Bilgisayar İŞLETİM SİSTEMLERİ

Bilgisayar İŞLETİM SİSTEMLERİ

Prof.Dr. Ali SAATÇİ

Hacettepe Üniversitesi
Bilgisayar Mühendisliği Bölümü

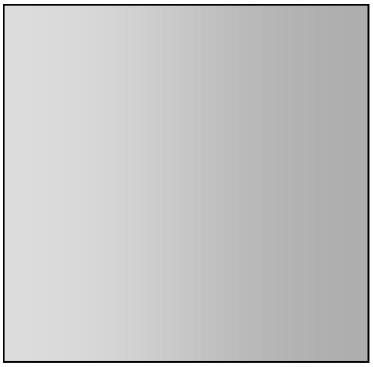
Genişletilmiş İkinci Baskı
Ankara, 2002

DEFNE ve YUNUS'a

Yazar Hakkında

Dr. Ali SAATÇİ, 1950 yılında Ankara'da doğdu. Orta öğrenimini Galatasaray Lisesinde tamamladı. Fransız Hükümetinin verdiği bursla Fransa'ya gitti. *Institut National des Sciences Appliquées de Lyon* adlı okuldan Elektrik Yüksek Mühendisliği diploması aldı. Aynı okulda, bilgisayarlı süreç denetimi konusunda doktora çalışmalarını tamamlayarak 1980 yılında yurda döndü. 1981 yılından başlayarak Hacettepe Üniversitesi, Bilgisayar Mühendisliği Bölümünde çalışmaya başladı. Bilgisayar Yapısı, Sistem Programlama, İşletim Sistemleri, Veri İletişimi, Bilgisayar Ağları, Ağ Güvenliği, Mikroişleyiciler konularında dersler verdi. Bilgisayar Donanımı anabilim dalında 1983'te Doçent, 1989 yılında da Profesör ünvanlarını aldı. 1986-1991 yılları arasında, Hacettepe Üniversitesi, Bilgisayar Mühendisliği Bölümünde Bölüm Başkanlığı yaptı. Dr. Ali SAATÇİ, halen aynı bölümde öğretim üyeliği ve Bilgisayar Donanımı Anabilim Dalı Başkanlığı görevlerini yürütmektedir.

İÇİNDEKİLER



İşletim Sistemleri konusu, bilgisayar bilimleri ve mühendisliği dallarının önemli bir konusunu oluşturur. Bilgisayar mühendisliği eğitimi, Yurdumuzda Hacettepe ve Ortadoğu Teknik Üniversiteleri tarafından, 1977 yılında başlatılmıştır. Bugün, hemen hemen tüm büyük üniversitelerimize yaygınlaşmıştır. Üniversitelerimizde yirmibeş yıldır bilgisayar bilimleri eğitimi yapılıyor olmasına karşın, bu dalın önemli konularında, birkaç istisna dışında, Türkçe kitap yayımlanmamıştır. Bunun temel nedeni, haklı ya da haksız, büyük üniversitelerimizde, yaygın bir biçimde İngilizce eğitim tercihinin yapılmış olmasıdır. Eğitim dilinin İngilizce olması Türkçe kitaba olan gereksinimi azaltmış ve kitap yazabilecek kişileri caydırmıştır. Ancak günümüzde bir yandan bilgisayar mühendisliğinde, diğer yandan da bilgisayar bilimlerinin temel konularıyla ilgili; başta elektrik, makina, endüstri, inşaat gibi mühendislik; fizik, matematik, istatistik gibi temel bilimler dallarında Türkçe eğitim yaygınlaşmıştır. Bu durum, Türkçe kitap gereksinimini artırmıştır. Bilgisayar İşletim Sistemleri adlı bu kitap, bu bağlamda kaleme alınmıştır.

Kitap, Hacettepe Üniversitesi Bilgisayar Mühendisliği Bölümünde, İşletim Sistemleri I ve İşletim Sistemleri II adlı dersler kapsamında, son yıllarda işlenen konuları içermektedir. Bu nedenle, öncelikle bilgisayar mühendisliği öğrencilerine yöneliktir. Ancak işletim sistemleri konusunda temel bilgiler edinmek isteyen diğer fen ve mühendislik öğrencilerinin de bu kitaptan, en üst düzeyde yararlanabilmeleri amaçlanmıştır. Bu bağlamda, kitabı oluşturan bölümler, birbirlerinden elverdiğince bağımsız bir biçimde kaleme alınmış, temel tanımlar, değişik bölümler içinde yinelenerek her bölümden ayrı ayrı yararlanma yolu açık tutulmaya çalışılmıştır. Bu kitaptan yararlanmak isteyen kişilerin, programlama ve bilgisayar yapısı konularında temel bilgilere sahip olmaları gerekmektedir.

Kitabın Giriş adlı ilk bölümde işletim sistemlerine ilişkin temel tanımlara, bilgisayar yapısının yalın bir modeline, işletim sistemini oluşturan değişik işlevsel kesimlerin kısa tanımlarına ve sistem çağrı kavramına derli toplu ve özlü bir biçimde yer verilmiştir. Kitabın ikinci bölümü Giriş/Çıkış Sistemi adını taşımaktadır. Bu bölümde giriş/çıkış

birimlerinin programlanmasında yararlanılan temel yaklaşımlar ve değişik yöntemler açıklanmıştır. Bilgisayar donanımına ilişkin ayrıntılara ilgi duymayan, alt düzey programlama kavramlarına yabancı okuyucuların bu bölümü tümüyle atlamalarında herhangi bir sakınca bulunmamaktadır. Kitabın Üçüncü ve Dördüncü Bölümleri Görev Yönetimi ve Birlikte Çalışan Görevler konularına ayrılmıştır. İşletim sistemlerinde görev yönetimi, iş yönetimi ve orta dönemli planlama ile görevler arası zamanuyumlama bu bölümlerde açıklanmıştır. Beşinci Bölüm tümüyle Ana Bellek Yönetimi adlı konuya ayrılmıştır. Sayfalama, kesimleme kavramları ile görüntü (sanal) bellek düzenine, bu bölümde yer verilmektedir. Altıncı Bölüm Kütük Yönetimi adını taşımaktadır. Ana bellek dışında saklanan verilerin bir yandan yalın mantıksal modeller çerçevesinde kullanılabilmesine diğer yandan da giriş/çıkış birimlerinin verimli kullanımına olanak veren düzenlemeler bu bölümde açıklanmaktadır. Yedinci bölüm işletim sistemlerinde Güvenlik ve Koruma konusuna ayrılmıştır. Verilerin bütünlüğünün, gizliliğinin ve kullanılabilirliğinin sağlanmasında kullanılan yaklaşım ve yöntemler bu bölümde yer almaktadır. Sekizinci Bölüm Aygıt Sürücüler konusuna ayrılmıştır. Aygıt sürücü, bir bilgisayar sisteminde belirli bir giriş/çıkış arabirim donanımını süren yazılıma verilen addır. Sisteme yeni bir arabirim katıldığında, bununla ilgili, çekirdek katman düzeyi sürücü yazılımın işletim sistemine nasıl eklendiği, *UNIX* bağlamında, bu bölümde açıklanmaktadır. Dokuzuncu Bölümde Dağıtımlı İşlem konu edilmektedir. Günümüzde bilgisayarlar bir ağ içinde yer almakta ve gerektiğinde kaynaklarını, istemci-sunucu yaklaşımı ile paylaşabilmektedir. Bu paylaşımın kuralları ve altyapısı, bu bağlamda, *TCP/IP* olarak anılan işletim sistemi ağ katmanı ve *socket* düzeneği bu son bölümde anlatılmaktadır.

Bu kitapta kullanılan bilişim terimleri, büyük oranda, Dr. Aydın KÖKSAL'ın, Türk Dil Kurumu'nca, 1981 yılında yayımlanan “Bilişim Terimleri Sözlüğü"ne uygun olarak seçilmiştir. Bu sözlükte yer alan bilişim terimleri, bilgisayar bilimlerine ilişkin temel kuramsal bir konuda, tutarlı bir Türkçe ile kitap yazabilmeye olanak vermiştir. Türkçe bilişim terimlerinin geliştirilmesine katkı veren ve bu alanda özen ve titizlik gösteren herkese minnettarım. Bu terimlerin yaşam çevriminde standartlaşarak yaygınlaşmasına katkı vereceğini umduğum bu kitabın yazılmasında yardımlarını esirgemeyen, başta Dr. Harun ARTUNER’e, tüm değerli meslektaşlarıma ve yazım hatalarının ayıklanmasında yardımcı olan tüm öğrencilerime teşekkürü bir borç bilirim.

Ali SAATÇİ
Ankara, (İkinci Baskı: Ekim 2002)

1. BÖLÜM

İ Ş L E T İ M S İ S T E M L E R İ

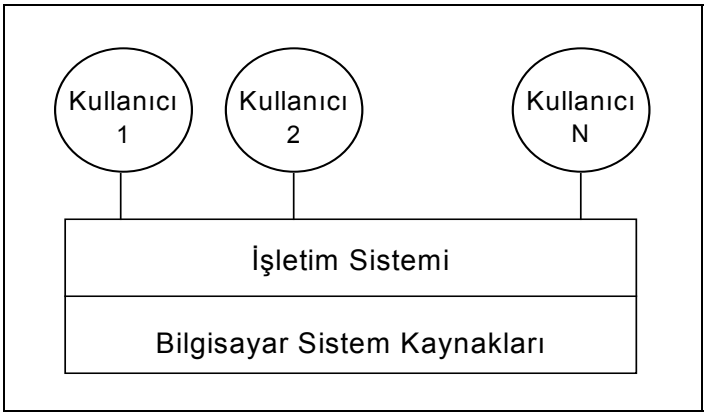
GİRİŞ

İşletim Sistemleri konusu, bilgisayar bilimleri içinde önemli ve kapsamlı bir konuyu oluşturur. Bu kapsamlı konu, doğal olarak, kendine özgü çeşitli özel tanım ve kavramları içerir. İşletim sistemleri konusunun gelişmesine koşut olarak ortaya çıkan bu özel tanım ve kavramların, başta, derli toplu ve özlü bir biçimde ele alınmasında yarar vardır. Bu amaçla, izleyen kesimde önce, işletim sisteminin genel tanımı ile kullanıcı yönünden konumuna değinilmiştir. Konumu gereği kullanıcı ve bilgisayar donanımı arasında ve donanıma en yakın bir yazılım katmanı olarak yer alan işletim sisteminin incelenmesi, çoğu kez bilgisayar donanımıyla ilgili kimi yapıların ve özel ayrıntıların bilinmesini gerektirir. Bu nedenle, konunun hemen başında, donanımla ilgili bilgilere başvurulması gerektiğinde yararlanılacak, bilgisayar yapısına ilişkin standart bir modele yer vermenin doğru olacağı düşünülmüştür. İşletim sisteminin genel tanımının yapıldığı kesimden sonra bu model açıklanmıştır. Bundan sonra, özlü olarak, program kavramının genişletilmiş bir biçimi olan iş kavramı ve bu kavrama ilişkin tek iş ve çok iş adlı işletim düzenleri tanımlanmıştır. İş kavramının ardı sıra, bu kez, program kavramına işletim boyutunu katan görev kavramı ve bununla bağıntılı çok görevli işlem söz konusu edilmiş; çok görevli işlem tanımına ek olarak toplu, etkileşimli ve gerçek zamanlı işlem türleri açıklanmıştır. Görev yönetimi, ana belleğin yönetimi, kütük yönetimi gibi, bir işletim sistemini oluşturan önemli alt kesimlere kısaca değinilmiştir. Bunun sonrasında

sistem çağrısı kavramı açıklanmış ve örneklenmiştir. Son olarak, kurum bilgisayar alt yapılarının ve çok kullanıcılı bilgisayar sistemlerinin evrimi.konu edilmiştir.

1.1. İşletim Sisteminin Genel Tanımı ve Konumu

Bilindiği gibi bilgisayar sistemleri donanım ve yazılım olarak adlandırılan iki temel birleşenden oluşur. Gözle görülür, elle tutulur yarı iletken yongalar, bunları taşıyan kartlar, görüntü ekranları gibi elektronik kökenli birimler ile disk, disket, manyetik şerit sürücüleri, tuş takımı ve yazıcılar gibi elektromekanik nitelikli ögeler bilgisayar donanımı olarak adlandırılırlar. Donanım ögelerinin anlamlı birlikteliğiyle ortaya çıkan bilgisayar sisteminin hizmet üretebilmesi programlanarak işe koşulmasıyla sağlanır. Bilgisayar donanımlarını, hizmet üretme yönünde denetleyen programlar, daha genel bir bağlamda yazılım diye adlandırılırlar.



Çizim 1.1. İşletim Sisteminin Konumu

Sorunların çözümünde bilgisayardan yararlanabilmek için, çözümlere ilişkin algoritmaları, bilgisayar sistemi üzerinde programlamak gerekir. Bu yolla oluşturulan bilgisayar programları, eldeki verileri, algoritmanın belirlediği yolla işleyerek sonuç üretirler. Verilerin işlem öncesi bilgisayar sistemine girilmesi, elde edilen sonuçların da kullanıcıya döndürülmesi, görüntülenmesi gerekir. Bu bağlamda programlar, sorunun çözümüne özgü ve verilerin giriş/çıkış işlemlerine özgü olmak üzere iki kesimden oluşuyormuş gibi düşünülebilir. Sorununun çözümüne özgü kesimde, örneğin, çözümün gerektirdiği kimi hesaplamalar, arama, sıralama gibi işlemler yerine getirilirken giriş/çıkış işlemleri kesiminde; verilerin, klavye, disk gibi birimlerden, fiziksel olarak okunması, yazıcılardan döküm alınması gibi işlemler gerçekleştirilir. Ortalama alma amacıyla geliştirilen bir programda, ortalamaya giren değerlerin toplanması, toplamın öge sayısına bölünmesi işlemleri sorunun çözümüne özgü işlemlerdir. Ortalamaya giren değerlerin klavyeden okunması, elde edilen sonuçların ekran, yazıcı gibi birimlerden görüntülenmesi ise giriş/çıkış işlemlerini oluşturur. Verilerin diskten okunması, ekranda görüntülenmesi, yazıcıdan dökülmesi gibi giriş/çıkış işlemleri, üst düzey programlama dilleriyle, birkaç komutla yerine getirilen işlemler olarak algılanır. Ancak derleme sonucunda elde edilen amaç programlar içinde, bu işlemlere ilişkin kesimler de önemli bir yer tutar.

Programlar içinde sorunların çözümüne ilişkin kesimler sorundan soruna farklılıklar gösterir. Giriş/çıkış işlemleriyle ilgili kesimler ise, donanımın fiziksel özelliklerine bağımlı ve bir bilgisayar sistemi için hep aynıdır. Örneğin, silindir, kafa ve sektör numaraları bilinen bir disk öbeğinin sürücüden okunması, tüm uygulama programlarınca, hep aynı biçimde yerine getirilmesi gereken bir işlemdir. Bu durumda programların, giriş/çıkış işlemlerine özgü kesimlerinin, aynı donanım üzerinde işletilen tüm programlar tarafından ortaklaşa kullanılabileceği söylenir. Ortaklaşa kullanılabilen bu kesimleri hazır yordamlar biçiminde, bilgisayar donanımlarıyla birlikte sağlamak, kullanıcıları bunların gerçekleştirim yükümlülüğünden kurtarır. Donanıma özgü karmaşık ayrıntıların bilinme zorunluluğu da, böylece ortadan kalkar. Bilgisayar sisteminin kullanımı, bu sistemden hizmet alan kullanıcılara daha yalın bir görünüm ve hazır işlevler çerçevesinde sunulabilir. Bu yolla program geliştirme süreçleri kısaltılarak kullanım kolaylığı sağlanır.

İşletim sistemleri, donanım birleşenlerinin sürülmesine dönük yordamları hazır işlevler olarak içerirler. Bu yolla kullanıcılara, bilgisayar sistemine dönük yalın bir görünüm ve kolay bir kullanım ortamı sunmayı amaçlarlar. Kullanım kolaylığı işletim sistemlerinin temel amaçlarından birini oluşturur. Salt bu amaç gözetilerek bir tanım vermek gerekirse: "İşletim sistemi, bilgisayar donanımı ile kullanıcı programları arasında yer alarak kullanıcıların bilgisayar sisteminden kolayca yararlanabilmelerini sağlayan hizmet yazılımı" olarak tanımlanır.

Ana işlem birimi, ana bellek, giriş/çıkış birimleri gibi, bir bilgisayar sistemini oluşturan donanım birleşenleri, kullanıcı programlarının işletilmesinde yararlanılan kaynaklar olarak da düşünülebilir. Bu bağlamda işletim sistemi, kaynak sözcüğünü de içerecek biçimde: "Kullanıcıların, bilgisayar sistem kaynaklarından kolayca yararlanabilmelerine olanak sağlayan hizmet yazılımı" olarak da tanımlanabilir.

Kullanıcılar bir bilgisayar sisteminden hizmet alırken donanımsal kaynakların yanı sıra yazılım nitelikli kaynaklardan da yararlanırlar. Örneğin üst düzeyli bir dil kullanarak program geliştiren bir kullanıcı, bunu yazabilmek için bir metin düzenleyiciye, programını makina dili programına dönüştürebilmek için bir derleyiciye, kimi hazır kitaplık yordamlarını kendi programına katabilmek için bir bağlayıcıya gereksinim duyar. Metin düzenleyici, derleyici, bağlayıcı gibi araçlar da sistem kaynakları arasında sayılırlar. İşletim sisteminin, donanımsal kaynakların yanı sıra yazılım nitelikli kaynakların kullanımına da olanak vermesi gerekir. Bu nedenle, yukarıdaki tanımda yer alan kaynak sözcüğünü, hem donanım hem de yazılım nitelikli kaynakları kapsar biçimde algılamak daha doğru olur (Çizim 1.1).

Kullanıcıların bilgisayar sistem kaynaklarından kolayca yararlanmalarını sağlama işlevi, işletim sisteminin yerine getirmekle yükümlü olduğu tek işlev değildir. Bu işlev, işletim sisteminin salt kullanıcılara dönük sorumluluğunu belirler. İşletim sisteminin bunun yanı sıra, sistem kaynaklarının verimli kullanımını sağlamak gibi, işletime dönük çok önemli bir diğer işlevi daha vardır. Kullanıcılar, sistemden, kolay kullanım olanaklarının yanı sıra hızlı bir işletim de beklerler. Sisteme sundukları programlarının hemen işletilerek sonlandırılmasını, bu amaçla gereksinim duydukları tüm kaynakların

kendilerine hemen sağlanmasını isterler. Ancak bilgisayar sistemlerinde kaynaklar kullanıcıların tüm istemlerini anında karşılayabilecek sayı ve sığada olamaz. Zira bu kaynaklar genelde pahalı kaynaklardır. Kaynakların kısıtlılığı, bunların kullanıcılar ya da programlar arasında bölüşülmeleri zorunluluğunu doğurur. İşletim sistemi, kısıtlı kaynakları programlar arasında, hem bu kaynakların verimli kullanımını, hem de hızlı bir işletimi gözeterek paylaştırmak zorundadır. Verimli kaynak kullanımı, kaynakların olabildiğince çok kullanıcı arasında bölüştürülerek boşta kaldıkları sürelerin en düşük düzeyde tutulmasını gerektirir. Bu yolla, birim zaman içinde, aynı sayıda kaynakla daha çok hizmet üretme olanakları yaratılır. Ancak programlar yönünden tek tek bakıldığında bu yaklaşımın, kaynak bekleme nedeniyle işletim hızlarını düşürdüğü görülür. Kaynak kullanım verimliliği ve program sonlandırma hızı gibi, genelde çelişebilen iki ayrı parametrenin, aynı anda eniyilenmesi, işletim sisteminin diğer temel ve zor işlevini oluşturur.

Bu durumda: "İşletim sistemi, bilgisayar sistemini oluşturan donanım ve yazılım nitelikli kaynakları, kullanıcılar (programlar) arasında kolay, hızlı ve nitelikli bir işletim hizmetine olanak verecek biçimde paylaştırırken bu kaynakların kullanım verimliliğini en üst düzeyde tutmayı amaçlayan bir yazılım sistemi" olarak düşünülebilir.

İşletim sisteminin, kolay ve hızlı kullanım, kaynak verimliliği gibi kıstasların dışında, bilgisayar ortamında saklanan bilgilerin, gerek bozulmalara, gerekse izinsiz erişimlere karşı korunmasının sağlanması gibi koruma ve güvenlikle ilgili başka işlevleri de vardır. Bu işlevler de tanım içine katıldığında: "İşletim sistemi, bilgisayar sistemini oluşturan donanım ve yazılım nitelikli kaynakları kullanıcılar (programlar) arasında kolay, hızlı ve güvenli bir işletim hizmetine olanak verecek biçimde paylaştırırken bu kaynakların kullanım verimliliğini en üst düzeyde tutmayı amaçlayan bir yazılım sistemi" olarak tanımlanabilecektir.

1.2. Bilgisayar Yapısının Gösteriminde Kullanılan Model

Daha önce de belirtildiği gibi, işletim sistemi, kullanıcı (ya da uygulama programı) ile bilgisayar sistem kaynakları arasında yer alan bir yazılım katmanıdır. Bu yazılımın incelenmesi, bu bağlamda denetimleri gerçekleştirilen donanım birleşenlerine ilişkin kimi özellik ve ayrıntıların bilinmesini gerektirir. Bu özellik ve ayrıntılar inceleneceği zaman, gereksiz ayrıntılardan soyutlanmış ancak fiziksel gerçekliği yeteri kadar yansıtan bir yapısal model kullanmak, bu model üzerinde yer alan birleşenlerin yalın ve açık tanımlarını vermek sözel anlatımı kolaylaştırmak ve bu anlatıma açıklık kazandırmak açısından yararlı olacaktır. Bu çerçevede, bilgisayar sistemi dendiğinde:

- Ana işlem birimi (AİB),
- Ana bellek ve
- Giriş/çıkış birimleri

olarak adlandırılan üç temel birleşenden oluşan bir bütün anlaşılacaktır. Bu bütün içerisinde tek bir ana işlem birimi bulunduğu, başka bir deyişle incelenen bilgisayar sisteminin tek işleyicili olduğu varsayılacaktır. İleride bu varsayımın, tek işleyicili

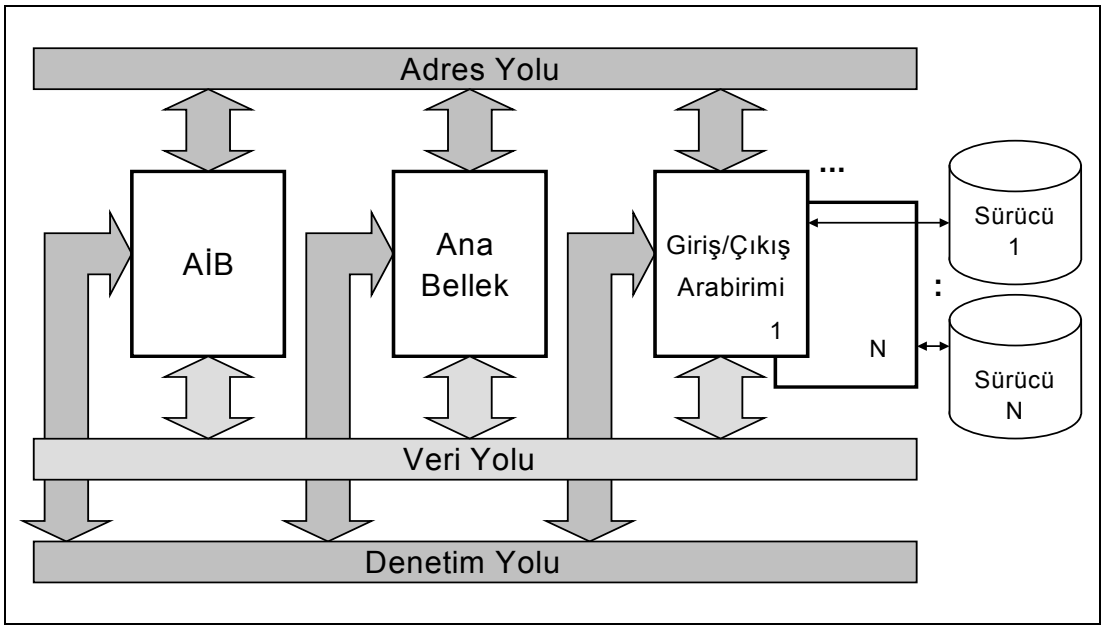
sistemlere dayalı ilkeleri çok işleyicili sistemlere uyarlamada önemli bir sınırlama getirmediği görülecektir. Ana işlem biriminin kendisi de:

- Yazmaç/Sayaç Takımı (kısaca yazmaç takımı),
- Aritmetik-Mantık Birimi (AMB) ve
- Denetim Birimi (DB)

olarak, üç alt birleşenden oluşan bir birim olarak modellenecektir. Ana bellek, sözcüklerden oluşan doğrusal bir dizi biçiminde düşünülecektir. Giriş/çıkış birimi,

- Giriş/çıkış sürücüsü (ya da aygıtı) ile
- Giriş/çıkış arabirimi

ikilisinin oluşturduğu bütünün adı olarak kullanılacaktır. Ana işlem birimi, ana bellek ve giriş/çıkış birimleri, bilgisayar sistem donanımını oluşturmak üzere bir yol yapısı çerçevesinde bütünleşeceklerdir. Bu bağlamda, giriş/çıkış arabirimleri, giriş/çıkış sürücülerinin ana işlem birimi ve ana bellek ikilisi ile bütünleşmesini sağlayan uyarlayıcı birimler olarak düşünülebilecektir (Çizim 1.2).



Çizim 1.2. Bilgisayar Yapısı Modeli

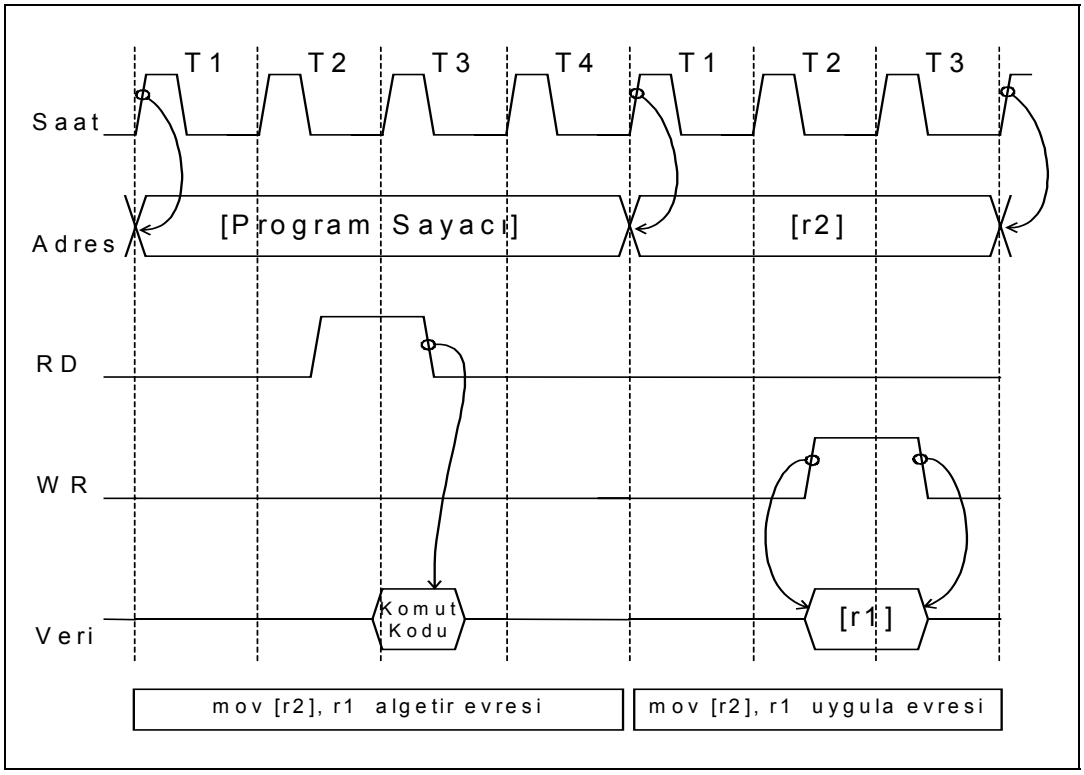
Yol, ortak işlevleri uyarınca gruplanmış im ileti hatlarına verilen addır. Bu bağlamda:

- Adres,
- Veri ve
- Denetim

yolları sözkonusu edilecektir. Adres yolu, ana işlem biriminin, okuma ya da yazma amacıyla erişmek istediği, ana bellek sözcüğü ya da giriş/çıkış arabirimi yazmacını belirlemede (adreslemede) kullandığı genelde tek yönlü hatlara; veri yolu, kimliği adres yolu üzerinde bulunan bellek sözcüğü ya da arabirim yazmaç içeriklerinin buralardan ana işlem birimi yazmaçlarına ya da ana işlem birimi yazmaçlarından buralara iletildiği

çift yönlü hat grubuna verilen addır. Denetim yolu ise, ana işlem birimi ile ana bellek ve giriş/çıkış arabirimleri arasındaki veri aktarımlarının zamauyumlanmasına yarayan imlerin iletildiği hat grubudur. Adres ve veri yolu üzerinden iletilen adres ve veri imleri bir bütün olarak anlamlıdır. Zira, adres ve veri yolunu oluşturan her hat, aynı anda, aynı adres ya da veri sözcüğünün değişik bit değerlerini taşır. Denetim yolunu oluşturan hatlar ise, saat, okuma - yazma, kesilme ve doğrudan bellek erişim istem, istem alındı imleri gibi, zaman içerisinde birbirlerinden bağımsız olarak gelişen imleri taşırlar.

Von Neumann ilkesine göre çalışan bir bilgisayar sisteminde, ana bellekte saklanan komutlar yorumlanarak ya ana bellekte ya da giriş/çıkış arabirim yazmaçlarında tutulan veriler üzerinde, temel aritmetiksel ve mantıksal işlemler gerçekleştirilir. Bunun için işlemin türünü belirleyen komutlara ve işlemin uygulanacağı verilere erişim gereklidir. Ana işlem biriminin temel etkinliğini oluşturan ana belleğe ya da giriş/çıkış arabirim yazmaçlarına erişimin mantığı, zaman çizeneği diye adlandırılan bir çizim yoluyla gösterilir. Zaman çizenekleri, bilgisayar donanımlarınca yürütülen alt düzey işlemleri açıklamada güçlü bir gösterim aracını oluştururlar.



Çizim 1.3. Örnek Zaman Çizeneği

Zaman çizenekleri, ana işlem biriminin gerek bellek sözcüklerine, gerekse arabirim yazmaçlarına erişimi sırasında etkili olan, adres, veri ve denetim nitelikli imlerin zaman eksenindeki çizimlerinden oluşur. Bu çizimlerde imlerin zamana göre değişen mantıksal düzeyleri gösterilir. Zaman çizenekleri üzerinde, tüm imlerin, mutlaka ayrı ayrı gösterimlerine yer verilmez. Adres ve verileri simgeleyen imler gibi birbirleriyle

bağımlı imler demetlenerek gösterilir. Bu durumda, çizim üzerinde, bu imlere ilişkin salt ortak düzey değişimlerine yer verilir. Zaman çizeneklerinde kullanılan zaman birimi, ana işlem birimi saat imi periyodudur (T_i). Bu nedenle, genellikle zaman çizeneklerinin en üst kesiminde saat imini simgeleyen bir kare im çizimine yer verilir. Zira bilgisayar sistemlerinde işlem akışı ve bu akışının gereği olarak ortaya çıkan im düzey değişimleri, bir saat imine uyumlu (zamanuyumlu) olarak gelişir.

Bu açıklamaları örneklemek üzere, Çizim 1.3' te, `mov [r2],r1` simgesiyle gösterilen, `r1` adlı yazmaç içeriğini `r2` adlı yazmacın gösterdiği adrese aktaran (yazan) örnek bir komutun, varsayımsal bir işleyici için zaman çizeneği verilmiştir. Bu zaman çizeneğinin ilk kesimi komut kodunun bellekten okunması (komut algetir evresi) sırasında, ikinci kesimi ise `r1` yazmaç içeriğinin `r2` yazmaç içeriğiyle adreslenen bellek sözcüğüne yazılması (komut uygula evresi) sırasında, adres, veri ve ilgili (RD: *read*/oku; WR: *Write*/yaz) denetim imlerinin değişimlerini gösteren çizimlerden oluşmuştur. Yukarıda belirtildiği üzere adres ve veri imleri, bunların bulunabileceği her iki olası düzeyi simgeleyen iki koşut doğru ile, demetli biçimde ve sadece değişimler gösterilerek çizilmiştir. Tek başına anlamlı olan RD ve WR gibi denetim imleri ise ayrı birer çizimle gösterilmiştir. İmlerim değişim anlarının diğer imler ve saat imi ile olan zaman uyumu eğri oklarla vurgulanmıştır. Ana bellek ve giriş/çıkış arabirimleri, ana işlem birimi ile iletişimlerinde, veri yolunu ortak kullanmak durumundadırlar. Bu birimler, veri yolunu salt ana işlem birimi buna izin verdiği sürece (bu birimleri seçtiği sürece) kullanabilirler. Bu süreler dışında, veri yolu ile bağlantılarını kopuk tutmak zorundadırlar. Bu nedenle, veri yolu üzerinde imler, salt aktarıldıkları süre boyunca ve kopuk kopuk gösterilirler. Aktarım anlarının dışında veri yolunun boş (yüksek empedans konumunda ya da birimlerden kopuk) tutulduğunu belirtmek için yarım düzeyli, düz bir çizgi kullanılır.

`mov [r2],r1` komutunun işletiminin ilk evresi olan algetir evresinde ana işlem birimi, adres yolu üzerine, işletilecek komutun adresini içeren program sayaç içeriğini koyar. Adres hatları üzerinde adres bilgisi varlığını sürdürürken, ikinci saat periyodunda, saat iminin düşen kenarıyla zamanuyumlu olarak RD denetim imini yükseltir. Bir saat periyodu süresince bekledikten sonra, RD iminin düşen kenarıyla zamanuyumlu olarak, veri yolu üzerinde bulunan veri değerini komut yazmacı içine aktarır. Bu işlem, ana işlem biriminin bellekten okuma yapma işlemidir. Algetir evresinin dördüncü periyodunda, komut yazmacı içinde yer alan komut kodunun çözümü (yorumlaması) yapılır. Komutun uygulaması, algetir evresini izleyen üç periyod içinde gerçekleşir. Bellekten okuma işleminde olduğu gibi, ana işlem birimi, adres yolu üzerine, bu kez `r2` yazmacının içeriğini koyar. Adres hatları üzerinde bu adres bilgisi varlığını sürdürürken, bu evreye ilişkin ikinci saat periyodunda, saat iminin düşen kenarıyla zamanuyumlu olarak, bir yandan WR denetim imini yükseltirken diğer yandan, veri yolu üstüne `r1` yazmaç içeriğini yükler. Bir saat periyodu boyunca bu durumda bekledikten sonra, WR imini düşürüp veri hatlarıyla olan bağlantısını koparır. Bir periyodluk bu süre içinde, ana işlem birimi, adreslenen bellek biriminin, veri yolu üzerinde hazır tutulan değeri (`r1` içeriğini), ilgili (r2 ile adreslenen) sözcük içine aktardığını varsayar. Bu işlem de, ana işlem biriminin belleğe yazma yapma işlemidir. Zaman çizeneğinin amacı, sözlü olarak

verilen bu uzun açıklamaları, görsel yolla, daha yalın, kısa ve standart bir biçimde aktarabilmektir.

1.3. İşin Tanımı

Kullanıcılar bilgisayar sistemlerinden ilgili programlarını çalıştırarak yararlanırlar. Bir programın çalıştırılması, işletim sisteminin, ileride göreceğimiz kabuk katmanınca yorumlanan, *run, execute* gibi kimi sistem komutları[1] aracılığıyla ya da sadece bu programın (derlenmiş biçiminin) saklandığı kütük adı verilerek sağlanır[2]. Kullanıcılar, kimi zaman ya değişik programları arka arkaya ya da aynı programı değişik veri takımlarıyla, belirli bir mantıksal sırada işletmek gereksinimini de duyabilirler. Örneğin *A* programını çalıştırmak, bu programın oluşturduğu *K1* kütüğünü *B* programı ile işlemek ve *K2* kütüğünü elde etmek istenebilir. Bu örnekte sıralanan işlemler bir işi oluşturur. Bu durumda iş, kullanıcıların sistemden bir bütün olarak ele alınmasını istedikleri işlem takımına verilen ad olarak tanımlanır. İş, program kavramını genişleterek içeren bir kavramdır. Bu bağlamda, tek bir program da, yine iş olarak adlandırılabilecektir.

Hemen hemen tüm işletim sistemleri, kullanıcılarına iş tanımlama olanakları sunarlar. Kişisel bilgisayarlara yönelik *MS-DOS* işletim sisteminde iş tanımlama, bu amaç için öngörülmüş kütükler kullanılarak yapılır. Kullanıcılar bir bütün olarak işletilmesini istedikleri komut ya da program adlarını, türü *.BAT* olan özel bir kütüğün içine, istedikleri işletim sırasında yazarlar. Tanımlanan iş, bu özel kütüğün adı verilerek çalıştırılır.

```
cd /usr/$altklv
sort < $kutuk1 > yenikut
cmp -s yenikut eskikut
       if test $? -eq 1
       then
         program < yenikut
       fi
```

Çizim 1.4. *UNIX*'te İş Tanım Kütüğü Örneği

UNIX işletim sisteminde de, *MS-DOS* kapsamında açıklanana benzer bir yolla iş tanımı yapılır. Kullanıcılar, gene bir bütün olarak işletilmesini istedikleri komut ya da program adlarını, metin türü bir kütüğün içine, istedikleri işletim sırasında yazarlar. İşletilir (X)

---

[1] İşletim sistemi komutları ya da kısaca sistem komutları, kullanıcıların program geliştirmek, işletmek, kütükleriyle ilgili kopyalama, listeleme, adlandırma, silme gibi kimi temel işlemleri yerine getirmek için kullandıkları özel hazır programlardır. Bu komutlar işletim sistemi tarafından yorumlanıp çalıştırılan komutlardır. Bu nedenle, işletim sisteminden işletim sistemine, gerek adlandırma gerekse kullanım mantığı yönünden ayrım gösterirler. İleride sistem komutu kavramına yeniden dönülecektir.

[2] *WINDOWS, X_WINDOWS* gibi kullanıcı grafik arayüzü sunan işletim sistemlerinde, programlar, ekranda bir ikon ile simgelenir. Bu durumda bir program, ekrandaki ilgili ikon tıklanarak da işletilir. Bir programın komut satırından adının girilmesi ile ikonunun tıklanması arasında, sonuç açısından herhangi bir ayrım sözkonusu değildir.

özelliği kazandırılan bu kütüğün adı verilerek tanımlanan iş çalıştırılır. Gerek *MS-DOS* gerek *UNIX* işletim sistemlerinde, bir kütük içinde sıralanan komutları, giriş kütüğü adı, seçenek numarası gibi işletim aşamasında sağlanacak parametrelerle donatmak ya da bu komutları, işlem akışı sırasında ortaya çıkan koşullara bağlı olarak gerçekleştirmek de olanaklıdır. Bu gerekçeye dayalı olarak, kimi zaman, bir programlama dili yetkinliğinde olabilen iş tanım ya da iş denetim dillerinden de söz edilir. Çizim 1.4 'te verilen iş tanımı içindeki, `if`, `fi` gibi komutlar *UNIX* iş tanım/denetim dili (*Shell* Programlama) komutlarıdır.

UNIX işletim sistemi ortamında, `usr/$altklv` adlı alt kılavuz altındaki `$kutuk` adlı kütüğün sıralanması, sıralı kütüğün `yenikut` içinde oluşturulması, `yenikut`'ün `eskikut`'le karşılaştırılması, bu kütükler aynı değilse `yenikut` ile program'ın çalıştırılmasını gerçekleştiren işle ilgili kütük içeriği Çizim 1.4 ' te verilmiştir. *UNIX* kuralları gereği `$` işareti ile başlayan değişken adları, işe, işletimin başlama aşamasında sunulacak parametreleri göstermektedir. Bu bağlamda, Çizim 1.4 ' teki örnekte, sıralanacak kütük adı ile bu kütüğün bulunduğu altkılavuz adı, işletim aşamasında işe aktarılacak parametreler biçiminde yazılmıştır. Sözkonusu iş çalıştırılırken iş adımlarını içeren kütük adının yanında, ilgili alt kılavuz adı ile sıralanacak kütük adı girilecektir. Çizim 1.4 ' te verilen `cmp` komutu, `-s` anahtarı ile çalıştırılarak karşılaştırma sonucunun değeri, bir alt satırda `?` ile belirtilen durum değişkeni içinde oluşturulmaktadır.

1.4. Tek İş Düzeni, Çok İş Düzeni Tanımları

Tek iş ve çok iş düzenleri bilgisayar sistemleri üzerinde yürütülen işletim düzenleridir. Bir bilgisayar sisteminde, aynı anda tek bir iş işletime alınabiliyor ise kurulan işletim düzenine tek iş düzeni denir. Bu düzen içinde tüm sistem kaynakları, aynı anda tek bir iş, dolayısıyla tek bir kullanıcı tarafından tüketilir. Tek iş düzeni, genel amaçlı bilgisayar sistemlerinin kısıtlı kaynaklara sahip olduğu ilk yıllarda kullanılmış, sonradan bilgisayar donanımlarının ucuzlamasına koşut olarak tüm kaynakların tek bir kullanıcıya adandığı kişisel bilgisayarlarda da geçerli olmuş bir işletim düzenidir.

Çok iş düzeni, tek iş düzeninin yetersizliklerini aşmak üzere ortaya çıkan bir düzendir. Bir bilgisayar sisteminde, birden çok iş, aynı anda işletime alınabiliyor ise kurulan işletim düzenine çok iş düzeni denir. Birden çok işin aynı anda işletime alınabilmesi, bir işin işletimi sonlanmadan diğer işlerin de işletimlerinin başlatılması demektir. Birden çok işin aynı anda işletime alınabilmesi, sistem kaynaklarının bu işler arasında, eşzamanlı olarak paylaştırılmasını gerektirir. Tek iş düzeninden çok iş düzenine, sistem kaynaklarını işler arasında paylaştırarak, bir yandan bu kaynakların kullanım verimliliğini, diğer yandan, birim zaman süresi içinde daha çok iş sonlandırıp işletim hızlarını artırmak amacıyla geçilir.

Bir bilgisayar sistemini oluşturan ana işlem birimi, ana bellek ve giriş/çıkış birimleri, işlem hızları yönünden ele alındıklarında farklılıklar gösterirler. Bu birimler üzerinde işlem hızlarından söz edebilmek için, öncelikle bu birimlere özgü temel işlemlerin tanımını vermek gerekir. Bilgisayar sistemini oluşturan donanım birleşenlerinin temel işlemleri dendiğinde bu birimlerin bir seferde gerçekleştirebildikleri, alt adımlara

bölünmeden (atomik olarak) ele alınan işlemler anlaşılır. Bu bağlamda, ana işlem birimi üzerinde yürütülen temel işlem komut işletimi, ana bellek üzerinde yürütülen temel işlem sözcüğe erişim, giriş/çıkış birimi üzerinde yürütülen temel işlem ise ilgili sürücü düzeyinde saklanan (fiziksel bir disk öbeği gibi) bir birimlik bilgiye erişim işlemidir. Ana işlem birimi ve ana bellek üzerinde yürütülen temel işlemler, hızları açısından büyük ayrım göstermezler[3]. Zira Çizim 1.3'te yer alan zaman çizeneğinden de görüleceği gibi, komut işletimi birkaç bellek erişiminden oluşmaktadır. Giriş/çıkış birimleri üzerinde yürütülen temel işlem hızları ise, ana işlem birimi ve ana bellek üzerinde yürütülen temel işlem hızlarına göre çok düşüktür. Bir disk sürücüden bir öbeklik bilginin okunması için gerekli ortalama süreyi, bir makina komutunun ortalama işletim süresiyle karşılaştırmak, bu ayrımı örneklemek için yeterlidir.

3600 *rpm* (döngü / dakika) hızında dönen, 12 ms'lik *(average seek time)* ortalama yatay erişim süresine sahip, her izinde 512 baytlık 63 sektör bulunan bir disk sürücüde, iki sektörden oluşan bir öbeğe[4] ortalama erişim süresi:

$t_o = t_y + t_d + t_a$ olarak hesaplanır.

Burada t_y ortalama yatay erişim süresini,
t_d ortalama döngüsel gecikme süresini,
t_a ise bir öbeklik bilginin sürücü-arabirim arası aktarım süresidir[5].

Bu durumda:

$t_y = 12$ ms;
$t_d = (1/60)(1/2) = 8{,}3$ ms;
$t_a = (1/60)(2/63) = 0{,}06$ ms olduğundan

t_o yaklaşık 20 ms olarak bulunur.

500 MHz'lik saat imi ile çalışan ve makina komutlarını 1 saat periyodu[6] içinde işletebilen bir ana işlem birimi için komut işletim süresinin, $(1 / 500) \times 10^{-6}$ saniye, ya da 2 nanosaniye (ns) olduğu hesaplanır.

---

[3] Ana işlem birimi ile ana bellek arasında da hız ayrımı sözkonusudur. Ana bellekte bir sözcüğe erişim için gerekli sürede, ana işlem birimi, onlarca/yüzlerce komut işletebilir. Bu nedenle, günümüz ana işlem birimlerinde, *pipe line* tekniği ile, belleğe erişim işlemi (örneğin al getir evresi), ana işlem birimi içi, özel bir işlevsel birim aracılığıyla, komut uygula evresiyle koşut yürütülür.

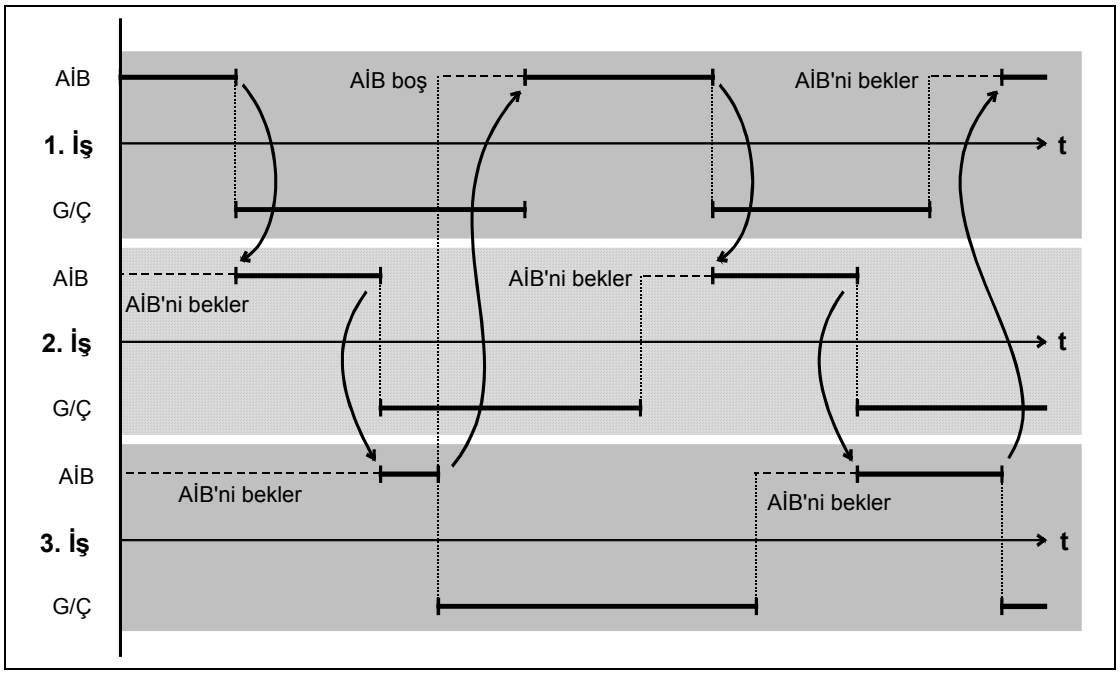
[4] Bilindiği gibi bir disk sürücü bir ya da birkaç dönen plakadan oluşur. Plakalar üzerinde bilgi içiçe izler biçiminde kaydedilir. İzler belirli sayıda sektöre bölünür. Giriş/çıkış biriminden bir seferde okunup yazılabilen veri birimi kısaca öbek olarak adlandırılmıştır.

[5] Sürücüden okunan bilginin arabirime aktarılmasından sonra arabirimden de ana belleğe aktarılması gerekir. Giriş/Çıkış Sistemi adlı bölümde açıklanacağı gibi, bu aktarımların süreleri erişim süreleri toplamına katılmayacak kadar küçüktür.

[6] Ana işlem biriminin çalışma ilkesi düşünüldüğünde (Çizim 1.3), bir saat periyodunda bir makina komutunun işletilmesi fiziksel olarak mümkün değildir. Ancak, yukarıda anılan *pipe line* tekniği ile, komut işletiminin alt evreleri, ana işlem birimi içi, özel işlevsel birimler aracılığıyla, koşut işletilerek bu gerçekleştirilebilir. Bu durumda *scalar* nitelikli ana işlem birimlerinden söz edilir.

Disk ile ana işlem birimine ilişkin temel işlem süreleri karşılaştırıldığında (2ns / 20ms) gibi, bire on milyonluk bir oran elde edilir. Başka bir deyişle, bir öbeğin disk sürücüden okunma süresinde ana işlem birimi 10 milyon makina komutluk bir programın işletimini tamamlayabilmektedir. Günümüzde işleyicilerin 2-3 GHz'lik saat imleri ile çalıştığı, disk sürücülerin erişim sürelerinin ise, yukarıdaki örnekte verilenin en çok yarısına düştüğü düşünüldüğünde, hız ayrımının, daha da derinleştiği söylenebilecektir.

Giriş/çıkış sistemi incelenirken görüleceği üzere, bir öbek bilginin disk sürücüden okunabilmesi için, bu sürücünün bağlı olduğu arabirim düzeyinde yer alan ve genellikle komut yazmacı olarak anılan yazmaç içine, okuma işlemini belirten bir kod değerini yazmak, bunun sonrasında arabirimin, okuma işlemini tamamlayarak ana işlem birimini, bir biçimde uyarmasını beklemek gerekecektir. Bu durumda, tek iş düzeninin uygulandığı bir sistemde, ana işlem birimi, program işletimini sürdürebilmek için ilgili öbeğin ana belleğe aktarılmasını beklemekten başka birşey yapamayacaktır. Başka bir deyişle, ana işlem birimi, milyonlarca komut işletebildiği bir süre boyunca bekler (boş) durumda tutulacaktır.



Çizim 1.5. Üç Değişik İşin Birlikte İşletimi

Bilgisayar sistemi donanım birleşenleri arasında en önemli kaynağı oluşturan ana işlem biriminin daha verimli kullanılabilmesi, bu birimin bekler durumda kaldığı süreleri kısaltmakla olanaklıdır. Bu sürelerin kısaltılması aynı anda birden çok işi işletime almak yoluyla gerçekleştirilir. Anımsanacağı gibi, aynı anda birden çok işin işletime alınması, bir işin işletimi sonlanmadan diğer işlerin de işletimlerinin başlatılması demektir. İşletimi sürdürülen bir işin giriş/çıkış işlemi başlatması durumunda, ana işlem biriminin

boş kalmasını önlemek üzere yeni bir iş işletime alınarak çok iş düzeni kurulur. Çizim 1.5'te, bu yolla, 3 değişik işin birlikte işletimi örneklenmiştir.

Herhangi bir anda, işletimi birlikte sürdürülen iş sayısı, sistemin o anki çok iş düzeyini belirler. Bu düzey, bilgisayar sisteminin, kaynak kullanım ve verilen işletim hizmetinin niteliği yönlerinden ölçülen başarımını önemli ölçüde etkiler. Bu düzeyin belirlenmesi Görev Yönetimi adlı konu başlığı altında açıklanan ve İş Yönetici olarak anılan işletim sistemi kesiminin temel işlevini oluşturur. Bu yönetim kesimi, aynı anda işletime alınan iş sayısını belirlemenin ötesinde işletime alınacak işlerin ana işlem birimi ve giriş/çıkış birimlerine yönelik istemleri açısından da dengeli biçimde harmanlanmalarını sağlar. Ana işlem birimi kullanımı ağırlıklı işlerle (örneğin yoğun ana işlem birimi kullanımı gerektiren görüntü işleme türü işlerle), giriş/çıkış birimleri kullanımı ağırlıklı işlerin (örneğin kütükler üzerinde günleme yapan işlerin) birlikte ele alınmaları, aynı anda hep aynı tür kaynaklara talep yaratarak kaynak darboğazına ve sistem başarımının düşmesine neden olabilecektir. Başka bir deyişle, işlerin, ana işlem birimine gereksinim duyduğu anların çakışması, ana işlem birimini daha uzun süreler beklemelerine; giriş/çıkış sürücülerine erişim gereksinimi duyduğu anların çakışmasının artması ise ana işlem biriminin daha uzun süreler boşta kalmasına neden olacaktır.

Bunu örneklemek üzere, Çizim 1.5'in ilk kesiminde, işlerin, ana işlem birimi kullanım sürelerinin kısa sürdüğü varsayılmış, bu durumda ana işlem biriminin, işlerin giriş/çıkış işlemlerinin tamamlanmasını beklemek üzere boşta kaldığı; çizimin ikinci kesiminde ise, bunun tersine, işlerin, ana işlem birimini uzunca süreler ellerinde tuttukları varsayılarak işlerin giriş/çıkış işlemleri sonlandıktan sonra ana işlem biriminin boşalmasını bekledikleri durum örneklenmiştir.

1.5. Görev ve Çok Görevli İşlem

Görev herhangi bir programın işletimine verilen addır. Program durgun komut dizisini tanımlarken görev bu komut dizisini işletim boyutuyla ele alan bir kavramdır. Yukarıda sözü edilen çok iş düzeninin uygulandığı bilgisayar sistemlerinde, aynı anda birden çok işin ele alınabilmesi, sistem kaynaklarının değişik işler ya da programlar arasında paylaştırılmasını gerektirir. Ana işlem biriminin de, önemli bir kaynak olarak programlar arasında paylaştırılması, işletilmekte olan bir programın kesilerek diğer bir programın işletime alınmasını gerektirir. İşletimi kesilen programın işletimine, ileride kalınan yerden devam edilebilmesi ve işletim bütünlüğünün korunabilmesi için işletimin kesildiği konuma ilişkin durum bilgilerinin saklanması gereklidir. Bu amaçla, işletilen her program için, bu bilgilerin saklandığı bir veri yapısı öngörülür. İlgili programın işletiminin her kesilişinde program sayacı, yığıt sayacı gibi programın kullandığı ana işlem birimi yazmaç içerikleri, programın saklandığı kütük kimliği, programca açılmış kütüklerin bulunduğu altkılavuz kimlikleri gibi bilgiler bu veri yapılarına saklanır. Bir program işletime alınacağı zaman ise ana işlem birimi yazmaç içerikleri ve diğer işletimle ilgili değişkenler bu bilgilerle günlenerek işletimin kalınan yerden sürdürülmesi ve işletim bütünlüğünün korunması sağlanır.

Böylece, programlar komut satırları olarak değil de, işletim bilgilerinin tutulduğu, görev iskeleti, denetim öbeği gibi adlarla anılan veri yapılarıyla ele alınmış olurlar. Sistem yönünden bakıldığında birlikte işletilen komut dizileri (programlar) yerine değişik görev iskeletleri yada denetim öbekleri arasında anahtarlanan bir ana işlem birimi söz konusu olur. Programın, işletim aşamasında, iskelet adlı veri yapısı ile ele alınan biçimine görev adı verilir. Görev, özellikle, ana işlem biriminin yönetimi açısından önemli ve temel bir kavramdır. Bu kavrama, Görev Yönetimi adlı konu başlığı altında, ayrıntılı olarak yeniden dönülecektir.

Çok görevli işlem, herhangi bir kullanıcının aynı anda birden çok görev tanımlayarak işletebilmesine olanak veren bir işlem türüdür. Çok iş düzeni için verilen tanımın çağrıştıracağı üzere çok görevli işlem birden çok görevin aynı anda çalışmasına olanak veren işlem türü olarak düşünülmemelidir. Bir bilgisayar sisteminde aynı anda birden çok görevin, işletim sisteminin denetiminde çalıştırılması, bu sistemde çok görevli işlemin yapıldığını söylemeye yetmez. Bu, çok iş düzeninin kurulmasında işletim sisteminin başvurduğu bir yoldur ve kullanıcılar yönünden saydamdır. Zira çok görevli işlemin yapılmadığı sistemlerde de, kimi özel sistem görevleri ile sistem tarafından tanımlanmış kullanıcı görevleri birlikte çalışmak zorundadır. Bu nedenle, bir sistemde kullanıcılar, aynı anda, bizzat, birden çok görev tanımlayıp işletimlerini birlikte başlatabiliyorlarsa, ancak o zaman bu sistemde çok görevli işlemin yapıldığı söylenebilir. Çok görevli işlem, işletim sisteminin kullanıcılara sunduğu bir olanak, bir işlem türü olarak algılanmalıdır. Bu bağlamda, *MS-DOS* işletim sistemi[7] kapsamında aynı anda birden çok görev çalışıyor olabilmesine rağmen kullanıcılara, aynı anda birden çok programı birlikte çalıştırma (çok görevli işlem) olanağı sunulmamaktadır. Başka bir anlatımla, *MS-DOS*'ta bir program ya da bir sistem komutunun işletimi sonlanmadan, ekranda imleci yeniden bularak yeni bir program ya da komut işletimi başlatma olanağı bulunmamaktadır. *UNIX* ve uyumlu işletim sistemleri ise çok görevli işleme olanak veren sistemlerdir. Örneğin bu işletim sistemlerinde birden çok görev tanımlayıp *background* adı altında, bunları birlikte çalıştırma olanağı bulunmaktadır. *MS-DOS*'tan türeyen *Windows* işletim sistemi de, çok görevli işleme olanak sunan bir işletim sistemidir.

1.6. Toplu İşlem ve Etkileşimli İşlem

Toplu işlem ve etkileşimli işlem, çok iş düzeninin kurulduğu bilgisayar sistemlerinde, aynı anda işletime alınan işlerin ele alınış biçimine verilen adlardır. Toplu işlemde (işletimde) işler sisteme, biriktirilerek, dönem dönem sunulurlar. Sisteme sunulan işler, sunuş anından başlayarak sonlanıncaya kadar kullanıcının her türlü müdahalesine kapalı biçimde işletilir. İşi oluşturan adımlar bir bütün olarak ele alınıp topluca işletilirler. Kullanıcılar adımlar arasında işletim akışını izleme ve denetleme (durdurma) olanaklarına sahip değildirler.

---

[7] *MS-DOS* işletim sistemi, günümüzde en yaygın kullanılan *Windows* işletim sisteminin atasıdır. *Windows* ortamında *Command Prompt* seçeneği ile kullanımı örneklenebilir.

Toplu işlem bilgisayar sistemlerinin yaygın biçimde kullanılmaya yeni yeni başlandığı (altmışlı) yıllardan kalan bir işletim türüdür. Bu yıllarda kullanıcılar delikli kart destelerinden oluşan programlarını, sistem işletmenine sunarlar ve bunların kart okuyucu üzerinden okutularak işletilmesini isterlerdi. Sunulan bu kart desteleri biriktirilerek dönem dönem okutulup sisteme sırayla sunulurdu. Program kart desteleri, çoğu kez, sistem işletmenine sunulmak üzere bilgi işlem merkezinde, bu amaçla öngörülmüş kutulara bırakılıp kutular doldukça okutulup sisteme sunulduğundan bu işlem türüne İngilizce'de *batch processing* deyimi kullanılmıştır. Günümüz bilgisayar sistemlerinde toplu işlem dendiğinde, işlerin sisteme, bu amaçla öngörülmüş işletim sistemi bekleme kuyrukları üzerinden sunulduğu işlem türü anlaşılmaktadır.

Kullanıcıların iş adımlarını adım adım işletebildiği, adımlar arasında işletim akışını izleyebildiği ve işletim akışına müdahale edebildiği işlem türü etkileşimli işlem olarak bilinir. Etkileşimli işlemde kullanıcılar sistemle terminaller aracılığıyla etkileşim kurarlar. Etkileşimli işlem, kullanıcılara sistemden yalnız başına yararlanıyorlarmış izleniminin verildiği bir işlem türü olarak da tanımlanabilir. Bu işlem türünde ana işlem birimi, kısa zaman dilimleri içinde (örneğin her 10 ms'de bir) kullanıcılara sırayla dönerek atanır. Bu zaman dilimlerinin bir ya da birkaçında, terminalleri başında oturan kullanıcıların, terminallerinden girdikleri işletim komutları çalıştırılabildiğinden kullanıcılar, komutlarına anında yanıt alırlar. Böylece sistem tümüyle kendileri için çalışıyormuş izlenimini edinirler. Bu izlenim doğal olarak bilgisayar işlem hızının insanınkine göre çok yüksek olmasından kaynaklanır. Etkileşimli işlemde, işleri oluşturan adımlar, kullanıcının sunuş sırasında işletildiğinden işletim akışının izlenmesi sağlanır. Ayrıca işletim, toplu işlemin tersine kullanıcının denetiminde istenen herhangi bir anda, terminallerdeki özel denetim tuşları kullanılarak kesilebilir. Etkileşimli işlem, ana işlem biriminin kullanıcılar arasında kısa sürelerle paylaştırılıyor olması nedeniyle zaman paylaşımlı işlem olarak da adlandırılır.

Çok kullanıcılı bilgisayar sistemlerinde toplu işlem ve etkileşimli işlem birlikte kullanılan işlem türlerini oluştururlar. Genelde kullanıcılar program geliştirme, metin düzenleme, program derleme ve sınama, veri tabanı sorgulama gibi kısa süreli işlerini etkileşimli ortamda gerçekleştirirken daha karmaşık, birden çok sayıda alt adımdan oluşan ve uzun süren işlerinin işletimi için toplu işleme başvururlar. Toplu işlem, alt adım akış sırası değişmeyen, bu nedenle adımlar arası işletim akış denetimine gerek duyulmayan işler için, kullanıcıyı terminali başında, iş adımlarını izleme ve ardarda girmenin getireceği bekleme zahmetinden de kurtarır.

İşletim sistemi, genelde, etkileşimli işlem ortamında sunulan (kısa sürede sonlanacağı varsayılan) işleri toplu işlem ortamında sunulan işlere göre daha öncelikli olarak ele alır. Sisteme toplu ya da etkileşimli işlem ortamlarında sunulan işler görevlere dönüştürülür. Etkileşimli işlem ortamında sunulan işler doğrudan görevlere dönüştürülüp ana işlem birimine anahtarlanmak üzere, hazır görevler kuyruğu diye adlandırılan görev bekleme kuyruğuna bağlanırlar. Toplu işlem ortamında sunulan işler ise, görevlere dönüştürülmeden önce toplu işlem kuyrukları üzerinde bekletilirler. İş yönetimi olarak anılan işletim sistemi kesimi, kuyruk başında bekleyen işleri, dönem dönem görevlere dönüştürerek hazır görevler kuyruğuna ekler. Toplu işlem ortamında sunulan işler,

etkileşimli işlem ortamında sunulan işlerden arda kalan sürelerde sistem kaynaklarını dolu tutmak ve bu yolla kaynak kullanım verimliliğini artırmak amacıyla işletime alınırlar. Bu nedenle kimi zaman işletim sistemi, dengesiz ve aşırı kaynak talep eden işleri toplu işlem kuyruklarından sunulmaya da zorlayabilir. Bu noktaya, Görev Yönetimi adlı konu altında da değinilecektir.

1.7. Gerçek Zamanlı İşlem

Yukarıda açıklandığı gibi, etkileşimli işlem, bilgisayar sisteminden terminaller aracılığıyla hizmet alan kullanıcıların işlettikleri komutlara, sistemden yalnız yararlanıyormuşçasına hızlı yanıtlar alabildikleri bir işlem türüdür. Etkileşimli işlem ortamında çalışan herkesin tanık olabileceği gibi, sistemin çok yüklü olduğu (örneğin sistemden yararlanan kullanıcı sayısının aşırı arttığı) durumlarda terminallerden girilen komutlara alınan yanıtlar, tek başına kullanım izlenimini silecek biçimde gecikebilmektedir. Etkileşimli işlemde, sistemin yanıt süresine bir üst sınır konabilmesi durumunda yapılan işlem türüne gerçek zamanlı işlem denir. Burada söz konusu üst sınır, doğal olarak saniyeler / milisaniyeler düzeninde düşünülmelidir. Gerçek zamanlı işlem, terminaller aracılığıyla hizmet alınan çok kullanıcılı bilgisayar sistemlerinden çok, yanıt süresinin üretilen hizmetin nitelik ve güvenliği yönünden çok kritik olduğu, endüstriyel süreç denetimi gibi özel uygulamaların yapıldığı bilgisayar sistemlerinde kullanılan bir işlem türüdür. Bu tür bilgisayar sistemlerinde, terminallerden girilen (sunulan) hizmet komutları yerine ısı, basınç, debi algılayıcıları gibi elektronik ölçüm birimlerinden sisteme dolaysız ulaşan ölçüm ve uyarılar sözkonusudur. Bu ölçüm ve uyarıların gerektirdiği (vanaların açılıp kapanması, alarmların kurulması gibi) güdümler sistemin yanıtlarını oluşturmaktadır. Yanıt süresi dendiğinde bu tür denetim nitelikli eylemlerin yerine getiriliş hızı anlaşılmalıdır. Buradan, sistem yanıt hızının, niçin bir üst sınırla garanti altına alınan kritik bir değer olduğu daha kolayca anlaşılacaktır.

Gerçek zamanlı işlem, genelde, çok görevli işlemle birlikte anılan bir işlem türüdür. Zira çok görevli işlem, gerçek zamanlı işlemin gerçekleştirilmesinde başvurulması gereken bir yöntemdir. Gerçek zamanlı uygulamalarda, işlemler çoğu kez, sisteme dış ortamdan gelen uyarılarla ele alınırlar. Bu uyarıların her biri için bir ya da birkaç görev tanımlanır. Bu görevler, ilgili oldukları uyarıların yanıt sürelerine dayalı öncelik sırasında ana işlem birimine anahtarlanırlar. Uyarılarla ilgili görevlerin kolayca tanımlanabilmesi ve tanımlanan bu görevlerin önceliklerine göre ana işlem birimini paylaşmaları çok görevli işlem ortamını gerektirir. Bu nedenle gerçek zamanlı işlem, genellikle çok görevli işlemle birlikte anılır ve gerçekleştirilir.

Gerçek zamanlı nitelemesi çoğu kez çevrim içi deyimi ile aynı anlama gelecek biçimde kullanılır. Çevrim içi, bilişim uygulamalarında verilerin sisteme sunuluş biçimini tanımlayan bir terimdir. Eğer işlenecek veriler bilgisayar sistemine dolaysız ve aracısız bir biçimde aktarılıyor ise yapılan uygulamanın çevrim içi bir uygulama olduğu söylenir. Örneğin bankacılık uygulamalarında müşteriler tarafından gerçekleştirilen para çekme, para gönderme gibi değişik bankacılık işlemlerine ilişkin veriler, telefon hatları aracılığıyla, doğrudan uygulamanın yürütüldüğü bilgisayar sistemine ulaşıyorsa (dolayısıyla hemen işleme konabiliyorsa) yapılan uygulamanın çevrim içi bir uygulama

olduğu söylenir. Öğrencilerin, derslere ya da sınavlara özel (optik) formları doldurarak kayıt oldukları bir uygulamada ise, bilgilerin, doldurulan formların toplanması ve optik okuma işlemleri sonrasında bilgisayar sistemine dolaylı olarak aktarılması nedeniyle çevrim içi bir uygulamadan söz edilemez.

Endüstriyel süreç denetimi gibi gerçek zamanlı uygulamalarda da, uygulamanın gerektirdiği veriler sisteme, örneğin elektronik algılayıcılar tarafından dolaysız olarak aktarılır. Zira bu uygulamalarda zaman çok önemlidir. Verilerin sisteme dolaylı bir biçimde aktarılması hoşgörülemez. Bu nedenle endüstriyel süreç denetimi ile örneklenen gerçek zamanlı uygulamalar her zaman çevrim içi uygulamalardır. Ancak bir uygulamanın gerçek zamanlı uygulama olması, verilerin sisteme aktarılış biçiminden kaynaklanan bir özellik değildir. Bir uygulamayı gerçek zamanlı kılan, daha önceden de belirtildiği gibi, yerine getirilen işlem yanıt hızıdır. Gerçek zamanlı uygulamalar, aynı zamanda çevrim içi uygulamalar olabilirken her çevrim içi uygulama, mutlaka gerçek zamanlı bir uygulama olmak zorunda değildir. Bu nedene dayalı olarak, gerçek zamanlı ve çevrim içi terimlerinin ayrı anlamlar içeren terimler olarak düşünülmesi ve bu kavramlar arasında ayrım gözetilmesi gerekmektedir.

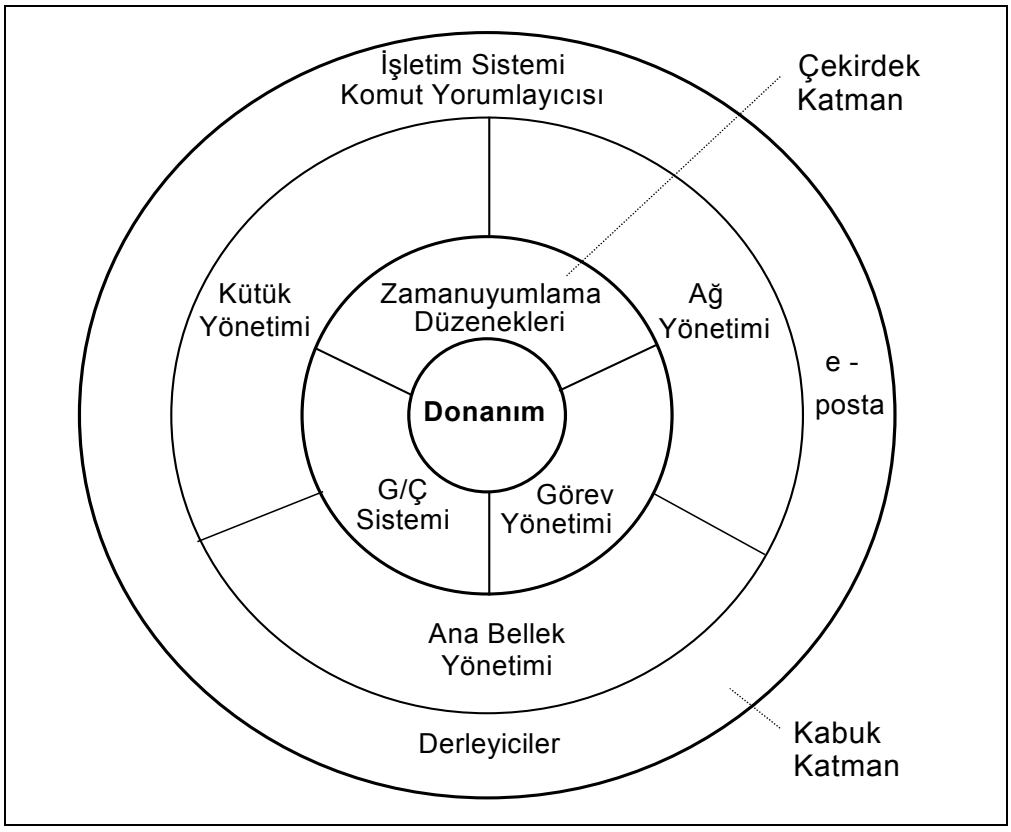
1.8. İşletim Sistemini Oluşturan Kesimler

İşletim sistemi, bilgisayar sistemini oluşturan donanım ve yazılım nitelikli kaynakları kullanıcılar arasında kolay, hızlı ve güvenli bir işletim hizmetine olanak verecek biçimde paylaştıran, bunu yaparken, bu kaynakların kullanım verimliliğini en üst düzeyde tutmayı amaçlayan bir yazılım sistemidir. Bu bağlamda, işletim sisteminin, sistem kaynakları olarak ana işlem biriminin, ana belleğin ve giriş/çıkış birimlerinin, kullanıcılar arasında verimli paylaşımını sağlayan işlevleri, ana işlem biriminin yönetimi, ana belleğin yönetimi, kütük yönetimi gibi adlarla anılan değişik kesimler içinde toplanır. İşletim sisteminin yerine getirdiği işlevler, çoğu kez yönetim işlevleri olarak tanımlanır. Yönetim sözcüğü Türkçe'de, daha çok insanlar için kullanılan bir sözcük olmakla birlikte burada, anlamı, kaynakları belirlenen amaca en yüksek verimi elde edecek biçimde yönlendirme biçiminde genişletilerek kullanılmıştır.

İşletim sisteminin, kolay kullanım ilkesi gereği, özellikle sistem giriş/çıkış birimlerinin kolayca algılanabilir, yalın mantıksal modeller çerçevesinde düşünülerek kullanılabilmelerine olanak vermesi de gerekir. Örneğin disk, mıknatıslı şerit gibi ikincil belleklerde saklanan verilerin, kullanıcılara, bayt, tutanak dizileri gibi görünümler altında sunulması, bunlar üzerinde yapılacak okuma yazma gibi işlemlerin de bu yalın modele uyumlu işlemler olarak öngörülmesi gerekir. Giriş/çıkış birimleriyle ilgili bu temel işlev, işletim sistemlerinde kütük yönetimi olarak anılan kesimce sağlanır. Bunun gibi, kullanıcıların çalıştıkları bilgisayar sisteminin bağlı olduğu diğer bilgisayar sistemlerinin kaynaklarına, örneğin o sistemlerde saklanan kütüklere, ayrıntılardan arındırılmış yapılar çerçevesinde erişebilmelerini ağ yönetimi kesimi sağlar.

İşletim sistemleri, yukarıda da belirtildiği gibi, bilgisayar sistemlerinin kolayca kullanılabilmelerine olanak sağlamak üzere, sistem birleşenlerini, kullanıcıların kolayca algılayabilecekleri yalın mantıksal yapılar olarak sunmaya çalışırlar. Örneğin

kullanıcılara kütüklerini, kendi uygulamalarıyla uyumlu uzunluk ve sayıda tutanaktan oluşan bir tutanaklar dizisi olarak düşünme ve ilgili işlemleri bu yapıya uyumlu olarak yürütebilme olanağı verilir. Kullanıcının düşüncesinde gerçekliği olan bu yapılar mantıksal yapılar olarak adlandırılır. Disk ortamında saklanan kütüklerin, fiziksel olarak, değişik disk yüzeylerinin, değişik izlerine dağılmış değişik sektörler üstünde saklandığı bilinir. Kolay kullanım yönünden, donanımın fiziksel ayrıntısının kullanıcılardan tümüyle gizli tutulması gereklidir. Bu amaçla işletim sistemi, kullanıcıya sunulan yalın kullanım modeli ile gerçekleştirime özgü fiziksel yapı arasında geçişi sağlamak zorundadır. Kütük kullanımı kapsamında bu geçiş, ileride ayrıntılarıyla incelenecek olan kütük yönetim kesimince ele alınır (Çizim 1.6).



Çizim 1.6. İşletim Sistemi Kesimleri ve Yer Aldıkları Katmanlar

Kütük kullanımının yanı sıra, ana belleğin kullanımı yönünden de benzer geçişlerin gerçekleştirilmesi gerekir. Zira kullanıcılar için ana bellek, mantıksal olarak, bir sözcükler dizisidir. Kullanıcılar programlarının, ana bellekte bitişken olarak yüklenip çalıştırıldığını düşünürler. Ancak gerek bellek alanlarının verimli kullanımı gerekse var olan bellekten daha büyük bellek sığaları yaratma gibi amaçlarla programlar, işletim sistemi tarafından sayfa ve kesimlere bölünebilir. Bu sayfa ve kesimler ana bellekte bitişken olmayan konumlara yüklenebilir. Programları üzerinde, inisyatifi dışında yapılan bu bölümlemeler kullanıcıdan gizlenmek zorundadır. Bu gizleme, sayfa ve kesimlere bölünmüş fiziksel program yapıları ile bitişken mantıksal program yapıları arasında geçiş kurularak işletim sistemi tarafından sağlanır. Bu bağlamda, mantıksal

kullanım modelleri ile fiziksel gerçekleştirim modelleri arasındaki otomatik geçiş işlevi, işletim sisteminin önemli bir işlevini oluşturur.

Mantıksal yapılardan fiziksel yapılara geçiş, genellikle tek bir adımda gerçekleşmez. Değişik soyutlama düzeylerinde, birkaç adımda ele alınır. Alt soyutlama düzeylerinde donanımın fiziksel ayrıntısını taşıyan veriler üzerinde işlem yapılır. Soyutlama düzeyi yükseldikçe fiziksel gerçekliği daha az yansıtan veri yapılarına dayalı işlemler söz konusu olmaya başlar. Fiziksel ayrıntılara dayalı alt soyutlama düzeyi işlemler, donanıma yakın işlemler olarak da bilinir. Donanıma yakın işlemler, ilgili donanımların fiziksel ayrıntılarını bilmeyi gerektirir. Bu bağlamda, örneğin bir öbeğin disk sürücüden okunabilmesi için sürücünün bağlı olduğu arabirimin programlanması gereklidir. Arabirimleri programlayabilmek için, çalışma ilkeleri, içerdikleri yazmaçların işlevleri, fiziksel adresleri gibi özel ayrıntıları yakından bilmek gerekir. Bunun tersine, örneğin, üst düzey programlar içinde yer alan sıradan tutanak okuma işlemleri ise üst soyutlama düzeyi işlemleri oluşturur. Üst soyutlama düzeyi işlemler, mantıksal tutanaklar gibi kullanıcının düşüncesinde gerçekliği olan mantıksal veri yapıları üzerinde işlem yapmaya olanak verirler. Üst soyutlama düzeyi işlemler donanıma uzak işlemler olarak da bilinir.

İşletim sistemini oluşturan değişik işlevsel kesimler, kullanıcıdan donanıma inen eksende, ilgili oldukları veri soyutlama düzeyine koşut olarak değişik düzeylerde düşünülebilirler. Bu düzeylerden en altta olanı çekirdek, en üstte olanı ise kabuk olarak anılır. Kabuk düzey kullanıcıya en yakın, dolayısıyla en üst soyutlama düzeyini, çekirdek düzey ise donanıma en yakın, dolayısıyla en alt soyutlama düzeyini ifade eder. Düzeyler katman olarak da anılır. Çizim 1.6'da, genel amaçlı bir işletim sisteminde bulunan işlevsel kesimler, yer aldıkları katmanlarla verilmiştir. Bunlardan giriş/çıkış sistemi, görev yönetimi ve görevler arası zamanuyumlama düzenekleri kesimi çekirdek katman içinde, ana bellek yönetimi, kütük yönetimi ve ağ yönetimi kesimleri, kabuk ile çekirdek arasında kalan ara katmanda gösterilmiştir. İşletim sisteminin kullanıcı ile dolaysız etkileşim kurduğu komut yorumlama kesimi ise, doğal olarak kullanıcıya en yakın katman olan kabuk katmanında yer almıştır.

1.8.1. Ana İşlem Biriminin Yönetimi

Ana işlem birimi bir bilgisayar sisteminin en önemli kaynağını oluşturur. Bu önemli kaynağın kullanıcılar arasında paylaştırılması ana işlem biriminin yönetimi kapsamında ele alınır. Ana işlem birimi, kullanıcı programlarının birlikte işletilmesiyle paylaştırılır. Bir program tümüyle sonlanmadan bir diğerinin de işletime alınması, bu programların birlikte işletilmesi olarak bilinir. Programlar, birlikte işletimin gereği olarak zaman zaman kesilerek ana işlem birimini diğer programlara bırakmak durumunda kalırlar. Bu biçimde kesilen bir programın, işletimini sonradan sorunsuz sürdürebilmesi için kesildiği andaki işletim ortamının saklanması gereklidir. Bu amaçla işletime alınan tüm programlar için iskelet adı verilen özel veri yapıları tutulur. Bir program işletime alınacağı zaman, işletim ortamını belirleyen ana işlem birimi yazmaçları bu programın iskeletindeki bilgilerle günlenir. Program işletimi, bu biçimiyle görev olarak adlandırılır. Ana işlem birimi yönetimi, bu nedenle görev yönetimi olarak da anılır.

Görev yönetimi, ana işlem birimi donanım yapısına doğrudan bağımlı alt düzey işlemlerin ele alındığı bir yönetim kesimidir. Görev iskeleti, görev denetim öbeği gibi, görev yönetiminin kullandığı temel veri yapılarında, yazmaç, program sayacı, yığıt göstergesi içerikleri gibi ana işlem birimine özel bilgiler tutulur. Görevler ana işlem birimine görev yönetici olarak adlandırılan özel bir sistem görevinin denetiminde anahtarlanırlar. Görev yönetici, kesilme olarak adlandırılan kimi özel donanım uyarıları ile ana işlem birimine anahtarlanır. Hangi görevin kendisinden sonra ana işlem birimini kullanıcağına karar verir. Ana işlem birimi yazmaçlarını, seçilen görevin iskeletindeki bilgilerle günleyerek anahtarlamayı gerçekleştirir. Gerek özel donanım uyarılarının ele alınması gerekse anahtarlama işlemleri donanımın ayrıntısına bağımlı işlemlerdir. Görev yönetimi, bu nedenle donanıma yakın işlemlerin ele alındığı çekirdek katman içinde düşünülür.

Görevler, işletimlerinin başından sonuna değin ya ana işlem birimi üzerinde çalışır durumda, ya bu birimi kullanmaya hazır durumda ya da başlattıkları bir giriş/çıkış işleminin (örneğin diskten bir öbeğin okunması işleminin) tamamlanmasını bekler durumda bulunurlar. Görevler çalışmadıkları sürece, hazır, bekler gibi adlarla anılan kuyruklara bağlı olarak tutulurlar. Görevler ana işlem birimine hazır görevler kuyruğu üzerinden anahtarlanırlar. Çalışmakta olan bir görevin işletimi herhangi bir nedenle kesildiğinde hazır görevler kuyruğundan hangi görevin ana işlem birimini kullanacağına görev yönetici karar verir. Bunun yanı sıra sisteme sunulan görevlerin iskeletlerinin çatılması, önceliklerinin belirlenmesi, işletimi sonlanan görevlerin varlıklarının sonlandırılması gibi işlemler de görev yönetimi kapsamında gerçekleştirilir. Ana işlem biriminin yönetimi, Görev Yönetimi adlı bölümde, ayrıntılı olarak incelenecektir.

1.8.2. Zamanuyumlama İşlevleri

Bir bilgisayar sisteminde tüketilen kaynak sayısı, sistemde aynı anda işletime alınan görev sayısından her zaman çok daha küçüktür. Görevler, örneğin yazıcılar gibi kimi kaynakları bu nedenle paylaşmak zorunda kalırlar. Bu zorunluluğun yanı sıra, görevler, koşut olarak günlenen bir kütük örneğinde olduğu gibi, uygulamanın gereği olarak da kaynakları ortak kullanmak durumunda kalabilirler. Görevlerin, kaynakları, gerekli önlemler alınmadan rasgele paylaşması işletim bütünlüğünün bozulmasına yol açabilir. Örneğin, bir görev, bir yazıcı üzerinden döküm yaparken, işletimi kesilerek diğer bir görevin işletime alınması ve aynı yazıcıdan döküm almaya kalkması, dökümlerin karışmasına yol açar. Bunun gibi, aynı kütük üzerinde koşut günleme yapan görevlerden birinin, bir tutanakla ilgili günleme işlemlerini tümüyle tamamlamadan işletiminin kesilmesi ve diğer bir görevin işletime alınarak aynı tutanak üzerinde günleme yapmaya kalkması, hatalı günlemeye, dolayısıyla veri bütünlüğünün bozulmasına yol açabilir. Bu nedenle bu tür günleme işlemlerinin bölünmez biçimde gerçekleşmesi gerekir. Başka bir deyişle, bir tutanak üzerinde yürütülen işlemler tümüyle sonlanmadan bu tutanağın erişimi diğer görevlere kapalı tutulur.

Kaynak paylaşımında işlem bölünmezliği, görev yapılarına zamanuyumlama işleçleri olarak adlandırılan özel programlama araçlarıyla katılır. Kimi zaman, ana işlem biriminin makina komutları gibi ele alınan bu işleçlerle ilgili düzeneklere, doğal olarak

donanım ayrıntısına en alt düzeyde açık olan çekirdek katman içinde yer verilir. İşletim sisteminin görevler arası zamanuyumlama işlevlerine Birlikte Çalışan Görevler adlı bölümde ayrıntılı olarak değinilecektir.

1.8.3. Giriş/Çıkış Sistemi

Giriş/çıkış birimleri, verilerin hem bilgisayar sistemi ile dış ortam arasında iki yönlü aktarımını sağlayan, hem de bunların bilgisayar ortamında saklanmasına yarayan birimlerdir. Bilindiği gibi giriş/çıkış birimleri giriş/çıkış sürücüleri ile giriş/çıkış arabirimlerinden oluşur. Giriş/çıkış sürücüleri, klavye, ekran, yazıcı, modem gibi, verilerin sistem ile dış ortam arasında çift yönlü aktarıldığı ya da disk, mıknatıslı şerit gibi sistem içinde saklandığı ögelerdir. Verilerin, sürücüler üzerinde okuma yazma işlemleri, giriş/çıkış arabirimleri aracılığıyla gerçekleştirilmektedir. Bu bağlamda, bir disk öbeğinin sürücü üzerinden okunması için, disk arabirimi düzeyinde yer alan kimi denetim yazmaçları kullanılarak okuma yazma kafasının ilgili ize taşınması, öbeğin sürücüden arabirim yastığına, buradan da ana belleğe aktarılması gerçekleştirilir. Benzeri işlemler, giriş/çıkış sistemi olarak adlandırılan kesim tarafından ele alınır.

Yukarıda tek bir sürücü için örneklenen işlemler, çok kullanıcılı bir bilgisayar sisteminde, aynı anda birden çok sürücü üzerinde gerçekleştirilmek zorundadır. Bunun için çoğu kez, sürücü-arabirim arası aktarım işlemleri bir kez başlatıldıktan sonra, arabirimlerin, aktarım sonunda ana işlem birimini uyarması öngörülür. Bu yolla ana işlem biriminin birden çok arabirimi eşanlı olarak denetleyebilmesi sağlanır. Arabirimlerin ana işlem birimine yolladıkları işlem sonu uyarıları, kesilme uyarıları olarak adlandırılır. Değişik giriş/çıkış arabirimlerinden aynı anda ve zamanuyumsuz olarak gelen bu uyarıların, hiçbir veri kaybına yol açmadan ve bu birimlerin öncelik sırası gözetilerek ele alınması kesilmelerin yönetimi olarak bilinir. Kesilmelerin yönetimi giriş/çıkış sistemi kapsamında düşünülür.

Verilerin giriş/çıkış birimleri ile ana bellek arasında aktarılması, her zaman, kesilme düzeneğine dayalı olarak yapılamaz. Özellikle disk gibi hızlı giriş/çıkış birimlerinden veriler, doğrudan bellek erişimi olarak adlandırılan bir başka yöntemle aktarılırlar. Bu yöntemde, ana bellek ile giriş/çıkış birimleri arasında veri aktarımı, doğrudan bellek erişim denetleme birimi olarak adlandırılan yardımcı bir işleyici aracılığıyla gerçekleştirilir. Doğrudan bellek erişim işlemleri de giriş/çıkış sistemi kapsamında ele alınır. Giriş/çıkış sistemi kesimi, giriş/çıkış arabirimleri, ana işlem birimi, doğrudan bellek erişim denetleme birimi gibi birimlerin en alt düzeyde programlandığı, donanım ayrıntısına bağımlı işlemlerin ele alındığı bir kesim olarak çekirdek katmanda yer alır. Bu kesim Giriş/Çıkış Sistemi adlı bölümde ayrıntılı olarak incelenecektir.

1.8.4. Ana Belleğin Yönetimi

Çok iş düzeninin kurulduğu bilgisayar sistemlerinde, birden çok program ana işlem birimini paylaşarak birlikte işletilir. Bir programın işletime alınabilmesi, öncelikle bu programın ana belleğe yüklenmesini gerektirir. Programların birlikte işletimi ana

belleğin bunlar arasında paylaştırılmasını zorunlu kılar. Ana belleğin programlar arasında paylaştırılma zorunluluğu, işletim sistemine yeni bir yönetim işlevi yükler.

Ana bellek, programlar arasında, değişik biçimlerde bölümlenerek paylaştırılır. Belleğin bölüşülmesi, kimi sorunları da beraberinde getirir. Bu sorunlardan en önemlisi, bellekte, programlar arasında kalan kullanılamaz boşlukların yarattığı parçalanma sorunudur. Parçalanma sorunu, programların yerlerinin işletim aşamasında değiştirilmesine olanak sağlanarak dönem dönem bitiştirilmeleri yoluyla aşılır. Bu olanak, fiziksel adreslerin, bellek erişim aşamasında, özel bir yazmaca göreli olarak hesaplanması ve programların mantıksal adres evrenlerinin fiziksel adres evreninden bağımsızlaştırılması yoluyla yaratılır. Program bitiştirme işlemleri, dönem dönem işletimin kesilmesini gerektirdiğinden sistem başarımının düşmesine neden olur. Bitiştirme işlemlerini ortadan kaldırmak üzere programların ana belleğe bir bütün olarak yüklenmesi koşulundan vazgeçilebilir. Programlar sayfa ya da kesim olarak adlandırılan parçalara bölünür. Bu parçalar ana bellekte boş bulunan yerlere yüklenir. Bellekte bitişken olmayan kesimlere serpiştirilmiş programların işletimi, yeni adres dönüştürme ve yönetim işlevlerini ortaya çıkarır. Bu bağlamda sayfalı ve kesimli bellek yönetimlerinden söz edilir.

Bilgisayar sistemlerinde, kimi zaman kullanıcılara, sistemde var olan fiziksel bellek sığasından daha büyük sığada bellek alanları kullanabilme olanakları sunulur. Bu, görüntü bellek olanağı olarak bilinir. Bu bağlamda görüntü sayfalı, kesimli ve kesimli-sayfalı bellek yönetimlerinden söz edilir. Görüntü bellek yönetiminin uygulandığı sistemlerde program işletimi, program tümüyle ana belleğe yüklenmeden başlatılır. Ancak programın tümü diskte tutulur. Programın sayfa ya da kesim olarak adlandırılan parçaları, işletim aşamasında, gerektikçe ana belleğe yüklenir. Bellekte bulunmayan bir sayfa ya da kesime sapıldığında, öncelikle bunun ana bellekte taşınabileceği bir yer aranır. Bulunamazsa bellekten bir sayfa ya da kesim diske taşınarak yer açılır. Bu bağlamda ortaya çıkan yeni yönetim işlevleri de bellek yönetimi kapsamında sağlanır.

Bilgisayar sistemlerinde, ana bellek yönetimini desteklemek üzere, eksik sayfa, kesimden taşma, izinsiz erişim uyarıları gibi uyarıların üretilmesini, adres dönüştürme işlemlerinin gerçekleştirilmesini sağlayan özgün ekler bulunur. Ana Belleğin Yönetimi adlı bölümde, çeşitli bellek yönetim yöntemleri ayrıntılı olarak incelenirken bu özgün donanım ekleri de, yeri geldikçe açıklanacaktır.

1.8.5. Kütük Yönetimi

Kütük yönetimi, kullanıcılara, ana bellek dışında saklanan verileri (kütükleri) üzerinde kolay ve hızlı işlem yapabilme olanağı veren işletim sistemi kesimidir. Kütükler üzerinde yapılan temel işlemler, okuma, yazma, açma ve kapama işlemleridir. Kullanıcılar bu işlemleri, bayt, tutanak dizileri olarak gördükleri mantıksal kütük görünümleri üzerinde, genellikle üst düzey programlama dillerinin sunduğu olanakları kullanarak gerçekleştirirler. Daha önceden de belirtildiği gibi, kullanıcıların düşüncelerinde gerçekliği olan mantıksal yapılı kütükler, disk gibi mıknatıslı ortamların yüzey, silindir ve sektörleri üzerinde fiziksel olarak yer alırlar. Mantıksal tutanaklar

üzerinde tanımlanan işlemlerin fiziksel olarak gerçekleştirilmesi ve donanım ayrıntısının kullanıcıdan gizlenmesi kütük yönetiminin temel işlevidir. Bu bağlamda, bir kullanıcının, üst düzey araçları kullanarak okuma, yazma gibi işlemlerden birini herhangi bir kütüğünün mantıksal bir tutanağına uygulamak istemesi durumunda, öncelikle ilgili kütük adından bu kütüğün bulunduğu sürücü kimliğinin bulunması, ilgili mantıksal tutanağın bu sürücü üzerinde hangi silindir, hangi yüzey ve hangi sektörde yer aldığının saptanması gereklidir. Mantıksal tutanak numaralarından, (silindir, okuma-yazma kafa numarası ve sektör üçlüsünden oluşan, 56-4-9 gibi) fiziksel disk öbek adreslerini hesaplayarak tutanak içeriklerinin, okuma-yazma işleminin yapılacağı ana bellek yastık alanlarına aktarılmasının sağlanması kütük yönetim kesiminin yükümlülüğündedir.

Mantıksal tutanak numaralarından fiziksel disk adreslerine geçiş, kütük yönetim sisteminin, yalın bir kullanım ortamı sağlama ilkesi çerçevesinde yerine getirdiği bir işlevdir. Bunun yanı sıra, kütük yönetim sisteminin, disk alanlarının verimli kullanımı, bu ortamlarda saklanan verilere en hızlı erişimi sağlayan düzenlemelerin yapılması gibi, sistem verimliliğine dönük başka işlevleri de vardır. Bu amaçla, disk alanlarının düzenlenmesi, buralardan kütüklere yer sağlanması, boş alanların izlenmesi, kütüklerin bu alanlar üzerindeki yerleşimlerinin, erişimin en hızlı olacağı biçimde düzenlenmesi gibi işlevler de kütük yönetimi kapsamında yerine getirilir.

Kütük yönetim sistemi, kullanıcılara kütükleri üzerinde, mantıksal yapılar çerçevesinde işlem yapabilme olanağı sağlarken sözkonusu kütüklere, `prog1.c, rapor.doc` gibi simgesel adlar (kimlikler) verme, bunları kılavuz, alt kılavuz gibi adlarla anılan yapılar altında gruplandırma, ayrıştırma ve düzenleme olanakları da sunar.

Kütük yönetim kesimi, çok görevli işlemin uygulandığı sistemlerde kütüklerin kullanıcılar arasında eşzamanlı paylaşım sorununa da etkin çözümler sunmak zorundadır. Bu kapsamda kütüklere erişimin, kütük, tutanak gibi değişik düzeylerde kilitlenmesini sağlayacak düzeneklerin kurulması ve bu düzeneklerin, işletim bütünlüğünü bozmadan çalıştırılması kütük yönetimi kapsamında ele alınır.

Kütük yönetimi kapsamında düşünülen ve yerine getirilen bir diğer işlev de güvenli bir işletim ve saklama ortamının yaratılmasıyla ilgilidir. Güvenli bir işletim ortamı, kütüklerin bozulma ve silinmelere karşı korunmasını, içerdikleri bilgiler yönünden de gizliliğin sağlanmasını gerektirir. Bu amaçla kütük yönetim sistemi, kütüklere erişimleri denetim altında tutar. Kütük yönetim sisteminin; yalın modellere dayalı kullanım ortamının yaratılmasına, disk alanlarının verimli kullanımının sağlanmasına ve kütüklerin paylaşılması ve korunmasına ilişkin temel işlevleri, Kütük Yönetimi adlı bölümde ayrıntılı olarak incelenecektir.

1.8.6 Sistem Komut Yorumlayıcısı

İşletim sisteminin kullanıcı ile iletişim kurduğu katman, kabuk katmanıdır. Kabuk katmanında sistem komut yorumlayıcısı yer alır. Sistem komut yorumlayıcısı, kullanıcıların, terminalleri başından girdikleri komutları yorumlayarak bu komutlarla

tanımlanan işlemleri yerine getirir. Sistem komutları, kullanıcının bilgisayar ortamında saklanan kütüklerini düzenleme, listeleme, görüntüleme, silme, kopyalama, adlandırma, program geliştirme, iş tanımlama, iş çalıştırma, iz sürme, durum sorgulama, ileti gönderme, oturum sonlandırma gibi işlemleri yerine getirmek için kullanılır. Yerine getirilen işlemin türünü anımsatacak biçimde *type, list, cd, cat, exec* gibi kısaltma adlarla anılan bu komutlar, belli yazım kurallarına uygun olarak, giriş ve çıkış parametreleriyle birlikte kullanılırlar. *UNIX*'te komut yorumlama işlevi shell adlı kesim tarafından yerine getirilir. *command, MS-DOS* işletim sisteminin komut yorumlayıcısına verilen addır. Sistem komutlarını örneklemek üzere, aşağıda *MS-DOS*'un sıkça kullanılan, kütük işlemleriyle ilgili kimi komutlarına yer verilmiştir:

```
dir   [kütük adı]          copy  [kütük adı-1] [kütük adı-2]
cd    [alt kılavuz adı]    del   kütük adı
md    alt kılavuz adı      type  kütük adı
```

Bu komutların sağında yer alan parametreler komutun uygulanacağı kütük ya da altkılavuz adlarını göstermektedir. Bu komutlardan, örneğin `dir` komutu adı verilen kütüğün sistemde yer alıp almadığını belirlemek üzere kullanılabilmektedir. Bu komut parametre yazılmadan girildiğinde, o an üzerinde işlem yapılan alt kılavuz altındaki tüm kütük adlarını listelemeye yaramaktadır. `copy` komutu ise bir kütüğün kopyasını çıkarmak üzere kullanılmaktadır. Parametre olarak verilen ilk kütük adı kopyalanacak kütüğü, ikinci ad ise, kopyanın yer alacağı yeni kütük adını belirtmektedir[8].

Bilgisayar sistemleri açıldığında, işletim sisteminin ana belleğe yüklenmesinden sonra çalışmaya başlayan kesim komut yorumlama kesimidir. Çok kullanıcılı bilgisayar sistemlerinde, sisteme giren her kullanıcı için, komut yorumlayıcısıyla ilgili bir görev çalıştırılır. Komut yorumlayıcısının çalışması, kullanıcıların bilgisayar sistemi ile etkileşim kurdukları terminal sürücü ekranında imleç olarak adlandırılan, `>` , `$`, `#` gibi özel damgaların satır başına yazılmasından anlaşılır.

Komut yorumlama kesimi uygulama programlarıyla aynı düzeyde düşünülen bir kesimdir. Kullanıcı komutlarının gerektirdiği kütük işlemleri, giriş/çıkış işlemleri gibi işlemler, aynı uygulama programlarında olduğu gibi, daha alt katmanlarda yer alan kütük yönetimi, giriş/çıkış sistemi gibi işletim sistemi kesimlerinden, izleyen kesimde açıklanan sistem çağrıları aracılığıyla hizmet alınarak gerçekleştirilir. Bu nedenle *MS-DOS*'ta, uygulama programlarına bellekte yer açmak üzere *command* adlı sistem komut yorumlayıcısı bellekten çıkarılabilmektedir. *UNIX*'te de *shell*, işletim sisteminin diğer kesimlerinden bağımsız düşünülmekte ve *Bourne shell*, *C shell*, *Korn shell* gibi kullanıcıya özel, ayrı komut yorumlayıcılar kullanılabilmektedir.

---

[8] *WINDOWS, X_WINDOWS* gibi kullanıcı grafik arayüzü sunan işletim sistemlerinde, komutlar, ekranda bir ikon ile simgelenir. Bu durumda komutlar, ekrandaki ilgili ikon tıklanarak işletilir. Bir komutun, komut satırından adının girilmesi ile ikonunun tıklanması arasında, yürütülen işlemler açısından herhangi bir ayrım sözkonusu değildir.

1.9. Sistem Çağrıları

İşletim sistemini oluşturan değişik işlevsel kesimler, kabuktan çekirdeğe uzanan eksende değişik katmanlarda yer alırlar. Bir kesimin yer aldığı katmanı, ele aldığı verilerin soyutlama düzeyi belirler. Çekirdek katmanda veriler, donanımın fiziksel özelliklerini taşırlar. Üst katmanlarda ise, kullanım kolaylığı sağlayan mantıksal modeller çerçevesinde tanımlanırlar ve gerçeklikten uzaklaşırlar. Bu bağlamda, örneğin kütükler, çekirdek katman içinde, fiziksel olarak yer aldıkları öbek adresleriyle tanımlanırlar. Aynı kütük, kabuk katmandan hizmet alan bir kullanıcı için ise, örneğin doğrusal bir damga dizisidir. Kütüğü oluşturan damgalar, kütük başına göreli sıra numaraları ile, mantıksal olarak ele alınırlar. İşletim sistemlerinde verilerin soyutlanmasına, yalın mantıksal modellere dayalı kolay kullanım hizmeti verebilmek amacıyla başvurulur. Kullanıcıların mantıksal olarak tanımladıkları işlemlerin yerine getirilebilmesi için üst soyutlama düzeyinde tanımlanmış verilerden fiziksel düzeyde tanımlı verilere geçiş sağlanması gerekir. Bu bağlamda, bir kütüğün başına göreli beşinci tutanağının okunabilmesi için bu tutanağın diskte hangi fiziksel öbek içinde yer aldığının belirlenmesi; başka bir anlatımla, kütük başına göreli mantıksal tutanak numarasından sürücü başına göreli fiziksel öbek adresine geçişin sağlanması gereklidir. İşletim sistemlerinde bu geçiş sistem çağrıları ile gerçekleştirilir. Sistem çağrı düzeneği, işletim sistemince verilen hizmetlerin, donanımın ayrıntısından soyutlanmış, kolay anlaşılır mantıksal modellere dayalı olarak yürütülebilmesini sağlayan bir düzenektir.

Sistem çağrı düzeneği, genelde programlama dillerindeki yordam çağırma düzeneğine benzer. Sistem çağrı düzeneğinin kurulmasında, ana işlem biriminin bu amaçla öngörülmüş komutlarından yararlanılır. Sistem çağrısı yapmak işletim sistemine sapmak anlamına gelir. Bir işletim sistemi işlevine gereksinim duyulduğunda `SVC (Supervisor Call)`, `INT (Interrupt)` gibi adlarla anılan özel makina komutları çalıştırılarak ilgili işletim sistemi kesimine sapılır. İşlem parametreleri sisteme, ana işlem birimi yazmaçları içinde aktarılır.

MS-DOS/Windows işletim sisteminde sistem çağrıları 80X86 türü işleyicilerin yazılım kesilmesi olarak bilinen `int xx` komutları kullanılarak gerçekleştirilmiştir. `xx` değeri istenen hizmetin türünü belirlemektedir. Kullanıcılar *MS-DOS/Windows* işletim sisteminde, `int 20h` ile `int 40h` arasındaki sistem çağrı komutlarını kullanarak hizmet alabilmektedirler. Bunlardan `int 21h`, en çok yararlanılan ve giriş/çıkış hizmetleriyle ilgili olan sistem çağrı komutudur. Bu komuta, yerine getirilmesi istenen işlevin kodu, giriş ve sonuç parametreleri gibi parametreler işleyici yazmaçları aracılığıyla aktarılmaktadır. Sistem çağrı kavramı, Çizim 1.7'deki programla örneklenmiştir.

Sözkonusu programda `\bbm\örnküt.doc` adlı bir kütüğe, 2048 inci baytından başlayarak, `yastık` adlı bellek alanında yer alan 256 baytın yazılması konu edilmiştir. Bu amaçla, önce kütüğün simgesel adı ile fiziksel konumu arasında bağlantı kurmak üzere `3DH` kodlu `open`, kütük açma işlemi gerçekleştirilmiştir. Bu işlem sonunda `ax` yazmacı içinde, `handle` adıyla, kütüğün fiziksel konumuna ilişkin bilgilerin tutulduğu

alan göstergesi elde edilmiştir. Bunun sonrasında kütüğün başına göreli 2048 inci mantıksal bayt konumundan başlayarak işlem yapılacağını belirten `move-file-pointer` çağrısı çalıştırılmıştır. Bunu izleyen kesimde 256 baytlık ana bellek yastık alan içeriğinin kütüğe yazılması gerçekleştirilmiştir.

```
asciiz    db    '\bbm\örnküt.doc',00H
handle    dw    ?
yastık    db    256 dup(?)
.
;open çağrısı
;
mov ah,3DH             ;open file işlevi
mov al,02              ;erişimin türü
mov dx,offset asciiz   ;kütük kimliği
int 21h                ;çağrı
jc  error1             ;hata
mov handle,ax
;
;2048 inci konumu belirleme çağrısı
;
mov ah, 42h            ;move file pointer işlevi
mov al, 0              ;kütük başına göreli
mov bx, handle         ;kütük kimliği
xor cx, cx             ;cx ← 0
mov dx, 2048           ;2048. konum
int 21h                ;çağrı
jc  error2             ;hatalı işletim
;
;yazma çağrısı
;
mov ah, 40h            ;write to file işlevi
mov bx, handle         ;kütük kimliği
mov cx, 256            ;tutanak boyu
mov dx, offset yastık  ;aktarım alan adresi
int 21h                ;çağrı
jc  error3             ;hatalı işletim
.
```

Çizim 1.7. *MS-DOS*'ta Sistem Çağrılarının Kullanımı

Çizim 1.7'deki programda yer alan `int N` komutu, 80X86 serisi işleyicilerin yazılım kesilmesi olarak tanımlanan bir makina komutudur. Bu komut (*real mode*'da) işletildiğinde, ana belleğin ilk 1 KBaytlık kesimindeki *N*4* ve *N*4+2* adreslerinde tutulan program sayacı ve kesim yazmacının gösterdiği yordama sapılmaktadır. *MS-DOS*'ta sistem çağrılarından giriş/çıkışlarla ilgili olanları `int 21h` komutu aracılığıyla çağrılmaktadır. Alınmak istenen hizmetin türü, *21*4* ve *21*4+2* adresindeki program sayacı ve kesim yazmacı ikilisiyle sapılan yordama, `ah` yazmacı içinde aktarılmaktadır.

İstenen hizmetin gerektirdiği giriş parametreleri `bx`, `cx`, `dx` gibi yazmaçlara yüklenmektedir. Yerine getirilen hizmetle ilgili, varsa durum bilgileri `al`, `psw` gibi yazmaçlar içinde geri dönmektedir. Bu bağlamda, Çizim 1.7'de verilen programın kullandığı çağrılardan `40h` kimlikli `write-to-file`, kütüğe yazma çağrısına, yazma yapılacak kütük kimliği `bx`, yazılacak yastık uzunluğu `cx` yazmaçlarıyla aktarılmakta, çağrı sonucu ise, program durum yazmacının elde (*carry)* biti içinde döndürülmektedir.

Sistem çağrı düzeneğinin, işletim sistemince verilen hizmetlerin, donanımın ayrıntısından soyutlanmış, kolay anlaşılır mantıksal modellere dayalı olarak yürütülebilmesini sağlayan bir düzenek olduğu yukarıda belirtilmişti. Bu bağlamda, sistem çağrı düzeneği, uygulama programlarının işletim sisteminden hizmet almak için kullandıkları bir düzenek olarak da düşünülebilir. Ancak sistem çağrı düzeneği kavramını, uygulama programı-işletim sistemi arası hizmet alış-verişi ile sınırlı düşünmek doğru olmaz. Tek parça *(monolithic)* olarak ele alınması mümkün olmayan yazılımlar, çoğu kez katmanlı yapıda tasarlanırlar. Bu bağlamda her katman, yazılımın üreteceği hizmetin bir alt adımını gerçekleştirmek için öngörülür. Katmanlar, yazılımın genel yapısı içinde hiyerarşik bir sırada yer alırlar. Her katman, bir üst katmana hizmet üretirken bir alt katmandan hizmet alır. Bu yapı çerçevesinde her katman belirli bir soyutlama düzeyini temsil eder ve bir üstünde yer alan katmana, hizmeti, bu soyutlama düzeyine uygun olarak üretir. Bu yolla her alt adıma ilişkin ayrıntı ilgili katmana gömülü kalır. Katmanlar arası hizmet alış-verişi sistem çağrı düzeneğine dayalı olarak yürütülür. Katmanlı yapıda gerçekleştirilmiş işletim sistemleri de bu genel yapıya uyarlar. Başka bir deyişle, uygulama programı-işletim sistemi arası hizmet alış-verişinin yanı sıra, ara katmanlar ile çekirdek katman arasında da hizmet alış-verişi sistem çağrı düzeneğine dayalı olarak yürütülür.

Örneğin, bir uygulama programı, simgesel kimliğiyle andığı bir kütüğün başına göreli belirli sayıda baytı okumak ya da yazmak istediğinde bu istemine karşı gelen hizmeti, bir sistem çağrısıyla kütük yönetim sisteminden talep eder. Kütük yönetim sistemi de, örneğin, ilgili kütüğün yer aldığı fiziksel sürücü kimliğini ve okunacak ya da yazılacak bayt dizisinin yer aldığı fiziksel öbek adresini, silindir-kafa-sektör üçlüsü olarak hesaplayıp (başka bir deyişle mantıksal yapı ile fiziksel yapıyı eşleyip) bu üçlüye karşı gelen sektörlerin okunup sözkonusu uygulama programının adres evreni içinde tanımlı ana bellek alanına aktarılmasını, fiziksel giriş-çıkış işlemlerini yürütmekle sorumlu çekirdek katmandan ister. Bu nedenle, katmanlı mimariye sahip işletim sistemlerinde değişik katmanlar için değişik sistem çağrı grupları bulunur. Örneğin *UNIX*'te sistem çağrıları, işletim sistemi C programlama dili kullanılarak yazıldığından C işlevleri olarak tanımlanırlar ve sözkonusu işlevlerin ait oldukları grup numaralarından söz edilir. Kimi gruplara ilişkin işlev ya da sistem çağrıları uygulama programlarınca, bir bütün olarak işletim sisteminden hizmet almak için kullanılabilirken kimileri de, salt işletim sistemi alt katmanlarınca, çekirdek katmandan hizmet almak üzere öngörülürler. *MS-DOS*'ta ise `int 21h`, uygulama programlarının kütük yönetim sisteminden hizmet almak için kullandıkları bir düzenektir. Ancak `int 21h` düzeneğinin yanı sıra, `int 1Ah`, `int 1Bh` gibi, daha çok, kütük yönetim sisteminin çekirdek katmandan hizmet alması amacıyla öngörülmüş başka sistem çağrıları da bulunur.

İşletim sistemleri konusuna Girişte, son olarak, çok kullanıcılı bir bilgisayar sistemi üzerinde program geliştiren bir kullanıcının, işletim sisteminin değişik kesimlerinden doğrudan ya da dolaylı olarak nasıl yararlandığını gösteren bir örnek incelenmiştir. Bu örnekte kullanıcının varsayımsal bir işletim sistemi altında `prog.pas` adlı kütüğünü `sed` (*screen editor*) adlı bir metin düzenleme yazılımı ile günlediği düşünülmüştür:

Kullanıcı, oturumun başında, kendisi için tanımlanan ve işletimi başlatılan komut yorumlayıcısı görevinin ekranına yazdırdığı imlecin sağına `>sed prog.pas` komutunu yazarak işlemleri başlatır. Komut yorumlayıcısı, öncelikle bu komut satırının sözdizimini denetler. Komut girilirken sözdizim hatası yapılmamışsa komut işletimine geçilir. Öncelikle kütük açma sistem çağrısı çalıştırılarak `prog.pas` kütüğünün açılma işlemi gerçekleştirilir. `sed` sözcüğü, metin düzenleme programının amaç kodunu taşıyan kütük adı olduğundan bu programla ilgili yeni bir görev tanımı yapılır. Bu bağlamda kullanıcının bilgilerini içeren yeni bir görev iskeleti oluşturulur. Oluşturulan bu iskelet hazır görevler kuyruğuna bağlanıp görev yöneticinin denetimine bırakılır. Görev tanımlama, görev iskeleti kurma, görevi hazır görevler kuyruğuna bağlama gibi işlemler, hep ilgili sistem çağrıları kullanılarak yerine getirilir.

Günleme işlemleri çerçevesinde, `prog.pas` adlı kütüğün tutanaklarına erişmek istendiğinde mantıksal adlandırmalardan fiziksel disk adreslerine (silindir, kafa, sektör üçlüsüne) kütük yönetimi ile ilgili sistem çağrıları kullanılarak geçilir. Elde edilen fiziksel adresler, çekirdek katmanda yer alan giriş/çıkış sistemine aktarılarak ilgili tutanak içeriklerinin sürücülerden görevin (`sed`'in) ana bellek yastık alanına aktarılması sağlanır. Giriş/çıkışlarla ilgili bir sistem çağrısını işlettiğinde, giriş/çıkış işlemlerinin göreli yavaşlığı nedeniyle, görev, ilgili giriş/çıkış sürücüsünün bekleme kuyruğuna bağlanır. Giriş/çıkış işleminin sonlanmasıyla (örneğin ilgili sürücü tutanak içeriğinin arabirim yastığında hazır olması ya da doğrudan bellek erişim düzeneğince ana bellekteki yastık alanına aktarılması sonrasında) yeniden hazır görevler kuyruğuna, ilgili sistem çağrısı aracılığıyla eklenir. Erişilen kütük, başka görevlerce de koşut olarak kullanılıyorsa (bu görev ilgili kütük üzerinde giriş/çıkış işlemi başlatan sistem çağrısını çalıştırdığı anda başka bir görevin de aynı kütük üzerinde günleme yapıyor olduğu durumlarda) görev, zamanuyumlama düzeneğine ilişkin bekleme kuyruklarına da bağlanabilir.

Ana bellek kullanımı, ana bellek yönetiminden sorumlu sistem çağrıları tarafından gerçekleştirilir. Herhangi bir görevin ve işlediği verilerin ana bellekte yer alabilmesi, ana bellek yönetim kesimince bu göreve yer sağlanması koşuluna bağlıdır. Bir görev ancak ana bellekte ilgili programına yer sağlandığı takdirde hazır görevler kuyruğuna bağlanabilir. Sistemde yürütülen bellek yönetiminin türüne bağlı olarak sayfalama, kesimleme gibi, çok sayıda karmaşık bellek işlevi de, bu görevle ilgili olarak ve kullanıcıya yansıtılmadan işletilir.

Metin düzenleme gibi yalın bir hizmetle ilgili olarak ortaya çıkan bu sistem çağrı trafiği, çok sayıda kullanıcı (ya da program) ve bunların sistemden talep edebilecekleri çeşitli hizmetler göz önüne alındığında çok daha karmaşık bir yapıya bürünür. Yukarıdaki

örnekte göz önüne alınmayan görüntü bellek, uzak bilgisayar sistemlerine erişim, güvenlik ve bilgi gizliliği gibi düzenekler de düşünüldüğünde işletim sistemi tarafından kullanıcılara sunulan hizmetin karmaşık bir alt yapıya dayalı olarak yürütüldüğü kolayca anlaşılır. Bilgisayar sistemini oluşturan donanım ve yazılım nitelikli kaynakları programlar arasında kolay, hızlı ve güvenli bir işletim hizmetine olanak verecek biçimde paylaştıran, bunu yaparken, kaynakların kullanım verimliliğini en üst düzeyde tutmayı amaçlayan işletim sistemi, var olan yazılım sistemlerinin en karmaşığıdır. Bu karmaşık sistem, temel işlevleri ve ana çizgileriyle bu kitabın izleyen konu başlıkları altında incelenecektir.

1.10. Çok Kullanıcılı Bilgisayar Sistemi Kavramının Evrimi

1970'li yıllara gelinceğe değin, bilgisayar sistemleri, kurumlar içinde, bilgi işlem merkezi olarak adlandırılan belirli bir mekanda yer almış, kullanıcılar bilgisayar sistemlerinden bu merkeze bizzat gelerek hizmet alma durumunda olmuşlardır. Bilgisayar sistemlerinden her türlü hizmetin alımı, verilere ve bunları işleyen programlara ilişkin delikli kart destelerinin sisteme okutulması ve sonuçların da bir yazıcı aracılığıyla elde edilmesi biçiminde olmuştur. Toplu işlem kapsamında düşünülen bu hizmet türü, 1970'li yıllarda yerini etkileşimli işlem olarak adlandırılan yeni bir işlem türüne bırakmıştır.

Etkileşimli işlemde kullanıcılar, bilgisayar sisteminden, terminal olarak adlandırılan birimler aracılığıyla yararlanmaktadırlar. Terminal, yalın ekran ve klavye ikilisine verilen addır. Verilerin ve bunları işleyen programların sisteme sunuluşu, bu işlem türünde, delikli kartlar yerine klavye aracılığıyla gerçekleşmekte, sonuçlar, yazıcı birimlerin yanı sıra ekran üzerinden de aracısız elde edilebilmektedir. Gerek toplu, gerekse etkileşimli işlemde bilgisayarın kurum içindeki konumu ve hizmet alma biçimi merkezi olmuştur.

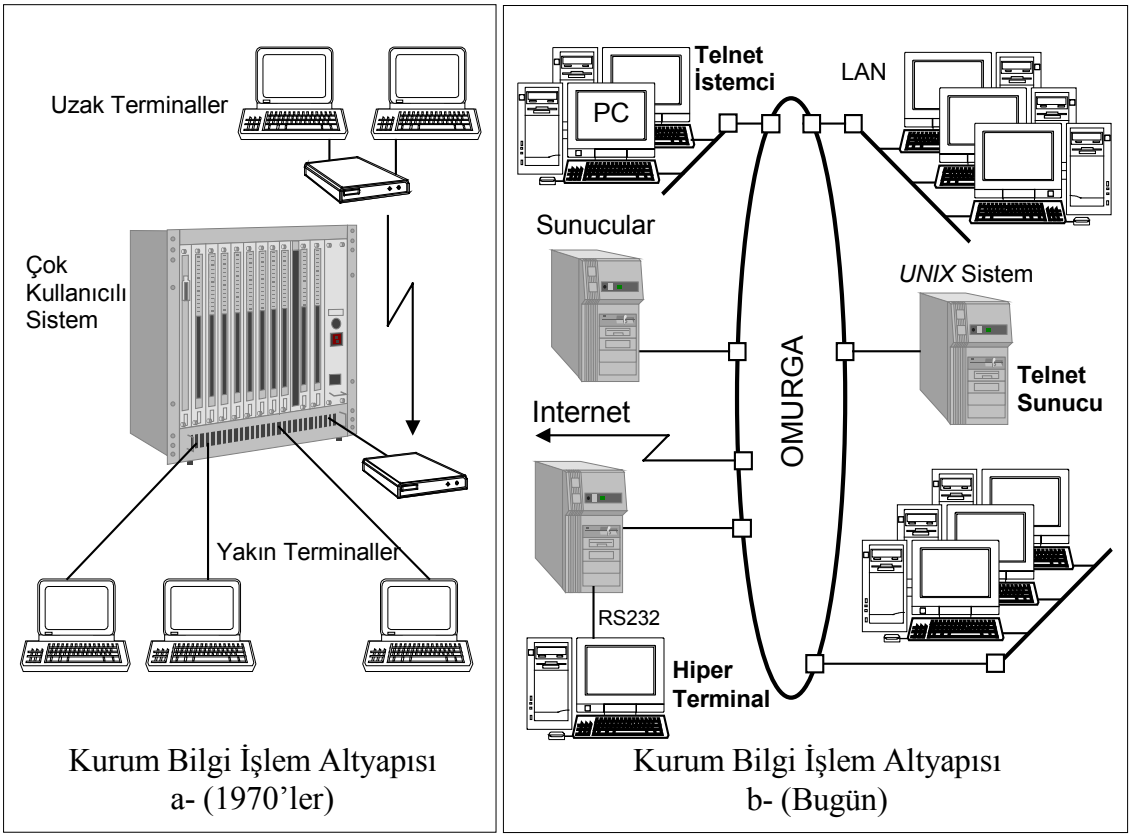
Klasik anlamda çok kullanıcılı bilgisayar sistemi dendiğinde, birden çok kullanıcının, terminal birimleri aracılığıyla, aynı bilgisayar sisteminden, etkileşimli işleme dayalı olarak eşanlı hizmet aldıkları sistem anlaşılmaktadır. Bu kapsamda terminal birimleri, bilgisayar sistemine, sisteme özel (yıldız ya da yol gibi) topolojilerle bağlı olarak çalışmaktadır (Çizim 1.8.a).

Mikro-elektronikteki teknolojik gelişmeler, bilgisayar donanımlarının, boyut ve fiyat yönünden olağanüstü ufalmasını sağlamış ve 1980'li yıllardan başlayarak kişisel bilgisayar *(PC)* sistemleri ortaya çıkmaya başlamıştır. 1970'li yılların odalar dolusu bilgisayar sistemlerinden kat kat daha yüksek işlem ve saklama kapasitesine sahip bu bilgisayarlar, günümüzde, masa üstü boyut ve standartlaşmış yapılarla üretilip pazarlanmaktadır. Bu gelişmeler, bilgi işlem hizmetlerini, bir merkez yerine, kişinin ofisinden ya da evinden alınabilir bir niteliğe büründürmüştür.

Bilgi işlem hizmetinin, kişisel bilgisayarlara dayalı olarak ofiste, masa üstünden alınıyor olması bu bilgisayarlar arası iletişim gereksinimini ortaya çıkarmıştır. Bu gereksinim, bir yandan doğası gereği merkezi bir konumda bulunan ortak verilere erişim, diğer

yandan da disk ve yazıcı birimleri gibi pahaca yüksek bilgi işlem kaynaklarının paylaşım zorunluluğundan doğmuştur. Sözkonusu gereksinim, bir ofis ya da kurumda yer alan kişisel bilgisayar sistemlerinin yerel ağlar *(LAN)* biçiminde birbirlerine bağlanması yoluyla karşılanmıştır.

Bu gelişmelere paralel olarak, kurumlarda çok kullanıcılı büyük boy bilgisayar sistemlerinin yerini yerel ağlar içinde kümelenen bilgisayar sistemleri almaya başlamıştır. Bu yapı içerisinde bir ya da birkaç bilgisayar sistemi (sunucu), diğerlerine (istemci) hizmet sağlama amacıyla öngörülmektedir. Bir kurumun, büyüklüğüne göre, değişik bilgi işlem uygulamalarına dönük birden çok yerel ağ sistemi bulunabilmektedir. Bunların tek bir yapı içinde bütünleşmeleri, omurga *(backbone)* adlı yüksek hız kapasiteli iletişim altyapıları ile mümkün olmaktadır. Bir üniversite kampüsü gibi büyükçe bir kurumun bilgi işlem donanımı artık günümüzde, bir omurga çevresinde kümelenen çeşitli yerel ağ sistemlerinden oluşmaktadır (Çizim 1.8.b). Bu durumda klasik çok kullanıcılı bilgisayar sistemi kavramı da anlam değiştirmiştir.



Çizim 1.8. Çok Kullanıcılı Bilgisayar Sisteminin Evrimi

Günümüzde, sunucu nitelikli bilgisayar sistemlerinin yaygın kullandığı işletim sistemleri, *UNIX* ve *Windows* işletim sistemleridir. Bilindiği gibi, *UNIX* 1970'li yıllarda geliştirilmiş bir sistemdir. Sonradan eklenen ağ katmanı *(TCP/IP)* sayesinde *UNIX*, mimari evrime (ya da metamorfoza) ve istemci-sunucu paradigmasına dayalı işlem türüne ayak uydurabilmiştir. Bununla birlikte *UNIX*, kullanıcılara, bir terminal

aracılığıyla sisteme girip *(logon* olup*)* sistemde bulunan programları çalıştırma yoluyla da hizmet üretebilme, başka bir deyişle klasik çok kullanıcılı sistem olma özelliğini de korumaktadır. Bu özellikten yararlanabilmek için, *UNIX* tanımlarına uygun terminal birimlerine gereksinim duyulacağı açıktır. Yeni mimari yapılanma içinde, terminal birimleri olarak, ağ içinde yer alan diğer bilgisayar sistemlerinin kullanılması en mantıklı yoldur. Bu amaçla, *Telnet (Terminal Emulation)* adlı, terminal öykünümü yapan bir uygulamadan yararlanılmaktadır. *Telnet* sunucu ve istemci olmak üzere iki birleşenden oluşmaktadır. Kendisini terminal olarak göstermek isteyen bilgisayar sistemi, *Telnet* istemci birleşenini çalıştırıp sunucu üzerindeki *Telnet* birleşeni aracılığıyla, klasik bir terminal gibi *UNIX* sistemine bağlanabilmektedir. Bu yolla, eskilerin çok kullanıcılı sistemleri yeni ağ yapıları içinde varlıklarını sürdürebilmektedir.

Telnet uygulamasının yanı sıra, *Windows* işletim sistemi altında çalışan bir bilgisayar sisteminin, *UNIX* işletim sistemi altında çalışan başka bir bilgisayar sistemine, terminal olarak bağlanmasını sağlayan bir başka uygulama da *Hiper Terminal* uygulamasıdır. Bu uygulamada sözkonusu bağlantı, ya Giriş/Çıkış Sistemi başlıklı konu altında açıklanacak *RS-232C* standartında ardıl bağlantı olarak, ya da her iki bilgisayar sisteminin de yer aldığı ağ üzerinden, *Telnet* uygulamasına benzer bir yapıda gerçekleşebilmektedir. Birinci seçenekte, her iki bilgisayar sistemi arasında, klasik anlamda, terminal-çok kullanıcılı bilgisayar sistemi nitelikli noktadan noktaya bağlantı bulunduğu varsayılmaktadır. Bu bağlamda, Kitabın ilerleyen kesimlerinde, özellikle Giriş/Çıkış Sistemi başlıklı kesimde, kimi kavramları, pedagojik nedenlerle yalın bir biçimde açıklamak üzere, sisteme *RS-232C* standartında bağlı klasik terminal sürücülerden söz edildiğinde, bilgisayar sistem konfigürasyonları içinde artık yer almayan bu birimleri, *Hiper Terminal* uygulaması çalıştıran kişisel bilgisayar sistemleri olarak düşünmek gerekecektir.

WindowsNT, yukarıda açıklanan ağ tabanlı yapılanma gözetilerek tasarlanmış yeni bir işletim sistemidir. Bu itibarla, bir terminal aracılığıyla sisteme girip *(logon* olup*)* o sistemde var olan bir programı, yine o sistem üzerinde çalıştırarak hizmet alma modeli yerine, istemci-sunucu ilişkisi içinde, istemci bilgisayar üzerinde çalışan bir programın (istemci programın), sunucu bilgisayar üzerinde çalışan diğer bir programdan (sunucu programdan) hizmet alması modelini kullanmaktadır. Bu yaklaşımın sonucu olarak, *WindowsNT* bağlamında klasik terminal kavramı söz konusu olmaksızın, ağ yapısı içindeki istemci nitelikli bilgisayar sistemlerinin, sunucu nitelikli bilgisayar sistemlerinden, yeni istemci-sunucu paradigması çerçevesinde hizmet alması mümkün olabilmektedir.

Günümüzde, sınanmışlık, alışkanlık gibi gerekçelerle, ağlar içindeki büyük boy sunucu nitelikli bilgisayar sistemlerinin çok büyük bir çoğunluğunun kullandığı işletim sistemi *UNIX* işletim sistemidir. Bu nedenle, çok kullanıcılı bilgisayar sistemi, terminal, *logon* gibi kavramların eski tanımları, bunlara yüklenen yeni tanımlarla birlikte varlıklarını sürdürmeye devam etmektedir. Ancak yakın bir gelecekte bu kavramların, salt ağ tabanlı sistemlere dönük, istemci sunucu paradigmasıyla uyumlu tanımlarının kalıcı olacağını söylemek yanlış olmayacaktır.

2. BÖLÜM

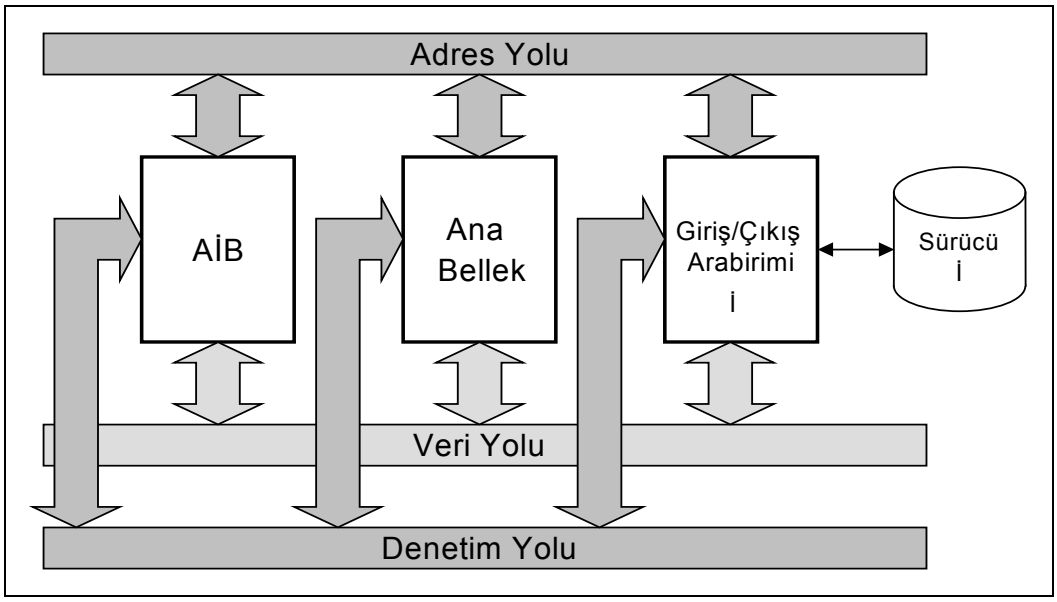
İ Ş L E T İ M S İ S T E M L E R İ

GİRİŞ/ÇIKIŞ SİSTEMİ

Bir bilgisayar sistemini oluşturan temel birleşenlerden giriş/çıkış birimlerinin işe koşulması; verilerin, fiziksel olarak bu birimlerden ana belleğe, ana bellekten de bu birimlere, hiçbir veri kaybı ve tutarsızlığa neden olunmadan aktarılması giriş/çıkış sistemi kapsamında düşünülen işlevler tarafından gerçekleştirilir. Giriş/çıkış birimleri, verilerin, kullanıcı ile bilgisayar ortamı arasında aktarılmasına ya da bu ortamda saklanmasına yarayan birimlere verilen genel addır. Bu bağlamda, terminal, yazıcı gibi giriş/çıkış birimleri veri aktarım; disk, mıknatıslı şerit gibi giriş/çıkış birimleri ise veri saklama (ikincil bellek) birimleridir. Veri saklama birimleri üzerinde saklanan verilere kütük adı verilir. İşletim sistemleri, bir yandan kütükleri saklandıkları fiziksel sürücülerden bağımsızlaştırarak, diğer yandan da, veri aktarım amaçlı giriş/çıkış birimlerini kütük tanımı içine alarak giriş/çıkış birimi kavramını, kütük kavramı ile özdeşleştirirler. Bu nedenle, giriş/çıkış işlemleri kütük işlemleri olarak ele alınır.

Kütük, kullanıcılar açısından, simgesel bir ad ve mantıksal bir yapıya sahiptir. Örneğin, yürütülen uygulamayla uyumlu boyda N adet tutanaktan oluşan bir yapıdır. Kullanıcının düşüncesinde gerçekliği olan bu mantıksal yapının, saklandığı sürücü ortamında, bir de, fiziksel yapısı bulunur. Bir kütüğün i inci mantıksal tutanağı, örneğin, silindir, kafa ve sektör numarasıyla belirlenen bir disk öbeği üzerinde yer alır. İşletim sistemleri, yerine getirmekle yükümlü oldukları kolay kullanım hizmeti gereği kütüklerin mantıksal ve fiziksel yapıları arasında geçişi sağlarlar. Bu bağlamda kütüklerin adlandırılması; giriş/çıkış sürücüleri üzerinde, verimli alan kullanımı ve hızlı erişim kıstasları gözetilerek yerleştirilmesi; tutanakların, kütük başına göreli mantıksal adreslerinden fiziksel öbek adreslerine geçişlerin sağlanması ve elde edilen adreslerdeki öbeklerin

sürücülere yazılıp okunması, işletim sisteminin giriş/çıkışlarla ilgili kesimlerince gerçekleştirilir. Bu işlevlerden adlandırma, yerleştirme, mantıksal adreslerden fiziksel adreslere geçiş işlemleri, kütük yönetim sistemi olarak anılan işletim sistemi kesimi tarafından yerine getirilir. Fiziksel adresleri kütük yönetim kesimince belirlenmiş giriş/çıkış öge içeriklerinin ana bellekten sürücülere ya da sürücülerden ana belleğe aktarılmaları, çekirdek katmanda yer alan giriş/çıkış yordamları tarafından gerçekleştirilir. Bu yordamlar ve ilgili özel düzeneklerin oluşturduğu bütün, giriş/çıkış sistemi olarak adlandırılır. Bu bağlamda, giriş/çıkış sistemi, kütük yönetim kesimine hizmet üreten bir alt katman olarak düşünülür.

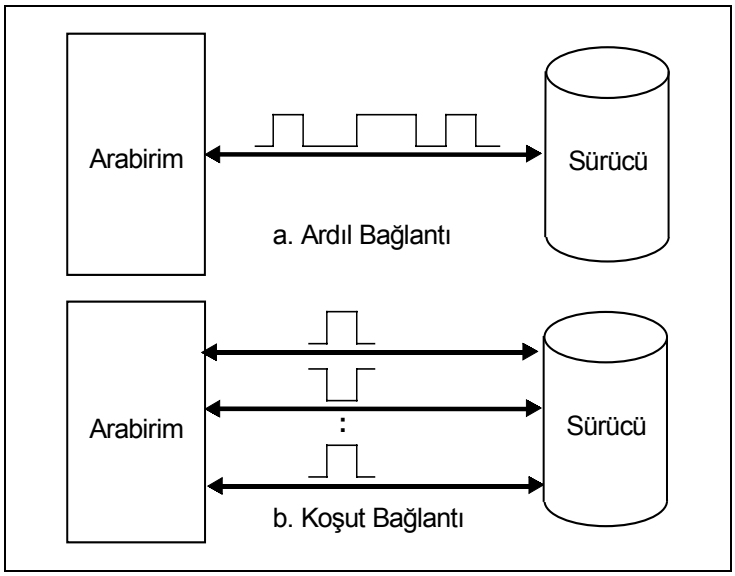


Çizim 2.1. Giriş/Çıkış Birimlerinin Konumu

Bilindiği gibi, giriş/çıkış birimleri giriş/çıkış sürücüleri ile bu sürücülerin ana işlem birimi - ana bellek ikilisiyle bütünleşmesine olanak veren giriş/çıkış arabirimlerinden oluşurlar. Giriş/çıkış birimi olarak terminal birimi; ekran ve klavyeden oluşan terminal sürücü ile bu sürücüye ilişkin arabirim çiftini tanımlar. Bunun gibi disk birimi, dönen plakalar ve ilgili elektronik aksamın yer aldığı disk sürücü ile ilgili arabiriminden oluşur. Ana işlem birimi ve ana bellek ikilisinin veriler üzerinde istenen işlemleri gerçekleştirebilmesi, bu verilerin, önce (dış) kullanıcı ortamından ana belleğe aktarılmasını gerektirir. Dış ortamdan ana belleğe aktarılan veriler, giriş/çıkış sürücüleri üzerinde saklanan ya da bu sürücüler aracılığıyla, kullanıcılar tarafından sisteme girilen verilerdir. Verilerin sürücülerden ana belleğe, ana bellekten sürücülere aktarılabilmesi için bu sürücülerin denetlenmesi gereklidir. Sürücülerin denetlenmesi arabirimlerinin programlanmasıyla gerçekleşir. Arabirimlerin programlanması, giriş/çıkışların programlanması kapsamında ele alınır.

İzleyen kesimde, giriş/çıkışların programlanmasıyla ilgili temel yaklaşımlar açıklanacaktır. Önce giriş/çıkış arabirimlerinin yapısal özellikleri incelenecektir. Bunun sonrasında giriş/çıkışların programlanmasında kullanılan yöntemler, bu yöntemlerin

gerektirdiği kimi özgün donanım düzenekleriyle birlikte verilecektir. Bu bağlamda, ilk olarak seçmeli programlama yöntemi açıklanacaktır. Bunun sonrasında, zamanuyumsuz biçimde gelişen giriş/çıkış istemlerini ele almaya olanak veren ana işlem birimi kesilme düzeneği, kesilmeli giriş/çıkış programlama yöntemiyle birlikte incelenecektir. Sürücü - ana bellek arası veri aktarımlarını, ana işlem birimine başvurmadan yerine getirmeye yarayan doğrudan bellek erişim düzeneği ile doğrudan bellek erişim denetleme birimleri ve kanallar / giriş/çıkış işleyicileri incelenecektir. Açıklanan tüm programlama yöntemleri, bir ya da birkaç G/Ç birimi içeren yalın örnek sistemler taban alınarak ayrı ayrı örneklenecektir. Bu örneklerde, daha çok sözkonusu yöntemlerin ve varsa destek donanımların çalışma ilkeleri öne çıkarılacaktır. Verilen örnekler içinde yer alan sürücü, kesilme gibi yordamlar, bu aşamada, pedagojik nedenlerle işletim sisteminin bütününden ve sistem çağrı düzeneğinden bağımsız olarak sunulacaktır. Bu yordamların, günümüz işletim sistemlerinin bütünü içinde nasıl ele alındıkları, Aygıt Sürücüler başlığı altında, işletim sistemini oluşturan temel birleşenlerin tümü incelendikten sonra Sekizinci Bölüm'de açıklanacaktır.



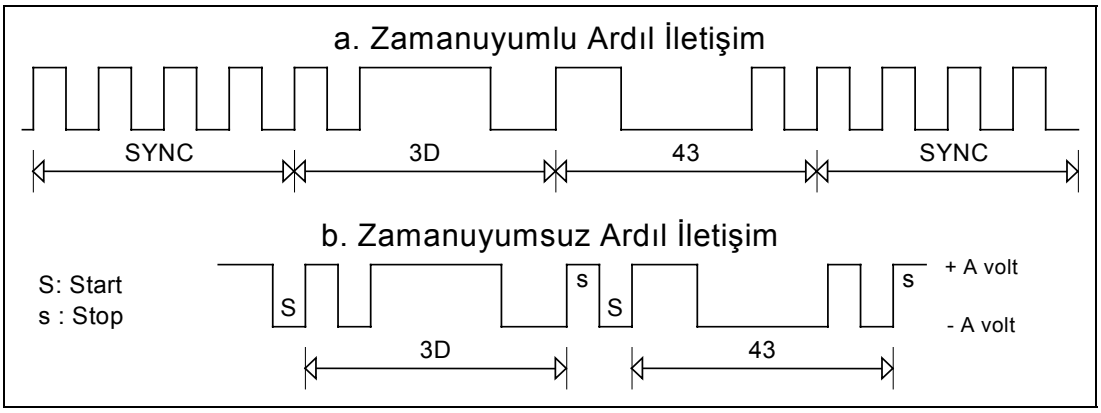
Çizim 2.2. Arabirim-Sürücü Bağlantı Biçimleri

2.1. Giriş/Çıkış Birimleri

Bilindiği gibi, giriş/çıkış birimleri giriş/çıkış sürücüleri ve giriş/çıkış arabirimlerinden oluşur. Giriş/çıkış arabirimleri, giriş/çıkış sürücülerinin ana işlem birimi - ana bellek ikilisi ile bütünleşmelerini sağlarken bu sürücülerin yalın bir biçimde programlana-bilmelerine de olanak verirler. Giriş/çıkış arabirimleri, ana işlem birimi ana bellek ikilisi ile (aynı kabin, aynı dolap gibi) aynı coğrafi ortamda bulunurken giriş/çıkış sürücüleri genelde bu ortamın dışında hatta çok uzağında olabilirler. Örneğin terminal sürücüler bilgisayar sisteminin bulunduğu oda, bina, yerleşim biriminin dışında bulunabilirler. Ancak bunların bağlı olduğu arabirimler ana işlem birimine en çok, bütünleştikleri yolun (adres, veri ve denetim hatlarının) izin verdiği uzaklıkta yer alırlar.

2.1.1. Arabirim-Sürücü arası Bağlantı

Giriş/çıkış sürücüleri ilgili oldukları arabirimlere, genelde iki biçimde bağlanırlar. Bunlar ardıl (seri) ve koşut (paralel) bağlantılardır. Sürücüler ile arabirimler arasında veriler, (genelde 1 bayt uzunluğundaki) damga tabanında aktarılırlar. Koşut bağlantıda aktarılan damga bitlerinin herbiri için ayrı bir hat bulunur. Ardıl bağlantıda ise damgayı oluşturan bitler aynı hat üzerinden, zaman içinde sırayla iletilirler. Ekonomik nedenlerle, sürücü ile arabirimin birbirlerine göreli yakın bulunabildiği durumlarda koşut, uzak olduğu durumlarda ise ardıl bağlantı kullanılır. Ekonomik nedenlerin yanı sıra, verilerin sürücüler üzerinde saklanış biçimi de aktarım biçimini belirleyebilir. Bilindiği gibi, disk sürücülerde veri bitleri izlere, bu bitler ardarda gelecek biçimde yazılır. Mıknatıslı şerit sürücülerde ise veriler şerit üzerindeki kanallara, her kanala bir bit gelecek biçimde, koşut olarak kaydedilirler. Bu nedenle disk sürücü ve arabirimi arası bağlantı genelde ardıl türde, mıknatıslı şerit sürücü ve arabirimi arası bağlantı ise koşut türdedir.



Çizim 2.3. Zamanuyumlu ve Zamanuyumsuz Ardıl İletişimde İm Görünümleri

Arabirim ile sürücü arasındaki bağlantının ardıl bağlantı olması durumunda, iletişimin başlatılması ve hatasız sürdürülmesinde kullanılan zamanuyumlama yöntemine göre zamanuyumlu ve zamanuyumsuz iletişimlerden söz edilir. Zamanuyumsuz ardıl iletişimde aktarılan damga bitlerinin başına ve sonuna başlama *(start)* ve durma *(stop)* bitleri eklenir. Başlama biti, gönderici birimin alıcı birimi, aktarımın başladığı konusunda uyarmasını sağlar. Durma biti, aktarılan damgalar arası sınırların belirlenmesine yarar. Zamanuyumlu ardıl iletişimde ise, veri bitlerine ek bitler katmak yerine, gönderici ve alıcı birimlerin birbirleriyle sürekli zamanuyumlu kalmalarını sağlayan ve damgalar arasında gerektikçe iletilen, özel zamanuyumlama damgaları *(SYNC)* kullanılmaktadır. Bunları örneklemek üzere, Çizim 2.3'te zamanuyumlu ve zamanuyumsuz iletişim kapsamında, `3DH` ve `43H` *(ASCII)* damga kodlarının hattaki im görünümleri verilmiştir.

İki birim arasında yapılan iletişim, radyo yayınlarında olduğu gibi hep göndericiden alıcıya doğru gerçekleşirse yapılan iletişime tek yönlü (*simplex*), her iki yönde de olabiliyorsa çift yönlü (*duplex*) iletişim denir. Çift yönlü iletişim, telefonda olduğu gibi aynı anda her iki yönde olabiliyorsa tam çift yönlü (*full duplex*), telsiz iletişiminde

olduğu gibi, aynı anda yalnız bir yönde, ancak zaman içinde her iki yönde de olabiliyorsa, bu kez yarı çift yönlü (*half duplex*) iletişim olarak adlandırılır. Sürücüler ile arabirimler arasındaki veri alış-verişi genelde tam çift yönlüdür. Örneğin terminal sürücü ile arabirimi arasındaki veri iletişimi tam çift yönlüdür. Ancak kimi yazıcılar için, hep arabirimden sürücüye doğru, tek yönlü de olabilmektedir.

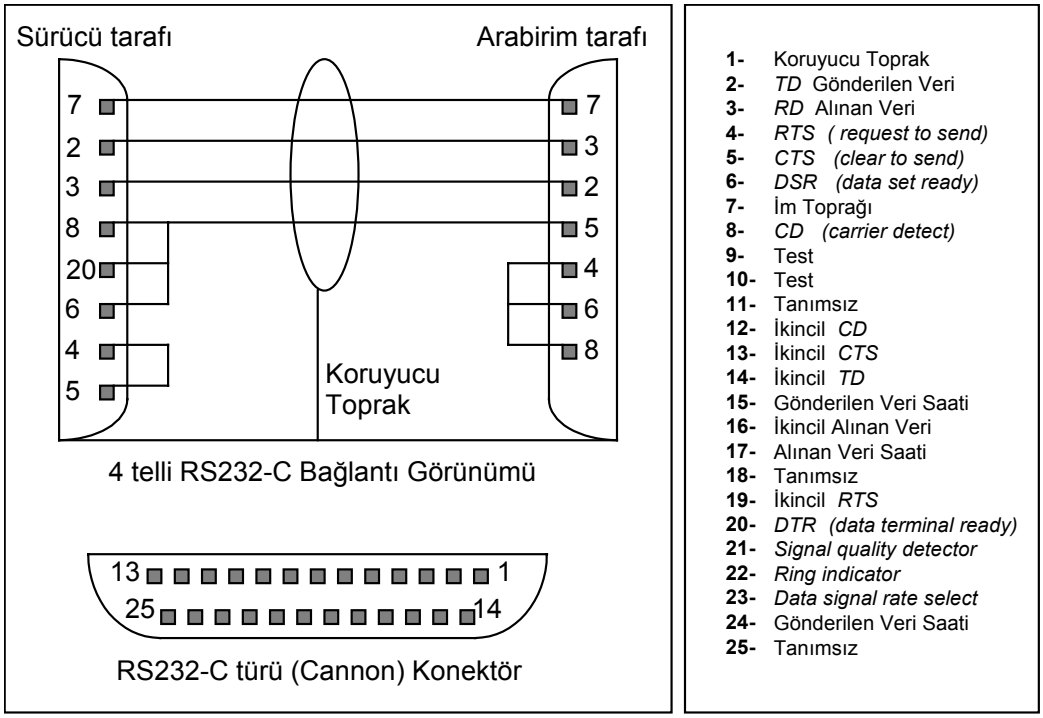
Giriş/çıkış sürücüsü ile arabirimin, her ikisi de aynı firma tarafından üretiliyorsa, genelde aralarındaki bağlantı biçimi ile iletişim protokolu da bu firmaya özel olmaktadır. Ancak terminal ve yazıcılarla başlayan ve özellikle *IBM* uyumlu kişisel bilgisayarlar ve *UNIX* tabanlı iş istasyonlarının yaygınlaşmasıyla süren gelişme çizgisinde, sürücü ve arabirim üreten firmalar çeşitlenmiştir. Değişik ellerde üretilen bu birimlerin bir bilgisayar sistemi içinde kolayca bütünleşebilmeleri için belirli bağlantı standartları ortaya çıkmıştır. Bu bağlamda, *EIA RS-232C*, *20 mA Courrent Loop*, *EIA RS-449*, *Centronix*, *SCSI* (*Small Computer System Interface*) gibi adlarla anılan, sürücü-arabirim bağlantı standartları en yaygın kullanılan standartlardır. Bu standartlar bağlantının (ardıl, koşut gibi) türünü, konektör, kablo gibi fiziksel bağlantı ögelerinin özelliklerini, iletilen bitleri simgeleyen elektriksel imlerin gerilim, akım düzeylerini, iletişim akışının denetlenmesinde kullanılan protokolu ayrıntılı olarak tanımlarlar. Bu tanımlar işlevsel, elektriksel, mekanik ve akış denetimi *(flow control)* gibi ana başlıklar altında gruplanırlar. Standartların tanımlanması, *RS-232C* örneğinde olduğu gibi, genelde, üreticilerin bir araya geldikleri meslek odaları eliyle olmaktadır. Ancak bir firmanın kullandığı özel tanımların çok yaygınlaşarak genel kabul görmesi de standartlara kaynaklık edebilmektedir. *Centronix* ve *SCSI* buna örnek gösterilebilen iki ünlü standarttır.

RS-232C standartı zamanuyumsuz ardıl iletişim kapsamında yaygın olarak kullanılan bir standarttır. Mekanik özellikler yönünden 25 iğneli özel bir konektör ile toprak korumalı örgülü kablo kullanmayı gerektirir. Konektör üstündeki iğneler, *TD* ve *RD* adlı (*transmitted data, received data*) gönderilen veri ve alınan veri hatlarının yanı sıra toprak (*GRD*), koruyucu toprak (*protective GRD*) hatları ile çeşitli akış denetim imlerinin iletildiği hatlara adanmıştır. Elektriksel özellikler yönünden, iletilen veri bitleri $\{\pm 3\text{volt}, \pm 25\text{volt}\}$ aralığındaki gerilim değerleriyle simgelenir. Mantıksal 0 $\{+3, +25\}$ volt aralığında, mantıksal 1 ise $\{-3, -25\}$volt aralığında bir değerle iletilir. *RS-232C* standartı, bitleri simgelemede gerilim düzeylerini kullandığından dengesiz (*unbalanced*) bir çevriminin oluşmasına neden olur.

RS-232C standartında alıcı ile gönderici arasında, izin verilen en büyük uzaklık yaklaşık 15 metredir. Bu uzaklıkları aşan bağlantılarda, elektriksel özellikler yönünden *RS-232C* standartından ayrılan ve mantıksal 0 ve 1'leri simgelemede gerilim düzeyleri yerine 4mA ve 20mA'lik akım değerlerini kullanan *20mA Courrent Loop* standartı kullanılır. Bu standart kapsamında kullanılan bağlantı yöntemine akım döngüsü anlamına gelen *current loop* adı verilir.

RS-232C standartı kapsamında kullanılan konektör görünümü ve bu konektör üzerinde yer alan 25 iğneye ilişkin im tanımları ile bu iğnelerden sadece 6'sını kullanan örnek bir bağlantı, Çizim 2.4'te gösterilmiştir. Bu çizim üzerinde yer alan 4, 5, 6, ve 20 numaralı

iğnelere karşıt gelen *RTS* (*request to send*), *CTS* (*clear to send*), *DSR* (*data set ready*) ve *DTR* (*data terminal ready*) imleri, yukarıda akış denetimi diye anılan gruba giren imlerdir. İletişim protokolunu gerçekleştirmede kullanılırlar. Bu bağlamda örneğin arabirim, *TD* (*transmitted data*) hattı üzerinden verileri, kendi *CTS* girişine bağlı, sürücünün, *DTR* ile gösterilen hazır durumunu saptamadan göndermez. Bu imler modem denetim imleri diye de anılırlar. *Modem*, İngilizce ***Mo****dulator* ***Dem****odulator* sözcüklerinden türetilmiş bir sözcük olup uzak bağlantılarda imlerin hatlar tarafından süzülmesini engelleyen özel iletişim gerecine verilen addır.



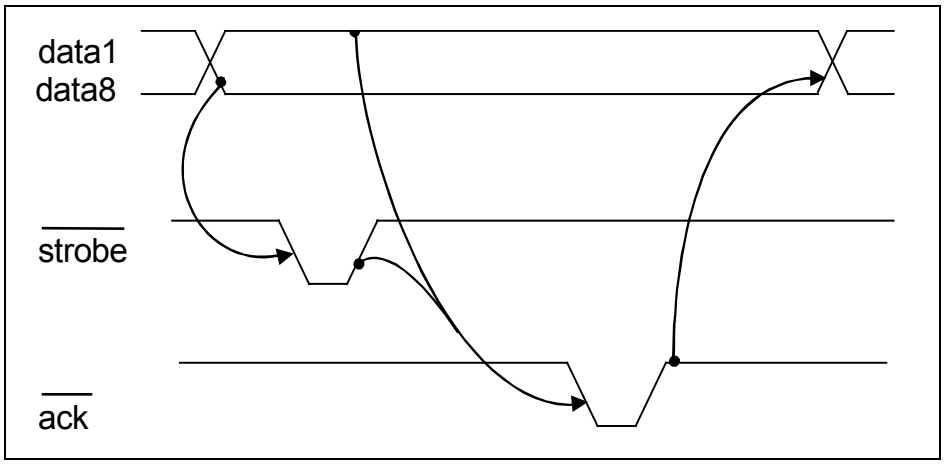
Çizim 2.4. *RS-232C* Standartında İğne Tanımları ve 4 telli Bağlantı Örneği

Zamanuyumsuz ardıl iletişimde, aktarım hızları, damga bit uzunlukları, bir bitlik iletişim hatalarını yakalamaya yarayan eşlik bitinin türü, *start* ve *stop* bitlerinden *stop* bitinin sayısı gibi özellikler, iletişim parametrelerini oluşturmaktadır.

- Aktarım hızı için 110, 200, 400, 600, 1200, 2400, 4800, 9600, 19200 *bps (bit per second*, bit / saniye);
- Damga bit uzunluğu için 5, 6, 7, 8 bit;
- Stop biti sayısı için 1, 1,5, 2 bit;
- Eşlik biti için tek, çift, yok

gibi değerler, telgrafçılıktan bu yana kullanılagelen, klasikleşmiş iletişim parametre değerleridir. Bu bağlamda damgaların, zamanuyumsuz ardıl iletişimle, 8 bit üzerinden, bir *start* bir de *stop* biti eklenerek 9600 *bps* hızında aktarıldığı varsayılırsa; saniyede, en çok 9600 / (1+8+1) = 960, yaklaşık 1 KB'lık damga aktarılabileceği düşünülmelidir.

Koşut iletişimde (özellikle yazıcı - arabirim bağlantılarında) yaygın olarak kullanılan standart *Centronix* standartıdır. Bu standartta da, *RS-232C* standartında olduğu gibi bağlantının işlevsel, elektriksel, mekanik ve akış denetimiyle ilgili tanımları bulunur. Bu bağlamda *Centronix* standartı, mekanik özellikler yönünden 36 iğneli (genelde *Amphenol* adlı markanın adıyla anılan) özel bir konektör ile toprak korumalı örgülü kablo tanımını içerir. 8 koşut veri hattı ile toprak hattının yanı sıra strobe, ack, busy gibi akış denetim imlerinin varlığını gerektirir. Çizim 2.5'te *Centronix* standartı konektör görünümü ile strobe ve ack imlerini içeren, akış denetimine ilişkin zaman çizeneği verilmiştir. Bu çizim üzerinde, *return* sözcüğünün kısaltılmış biçimi olarak rtn diye tanımlanan iğneler, data diye tanımlanan iğnelerden aktarılan verilerin karşı yönünde, tam çift yönlü aktarıma olanak vermek üzere öngörülmüştür.



Centronix Standartı Konektör İğne Tanımları

1	strobe	**19**	rtn	**10**	ack	**28**	rtn
2	data1	**20**	rtn	**11**	busy	**29**	rtn
3	data2	**21**	rtn	**12**	pe	**30**	rtn
4	data3	**22**	rtn	**13**	select	**31**	init
5	data4	**23**	rtn	**14**	af	**32**	error
6	data5	**24**	rtn	**15**		**33**	
7	data6	**25**	rtn	**16**		**34**	
8	data7	**26**	rtn	**17**		**35**	
9	data8	**27**	rtn	**18**		**36**	select in

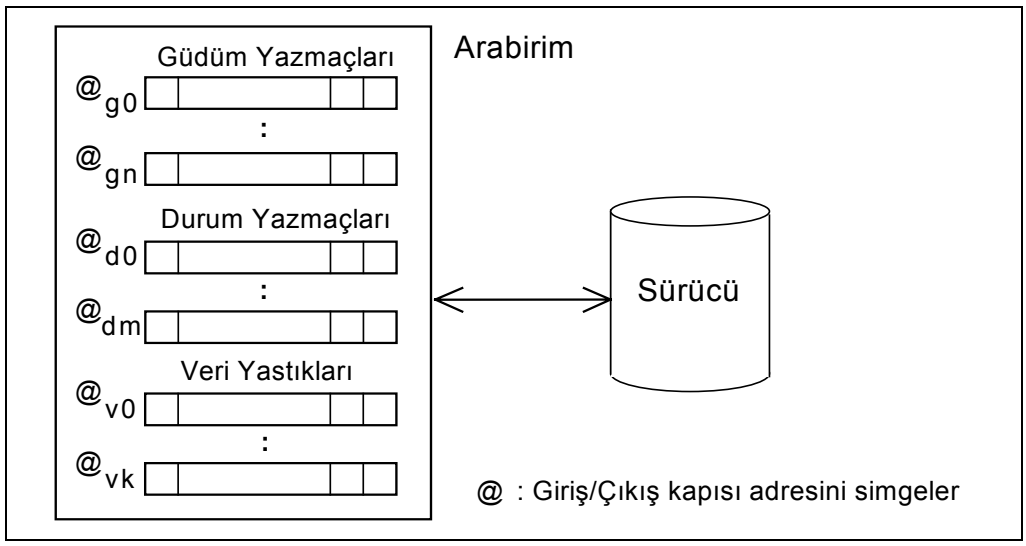
Çizim 2.5. *Centronix* Standartında Veri Aktarım Zaman Çizeneği ve İğne Tanımları

Centronix standartında mantıksal 1'in karşılığı 5 volt, mantıksal 0'ınki ise 0 volttur. Başka bir deyişle, bu standartta veri ve denetim imleri, *RS-232C* standartının tersine herhangi bir gerilim düzey değişikliği uygulanmadan, *TTL* düzeyi diye anılan {0 , 5} volt arabirim gerilim aralığında aktarılırlar. *Centronix* standartında arabirim sürücüye veri aktaracağı zaman önce busy (meşgul) hattını sınar. busy hattında sıfır değeri görülmeden aktarım işlemleri başlatılmaz. Arabirim busy hattında sıfır değeri bulmuşsa (sürücü meşgul değilse) aktarılacak bayt değerini data1 - data8 hatları üzerine yükler ve sürücüye bunu bildirmek üzere strobe imini kurar (sıfırlar). Damganın sürücü tarafından alındığı, ack imi sınanarak saptanır. Açıklanan bu imlerin yanı sıra, *Centronix* hat

tanımları arasında pe (*parity error*), error gibi hata sınama hatları ile arabirimin sürücüyü denetlemesine yarayan af (autofeed), init, select, select in gibi başka özel denetim ve adresleme hatları da bulunur.

2.1.2. Giriş/Çıkış Arabirimleri

Veriler, kullanıcı ortamı olarak da tanımlanan dış ortamdan ana işlem birimi ana bellek ikilisine giriş/çıkış sürücüleri aracılığıyla aktarılırlar. İşlem sonucu elde edilen sonuçlar da, gene giriş/çıkış sürücüleri aracılığıyla kullanıcılara iletilirler. Bunun yanı sıra, disk, mıknatıslı şerit gibi kimi giriş/çıkış sürücüleri, verilerin bilgisayar ortamında saklanmasına yararlar. Giriş/çıkış arabirimleri, verilerin sürücüler ile ana bellek arasında aktarılmasında kullanılır. Bir bilgisayar sisteminde yer alan giriş/çıkış sürücüleri tür, hız gibi değişik özellikleri yönünden çok büyük farklılıklar gösterirler. Değişik türdeki sürücüleri aynı ana işlem birimi ana bellek ikilisi ile bütünleştirmek ve sürücü - sistem arası veri aktarımlarını benzer kalıplar içinde ve sürücülerin karmaşık fiziksel yapılarını gizleyerek yürütmek giriş/çıkış arabirimlerinin temel işlevini oluşturur. Bu bağlamda giriş/çıkış arabirimleri, ilgili oldukları sürücünün türü ne olursa olsun bir dizi yazmaç ve yastıktan oluşan bir bütün olarak düşünülebilir.



Çizim 2.6. Giriş/Çıkış Arabirimi Görünümü

Veriler, sürücüler ile ana bellek arasında aktarılırken, önce arabirim düzeyinde yer alan yastıklara taşınırlar. Arabirim yastıklarındaki bu veriler, ikinci bir aşamada, aktarımın yönüne göre, ya ana belleğe ya da sürücülere aktarılırlar. Disk sürücüye yazılacak bir öbeklik veri, ilk aşamada disk arabirim yastığına taşınır. İkinci bir aşamada da buradan sürücüye aktarılır. Verilerin ana bellekten arabirim yastığına ya da buradan ana belleğe taşınması, arabirim yastığına yazma ya da okuma biçiminde, giriş/çıkış yordamlarının denetiminde gerçekleşir. Bu verilerin arabirim ile sürücüler arasında aktarımı ise, arabirim tarafından, kendiliğinden yerine getirilir. Bu bağlamda, bir damga, terminal ekranından görüntülenmek istendiğinde, damga kodunun terminal arabirimi çıkış yastığına yazılması yeterlidir. Arabirim yastığına yazılan damganın ekrana aktarılması

(görüntülenmesi), ek denetime gerek duyulmaksızın arabirim tarafından kendiliğinden gerçekleştirilir. Bu gerekçeyle, giriş/çıkış arabirimlerinin anlayışlı birimler oldukları söylenir.

Arabirim düzeyinde bulunan yazmaçlar, genelde güdüm ve durum yazmaçları adıyla anılır. Verilerin, sürücü ile arabirim yastığı arasında aktarım işlemleri güdüm yazmaç(lar)ının içine kimi özel kodlar yazılarak gerçekleştirilir. Bu özel kodlara karşı gelen karmaşık elektroniksel işlemler arabirim tarafından otomatik olarak yerine getirilir. Bu yolla sürücüler üzerinde yerine getirilen ve fiziksel yapılara bağımlı işlemlerin karmaşıklığı denetim yordamlarından (sistem programlarından) gizlenmiş olur. Arabirimler güdüm yazmaçları aracılığıyla programlanabilen anlayışlı birimlerdir. Durum yazmaçları, arabirim tarafından yürütülen aktarım işlem akışının ve sonuçlarının izlendiği yazmaçlardır. Bu yazmaç ve yastıkların herbirinin, genelde, ana bellek adres evreninin dışında, giriş/çıkış adres evreni olarak adlandırılan adres evreni içinde bir adresleri (kimlikleri) bulunur. Arabirim yazmaç ve yastıklarının herbiri giriş/çıkış kapısı olarak da adlandırılır.

Arabirim üstünde yer alan güdüm yazmaç(lar)ı:

- Arabirimin yerine getirebildiği bir dizi işlev arasından birinin seçilmesi,
- Sürücü üzerinde yerine getirilmesi istenen işlemlerin arabirime bildirilmesi

amacıyla kullanılır. Örneğin hem zamanuyumsuz, hem de zamanuyumlu türde iletişim yapmaya olanak veren bir arabirimin, sürücüsünün özelliğine göre iletişim türü, iletişim hızı gibi seçenekler güdüm yazmacı içeriğiyle belirlenir. Bunun gibi, bir disk sürücü için okuma yazma kafalarının izden ize hareket ettirilmesi, bir sektörün okunması, ya da yazılması gibi değişik işlemlerden (güdümlerden) birinin seçilmesi de, güdüm yazmacı aracılığıyla olur. Güdüm yazmacını oluşturan bitlerin herbirinin özel anlamı bulunur ve bu anlamlar arabirim donanım başvuru elkitaplarında açıklanır. Örneğin bir terminal arabiriminin sürücüsü ile yapacağı iletişimin hızı, arabirimi düzeyinde yer alan güdüm yazmacının üç biti ile belirlenebilir. Bu üç bit ile oluşturulabilen 8 değişik koddan herbiri 200, 400, 600, 1200, 2400, 4800, 9600, 19200 *bps* gibi iletişim hızlarından birini seçmeye yarayabilir. Bu bitler üzerinde, örneğin 110_b değeri, 9600 *bps* hızını seçmeyi sağlar. Güdüm yazmaçları çoğu kez salt yazılır özellikte olurlar.

Durum yazmaçları, güdüm yazmaçları ile belirtilen işlemlerin akış ve sonuçlarına ilişkin bilgilerin tutulduğu yazmaçlardır. Bir arabirim üzerinde bir ya da daha çok durum yazmacı bulunabilir. Durum yazmacını oluşturan her bit özel bir durum bilgisini ifade eder. Bir disk sürücü arabirim durum yazmacı, örneğin, okuma - yazma kafasının istenen ize götürülmesini sağlayan güdümün sonlanıp sonlanmadığına ilişkin durum bitini içerir. Tutanakların ana bellekten disk sürücüye aktarılmasını sağlayan yordamlar, örneğin bu biti sınayarak, ize erişim sonrasında öbek yazma işlemine geçebilirler. Bunun gibi, arabirimi aracılığıyla terminal sürücüye gönderilen bir damga kodunun gönderme işleminin sonlanıp sonlanmadığı, arabirim durum yazmacının ilgili biti üzerinden gözlenir. Durum yazmaçları, işlem sonuçlarının yanı sıra işlem hatalarının izlenmesine de olanak verir. Sürücüden arabirime aktarılan verilerin örneğin eşlik

bitleri, *CRC (Cyclic Redundancy Check)* damgaları otomatik olarak sınanıp varsa aktarım hataları, ilgili durum yazmaç bitleri kurularak belirtilir. Durum yazmaçları genelde salt okunur tür yazmaçlardır.

Arabirim veri yastıkları, verilerin, sürücü - ana bellek arası aktarımlar sırasında, arabirim düzeyinde geçici olarak tutuldukları bellek ögeleridir. Giriş/çıkış birimleri, ilgili oldukları sürücülere, bir seferde aktarılabilen veri birimi ile sınıflandırılabilirler. Eğer bir sürücüye, bir seferde:

- bir damga aktarılabiliyorsa bu sürücüyle ilgili giriş/çıkış birimine damga tabanlı;
- en az bir öbek aktarılabiliyorsa bu sürücüyle ilgili giriş/çıkış birimine ise öbek tabanlı

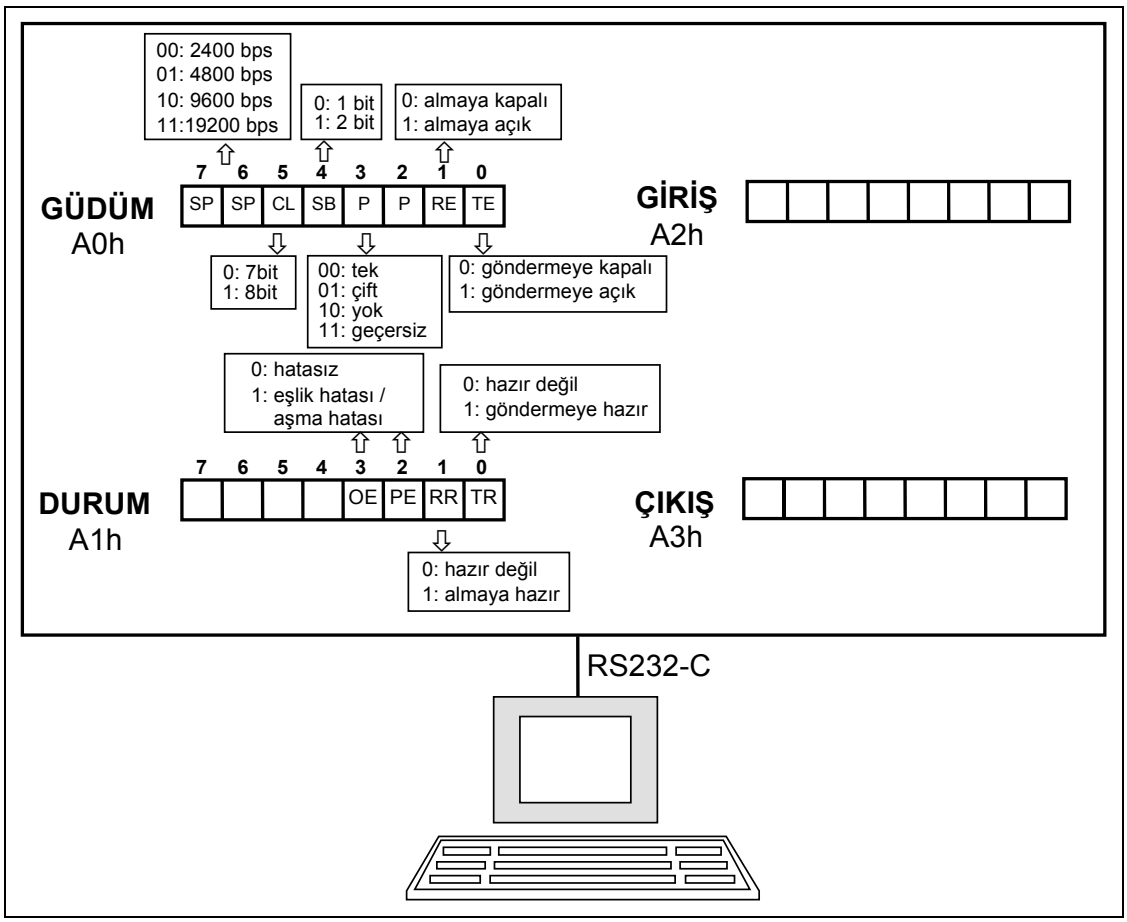
giriş/çıkış birimi denmektedir. Arabirimler üzerinde yer alan veri yastık sayısı bu sınıflandırmaya koşut olarak belirlenmektedir. Damga tabanlı bir giriş/çıkış arabirimi düzeyinde en az bir damgalık, öbek tabanlı bir arabirim düzeyinde ise, en az bir öbeklik giriş/çıkış yastığı bulunur. Bu bağlamda, damga tabanlı bir giriş/çıkış birimi olan terminal biriminde, örneğin bir damgalık giriş, bir damgalık da çıkış yastığı bulunabilir. Öbek tabanlı bir giriş/çıkış birimi olan disk biriminin ise, sürücüden bir seferde okunabilen öbeğin (bir ya da birkaç sektörün) içerdiği bayt sayısı uzunluğunda giriş/çıkış yastık alanları içerdiği düşünülebilir. *RAM* türü bellek yongalarının ucuzlamasına koşut olarak giriş/çıkış yastıklarının sığası da artmıştır. Arabirim düzeyinde yer alan giriş/çıkış yastıkları, gene bu ucuzlamaya koşut olarak, sürücüler üstüne eklenen, özellikle disk sürücülerinde mıknatıslı ortama erişim sayısını azaltma amacını güden sürücü ön (*cache*) bellekleriyle karıştırılmamalıdır.

Giriş/çıkış arabirimleriyle ilgili olarak, şimdiye değin açıklanan noktalar, varsayımsal bir terminal ve bir de disket arabirimi üzerinde örneklenmiştir (Çizim 2.7, 2.8). Sözkonusu özel terminal arabirimiyle ilgili varsayımlar şunlardır:

- Terminal arabirimi, terminal sürücüye[9] *RS-232C* standartında, zamanuyumsuz ve tam çift yönlü olarak bağlıdır.
- Arabirim düzeyinde birer damgalıklık bir Güdüm ve bir Durum yazmacı ile bir Giriş, ve bir Çıkış yastığı bulunmaktadır. Giriş yastığı klavyeden basılan tuş kodlarının ana belleğe aktarılmadan önce saklandığı, Çıkış yastığı ise ekrandan görüntülenmesi istenen damga kodlarının arabirim düzeyinde yazıldığı yastıklardır.
- Arabirimin yer aldığı bilgisayar sistemi giriş/çıkış adres evreninde:
  - Güdüm Yazmacının `A0H`
  - Durum Yazmacının `A1H`
  - Giriş Yastığının `A2H`
  - Çıkış Yastığının ise `A3H` fiziksel adresine atandığı varsayılmıştır.

[9] Terminal sürücü, *Hiperterminal* programı çalıştıran ve bu program aracılığıyla *RS232C* ardıl bağlantı kapısı üzerinden arabirime bağlanan bir kişisel bilgisayar *(PC)* sistemi olarak düşünülebilir.

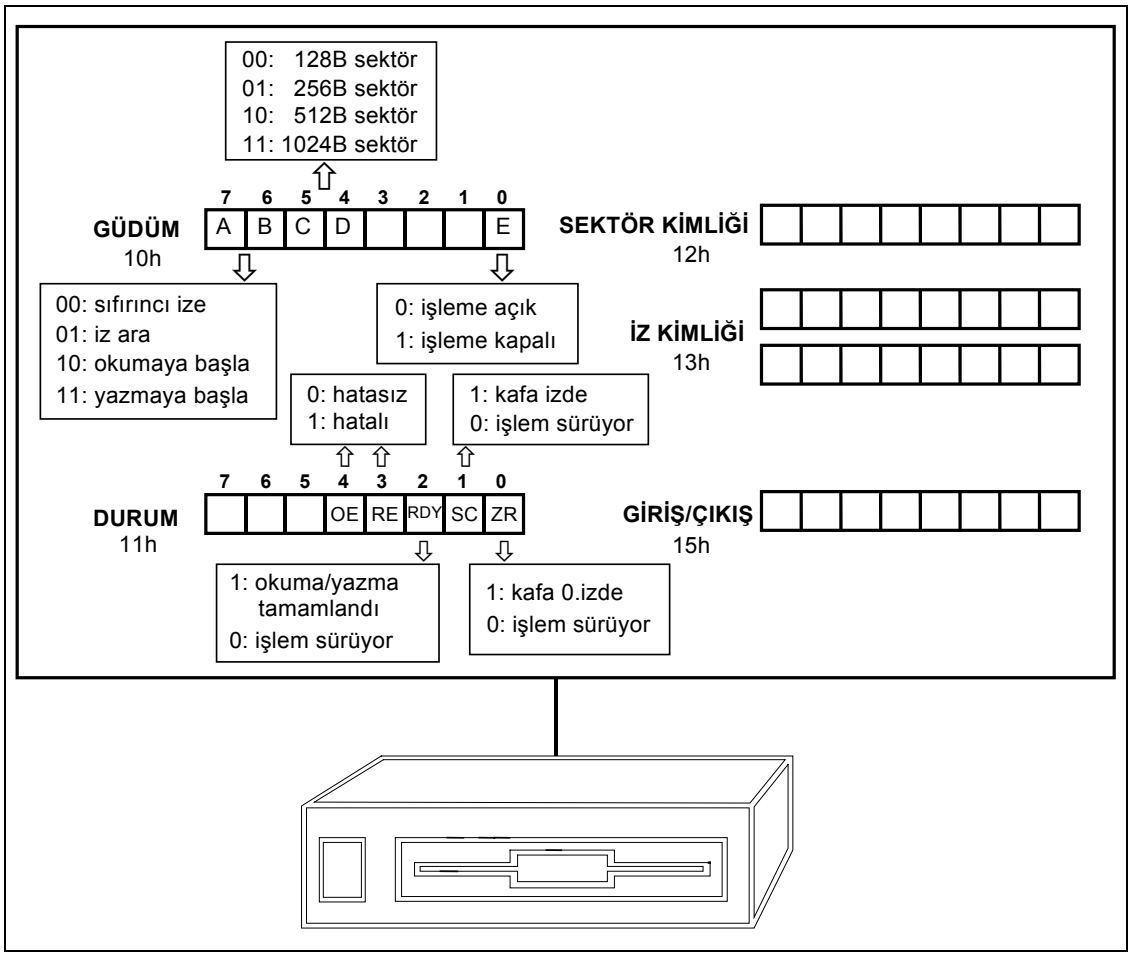
- Arabirim - terminal iletişiminde:
  - SP iletişim hızı, {19200, 9600, 4800, 2400} *bps*,
  - CL damga boyu, {7, 8 bit},
  - SB stop biti sayısı, {1, 2},
  - P eşlik sayısı, {tek, çift, yok} değerlerinden birine kurulabilmektedir.
- Arabirim hem almaya hem göndermeye, ayrı ayrı kurulabilmektedir.
- Arabirim Durum yazmacı içinde, arabirim veri göndermeye hazır (TR), almaya hazır (RR), eşlik hatası (PE), aşma hatası (OE) gibi bilgiler üretilebilmektedir.



Çizim 2.7. Örnek Terminal Arabirimi Görünümü

Bu durumda, *ascii* kodlu bir damganın, terminal ekranından görüntülenebilmesi için:

- Arabirim Güdüm yazmacına, terminal sürücü ile; bir stop bit, çift eşlik sayısı ve 9600 *bps* hızında iletişim kurmayı sağlayan; gönderme ve alma işlemlerine arabirimi açan (`10000111`) değerin yazılması,
- TR durum bitiyle, arabirimin gönderime hazır olduğunu sınayıp, sözkonusu damga kodunu arabirim Çıkış yastığına yazmak gereklidir.



Çizim 2.8. Örnek Disket Arabirimi Görünümü

Özel disket arabirimiyle ilgili varsayımlar şunlardır:

- Disket arabirimi sürücüye ardıl ve standart dışı bir biçimde bağlıdır.
- Arabirimde birer baytlık bir Güdüm Yazmacı, bir Durum Yazmacı, bir Sektör Kimliği Yazmacı ile 2 bayt uzunluğunda İz Kimliği Yazmacı bulunmaktadır.
- Arabirimde, okumada giriş, yazmada da çıkış yastığı gibi davranabilen bir baytlık Giriş/Çıkış adlı bir yastık yer almaktadır.
- Arabirimin yer aldığı bilgisayar sistemi giriş/çıkış adres evreninde:
  - Güdüm Yazmacının `10H`
  - Durum Yazmacının `11H`
  - Sektör Kimliği Yazmacının `12H`
  - İz Kimliği Yazmacının `13H`
  - Giriş/Çıkış Yastığının ise `15H` adresine atandığı varsayılmıştır.
- Güdüm yazmacının en büyük ağırlıklı iki biti ile:
  - okuma - yazma kafasının sıfırıncı ize götürülmesi,

- okuma - yazma kafasının, kimliği, İz Kimliği Yazmacı içinde bulunan ize götürülmesi,
- kimliği Sektör Kimliği Yazmacı içinde bulunan sektör baytlarının, ardarda Giriş/Çıkış yastığına okunmaya başlanması ve
- Giriş/Çıkış yastığından, kimliği, Sektör Kimliği Yazmacı içinde bulunan sektöre yazmanın başlatılması işlemleri kodlanabilmektedir.

Bu komutlar çerçevesinde, disket sürücü üzerinde bulunan bir sektörün okunması ya da yazılması için, önce okuma - yazma kafasının ilgili iz üstüne getirilmesi, bunun sonrasında okuma - yazma işlemine başlanması gerekmektedir.

- Güdüm yazmacının en büyük ağırlıklı iki bitini izleyen diğer iki bit ile 128, 256, 512 ve 1024 baytlık sektör boyları seçilebilmektedir.
- Güdüm yazmacının en küçük ağırlıklı biti ile disk işlem izni (E) açılıp kapanabilmektedir.
- Durum yazmacının değişik bitleri ile gösterilen:
  - ZR okuma-yazma kafası sıfırıncı izde,
  - SC okuma-yazma kafası, kimliği İz Kimliği yazmacında bulunan izde,
  - RDY giriş/çıkış yastığı okunmaya ya da yazılmaya hazır,
  - RE sürücüden arabirime aktarımda hata (*CRC* hatası) oluştu,
  - OE aşma hatası oluştu durum bilgileri üretilmektedir.

  RDY biti, okuma-yazma işleminin başlatılmasının peşisıra, ardarda (sürücüden arabirim giriş/çıkış yastığına) okunan ya da (giriş/çıkış yastığından sürücüye) yazılan baytların herbirinin okuma ya da yazma işleminin gerçekleşmesi sonrasında kurulmaktadır.
- SC durum biti, okuma-yazma kafası, ilgili güdüm gereği istenen ize ulaştığında kurulmakta, arabirime yeni bir güdüm değeri yazılana değin kurulu kalmaktadır. Benzer açıklama ZR durum biti için de geçerlidir.

Bu durumda örneğin, 512 baytlık bir tutanağın ana bellekten disket sürücü üzerinde n inci izdeki m kimlikli sektöre yazılabilmesi için:

- Arabirim İz Kimliği yazmacı içine (n) değerinin yazılması,
- Sektör Kimliği yazmacının içine (m) değerinin yazılması,
- Güdüm yazmacının içine okuma-yazma kafasının n inci ize götürülmesini sağlayacak güdümün (`01100001` değerinin) yazılması,
- İstenen işlemin gerçekleştiğinin, SC durum biti aracılığıyla algılanması,
- Bunun sonucunda, Güdüm yazmacının içine yazma işlemini başlatacak güdümün (`11100001` değerinin) yazılması,

- RDY durum bitinin kurulmasıyla, sektörün ilk baytının Giriş/Çıkış yastığına yazılarak sürücüye aktarılmasının sağlanması ve bu işlemin, sektörü oluşturan diğer baytlar için de (512 kez) yinelenmesi gereklidir.

Gerek terminal arabirimi gerekse disket arabirimi örneğinde belirtildiği gibi, verilerin ana bellekten sürücüye ya da sürücüden ana belleğe aktarılabilmeleri için arabirim güdüm yazmacı ile aktarım işlemi alt adımlarını sırayla gerçekleştirmek ve bir adımdan izleyen adıma geçişi durum bitlerini kullanarak denetlemek gerekmektedir. Arabirim güdüm yazmaç içeriklerini günlemek, durum yazmacını okuyarak ilgili bitleri sınamak, giriş/çıkış yastık içeriklerini okumak ya da yazmak gibi işlemler, doğal olarak sistemde çalışan kimi yordamlar tarafından gerçekleştirilir. Bu yordamlar giriş/çıkış sürücü yordamları olarak adlandırılır.

Giriş/çıkışların programlanması, giriş/çıkış işlem akışına göre:

- Seçmeli,
- Kesilmeli ve
- Doğrudan Bellek Erişimli

diye üç değişik yöntemle ele alınmaktadır. Bu yöntemler ve kullandıkları özel donanım düzenekleri, izleyen kesimde açıklanacak ve örneklenecektir.

2.2. Seçmeli Giriş/Çıkış Programlama

Seçmeli giriş/çıkış programlamada işlem akışı, sürücü yordamın, giriş/çıkış arabirimi durum yazmaç içeriğini okuyup ilgili durum bitini sınaması yoluyla gerçekleştirilir. İlgili durum bitinin kurulu bulunduğu durumlarda, giriş/çıkış yastığı üzerinde okuma ya da yazma işlemlerinden biri yerine getirilir. İlgili bitin kurulu olmadığı durumlarda ise durum yazmacı yeniden okunarak sınama işlemleri sürdürülür. Sözlü olarak açıklanan bu denetim algoritması, yukarıda verilen terminal arabirimine bağlı klavyeden girilen bir damga kodunu, `dizgi` adlı bir ana bellek alanına aktaran *ASM86* komut dizisi ile aşağıda örneklenmiştir:

```
            .
            .
yine:       in    al, durum
            test  al, 00000010B
            jz    yine
            in    al, giriş
            mov   dizgi[si], al
            .
            .
```

Seçmeli giriş/çıkış programlamanın temel ilkesi, aktarım yapılmak üzere seçilen birimin, istenen işlem gerçekleşene, ya da aranan koşul oluşana değin sürekli sınanmasıdır. Yukarıda verilen komut dizisinde de, arabirime bağlı klavyeden herhangi bir tuşa basılana değin, `yine` etiketi ile kurulan döngü içinde kalınmakta, bu döngüden ancak basılan damga kodunun giriş yastığı içinde okunmaya hazır olması durumunda çıkılabilmektedir.

Seçmeli giriş/çıkış programlama yöntemini kullanan ve `ctrl-z` damgasına kadar girilen damga kodlarını arabirimden okuyan ve bunları `dizgi` adlı ana bellek alanına yazan örnek program Çizim 2.9'da verilmiştir. Normal koşullarda, giriş/çıkış birimlerini süren yordamlar, giriş/çıkış sisteminin üst katmanlara sunduğu sistem çağrı hizmetlerini karşılamak üzere öngörülen ve giriş/çıkış sisteminin içinde düşünülen yordamlardır. Çizim 2.9'da verilen program (ve izleyen kesimde verilen diğer örnek programlar), pedagojik nedenlerle, kullanıcı düzeyi bir program gibi düzenlenmiştir. Burada birincil amaç, giriş/çıkış sürücü yordamların giriş/çıkış sistemi içinde nasıl yeraldıklarını örneklemek yerine seçmeli giriş/çıkış programlama ilkesini, ayrıntılardan arınmış bir biçimde örneklemek olmuştur. Giriş/çıkış sürücü yordamların giriş/çıkış sistemi içinde nasıl yeraldıkları Aygıt Sürücüler başlıklı 8. kesimde açıklanacaktır.

Bu program içinde `dizgi` adlı alan 256 baytlık bir alan olarak öngörülmüştür. Klavyeden 256 damga girilmeden `ctrl-z` tuşuna basılacağı varsayılarak alan taşma denetimi ile durum yazmacı içinden sınanabilen eşlik ve taşma hata denetimleri, programı karmaşıklaştırmamak amacıyla yapılmamıştır. Programın hemen başında veri kesim yazmacını (`ds`) günleyen komutlar, sistemi kesilmelere kapayıp açan `cli` *(clear interrupt)* ve `sti` *(set interrupt)* komutlarıyla ayraç arasına alınmıştır. Bu komutlar ve kesilme kavramı izleyen kesimde açıklanacaktır. Bu ayracın peşi sıra güdüm yazmacı içine `ilk-güdüm` adlı `10000111B` değeri yazılarak terminal arabiriminin 9600 *bps* hızında, bir stop bit ve çift eşlik sayısıyla, hem göndermeye hem de almaya izinli olarak çalışmaya başlaması sağlanmıştır.

Okunan her damganın, klavye ile ilgili ekrandan da görüntülenmesi sağlanmıştır. Bunun amacı, damgaları giren kullanıcıya, girdiği damgaların sistem tarafından alındığı konusunda geri bildirim sağlamaktır. Klavyeden girilen damgaların arabirime ulaştıktan sonra arabirim tarafından, ekrandan görüntülenmek üzere sürücüye geri gönderilmesi işlemi damga yankılama olarak adlandırılır. Program içinde bu:

```
yine2:      in    al, durum
            test  al, gönd-hazır
            jz    yine2
            pop   ax
            out   çıkış, al
```

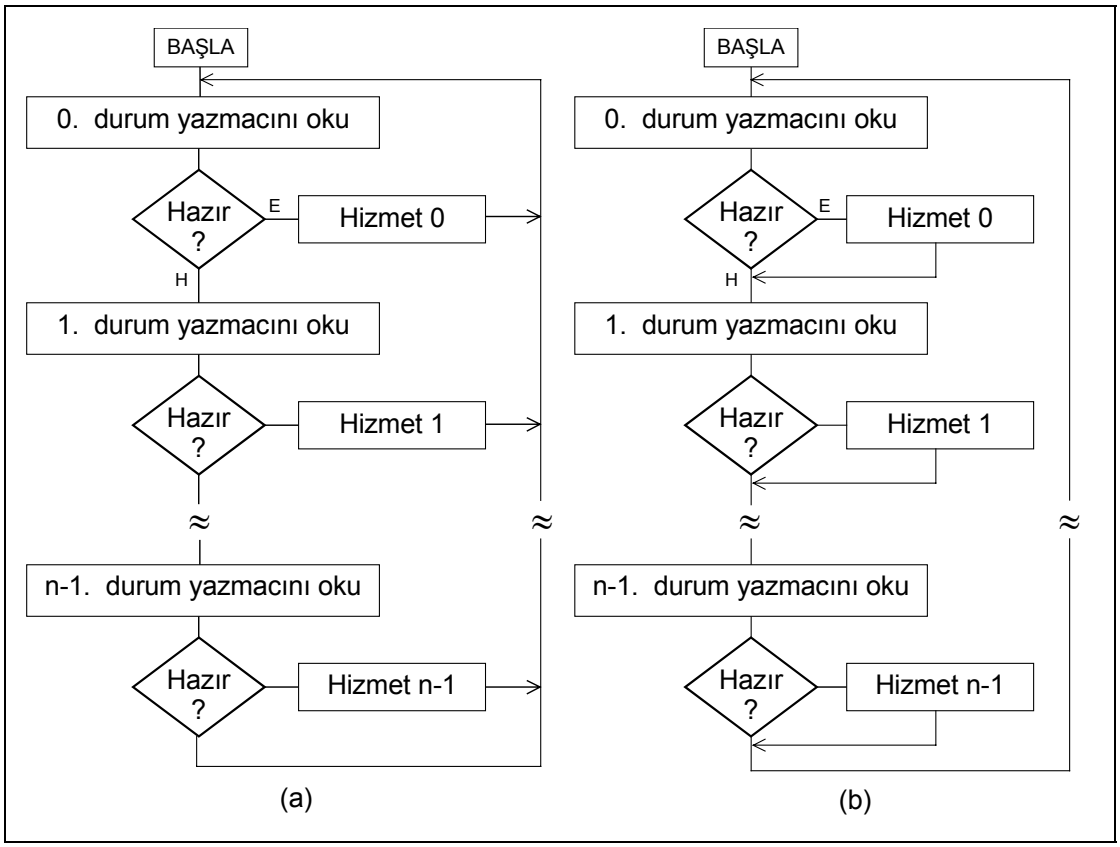
komutlarıyla gerçekleştirilmiştir. `al` yazmacı, hem damga kodunu hem de durum yazmacı içeriğini okumada kullanıldığından damga kodu önce yığıta saklanmış sonra da `pop` komutu ile geri alınarak çıkış yazmacına yazılmıştır. `push` ve `pop` komutlarının kullanılıyor olması programın başında yığıt kesim yazmacı (`SS`) ile yığıt göstergesinin (`SP`) günlenmesini gerekli kılmıştır.

Yukarıda da belirtildiği üzere, Çizim 2.9'da verilen program, örnek bir giriş/çıkış arabiriminin seçmeli giriş/çıkış programlama tekniği kullanılarak denetlenmesini örneklemek üzere, giriş/çıkış sistemi dışında, kullanıcı düzeyi bir program gibi düzenlenmiştir. Bu nedenle, bu program sonlandığında, bu programın kullanıcı düzeyinde çalıştırıldığı işletim sistemine geri dönüşün de öngörülmesi gerekmiştir. `ctrl-z` damgasına rastlanması durumunda sapılan ve bir makro gibi düşünülen `terminate` komutu bu amaçla kullanılmıştır.

```
;Tek terminal için Seçmeli Giriş/Çıkış Sürücü Programı
 veri            segment
 güdüm           equ   0a0h
 durum           equ   0a1h
 giris           equ   0a2h
 çıkıs           equ   0a3h
 ilk-güdüm       equ   10000111B
 gönd-hazır      equ   00000001B
 almaya-hazır    equ   00000010B
 ctrl-z          equ   1ah
 dizgi           db    256 dup(?)
 veri            ends

 yığıt           segment stack
                 db    100dup(?)
 yığıt-başı      equ this word
 yığıt           ends

 kod             segment
                 assume cs:kod,ds:veri,ss:yığıt
 damga-oku:      cli
                 mov   ax, veri
                 mov   ds, ax
                 mov   ax, yığıt
                 mov   ss, ax
                 mov   sp, offset yığıt-başı
                 mov   al, ilk-güdüm
                 out   güdüm, al
                 xor   si, si
                 sti
 yine1:          in    al, durum
                 test  al, almaya-hazır
                 jz    yine1
                 in    al, giriş
                 push  ax
                 mov   dizgi[si], al
                 cmp   al, ctrl-z
                 jz    son
 yine2:          in    al, durum
                 test  al, gönd-hazır
                 jz    yine2
                 pop   ax
                 out   çıkış, al
                 inc   si
                 jmp   yine1
 son:            terminate   ;işletim sistemine sapış
 kod             ends
                 end   damga-oku
```

Çizim 2.9. Tek Terminal için Seçmeli Giriş/Çıkış Programlama Örneği

Seçmeli giriş/çıkış programlamanın çalışma ilkesi, şimdiye değin, sistemde tek bir giriş/çıkış birimi varmış gibi açıklanmıştır. Ancak en yalın bir bilgisayar sisteminde bile birçok giriş/çıkış biriminin yer aldığı bilinir. Giriş/çıkış birimlerinin birden çok olduğu durumlarda seçmeli giriş/çıkış programlama yönteminin temel ilkesi değişmez. Tek bir arabirime ilişkin durum yazmacını sınamak yerine sistemde yer alan tüm arabirimlerin durum yazmaçları sırayla okunarak hazır durumda olanların gerektirdikleri hizmetler (hizmet yordamlarına sapılarak) sağlanır. Arabirimlerin durum yazmaçlarının sırayla sınanması arabirimlerin taranması olarak adlandırılır.



Çizim 2.10. Seçmeli Giriş/Çıkış Programlamada Önceliklerin Ele Alınması

Birden çok giriş/çıkış biriminin bulunduğu bir sistemde, bunlardan iki ya da daha çoğunun aynı anda (veri yastıklarının okunup yazılması gibi) hizmet istemleri söz konusu olduğunda, hangi birime daha önce hizmet verileceği sorunu ortaya çıkar. Bu sorun öncelik sorunu olarak bilinir. Seçmeli giriş/çıkış programlama yönteminde birimler arası öncelikler arabirim durum yazmaçlarının taranma sıra ve sıklığıyla belirlenir. En öncelikli birim en sık taranan birimdir. Eğer bir sistemde, 0 'dan (n-1)'e kadar numaralanan n adet giriş/çıkış birimi bulunduğu düşünülürse, en öncelikli olduğu varsayılan 0ıncı birimin durum yazmacı öncelikle okunur, ilgili durum bitleri aracılığıyla birimin hizmet gereksinimi bulunup bulunmadığı sınanır. Eğer birim hizmet gerektiriyorsa ilgili yordama sapılır. Yordam dönüşünde yine bu birimden başlamak üzere tarama işlemi yinelenir. Eğer birim hizmet gerektirmiyorsa bir sonraki birimin

durum sınamasına geçilir. i inci birimin hizmet almak için hazır olmadığı durumda (i+1)inci birimin durum sınamasına geçilir. i inci birimin durum sınaması sonucu hizmet yordamına sapılmışsa bu yordamdan geri dönüşte sınamalara 0ıncı birimden, yeniden başlanır. Seçmeli giriş/çıkış programlama yönteminde birimler arası önceliklerin ele alınışını gösteren akış çizgesi Çizim 2.10.a' daki gibidir. Sistemde yer alan tüm birimlerin eş öncelikli oldukları varsayıldığında taramalar, Çizim 2.10.b'deki akış çizgesine göre yapılır. Bu iki yaklaşım arasındaki tek ve önemli ayrım, hizmet yordamlarından geri dönüşte başa (0ıncı birime) ya da izleyen (i+1inci) birimin sınanmasına geçilmesidir.

Seçmeli giriş/çıkış programlama yönteminde önceliklerin ele alınışı Çizim 2.11'de verilen program ile örneklenmiştir. Bu örnek programda, 3 değişik terminal birimi klavyelerinden girilen damgaların, `ctrl-z` damgasına rastlanana değin ilgili ekrandan görüntülenmesi ve `dizgi(i)` olarak adlandırılan ilgili bellek alanlarına yazdırılması gerçekleştirilmektedir. Terminal arabirimleri üzerindeki yazmaç ve yastıkların, sistem giriş/çıkış adres evreni içindeki adreslerinin, i inci terminal için (i = {0,2}):

Güdüm Yazmacı adresi: `A0+4xi`
Durum Yazmacı adresi: `A0+4xi+1`
Giriş Yastığı adresi: `A0+4xi+2`
Çıkış Yastığı adresi: `A0+4xi+3` olduğu varsayılmıştır.

Terminallerin, bir önceki örnekte olduğu gibi 9600 *bps* hızında, bir stop bit, çift eşlik sayısı ile çalıştığı varsayılmış ve arabirimlerin önbelirlemeleri buna göre yapılmıştır. Çizim 2.9'da verilen programda olduğu gibi, `push` ve `pop` gibi yığıtı kullanan komutlardan yararlanılıyor olması yığıt kesim yazmacı ile yığıt göstergesinin günlenmesini gerekli kılmıştır. Program, her üç terminalin eş öncelikli olduğu varsayılarak yazılmıştır. Bu nedenle terminal arabirimlerinin durum yazmaçları sırayla okunarak `almaya-hazır` damga bulunup bulunmadığı sınanmakta; varsa o terminalle ilgili hizmet kesimine sapılmakta; hizmet kesiminin sonunda ise, izleyen terminalin durum sınamasının yapıldığı komuta dönülmektedir.

Hizmet kesiminde, önce, eşlik ve aşma hataları ile terminalin açık olup olmadığı sınanmaktadır. `hata` durumunda ilgili terminalin giriş yastığı içeriği okunarak durum bitlerinin sıfırlanması sağlanmaktadır. Klavyeden arabirime gelen damga `ctrl-z` damgası ise terminalin, arabirimi düzeyinde alma ve gönderme işlemlerine kapatıldığı program kesimine sapılmaktadır. Burada ilgili güdüm yazmacını, arabirimi alma ve göndermeye kapayan değerle günlemenin yanı sıra `sayaç` adlı bir değişken içeriği de bir artırılmaktadır. Bunun amacı, her üç terminalden de `ctrl-z` damgasının alınması durumunda programın işletimini sonlandırabilmektir. `sayaç` içeriğinin üçe ulaşması, her üç terminalde de, damga giriş işlemlerinin sonlandığını göstermektedir. Bu nedenle, terminallerin durum yazmaçlarının sırayla sınandığı kesimde `sayaç` 3 değeriyle karşılaştırılarak program işletiminin sonlanmasına karar verilebilmektedir.

```
;Üç terminal için Seçmeli Giriş/Çıkış Sürücü Programı

veri            segment
güdüm           equ   0a0h
durum           equ   0a1h
giris           equ   0a2h
çıkıs           equ   0a3h
ilk-güdüm       equ   10000111B
gönd-hazır      equ   00000001B
almaya-hazır    equ   00000010B
hatalı          equ   00001100B
kapat           equ   10000100B
ctrl-z          equ   1ah

dizgi0          db    256 dup(?)
dizgi1          db    256 dup(?)
dizgi2          db    256 dup(?)
dizin0          db    0
dizin1          db    0
dizin2          db    0
sayaç           db    0
veri            ends

yığıt           segment stack
                db    100 dup(?)
yığıt-başı      equ this word
yığıt           ends

kod             segment
                assume cs:kod, ds:veri, ss:yığıt
ana-yor:
                cli
                mov   ax, veri
                mov   ds, ax
                mov   ax, yığıt
                mov   ss, ax
                mov   sp, offset yığıt-başı

                mov   al, ilk-güdüm
                out   güdüm, al
                out   güdüm+4, al
                out   güdüm+8, al

                mov   dizin0,0
                mov   dizin1,0
                mov   dizin2,0
                mov   sayaç,0
                sti
```

Çizim 2.11. Üç Terminal için Seçmeli Giriş/Çıkış Programlama Örneği

```
  yine0:
                  in    al, durum
                  test  al, almaya-hazır
                  jnz   hizmet0
  yine1:
                  in    al, durum+4
                  test  al, almaya-hazır
                  jnz   hizmet1
  yine2:
                  in    al, durum+8
                  test  al, almaya-hazır
                  jnz   hizmet2
                  cmp   sayaç, 3
                  jz    bitti
                  jmp   yine0

  bitti:          terminate

.....................................................

  hizmet0:
                  test al, hatalı
                  jnz   hata0
                  lea   bx, dizgi0
                  mov   si, dizin0
                  in    al, giriş
                  mov   [bx][si], al
                  push  ax
  dön0:
                  in    al, durum
                  test  al, gönd-hazır
                  jz    dön0
                  pop   ax
                  out   çıkış, al
                  cmp   al, ctrl-z
                  jz    son0
                  inc   dizin0
                  jmp   yine1
  hata0:
                  in    al, giriş
                  jmp   yine1
  son0:
                  mov   al, kapat
                  out   güdüm, al
                  inc   sayaç
                  jmp   yine1
```

Çizim 2.11. Üç Terminal için Seçmeli Giriş/Çıkış Programlama Örneği (devam)

```
hizmet1:
                test al, hatalı
                jnz  hata1
                lea  bx, dizgi1
                mov  si, dizin1
                in   al, giriş+4
                mov  [bx][si], al
                push ax
dön1:
                in   al, durum+4
                test al, gönd-hazır
                jz   dön1
                pop  ax
                out  çıkış+4, al
                cmp  al, ctrl-z
                jz   son1
                inc  dizin1
                jmp  yine2
hata1:
                in   al, giriş+4
                jmp  yine2
son1:
                mov  al, kapat
                out  güdüm+4, al
                inc  sayaç
                jmp  yine2


hizmet2:
                test al, hatalı
                jnz  hata2
                lea  bx, dizgi2
                mov  si, dizin2
                in   al, giriş+8
                mov  [bx][si], al
                push ax
dön2:
                in   al, durum+8
                test al, gönd-hazır
                jz   dön2
                pop  ax
                out  çıkış+8, al
                cmp  al, ctrl-z
                jz   son2
                inc  dizin2
                jmp  yine0
```

Çizim 2.11. Üç Terminal için Seçmeli Giriş/Çıkış Programlama Örneği (devam)

```
hata2:
                in    al, giriş+8
                jmp   yine0
son2:
                mov   al, kapat
                out   güdüm+8, al
                inc   sayaç
                jmp   yine0

kod             ends
                end   ana-yor
```

Çizim 2.11. Üç Terminal için Seçmeli Giriş/Çıkış Programlama Örneği (devam)

Bilgisayar sistemlerinde verilerin, giriş/çıkış sürücülerinden sisteme sunuluşları genelde hep rasgele anlarda ortaya çıkmaktadır. Bir klavye aracılığıyla girilen damgaların sisteme hangi anlarda sunulacağını kestirmek, tuşlara ne zaman basacağını öngörmek gibi olanaksızdır. Kullanıcıların tuşlara basmaları gibi, zaman içinde rasgele anlarda oluşan eylemler, sistem açısından zamanuyumsuz gelişen eylemlerdir. Seçmeli giriş/çıkış programlama yönteminde, rasgele gelişen olayları izlemek için arabirim durum yazmaç bitlerini sürekli sınamak gerekmektedir. Bu durum, giriş/çıkış sürücülerinden veri aktarım istemi olmadığı zamanlarda bile, bu istemin ortaya çıkış anını yakalayabilmek üzere, sürekli sınamaları zorunlu kılmaktadır. Bu sınamalar ana işlem birimi zamanının gereksiz işlemlerle yitirilmesi sonucunu doğurmaktadır.

Seçmeli giriş/çıkış programlama yönteminin bir diğer sakıncası da, sistem işlem hızını sürücünün hızına indirmesidir. Bu sakınca, `dizgi` adlı bir bellek alanının terminal ekranından görüntülenmesini, seçmeli giriş/çıkış yöntemiyle gerçekleştiren, aşağıdaki program kesimiyle örneklenebilir:

```
                .
                .
                xor   si, si
dön:            in    al, durum
                test  al, gönd-hazır
                jz    dön
                mov   al, dizgi[si]
                out   çıkış, al
                cmp   al, ctrl-z
                jnz   son
                inc   si
                jmp   dön
son:            .
                .
```

Terminal sürücü ile arabirimi arasında 9600 *bps* hızında bir iletişim sözkonusu edildiğinde, on bitlik damgalardan bir saniyede 960 tanesi iletilir. Bu durumda bir damganın arabirimden sürücüye ulaşması yaklaşık 1 ms sürer. Makina komutlarının

ortalama 0,1 mikrosaniyede işletildiği varsayılırsa, bu süre boyunca, ana işlem birimi `in, test, jz` komutlarının herbirini, yaklaşık 3000' er kez çalıştırarak bekleme durumunda kalır. Başka bir anlatımla, ana işlem biriminin hızı, aktarım işlemlerinin hızına indirilmiş olur. Seçmeli giriş/çıkış programlama, ana işlem biriminin verimsiz kullanımına neden olan bir yöntemdir.

Ana işlem birimi kullanımındaki verimsizliği aşmanın yolu, yukarıdaki örneklerden de açıkça görüleceği üzere, durum sınamalarını ortadan kaldırmaktan geçmektedir. Bu sınamaların ortadan kaldırılabilmesi, kesilme düzeneği olarak bilinen bir ana işlem birimi uyarı düzeneğinin kullanılması ile olanaklıdır. Kesilme düzeneği ve bunun giriş/çıkışların programlanmasında kullanımı izleyen kesimde açıklanacaktır. Bu kesime geçmeden önce, *80X86* türü işleyicilerin `in` ve `out` adlı giriş/çıkış komutlarıyla ilgili özlü açıklamalar verilecektir.

Bilindiği üzere ana işlem biriminin erişebildiği ögeler:

- Bellek adres evreninde ana bellek sözcükleri,
- Giriş/çıkış adres evreninde ise, giriş/çıkış kapıları olarak da nitelenen arabirim yazmaç ve yastıklarıdır.

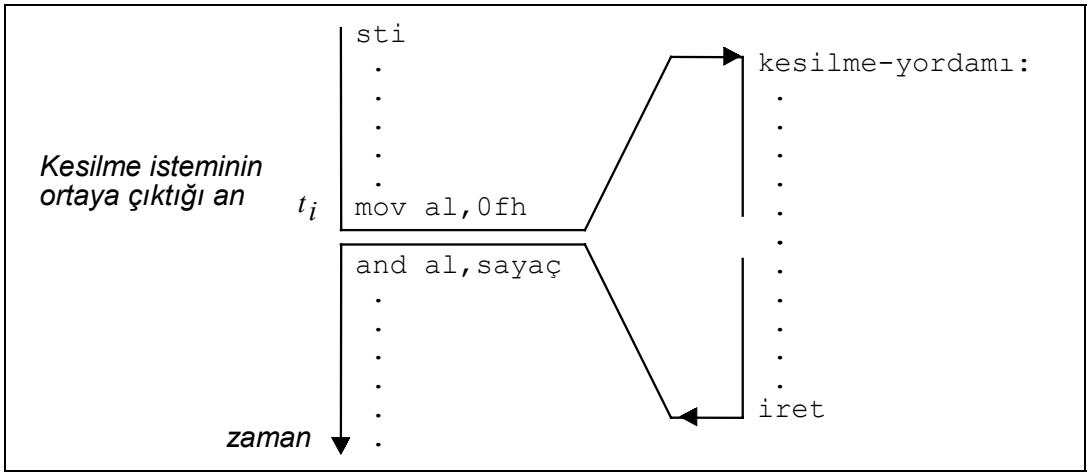
Ana bellek sözcüklerine erişim bellek erişimli olarak nitelenen komutlarla, giriş/çıkış kapılarına erişim ise giriş/çıkış komutlarıyla gerçekleşir. *80X86* türü işleyicilerin giriş/çıkış adres evreninde yer alan giriş/çıkış kapılarına erişimi `in` ve `out` komutlarıyla gerçekleşmektedir. Bu komutlar `ax` ya da `al` yazmacından giriş/çıkış kapısına ya da giriş/çıkış kapısından bu yazmaçlara veri aktarmak için kullanılırlar. Başka bir deyişle, bu komutların işlenenlerinden biri mutlaka `ax` ya da `al` yazmacıdır. Diğer işlenen ise, erişilen giriş/çıkış kapı adresidir. Giriş/çıkış kapılarının adreslenmesi komut içi ve yazmaç dolaylı olmak üzere iki biçimde yapılabilir. Yazmaç dolaylı adreslemede `dx` yazmacı kullanılır. Komut içi adresleme yapan giriş/çıkış komutları bir bayt uzunluğundaki adreslere izin verirken, yazmaç dolaylı adreslemeyi kullanan komutlar 16 bit üzerinden adresleme yapabilmektedir. Komut içi adresleme yapan `in` ve `out` komutları 2 bayt uzunluğunda olurken `dx` yazmacını dolaylı giriş/çıkış kapısı adreslemede kullanan `in` ve `out` komutları tek bayt uzunluğunda olmaktadır. `in` ve `out` komutlarının değişik türleri ve yerine getirdikleri işlemler aşağıda özetlenmiştir:

```
in   al, kapı    (al) <-- (kapı)
in   ax, kapı    (ax) <-- (kapı+1:kapı)
in   al, dx      (al) <-- ((dx))
in   ax, dx      (ax) <-- ((dx)+1:(dx))

out  kapı, al    (kapı)          <-- (al)
out  kapı, ax    (kapı+1:kapı)   <-- (ax)
out  dx, al      ((dx))          <-- (al)
out  dx, ax      ((dx)+1:(dx))   <-- (ax)
```

Bu tanımlar içinde `kapı` olarak adlandırılan bir baytlık değişmez, 0'dan 255'e kadar, komut içi bir giriş/çıkış kapı adresini temsil etmektedir.

2.3. Kesilme Düzeneği

Giriş/çıkış arabirimlerinin rasgele anlarda ortaya çıkan hizmet istemlerinin hemen ele alınabilmesi, o an işletilmekte olan programın kesilerek sözkonusu hizmetle ilgili yordama sapılması ile mümkündür. Ana işlem biriminin, işletmekte olduğu bir programı rasgele bir anda keserek özel bir yordama sapması kesilme olarak bilinir. Kesilme, ana işlem birimine giriş/çıkış arabirimlerinden ulaşan özel imler aracılığıyla gerçekleşir. Bu imler kesilme istem imleri olarak adlandırılır. Kesilme istem imleri ana işlem birimine kesilme girişleri üzerinden uygulanır. İşleyicilerin genelde tek bir kesilme girişleri bulunur. İzleyen kesimdeki açıklamalar, ilgili ana işlem biriminin tek bir kesilme girişi içerdiği varsayılarak verilmiştir. Giriş/çıkış arabirimlerinin kesilme istemleri, kimi durum yazmaç bitleriyle simgelenir. Kesilme istem imleri, ilgili durum bitleriyle aynı mantıksal değerleri taşırlar. Kesilme istemleri kesilme uyarıları olarak da adlandırılır. Örnek terminal arabirimindeki RR (almaya hazır), TR (göndermeye hazır) bitleriyle aynı mantıksal değerlere sahip imler, kesilme istem imlerine örnek olarak verilebilir.

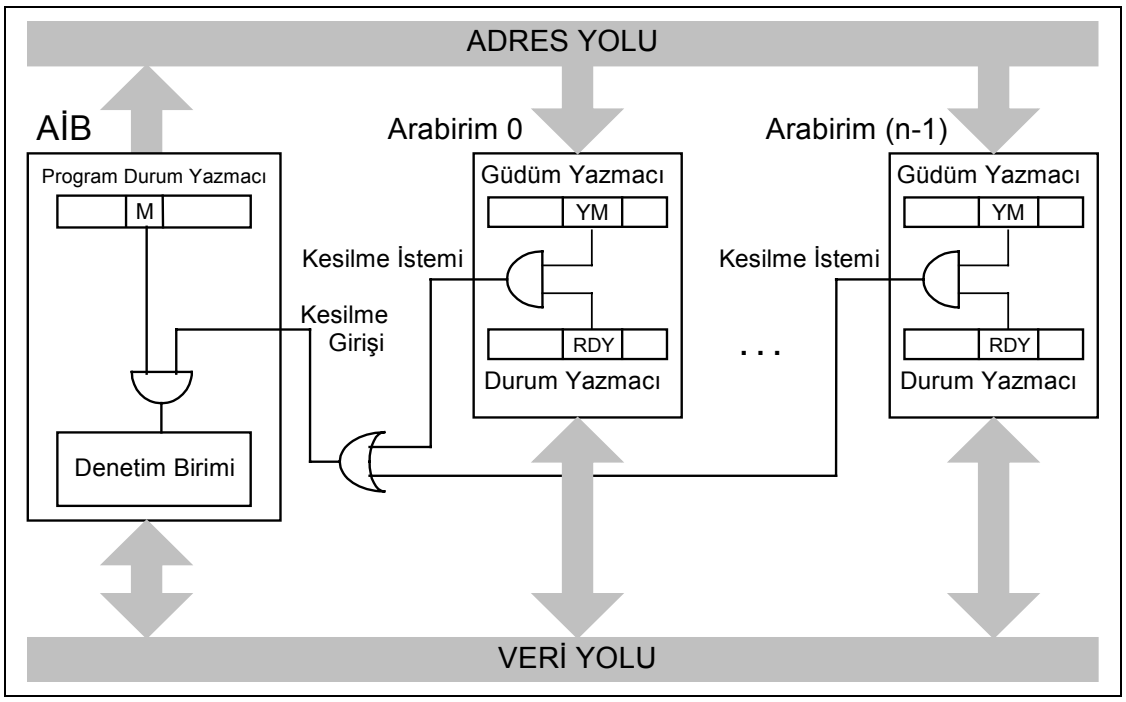


Çizim 2.12. Kesilme Yordamına Sapış

Ana işlem birimi kesilme girişi, denetim birimi tarafından sınanır. Denetim birimi her komut işletimi sırasında, saat periyotlarının birinde, örneğin uygula evresinin son saat periyodunda kesilme girişini sınar. Eğer giriş üzerinde kesilme istemini simgeleyen bir im düzeyi (örneğin 5 voltluk bir gerilim düzeyi) bulursa işlettiği komutu izleyen komutun işletimi yerine, kesilme yordamı olarak adlandırılan özel bir yordama sapmayı sağlar. Başka bir deyişle, program sayaç içeriğini bir artırmak yerine yeni bir değerle günler. Kesilme yordamı kesilme imini üreten arabirimin hizmet istemini yerine getiren yordamdır.

Kesilme girişi denetim birimine, ana işlem birimi program durum sözcüğü adlı yazmacın özel bir bitiyle denetlenen bir anahtarlama düzeneği üzerinden bağlanır. Bu bitin kurulu olmadığı durumlarda ana işlem birimi kesilme girişine ulaşan kesilme imleri, denetim birimine ulaşamaz ve dolayısıyla komut işletim akışını değiştiremezler. Anılan bu anahtarlama düzeneği, kesilme maske düzeneği olarak bilinir. İleride görüleceği üzere, aynı düzeneğin kesilme istemlerinin kaynaklandığı arabirim düzeyinde de bulunması nedeniyle ana işlem birimi içinde bulunan ve kesilme

istemlerini genel olarak denetleyen bu maske düzeneğine genel maske, arabirimler düzeyinde yer alan düzeneklere ise yerel maske düzenekleri denir. Kesilme istemlerinin denetim birimine ulaşıp ulaşamamasını denetleyen ana işlem birimi program durum sözcüğü yazmacının ilgili biti genel maske (M) biti olarak adlandırılır. Arabirimler düzeyindeki yerel maske (YM) bitleri ise güdüm yazmaçları içinde bulunan işlem izni açma-kapama bitleridir. Genel maske bitinin denetlenmesi, genelde *enable interrupt, disable interrupt; set interrupt mask, clear interrupt mask* gibi makina komutlarınca sağlanır. Bu komutlar genel maske bitinin kurulmasını ya da sıfırlanmasını gerçekleştirirler. Genel maske bitinin kurulması, kesilme isteminin denetim birimine kadar ulaşmasını ve komut işletim akışını etkileyebilmesini sağlar. Bu, ana işlem biriminin ya da sistemin kesilmelere açılması işlemi olarak nitelenir. Genel maske bitinin sıfırlanması ana işlem birimini kesilmelere kapar.



Çizim 2 . 13. Kesilme Düzeneği

Kesilme isteminin denetim birimi tarafından algılanması sonucu kesilme yordamının işletimine sapış rasgele yapılamaz. Kesilme istemi algılandığında işletilmekte olan programın işletim bütünlüğünün korunması kimi önlemlerin alınmasını gerekli kılar. Bu önlemler:

- Kesilme yordamına sapılmadan önce, ana işlem birimi tarafından kendiliğinden yerine getirilen önlemler ve
- Kesilme yordamı içinde ele alınan önlemler

olmak üzere iki sınıfa ayrılır. Ana işlem birimi tarafından otomatik olarak ele alınan önlem, kesilme yordamının işletimi sonlandığında işletimine geri dönülecek komut adresinin saklanmasıdır. Bu adrese geri dönüş adresi adı verilir. Geri dönüş adresi, o anda yığıt göstergesinin gösterdiği adrese saklanır. Bu, kesilme yordamına sapma işleminin ilk aşamasıdır. Kesilme yordamına sapma bu yordamın ana işlem birimine

anahtartarlanması olarak da nitelenir. Anahtarlanmanın bu aşamasında geri dönüş adresinin yanı sıra, işletime özel, program durum yazmacı, görev yazmacı gibi kimi diğer yazmaçlar da, ana işlem birimince, otomatik olarak yığıta saklanabilir.

Geri dönüş adresinin saklanmasından sonra ikinci aşama kesilme yordamına sapma aşamasıdır. Kesilme yordamına sapmada değişik işleyici mimarileri değişik yöntemler kullanırlar. Bu yöntemlerden en yalınları:

- Kesilme girişiyle eşleşmiş değişmez bir ana bellek sözcüğü üzerinden, dolaylı adresleme yoluyla kesilme yordamına sapma yöntemi ve
- Kesilme istem imini üreten sistem birleşeninin (örneğin arabirimlerin) sapılacak kesilme yordamı başlangıç adresini de sağladığı yöntemdir.

Birinci yöntemle, sistemin kesilmelere açık olduğu bir evrede, kesilme girişi üzerinde kesilme istem imi oluştuğunda, başlangıç adresi ana belleğin önceden bilinen, değişmez bir konumunda (örneğin sıfır adresli sözcüğünde) yer alan yordama sapma sağlanır. Kesilme yordamına sapmada yararlanılan ana bellek dolaylı adresleme sözcüğü vektör olarak adlandırılır ve sistem işletime açıldığında sistem programları tarafından, kesilme hizmet yordamının başlangıç adresiyle günlenir.

İkinci yöntemde ise, kesilme istem imi denetim birimince algılandığında, kesilme istemini üreten birleşene, kesilme istemi alındısı olarak adlandırılan bir geri bildirim imi gönderilir. Bunun üzerine söz konusu birleşen kendi istemiyle ilgili yordam başlangıç adresini ya da bunun hesaplanmasını sağlayacak parametreyi ana işlem birimine ulaştırarak kesilme yordamına sapılmayı sağlar. İlgili yordam başlangıç adresi ya da parametresi, arabirim üstünde, kesilme kimlik yazmacı diye adlandırılan bir yazmaç içinde saklanır. Kesilme kimlik yazmaçları salt okunur tür yazmaçlar değilse, içerikleri sistem işletime açıldığında, sistem programları tarafından, kesilme yordam başlangıç adresleriyle günlenir.

Kesilme yordamından geri dönüş diğer yordamlardan geri dönüşte olduğu gibi `return` türü komutlarla gerçekleşir. Bu komutun işlevi, kesilme istem iminin algılanmasından sonra ana işlem birimi tarafından, otomatik olarak yığıta saklanan değerlerin ilgili yazmaçlara geri aktarılmasıdır. Kimi işleyiciler için `call` türü komutlarla çağrılan yordamlardan geri dönüş ile kesilme yordamlarından geri dönüşü sağlayan komutlar birbirinden ayrıştırılır. Bunun nedeni `call` komutunun işletiminde yığıta saklanan içerikler ile kesilmeye sapışta saklanan içeriklerin aynı olmamasıdır.

Kesilme istemi, oluşmasına neden olan durum ortadan kaldırılıncaya kadar varlığını sürdürür. Kesilme isteminin ortadan kalkması, bu isteme ilişkin yordama sapılıp gerekli hizmet işlemleri yerine getirildikten sonra gerçekleşir. Örneğin, terminal arabirimi, klavyeden girilen bir damga kodunun, giriş yastığında okunmaya hazır olduğu andan başlayarak RR (almaya hazır) bitini kurar. Arabirim RR bitini ancak arabirim giriş yastığının okunması durumunda sıfırlar. Bu nedenle, kesilme yordamına sapılması, kesilme isteminin ortadan kalkması sonucunu doğurmaz. Yukarıda açıklandığı gibi denetim birimi, ayrıcalıksız, tüm komutlar için, komut işletiminin belirli bir periyodunda kesilme girişini sınamaktadır. Eğer sistem (ana işlem birimi) kesilmelere,

kesilme yordamının birinci komutundan başlayarak açık tutulacak olursa, bu ilk komutun işletimi sırasında da varlığını sürdüren aynı kesilme istemi, aynı yordama yeniden sapılmaya, tekrar tekrar neden olacaktır. Bu durum, örneğin, ana belleğin yönetimi incelenirken açıklanacak bellek koruma düzeneklerinin bulunmadığı sistemlerde tüm ana belleğin, yığıt gibi kullanılarak geri dönüş adres değerleri ile dolması ve dolayısıyla sistemin tümüyle işlemez hale gelmesi sonucunu doğuracaktır. Bellek koruma düzeneğinin bulunduğu sistemlerde ise yığıttan taşma hatası sonucu, işletim sonlanmadan durdurulacaktır. Bu gerekçeye dayalı olarak, kesilme imi denetim birimince algılanır algılanmaz program durum yazmacındaki genel maske biti otomatik olarak sıfırlanır. Kesilme yordamının içinde, gerekli önlemler alınarak sistem kesilmelere yeniden açılabilir. Bu önlemler, kesilme imlerinin arabirimler düzeyinde maskelenerek sıfırlanması yoluyla yerine getirilir. Arabirimler arası önceliklerin ele alınma zorunluluğu, sistemin, kesilme yordamının içinde kesilmelere yeniden açılmasını gerektirir.

```
kesilme-yordamı:
                    . aib yazmaçlarının yığıta saklanması;
                    . kesilme hizmeti;
                    . sistemin kesilmelere kapatılması (daha
                      önce, kesilme hizmeti diye anılan
                      kesimde açılmışsa);
                    . aib yazmaçlarının yığıttaki
                      içeriklerle günlenmesi;
                    . kesilen yordama geri dönüş
```

Çizim 2.14. Kesilme Yordamının Genel Yapısı

Kesilme istemi algılandığında, işletilmekte olan programın işletim bütünlüğünün korunması amacıyla kimi önlemlerin alınmasının gerektiği ve bu önlemlerin, kesilme yordamına sapılmadan önce ana işlem birimi tarafından otomatik olarak yerine getirilen önlemler ile kesilme yordamı içinde ele alınan önlemler olmak üzere ikiye ayrıldığı daha önce belirtilmişti. Bu bağlamda, kesilme yordamının hemen başında, ana işlem birimi tarafından otomatik olarak saklanmayan ve kesilme yordamı içinde kullanılarak içerikleri bozulacak ana işlem birimi yazmaçlarının yığıta saklanması gerçekleştirilmelidir. Bunun doğal sonucu olarak, kesilme yordamından kesilen programa geri dönülmeden önce de, ana işlem birimi yazmaç içeriklerinin bu programa ilişkin, yığıtta saklı içeriklerle günlenmesi gereklidir. Bu günleme öncesi, eğer sistem kesilme hizmet yordamı içinde, daha önceden kesilmelere açılmışsa kesilmelere yeniden kapatılır. Bu yapılmadığı takdirde yazmaç içeriklerinin günlendiği bir anda ortaya çıkacak bir kesilme istemi, o ana kadar günlendiği varsayılacak yazmaç içeriklerinin ve dolayısıyla ilk kesilen programın işletim bütünlüğünün bozulmasına neden olabilir. Kesilme istemi algılandığında işletilmekte olan programın işletim bütünlüğünün korunması amacıyla, bir kesimi ana işlem birimi tarafından otomatik olarak yerine getirilen yazmaç saklama ve günleme işlemleri, genel olarak bağlam anahtarlama

işlemleri olarak adlandırılır. Bağlam anahtarlama işlemlerine görev yönetimi kapsamında da değinilecektir. Bu aşamaya kadar verilen bilgiler ışığında, kesilme yordamının genel yapısı Çizim 2.14'te verildiği biçimdedir.

```
kesilme-yordamı:
                    . aib yazmaçlarının yığıta saklanması;
                    . Kesilme girişine bağlı arabirimlerin
                      durum yazmaçlarının okunarak kesilme
                      istemlerinin taranması;
                    . Kesilme istemine rastlanan ilk
                      arabirimle ilgili özel hizmet
                      yordamını çağırma
                    . aib yazmaçlarının yığıttaki
                      içeriklerle günlenmesi;
                    . kesilen yordama geri dönüş
```

Çizim 2.15. Ana Bellek üzerinden sapılan Kesilme Yordamının Genel Yapısı

2.3.1. Kesilme Yordam Adresinin Ana Bellekten Alınması

Kesilme yordamı başlangıç adresinin, ana belleğin kesilme vektörü olarak adlandırılan belirli bir sözcüğünden alındığı yöntemde, kesilme girişi üzerinde oluşan istem imi denetim birimince algılanır algılanmaz, bu girişle eşleşmiş tek bir yordama sapılır. En yalın bir bilgisayar sisteminde bile kesilme istemi üreten birden çok sistem birleşeni bulunur. Kesilme istemi üreten birleşen (arabirim) çıkışları kesilme girişine "ya da" türü mantıksal bir geçitten geçerek ulaşırlar. Bunun sonucunda ana işlem birimi düzeyinde algılanan kesilme isteminin hangi birleşenden kaynaklandığını dolaysız çıkarabilme olanağı bulunmaz. Bu nedenle sapılan ortak kesilme yordamı içinde, ana işlem birimine kesilme istemi üretebilen tüm birleşenlerin sınanarak istemin hangi birimden geldiğinin belirlenmesi gereklidir. Kesilme istemi üreten tüm birleşenlerin sınanması ve istemin hangi birimden geldiğinin belirlenmesi kesilme istemlerinin taranması olarak adlandırılır. Bu tarama, kesilme istem imi ile aynı mantıksal değeri gösteren durum yazmacı bitlerinin, seçmeli programlamada olduğu gibi sırayla okunması yoluyla gerçekleştirilir. Tarama sonucu istem ürettiği saptanan birleşenin özel hizmet yordamına sapılarak istemin gereği yerine getirilir. Burada kesilme yordamı biricik, kesilme isteminin gereğini yerine getiren özel hizmet yordamları ise kesilme istemi üretebilen arabirim sayısı kadardır. Bu yönteme göre, tüm birleşenlerce ortaklaşa kullanılan kesilme yordamının genel yapısı Çizim 2.15'te verilmiştir.

Seçmeli giriş/çıkış programlama yönteminde de görüldüğü gibi, birden çok arabirimin kesilme istem imi üreterek aynı anda hizmet istemesi öncelik sorununu ortaya çıkarır. Kesilme istem iminin algılanması sonucu sapılan kesilme yordamı içinde arabirimler öncelik sırasında taranırlar. Bu tarama sonucunda durum biti kurulu bulunan ilk arabirimin hizmet yordamı çağrılır. Bu yordamın işletiminden sonra kesilme

yordamından geri dönülür. Eğer bu aşamada, daha az öncelikli arabirimlerin de taranmamış istemleri kaldı ise, kesilme girişi üzerinde varlığını sürdüren kesilme imi, kesilme yordamına yeniden sapılmayı sağlar ve tarama yordamı yeni baştan çalıştırılır. Kesilme girişinin denetim birimince sınandığı anda var olan birden çok kesilme isteminin ele alınması bu biçimde gerçekleştirilir. Bu, eşanlı kesilme istemlerinde önceliklerin ele alınışıdır. Ancak bir arabirimle ilgili özel hizmet yordamına sapılıp bununla ilgili işlemler yürütülürken de, daha öncelikli arabirimlerin kesilme istemleri ortaya çıkabilir. Özellikle bu işlemlerin göreli uzun sürdüğü durumlarda, işletilen özel hizmet yordamına ara verilip daha öncelikli arabirimin hizmet yordamına sapmak gerekli olabilir. Bu, özel hizmet yordamları içinde, gerekli önlemler alınarak sistemin kesilmelere yeniden açılmasıyla gerçekleşebilir. Bunun yapılması, tüm kesilme istemlerini tek bir kesilme yordamıyla yanıtlayan bu yöntemle, aynı kesilme yordamına tekrar tekrar sapmayı gerektiren karmaşık bir yapı ortaya çıkarır. Bu nedenle, bu tür önceliklerin ele alınışı izleyen kesimde, diğer yöntem kapsamında açıklanacaktır.

Kesilme yordamlarına, belirli ana bellek sözcükleri üzerinden, dolaylı adresleme yoluyla sapma yönteminin kimi önemli sakıncaları vardır. Bu sakıncalardan en önemlisi, arabirimlere özgü özel hizmet yordamlarına dolaysız sapılamamasıdır. Kesilme istemi söz konusu olan arabirimin hizmet yordamına sapış, yukarıda tarama kesimi olarak adlandırılan komut dizisinin işletilmesini gerektirmektedir. Çoğu kez bir giriş/çıkış kapısını okuma, güdüm yazmacına yeni bir güdüm değeri yazma gibi birkaç komutluk hizmet gereksinimi nedeniyle üretilen her kesilme istemi için, kesilme yordamı içinde tüm arabirimlere ortak bu kesimin işletilmesinin, zaman açısından bedeli yüksektir. Özellikle aynı kesilme girişine bağlı arabirim sayısının kabarık olduğu sistemlerde bu bedelin daha da ağırlaşacağı açıktır. Bu temel sakıncayı ortadan kaldırmanın yolu tarama kesiminin işlevini arabirimlere (kesilme istemi üretebilen birleşenlere) yüklemekten geçer. Kesilme yordamına sapmada, sapma adresinin arabirimce sağlanmasını gerektiren yöntem bunu gerçekleştirir.

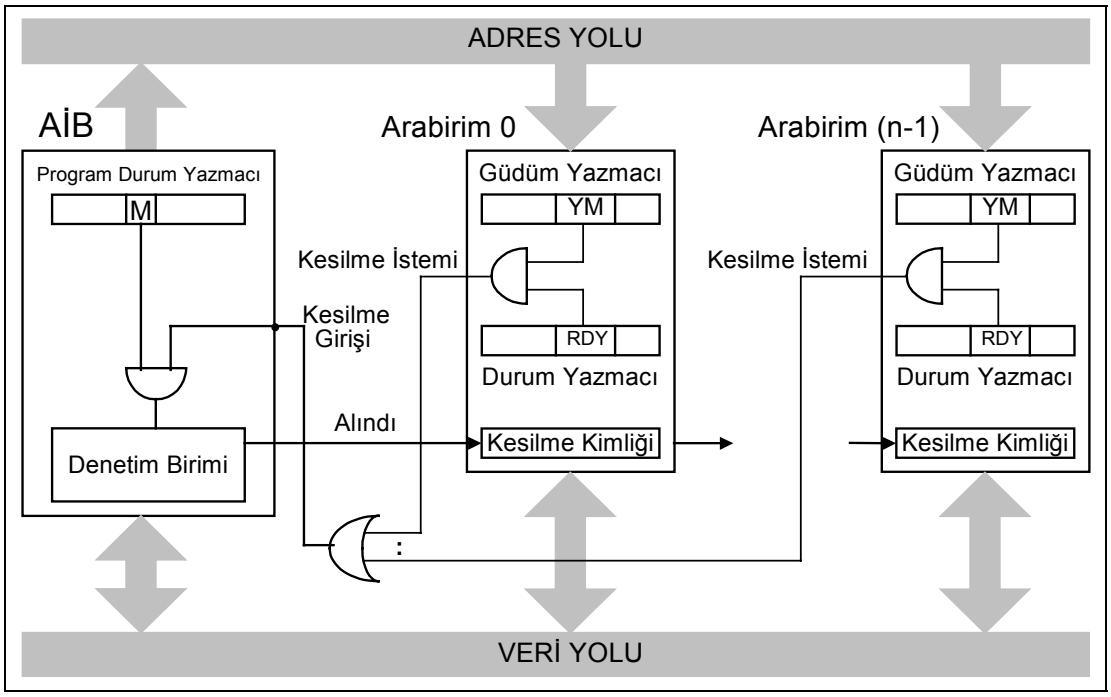
2.3.2. Kesilme Yordam Adresinin Arabirimce Sağlanması

Kesilme yordamına sapma adresinin arabirimce sağlandığı yöntemde, kesilme girişi üzerinden bir istem geldiğinde ana işlem birimi, "alındı" olarak adlandırılan bir imi arabirimlere gönderir. Kesilme istemini üreten arabirim, alındı imini alır almaz kendisiyle ilgili kesilme yordamının başlangıç adresini ya da bunu hesaplamaya yarayacak parametreyi, veri yolu üzerinden ana işlem birimine ulaştırır. Bu biçimde ana bellekte kesilme vektörü kullanımına gerek kalmaz. Çoğu kez, arabirimle ilgili kesilme yordamı başlangıç adresi (ya da parametresi) arabirim düzeyinde bir yazmaç (kesilme kimlik yazmacı) içinde tutulur. Kesilme alındı imi bu özel giriş/çıkış kapısını okumada kullanılan bir denetim imi olarak da düşünülebilir.

Ana işlem birimi kesilmelere açıldığı anda, kesilme girişi üzerinde birden çok kesilme istemi bulunabilir. Kesilme alındısının, bu istemleri üreten arabirimlerden hangisine gönderileceği ve bu yolla hangi arabirime ilişkin kesilme yordamına sapılacağı

arabirimlerin öncelikleri gözetilerek belirlenir. Arabirimler arası önceliklerin belirlenmesinde iki değişik donanımsal yapı kullanılır. Bunlar:

- Zincirleme Bağlantı ve
- Kesilme Önceliği Denetleme Birimi yapılarıdır.

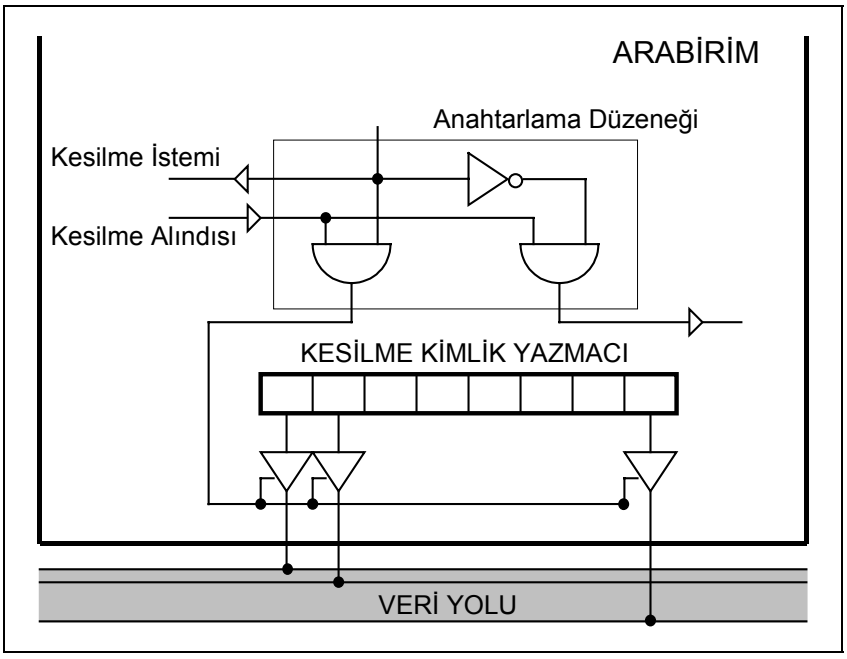


Çizim 2.16. Kesilme Alındısı Zincirleme Bağlantı Yapısı

Zincirleme Bağlantı yapısında, kesilme alındı imi arabirimlere, öncelikleriyle uyumlu, ardıl bir sırada ulaşır. Bu yapı içerisinde, alındı imi kendisine ulaşan arabirim; eğer kendisi kesilme istemi üretmiş durumda ise bu alındıyı kullanarak ana işlem birimine, kendi kesilme yordamı başlangıç adresini yollar. Yok eğer kendisinin kesilme istemi söz konusu değilse alındı imini, öncelik sırasında kendisini izleyen arabirime anahtarlar (gönderir). Alındı iminin ilk bağlandığı arabirim en öncelikli, son bağlandığı arabirim ise en düşük öncelikli (son öncelikli) birimi oluşturur. Kesilme alındısının bir arabirimden diğerine bağlanışı zincirleme bağlantı olarak adlandırılır. Bu bağlantı Çizim 2.16'da verilmiştir. Zincirleme bağlantı yapısında kesilme istem imleri, kesilme girişlerine, yine "ya da" türü geçitler üzerinden ulaşırlar. Bu yapıya ana işlem birimi yönünden bakıldığında, kesilme isteminin hangi arabirimden geldiği anlaşılamaz. Kesilme istemi denetim birimince algılır algılanmaz kesilme alındı imi kurulur. Bu imin kurulmasıyla veri yolu üzerinde bulunan değer, ana işlem birimince sapılacak kesilme yordamı başlangıç adresi ya da bunu hesaplamaya yarayacak parametre olarak yorumlanır. Çizim 2.17'de, kesilme alındı iminin, kesilme kimlik yazmacı içeriğini okumada nasıl kullanılabileceği gösterilmiştir.

Sistem kesilmelere açıldığı anda, birden çok arabirimin kesilme isteminin varlığı sözkonusu ise bu istemlere eşanlı istemler denir. Zincirleme bağlantı yönteminde, eşanlı

kesilme istemlerinde önceliklerin yerine getirilmesi kesilme alındısının arabirimleri ardıl biçimde dolaşması yoluyla sağlanır. Kesilme istemi üreten arabirimlerden kesilme alındısı kendisine ilk ulaşan arabirimin kesilme yordamına sapılır. Böylece, eşanlı istemlerde hangi isteme daha önce yanıt verileceği sorunu aşılmış olur. Arabirimler arası gözetilmesi gereken öncelikler eşanlı önceliklerle sınırlı değildir. Bir kesilme yordamı işletilirken daha öncelikli bir arabirimden kesilme istemi gelmesi durumunda işletilen kesilme yordamının kesilerek daha öncelikli bu arabirimin kesilme yordamına sapılabilmesi gereklidir. Bir kesilme yordamı işletilirken, bu işletimin kesilip daha öncelikli olduğu varsayılan bir diğer kesilme yordamının işletimine geçme ve bu yordam sonlandıktan sonra kesilen yordama geri dönme kesilmelerin içiçe ele alınması olarak bilinir. Öncelikler gözetilerek kesilmelerin içiçe ele alınabilmesi, yerel maske düzeneği diye adlandırılan bir düzeneğin kullanılmasını gerektirir.



Çizim 2.17. Kesilme Alındısı Örnek Anahtarlama Düzeneği

Herhangi bir arabirimle ilgili kesilme yordamına sapıldıktan sonra, daha öncelikli arabirimlerden gelebilecek kesilme istemlerinin gözönüne alınabilmesi için bu yordam içerisinde, bağlam saklama işlemlerinden hemen sonra, kesilmelerin, ana işlem birimi düzeyinde yeniden açılması zorunludur. Ancak bu işlemin, öncelik sırasını bozmaması için, daha düşük öncelikli arabirimlerin, bu aşama sonrasında kesilme üretmelerinin engellenmesi gereklidir. Bu amaçla arabirimlerde kesilme istem imlerini simgeleyen durum biti çıkışları, arabirim dışına, işlem izni güdüm biti çıkışlarıyla, "mantıksal ve" türü geçitlerden geçerek taşınırlar. Arabirim işlem izni güdüm biti sıfır olduğu sürece durum bitiyle simgelenen kesilme istemi ana işlem birimine ulaşamaz. İşlem izin bitinin sıfırlanması, kesilme isteminin yerel olarak maskelenmesi diye bilinir.

Kesilme yordamları içinde, düşük öncelikli arabirimlerin kesilme üretmelerinin engellenmesi, maskelenmeleri yoluyla sağlanır. *n* arabirimin yer aldığı bir sistemde,

kesilme önceliklerinin sıfırdan *(n-1)*'e azalarak sıralandığı varsayılırsa, *(i)*inci arabirimin kesilme yordamına sapıldıktan sonra, ana işlem birimi kesilmelere ancak *i, (i+1),…, (n-1)*inci arabirimlerin maskelenmesinden sonra açılabilir. Bu yolla, *(i)*inci arabirim kesilme yordamının işletimi kesilerek, daha öncelikli *(i-1), (i-2),…,0* ıncı arabirimlerin kesilme yordamlarına içiçe sapma sağlanmış olur. Bu durumda *(i)*inci arabirime ilişkin kesilme yordamının yapısı Çizim 2.18'de verilen biçime dönüşür.

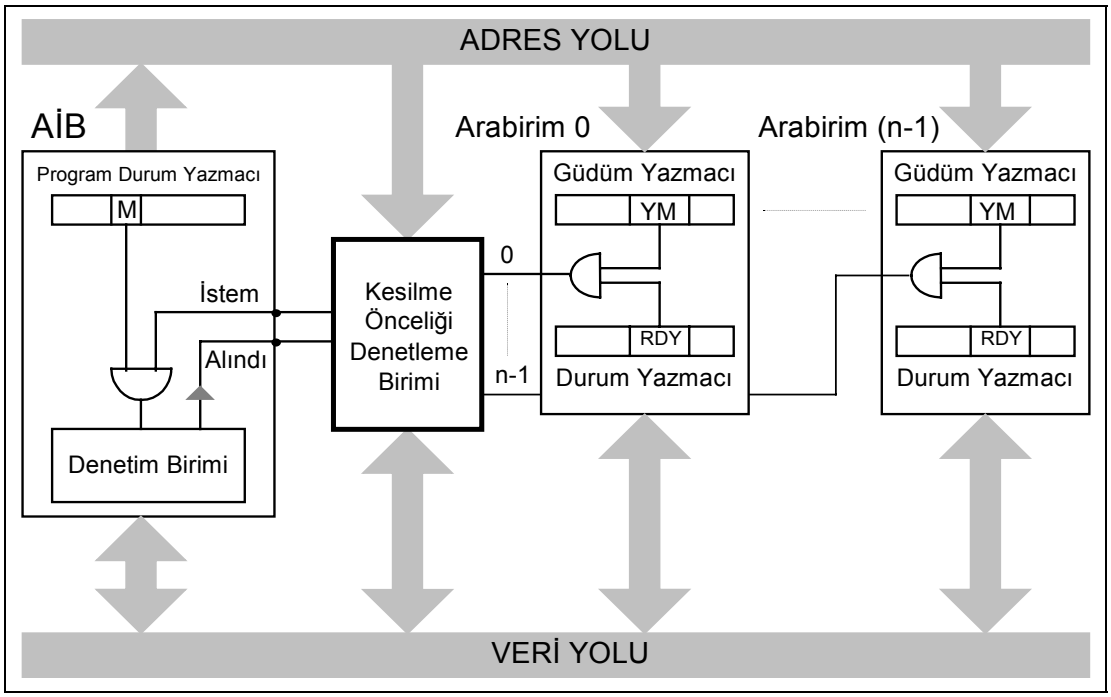
```
kesilme-yordamı-i:
                    . aib yazmaçlarının yığıta saklanması;
                    . i, (i+1),..,(n-1) inci arabirimlerin
                      yerel maske bitlerinin sıfırlanması;
                    . aib'nin kesilmelere açılması;
                    . (i) inci arabirimin hizmet yordamını
                      çağırma;
                    . aib'nin kesilmelere kapatılması;
                    . i, (i+1),..,(n-1) inci arabirimlerin
                      yerel maske bitlerinin kurulması;
                    . aib yazmaçlarının yığıttaki
                      içeriklerle günlenmesi;
                    . kesilen yordama geri dönüş
```

Çizim 2.18. Kesilme Yordamında Öncelikleri ele alan Yapı

Çizim 2.18'de verilen yapı gereği, *(i)*inci arabirim kesilme yordamında sistem, *(i-1)'*den sıfıra kadar olan, daha öncelikli arabirimlerin kesilmelerine, kesilme isteminin geldiği *(i)*inci arabirimin kendisi de yerel olarak maskelendikten sonra açılmaktadır. Daha önce de belirtildiği üzere, kesilme istemi üreten bir arabirim, bu istemini ancak gerekli hizmeti aldıktan sonra (hizmet yordamı işletildikten sonra) sıfırlar. Oysa sistem kesilmelere daha hizmet yordamına sapılmadan açılmak durumundadır. Arabirimin istemi daha sıfırlanmadan sistemin kesilmelere yeniden açılması durumunda, sistemi kesilmelere açan komutun hemen sonrasında, içinde bulunulan kesilme yordamının başına yeniden sapılacaktır. Bunun yaratacağı sonsuz döngü bellek koruma düzeneğine sahip bir sistemde, ancak "yığıttan taşma" hata uyarısı ile sonlanacaktır. Bu nedenle kesilme yordamı içinde, daha düşük öncelikli arabirimlerin yanı sıra kesilme istemini üreten arabirimin kendisi de maskelenir.

Kesilme önceliklerinin, iç içe kesilmelere olanak verecek biçimde denetlenebilmesi için arabirim maske bitlerinin kurulma işlemleri, çoğu kez kesilme yordamlarını karmaşıklaştıran, bu yordamların işletim sürelerini uzatarak sistem başarımını düşüren işlemlerdir. Ayrıca kesilme önceliklerinin denetiminin çok sayıda kesilme yordamının içine dağılmış komutlarla yapılması, sistem yazılımlarının okunurluk ve bakılırlığını olumsuz yönde etkileyen bir husustur. Zincirleme bağlantı yapısının sistem programlarına yansıyan bu sakıncalarının yanı sıra bir diğer sakıncası daha vardır. Bu

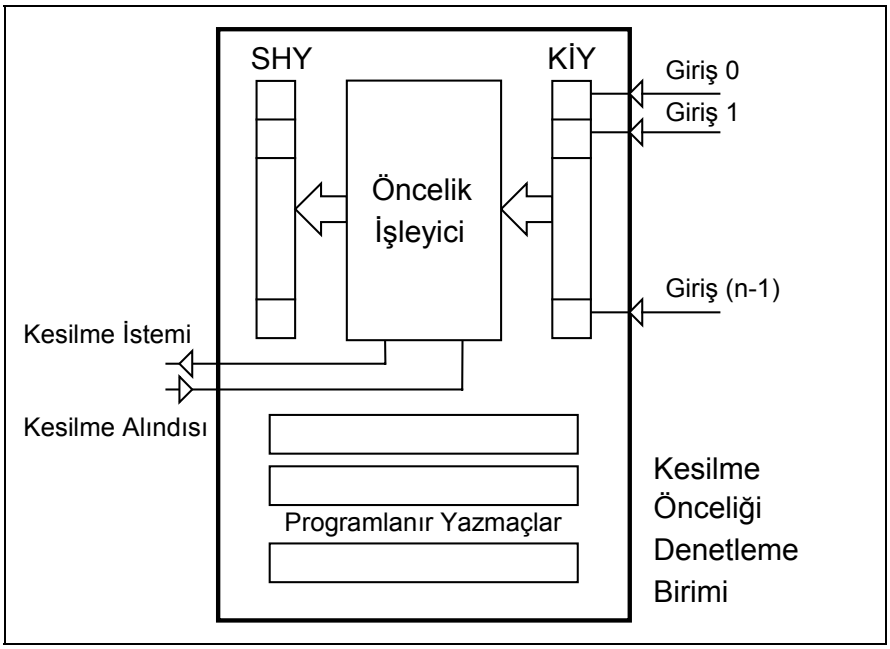
yapıda arabirimlerin kesilme öncelikleri, sistem donanım yapısına gömülüdür. Başka bir deyişle, bir arabirimin önceliğinin değiştirilebilmesi o arabirimin, zincirleme bağlantıdaki yerinin, fiziksel olarak değiştirilmesini gerektirir. Bu, sistem kutusunun açılarak kimi tel sargıların çözülüp yerine yeni sargı ya da lehimlerin yapılması demektir. Sistem konfigürasyonuna yeni arabirimlerin katılması durumunda da bu arabirimin önceliğinin gerektirdiği yere takılması ve diğer arabirimlerin alındı imi bağlantılarının değiştirilmesi gerekir. Sisteme yeni arabirimlerin katılması, çoğu kez müşteri ortamında gerçekleştiğinden sistemin genişletilmesindeki firma donanım desteği gereksinimi artar. Halbuki arabirim önceliklerinin sistem yazılımları aracılığıyla değiştirilebilmesi, çok daha esnek bir yaklaşımı oluşturacaktır.



Çizim 2.19. Kesilme Önceliği Denetleme Biriminin Konumu

Kesilme Önceliği Denetleme Birimi, kesilme önceliğinin işletim içinde devingen olarak değiştirilebilmesine olanak verir. Bunun yanı sıra kesilme yordamlarının, maskeleme yükünden arındırılarak yalınlaşmasını ve kısalmasını sağlar. Kesilme önceliği denetleme birimi kullanımı durumunda arabirimler kesilme istemlerini, doğrudan ana işlem birimine göndermek yerine bu birime gönderirler. Ana işlem birimine, arabirimler adına kesilme istem imi gönderme yükümlülüğü ise öncelik denetleme birimindedir. Ana işlem biriminin kurduğu istem alındı imi de, arabirimlere dağılacağına salt bu birime gönderilir. Alındı iminin gelmesiyle öncelik denetleme birimi, o anda en öncelikli birimin yordam başlangıç adresini (ya da parametresini) veri yolu üzerinden ana işlem birimine ulaştırır. Kesilme önceliği denetleme biriminin bir bilgisayar sistemi içindeki konumu Çizim 2.19'da gösterilmiştir.

Kesilme önceliği denetleme birimine ilişkin örnek bir yapı ise Çizim 2.20'de verilmiştir. Bu yapı, öncelikle Kesilme İstem Yazmacı (KİY) ve Süren Hizmet Yazmacı (SHY) olarak adlandırılan iki yazmacı içerir. Bu yazmaçlar öncelik denetleme biriminin içerdiği kesilme istem girişi kadar bitten oluşurlar. Kesilme İstem Yazmacının bitleri arabirimlerden gelen istemlerle kurulmakta ve arabirim istemini çektiğinde sıfırlanmaktadır. Süren Hizmet Yazmacının bitleri ise kesilme yordamına sapılan (kesilme hizmeti süren) arabirimler için kurulmaktadır. Öncelik denetleme biriminin içerdiği tüm kesilme girişlerinin bir önceliği bulunmaktadır. Örneğin 8 girişli bir denetleme birimi için, sıfırıncı girişin en öncelikli, yedinci girişin de en düşük öncelikli giriş olduğu bir durumda, üçüncü girişten bir istem geldiğinde, Kesilme İstem Yazmacının üçüncü biti kurulur ve ana işlem birimine kesilme istemi yollanır. Bu istemle ilgili alındı geldiğinde üçüncü girişe bağlı arabirimin kesilme yordamı başlangıç adres bilgisi (ya da parametresi) ana işlem birimine ulaştırılırken Süren Hizmet Yazmacının üçüncü biti kurulur. Bu aşamadan sonra, üçüncü girişten daha öncelikli olan ikinci girişten kesilme istemi gelirse üçüncü giriş için yerine getirilen süreç, bu kez ikinci giriş için aynı biçimde yinelenir. Bu yolla üçüncü girişe ilişkin kesilme yordamı kesilerek ikinci girişe ilişkin kesilme yordamına sapma sağlanmış olur.



Çizim 2.20. Kesilme Önceliği Denetleme Birimi Örneği

Üçüncü girişle ilgili kesilme yordamı işletilirken, bu kez ikinci giriş yerine dördüncü girişten kesilme isteminin geldiği düşünülecek olursa, öncelik denetleme birimi bu istemle ilgili hiçbir girişimde bulunmaz. Zira öncelikler gereği dördüncü girişe bağlı arabirimin, işletilmekte olan üçüncü girişle ilgili kesilme yordamını kesme hakkı yoktur. Bu istemin göz önüne alınabilmesi, ancak üçüncü girişle ilgili yordamın sonlanıp Süren Hizmet Yazmacının üçüncü bitinin sıfırlanmasından sonra olanaklı olacaktır. Daha genel bir bağlamda, kesilme önceliğinin *0*'dan *(n-1)*'e, azalan sırada uzandığı varsayıldığında, *(i)*inci kesilme girişine bağlı arabirim adına ana işlem biriminden

kesilme isteminde bulunulabilmesi için, Süren Hizmet Yazmacının *i, (i-1)* inci bitlerinden başlayarak *0* ıncı bitine kadar tüm bitlerinin sıfırlanmış olması koşulu aranacaktır.

Yukarıda verilen açıklamalardan anlaşılacağı gibi, kesilme önceliği denetleme biriminin kullanılması ile kesilme yordamlarının, öncelik gözetim işlemlerini ele almalarına gerek kalmaz. Kesilme yordamları, arabirimler düzeyindeki yerel maskeleri açıp kapayan kesimlerinden arınarak yalınlaşırlar. Ancak kesilme yordamlarından geri dönülmeden, Süren Hizmet Yazmacının ilgili bitinin sıfırlanması, bu yordamların içinde, geri dönüş komutundan önce gerçekleştirilmesi gereken bir yükümlülük olarak ortaya çıkar. Kesilme önceliği denetleme biriminin kullanılması durumunda kesilme yordamlarının genel görünümü Çizim 2.21'deki göreli yalın biçime dönüşür.

```
kesilme-yordamı-i:
                    . aib yazmaçlarının yığıta saklanması;
                    . aib'nin kesilmelere açılması;
                    . Hizmet yordamını çağırma;
                    . aib'nin kesilmelere kapatılması;
                    . Kesilme Önceliği denetleme biriminin
                      i inci bitinin sıfırlanması;
                    . aib yazmaçlarının yığıttaki
                      içeriklerle günlenmesi;
                    . kesilen yordama geri dönüş
```

Çizim 2.21. Kesilme Önceliği Denetleme Birimini Kullanan Yordamının Yapısı

Kesilme istemlerini öncelik sırasında işleme koyan kesilme önceliği denetleme birimi, bu işlevi nedeniyle yardımcı bir işleyici olarak düşünülebilir. Ancak bu birim, sistem programları yönünden diğer arabirimlerden farklı davranılan bir birim olarak görülmez ve diğer arabirimler gibi, giriş/çıkış adres evreni içerisinde giriş/çıkış kapılarından oluşan bir bütün olarak düşünülür ve programlanır. Bu bağlamda, kesilme önceliği denetleme birimi; yerine getirebildiği işlevler arasında seçim yapılmasına olanak veren güdüm yazmaçları, kesilme girişleriyle ilgili yordam başlangıç adres yazmaçları, giriş önceliklerini belirlemeye yarayan öncelik sırası yazmacı gibi programlanır yazmaçlar içerir. İleride, *80X86* türü işleyicilerin kesilme düzeneği açıklandıktan sonra bu düzenekle uyumlu bir kesilme önceliği denetleme birimi örneğine yer verilecektir.

2.3.3. Kesilme Türleri

Şimdiye kadar açıklanan kesimde, ana işlem birimine, salt diğer sistem birleşenlerinden ulaşan kesilme uyarılarından ve bu uyarıların ortaya çıkması durumunda yerine getirilen işlemlerden söz edilmiştir. Ancak kesilme düzeneği çerçevesinde, ana işlem birimini

uyarabilen birimler ana işlem biriminin dışında yer alan birimlerle sınırlı değildir. Kesilme uyarıları:

- İç kesilme uyarıları ve
- Dış kesilme uyarıları

olarak iki sınıfa ayrılır. Dış kesilme uyarıları, şimdiye kadar açıklanan, ana işlem birimi dışında yer alan arabirimler, yardımcı işleyiciler gibi sistem birleşenlerinin ürettiği kesilme uyarılarıdır. İç kesilme uyarıları ise ana işlem birimi içinden denetim birimine ulaşan uyarılardır. Bir sayıyı sıfıra bölme gibi hatalı bir durum ortaya çıktığında ya da görüntü bellek düzeneği kapsamında açıklanacak, bellekte bulunmayan bir sayfaya erişim sözkonusu olduğunda, aritmetik-mantık birimi, adres dönüştürme birimi gibi ana işlem birimi birleşenlerince denetim birimine yollanan uyarılar iç kesilme uyarılarına örnek oluştururlar. Bu uyarılar aracılığıyla ana işlem birimi, işletmekte olduğu programı keserek sıfıra bölme hata uyarı yordamı, bellek yönetim yordamı gibi sistem yordamlarının işletimine geçer.

Kesilme düzeneği açıklanırken, dış kesilme uyarılarının dış ortamdan denetim birimine doğrudan bağlanmadığı; "ve" türü mantıksal geçitler üzerinden, kesilme maskesi olarak adlandırılan program durum yazmacı bitleriyle birleşerek bağlandığı belirtilmişti. Ancak kimi dış kesilme imleri denetim birimine herhangi bir maske düzeneğinden geçmeden bağlanırlar. Bu uyarılar, oluştukları anda denetim birimince algılanarak işleme konurlar. Bu bağlamda kesilme uyarıları:

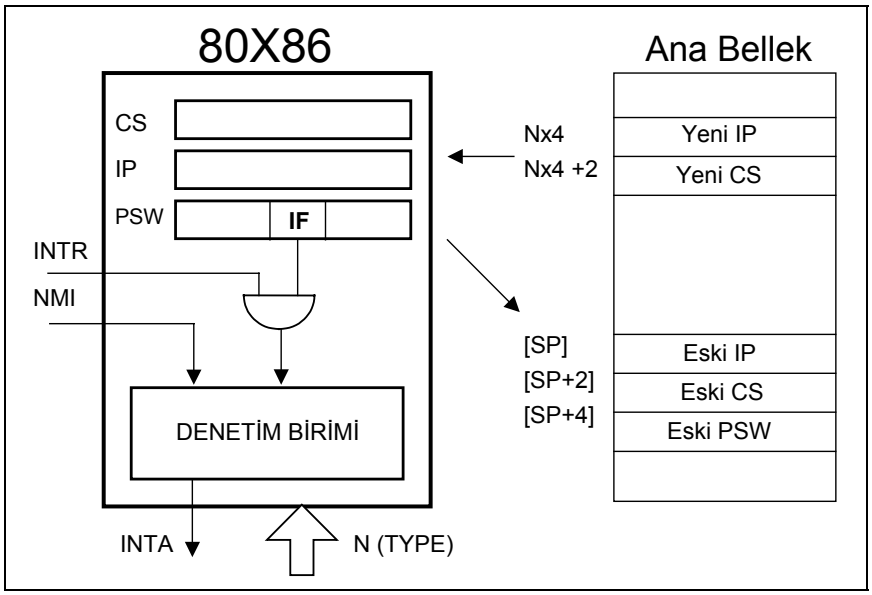
- Engellenir kesilme uyarıları
- Engellenemez kesilme uyarıları

olarak sınıflandırılırlar. Engellenemez kesilme uyarıları, ortaya çıktığı anda önlem alınmasını gerektiren, çok acil ve ertelenemez istemlere ilişkin uyarıları oluştururlar. İşleyicilerin büyük bir çoğunluğunda, engellenir uyarıların bağlandığı kesilme girişinin yanı sıra, bir de engellenemez tür uyarıların bağlandığı kesilme girişi bulunur. Bilgisayar sistemini besleyen elektrik şebekesindeki kesintilerde, bu durumu algılayan güç kaynağından gelen kesilme uyarısı, engellenemez kesilme uyarısına örnek verilebilen bir uyarıdır. Bu uyarının gelmesiyle, geriye kalan birkaç saniye içinde, önemli kütüklerin kapatılması gibi, sistem yeniden işletime açıldığında işletim bütünlüğünün korunmasını sağlayacak kimi acil önlemlerin işleme konması gerçekleştirilir. Herhangi bir maskeleme düzeneğinden geçmeden denetim birimine ulaştıklarından iç kesilme uyarıları da engellenemez kesilme uyarıları olarak düşünülürler.

Kesilme uyarıları, ana işlem biriminin, kesilme yordamı olarak adlandırılan özel bir yordamın işletimine geçmesini sağlarlar. İşleyicilerin makina komutları arasında, kesilme yordamlarına sapmada yararlanılan donanımsal düzeneği program içinden kullanmaya olanak veren komutlara da yer verilir. Altyordam çağırma komutlarına benzer bu komutlar yazılım kesilme komutları olarak bilinirler. Bu bağlamda kesilmeler bir de:

- Donanım kesilmeleri
- Yazılım kesilmeleri

olmak üzere ikiye ayrılır. Dış uyarıların algılanmasıyla başlatılan kesilme yordamına sapma süreci, bu komutlarla komut uygula evresinde gerçekleştirilir. Sapılacak kesilme yordam başlangıç adresi, donanım kesilmelerinde arabirimce sağlanırken yazılım kesilmelerinde komut işleneni olarak yer alır. Yazılım kesilme komutları, genellikle ayrıcalıklı komutları oluştururlar. Ayrıcalıklı komutlar sıradan kullanıcıların kullanamadığı, sistem yazılımları üzerinde günleme yapma hakkı verilen ve ayrıcalıklı kullanıcılar olarak da bilinen sistem programcılarına açık komutlardır. Ayrıcalıklı kullanıcı ve ayrıcalıklı komut kavramlarına, Koruma ve Güvenlik başlığı altında geri dönülecektir. Yazılım kesilmesi komutları, aynı zamanda, Giriş bölümünde açıklanan sistem çağrı düzeneğini gerçekleştirmede yararlanılan sistem altyapısını da oluştururlar.

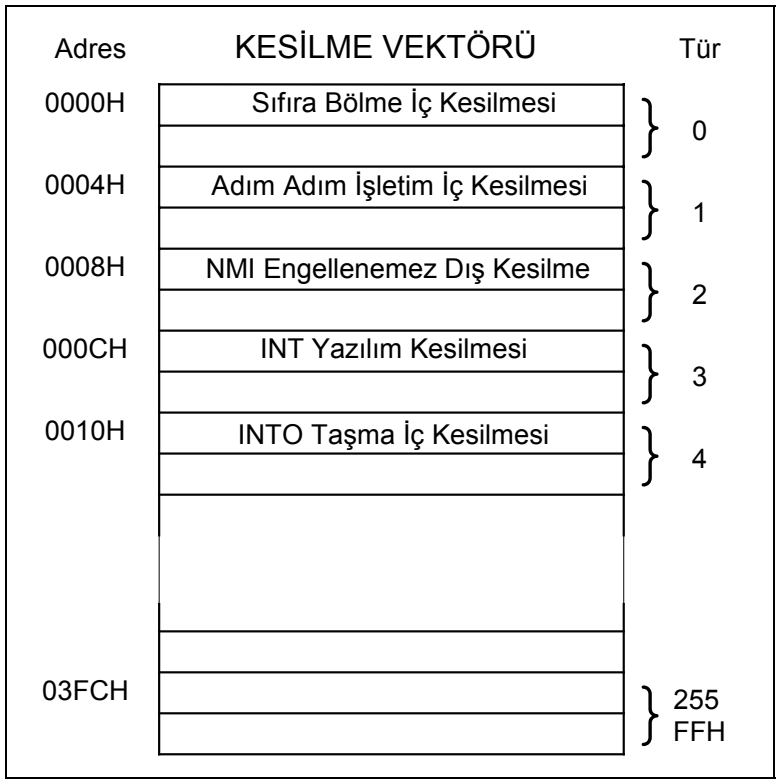


Çizim 2.22. *80X86* türü İşleyicilerde Kesilme Düzeneği

2.3.4. *80X86* Türü İşleyicilerin Kesilme Düzeneği

80X86 türü işleyicilerde iki kesilme girişi bulunur. Bunlar INTR ve NMI kısaltmalarıyla anılan girişlerdir. INTR *(interrupt)* girişi engellenir kesilme imlerinin bağlandığı varolan biricik kesilme girişidir. NMI *(non-maskable interrupt*) ise engellenemez kesilme girişidir. INTR girişine bağlı kesilme istem imleri `PSW` adlı program durum yazmacının `IF` (*interrupt flag*) adlı bitiyle maskelenerek denetim birimine ulaşırlar. Bu bit, `sti` *(set interrupt flag)* ve `cli` *(clear interrupt flag)* makina komutlarıyla kurulup sıfırlanır. Kesilme girişleri, denetim birimince, komut uygula evresinin son periyodunda sınanır. `IF` biti kurulu iken, bu son periyot süresince INTR kesilme girişinde kesilme istem imi bulunursa, izleyen komutun algetir evresi yerine kesilme yordamına sapma süreci başlatılır. Bu süreç içinde INTA *(interrupt acknowledge)* adlı alındı imi aracılığıyla, kesilme imini üreten arabirimden, sapılacak kesilme yordamı başlangıç adresini belirlemeye yarayacak bir baytlık tür *(type)* bilgisi okunur.

80X86 türü işleyicilerde kesilme yordamı başlangıç adresini belirlemede, yukarıda açıklanan yöntemlerin birlikte kullanıldığı karma bir yöntem uygulanır. INTA imi aracılığıyla, arabirimden okunan tür bilgisi (N), 4 ile çarpılarak elde edilen ana bellek adresinden kesilme yordamı başlangıç adresinin (IP' nin), bu değere 2 ekleyerek elde edilen adresten de, bu yordamın kesim (CS) başlangıç adresinin elde edileceği varsayılır. Bu durumda, her tür değeri için 2 bayt kesim (CS), 2 bayt da kesim içi göreli adres (IP) değerini tutmak üzere ana bellekte 4 baytlık dolaylı adresleme konumu bulunur. Ana belleğin sıfırdan 3FF adresine kadar olan 1 KB'lık kesimi, 256 değişik türe ilişkin 4'er baytlık kesilme vektörlerine ayrılır[10]. Kesilme vektörlerinden kimileri (*8086* da ilk 5 tanesi, *80486* da ilk 32 tanesi gibi) engellenemez kesilme girişinin, iç kesilmelerin ya da INT, INTO gibi özel yazılım kesilmelerinin kullanımına ayrılmıştır. Geriye kalan vektörler, INTR girişinden gelen kesilme istemleriyle ya da int N komutlarıyla ilgili yordamlara sapmada kullanılır.



Çizim 2.23. *80X86* türü İşleyicilerde Kesilme Vektörü

80X86 türü işleyicilere INTR girişinden kesilme istemi geldiğinde:

---

[10] Özellikle *80486*'dan başlayarak açıklanan bu düzenek biraz farklılaşır. Örneğin, kesilme yordamlarına sapışta, 4 baytlık vektörler yerine 8 baytlık vektörler kullanılır. Bunun yanı sıra, sözkonusu vektörler, ana belleğin ilk 1KB'lık kesimi yerine, ana belleğin yönetimi incelenirken tanımlanacak görüntü bellek düzeniyle uyumlu olarak, *Interrupt Descriptor Table Register (IDTR)* adlı yazmacın gösterdiği *IDT (Interrupt Descriptor Table)* adlı kesimde tutulurlar. Pedagojik nedenlerle ve anlatım kolaylığı sağlamak üzere, burada verilen açıklamalar *80386*'ya kadar olan *Intel* işleyicileriyle kısıtlı tutulmuştur.

1. İstemi üreten birimden, INTA imi aracılığıyla tür bilgisi (N) okunur.
2. PSW,IP yazmacları, yığıt göstergesinin (SS:SP'nin) gösterdiği yere (yığıta) saklanır:

```
(SP)<-- (SP)-2
((SP)+1:(SP))  <-- (PSW)
(SP)<-- (SP)-2
((SP)+1:(SP))  <-- (CS)
(SP)<-- (SP)-2
((SP)+1:(SP))  <-- (IP)
```

3. IF genel kesilme maske biti sıfırlanır.
4. IP yazmacı (N*4) adresindeki içerik ile günlenir:

```
(IP) <-- (N*4)
```

5. CS/SELECTOR yazmacı (N*4+2) adresindeki içerik ile günlenir:

```
(CS <-- (N*4+2)
```

6. Yeni bir komut işletim döngüsü başlatılır.

int N türü yazılım kesilmesi komutları işletilirken de, birinci adım dışında aynı işlem adımları gerçekleştirilir. N değeri komut içi işlenenden elde edilir. NMI girişinden bir istem geldiğinde ise, yine birinci adımın dışında tüm adımlar, bu kez N=2 için gerçekleştirilir.

80X86 türü işleyicilerde kesilme yordamından iret adlı komutla geri dönülür. iret komutunun işletimi aşağıdaki adımlardan oluşur:

```
(IP)<-- ((SP)+1:(SP))
(SP)<-- (SP)+2
(CS)<-- ((SP)+1:(SP))
(SP)<-- (SP)+2
(PSW)<--((SP)+1:(SP))
(SP)<-- (SP)+2
```

Kesilme vektöründe yer alan ve hem *8086* hem de *80386* türü işleyicilere ortak ilk beş vektörün anlamları aşağıda verilmiştir:

- Sıfıra bölme iç kesilmesi, DIV ya da IDIV komutları işletilirken elde edilen bölüm değerinin taşmaya neden olması durumunda, aritmetik-mantık birimi tarafından üretilmektedir. *8086* türü işleyicilerde bu hatalı durumu ele alacak yordamın kesim yazmaç (CS) içeriğinin 0 ıncı (16 bitlik) sözcükte; program sayaç içeriğinin ise 2 nci sözcükte yer alması gerekmektedir. *80386* türü işleyiciler *(real mode)* için ise sapılacak 32 bitlik yordam başlangıç adresini oluşturan, 16 şar bitlik *selector* ve *offset* adlı birleşenlerin aynı konumlarda bulunacağı varsayılmaktadır.

- Program durum yazmacı içinde bulunan TF *(Trap Flag)* adlı bitin kurulu olması durumunda, her makina komutunun sonunda adım adım işletim iç

kesilmesi oluşturulup 4 ve 6 ncı sözcüklerde yer alan adres birleşenlerinin gösterdiği yordama sapma sağlanır. Program geliştirme ve sınama amacıyla kullanılan adım adım işletim sırasında sapılan bu yordam içinde önemli yazmaç içeriklerinin görüntülenmesi gerçekleştirilerek program akışının izlenmesine ve varsa hataların giderilmesine yardımcı olmak amaçlanır.

- *80X86* türü işleyicilerde program geliştirme ve sınama amacıyla kullanılabilen tek olanak adım adım işletim değildir. Bir baytlık `int` komutu da bu amaçla öngörülmüştür. Bu komut program içinde, sınama amacıyla işletimin kesilmesi istenen yerlere yerleştirilir. Bu yolla değişik noktalarda denetimin 8 ve 10 uncu sözcüklerde başlangıç adresi bulunan yordama devredilmesi sağlanır. Bu yordam içinde de, adım adım işletim kesilme yordamında olduğu gibi kimi yazmaç ve göstergelerin görüntülenmesi sağlanarak program akışını izleme olanağı yaratılır. Bunun `int N` türü komutlarla da gerçekleştirilmesi, doğal olarak olanaklıdır. Ancak tek bayt uzunluğunda olması bu komut için önemli bir üstünlüktür.

- Aritmetiksel işlemlerde taşma olarak tanımlanan, işlem sonucunu ifade etmede sözcük bit sayısının yetersiz kaldığı durumlarla sıkça karşılaşılır. Herhangi bir aritmetiksel işlem komutunun hemen sonrasına `INTO` *(interrupt on overflow)* olarak adlandırılan özel bir yazılım kesilme komutu eklenerek taşma durumunda özel bir yordama sapma sağlanır. Bu yordamla ilgili başlangıç adresi bilgilerinin 12 ve 14 üncü bellek sözcüklerinde bulunması gerekir (Çizim 2.24).

```
  .
add ax,cx
into        {PSW(OF) = 1 ise → ((0C)) : ((0E)) adresine sapılır}
↓
  .
```

Çizim 2.24. `into` Komutunun Kullanımı

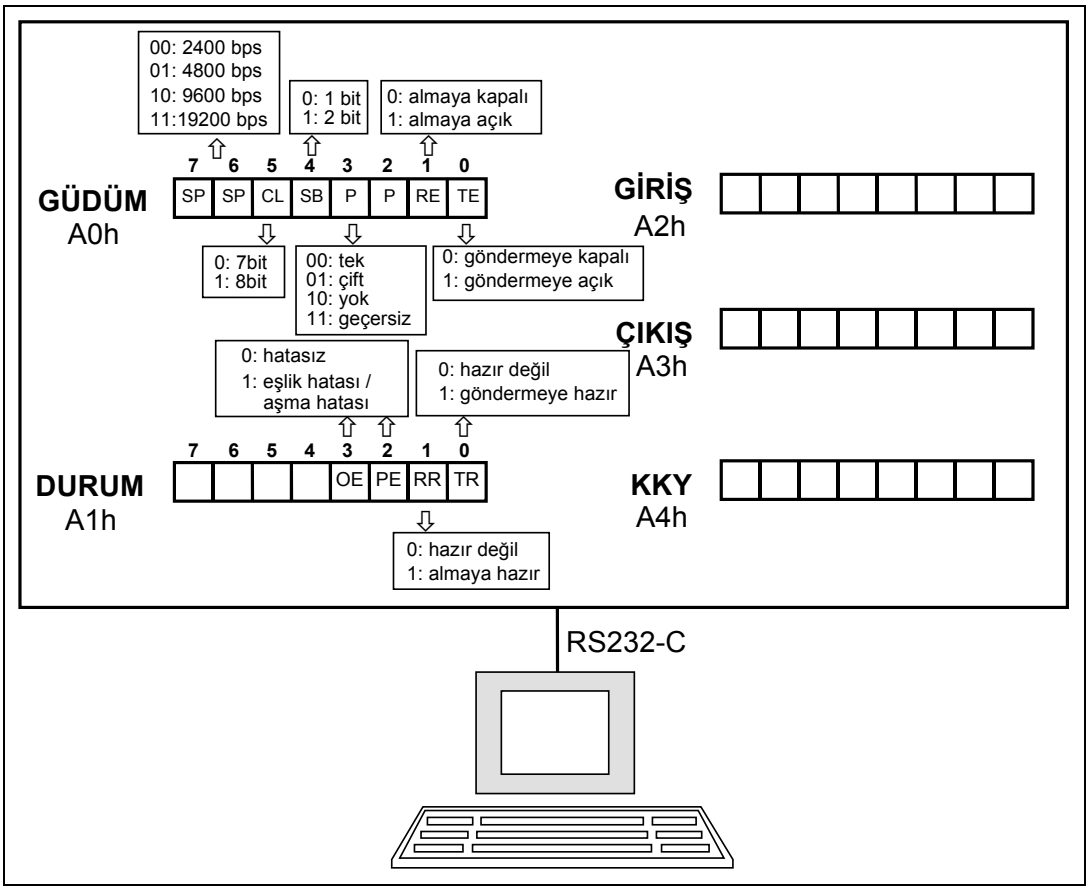
2.4. Kesilmeli Giriş/Çıkış Programlama

Daha önce, seçmeli giriş/çıkış programlama yöntemiyle ele alınan, örnek terminal sürücülerinden, `ctrl-z` damgasına rastlanana değin girilen damgaların ana bellekte `dizgi-i` adlı ilgili yastık alanlarına aktarılması örneği, kesilmeli giriş/çıkış programlama bağlamında yeniden ele alınmıştır. Bu yolla hem kesilme düzeneğinin giriş/çıkışların programlanmasında kullanımı örneklenmiş hem de seçmeli giriş/çıkış programlama yöntemiyle kesilmeli giriş/çıkış programlama yöntemini karşılaştırma olanağı sağlanmıştır.

Sözkonusu örnek kapsamında, diğer birleşenlerin yanı sıra örnek terminal birimlerinden üç adedinin yer aldığı *80X86* tabanlı bir bilgisayar sisteminde, terminal klavyelerinden girilen damgaların, `ctrl-z` damgasına rastlanana değin ilgili ekrandan görüntülenmesi

ve `dizgi-i` adlı ilgili bellek alanına yazılması amaçlanmaktadır. Terminal arabirimleri üzerindeki yazmaç ve yastıkların sistem giriş/çıkış adres evreni içindeki adreslerinin, *(i)*inci terminal düzeyinde :

Güdüm Yazmacı	için:	A0 + 5 x i (i = {0,2})
Durum Yazmacı	için:	A0 + 5 x i + 1
Giriş Yastığı	için:	A0 + 5 x i + 2
Çıkış Yastığı	için:	A0 + 5 x i + 3
Kesilme Kimlik Yazmacı	için	A0 + 5 x i + 4 olduğu varsayılmıştır.

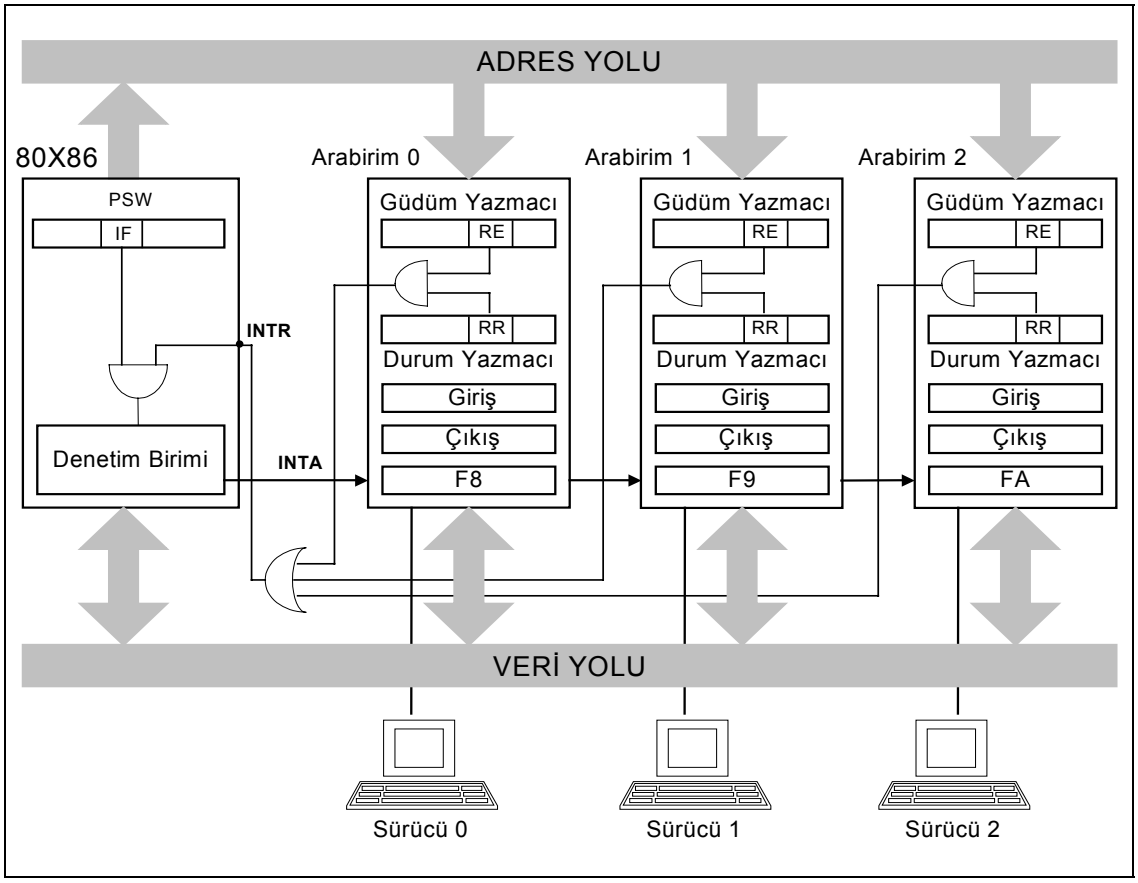


Çizim 2.25. Örnek (0 ıncı) Terminal Arabirimi Görünümü

Terminallerin, bir önceki örnekte olduğu gibi 9600 bps hızında, bir stop bit, çift eşlik sayısı ile çalışacağı öngörülmüştür. Bunun yanısıra, terminal arabirimlerinin klavyelerinden herhangi bir tuşa basılması durumunda kurulan "almaya hazır" biti üzerinden kesilme istemi ürettikleri; almaya hazır (RR) bitinin, alma işlem izin (RE) bitiyle maskelenerek üç girişli bir "ya da" geçiti üzerinden işleyiciye ulaştığı varsayılmıştır.

2.4.1. Zincirleme Bağlantı Yönteminin Kullanımı

Kesilme alındı (INTA) iminin arabirimlere zincirleme yöntemle bağlandığı durumda örnek terminal arabirim görünümü ile örneğe taban oluşturan sistem donanım ilke çizimi, sırayla Çizim 2.25 ve Çizim 2.26'da verilmiştir. Çizim 2.25'te verilen arabirim görünümü Çizim 2.7'de verilen çizimle hemen hemen aynıdır. Açıklanan örneğin izlenmesini kolaylaştırmak amacıyla yinelenmiştir. Ancak eski çizimde gösterilmeyen kesilme kimlik yazmacı yeni çizime eklenmiştir. Anımsanacağı üzere INTA kesilme alındı iminin arabirimlere zincirleme yöntemle bağlandığı durumlarda, kesilme kimliği (*type* `N`), işleyiciye arabirim tarafından sağlanmak zorundadır. Bu kimlikler: *0* ıncı arabirim için `F8H`, *1* inci arabirim için `F9H`, *2* nci arabirim için `FAH` olacaktır.



Çizim 2.26. Kesilmeli G/Ç Programlama Örneği-1 için Sistem İlke Çizimi

Kesilmeli giriş/çıkış programlama yönteminde, ana belleğin ilk 1 KBaytında yer alan kesilme vektörü ile arabirim kesilme kimlik yazmaç içeriklerinin günlenmesi, arabirimlerin işlem izin bitlerinin kurulması gibi önbelirleme işlemlerinden sonra, ana işlem birimi, seçmeli giriş/çıkış programlama yönteminin tersine başka programların işletimine geçer. Bu programların işletimi sırasında, arabirimlerden gelen her kesilme istemi için ilgili kesilme yordamına sapılır ve giriş/çıkış işlemleri diğer programlarla koşut olarak yürütülür. Çizim 2.27'da verilen örnek programda, önbelirlemelerden sonra

işletimine geçileceği varsayılan diğer programlar `dön: jmp dön` sonsuz döngü komutu ile benzetilmiştir. Doğal olarak sistem programları arasında bu tür komutların bulunması söz konusu olmayacaktır.

Ana bellek kesilme vektörünün günlenmesi, arabirim kesilme kimlik yazmaç içeriklerinin yazılması gibi kesilme düzeneğinin sağlıklı çalışması açısından kritik olan önbelirleme işlemleri ana işlem biriminin kesilmelere kapalı olduğu bir evrede yapılır. Bu nedenle örnek programın hemen başında yer alan bu kesimler `cli, sti` komutlarıyla ayraç arasına alınmıştır.

Sistemde yer alan her terminal birimi için bir kesilme yordamı öngörülmüştür. Bunlar `t-kes-0, t-kes-1` ve `t-kes-2` olarak adlandırılmıştır. Kesilme yordamlarının herbiri: bağlam saklama, yerel maskelerin sıfırlanması, hizmet, yerel maskelerin ve bağlamın eski değerlerle günlenmesi kesimlerinden oluşmaktadır. Bu kesimlerden hizmet olarak adlandırılanı, birimler arası öncelikleri gözetmek üzere kesilmelere açık olarak çalıştırılmaktadır. Terminal arabirimlerinin zincirleme bağlantı içerisindeki konumları gereğince, sıfırıncı terminal en öncelikli, ikinci terminal ise son öncelikli birimi oluşturmaktadır. `t-kes-0` adlı sıfırıncı terminalle ilgili kesilme yordamı içinde, bu birimin en öncelikli birim olması dolayısıyla sistem kesilmelere açılmamış ve bu nedenle yerel maske sıfırlama ve günleme işlemlerine de yer verilmemiştir. En düşük öncelikli birim olan *2* numaralı terminal birimine ilişkin kesilme yordamında, yerel maskeleme bağlamında *0* ve *1* numaralı terminallerin kesilme isteminde bulunabilmelerine izin verilmiş, bu amaçla salt ikinci arabirimin maskelenmesiyle yetinilmiştir. *1* numaralı terminal kesilme yordamı içinde, *1* ve *2* numaralı terminal arabirimleri maskelenerek, bu yordamdan geri dönülene değin *2* numaralı arabirimin kesilme üretmesinin engellenmesi amaçlanmıştır. *1* ve *2* numaralı arabirim kesilme yordamlarında, kesilme maskelerinin yukarıda açıklandığı biçimde kurulmasından sonra sistem kesilmelere hizmet kesimi öncesi açılmış ve bu yolla öncelikleri gözeten bir hizmet üretimi gerçekleştirilebilmiştir.

Hizmet kesimi içinde önce hata denetimi yapılmıştır. Bu denetim kapsamında arabirim durum yazmacı eşlik ve taşma hata bitleri sınanmıştır. Hata denetimini gerçekleştiren `hata sınama` adlı altyordam, hatalı durumu `dl` yazmacı içinde 0 döndürerek belirtmektedir. Bu yordamdan dönüşte, `dl` yazmacının sıfır içerdiği durumlarda kesilme yordamından herhangi bir işlem yapılmadan dönülmekte, bu yolla herhangi bir geri bildirim alamayan kullanıcının girdiği son damgayı yeniden girmeye zorlanması amaçlanmaktadır. Hatalı durumların dışında, her terminal için özel, `t-hizmet-0, t-hizmet-1 ve t-hizmet-2` adlı altyordamlar çağrılmaktadır. Hizmet altyordamları içinde hatasız okunan damganın ilgili `dizgi` alanına aktarılma ve damga yankılama işlemleri gerçekleştirilmektedir. Okunan damganın `ctrl-z` damgası olması durumunda ilgili arabirim, alma izni kapatılarak işlem dışı bırakılmaktadır.

```
;Üç terminal için Kesilmeli G/Ç Programlama Örneği-1

veri            segment
güdüm           equ   0a0h
durum           equ   0a1h
giris           equ   0a2h
çıkıs           equ   0a3h
kimlik          equ   0a4h
süren-hiz       equ   0b0h
tür0            equ   0F8H
tür1            equ   0F9H
tür2            equ   0FAH
ilk-güdüm       equ   10000111B
alma-maskesi    equ   11111101B
gönd-hazır      equ   00000001B
almaya-hazır    equ   00000010B
eşlik-hatası    equ   00000100B
taşma-hatası    equ   00001000B
kapat           equ   10000100B
ctrl-z          equ   1ah

dizgi0          db    256 dup(?)
dizgi1          db    256 dup(?)
dizgi2          db    256 dup(?)
dizin0          db    0
dizin1          db    0
dizin2          db    0
gösterge        db    0
veri            ends

yığıt           segment stack
                db    100 dup(?)
yığıt-başı      equ this word
yığıt           ends

kod             segment
                assume cs:kod, ds:veri, ss:yığıt
program:
                cli
; Kesim Yazmaçlarının günlenmesi
                mov   ax, veri
                mov   ds, ax
                mov   ax, yığıt
                mov   ss, ax
                mov   sp, offset yığıt-başı
```

Çizim 2.27. Üç Terminal için Kesilmeli Giriş/Çıkış Programlama Örneği-1

```
; kesilme vektörünün günlenmesi
                  xor   ax, ax
                  mov   es, ax
                  mov   ax, cs
                  mov   bx, tür0*4
                  mov   word ptr es:[bx], offset t-kes-0
                  mov   es:[bx+2], ax
                  mov   bx, tür1*4
                  mov   word ptr es:[bx], offset t-kes-1
                  mov   es:[bx+2], ax
                  mov   bx, tür2*4
                  mov   word ptr es:[bx], offset t-kes-2
                  mov   es:[bx+2], ax
; Kesilme Kimlik Yazmaçlarının günlenmesi
                  mov   al, tür0
                  out   kimlik, al
                  inc   al
                  out   kimlik+5, al
                  inc   al
                  out   kimlik+10, al
; Terminallerin işleme açılması, Dizinlerin günlenmesi
                  mov   al, ilk-güdüm
                  out   güdüm, al
                  out   güdüm+5, al
                  out   güdüm+10, al
                  mov   dizin0,0
                  mov   dizin1,0
                  mov   dizin2,0
                  sti
; Diğer işlemler
  dön:            jmp   dön               ;Benzetim

  t-kes-0:        push  ax                ;Bağlam saklama
                  push  bx
                  push  dx
                  push  si

  xor             dx, dx                  ;Hizmet
                  mov   dl, durum
                  call  hata-sınama
                  cmp   dl, 0
                  jz    son0
                  call  t-hizmet-0

  son0:           pop   si                ;Bağlam günleme
                  pop   dx
                  pop   bx
                  pop   ax
                  iret                    ;Geri dönüş
```

Çizim 2.27. Üç Terminal için Kesilmeli Giriş/Çıkış Programlama Örneği-1 (devam)

```
  t-kes-1:        push  ax              ;Bağlam saklama
                  push  bx
                  push  dx
                  push  si
                  in    al, güdüm+5     ;Maske sıfırlama
                  push  ax
                  and   al, alma-maskesi
                  out   güdüm+5, al
                  in    al, güdüm+10
                  push  ax
                  and   al, alma-maskesi
                  out   güdüm+10, al
                  sti
                  xor   dx, dx          ;Hizmet
                  mov   dl, durum+5
                  call  hata-sınama
                  cmp   dl, 0
                  jz    son1
                  call  t-hizmet-1
  son1:           cli
                  pop   ax              ;Maske günleme
                  out   güdüm+10, al
                  pop   ax
                  cmp   gösterge, 0ffh
                  jz    devam1
                  out   güdüm+5, al
  devam1:         pop   si              ;Bağlam günleme
                  pop   dx
                  pop   bx
                  pop   ax
                  iret                  ;Geri dönüş
;.....................................................

  t-kes-2:        push  ax              ;Bağlam saklama
                  push  bx
                  push  dx
                  push  si
                  in    al, güdüm+10    ;Maske sıfırlama
                  push  ax
                  and   al, alma-maskesi
                  out   güdüm+10, al
                  sti
                  xor   dx, dx          ;Hizmet
                  mov   dl, durum+10
                  call  hata-sınama
                  cmp   dl, 0
                  jz    son2
                  call  t-hizmet-2
```

Çizim 2.27. Üç Terminal için Kesilmeli Giriş/Çıkış Programlama Örneği-1 (devam)

```
  son2:           cli
                  pop   ax                ;Maske günleme
                  cmp   gösterge, 0ffh
                  jz    devam2
                  out   güdüm+10, al
  devam2:         pop   si                ;Bağlam günleme
                  pop   dx
                  pop   bx
                  pop   ax
                  iret
;.....................................................
  t-hizmet-0      proc  near
; girilen damganın ilgili dizgiye konması
                  lea   bx, dizgi0
                  mov   si, dizin0
                  in    al, giriş
                  mov   [bx][si], al
                  push  ax
; damga yankılama
  dön0:           in    al, durum
                  test  al, gönd-hazır
                  jz    dön0
                  pop   ax
                  out   çıkış, al
; dizgi sonu sınama
                  cmp   al, ctrl-z
                  jz    son0
                  inc   dizin0
                  ret
  son0:           mov   al, kapat
                  out   güdüm, al
                  ret
  t-hizmet-0      endp

;.....................................................
  t-hizmet-1      proc  near
; girilen damganın ilgili dizgiye konması
                  mov gösterge, 0
                  lea   bx, dizgi1
                  mov   si, dizin1
                  in    al, giriş+5
                  mov   [bx][si], al
                  push  ax
; damga yankılama
  dön1:           in    al, durum+5
                  test  al, gönd-hazır
                  jz    dön1
                  pop   ax
                  out   çıkış+5, al
```

Çizim 2.27. Üç Terminal için Kesilmeli Giriş/Çıkış Programlama Örneği-1 (devam)

```
; dizgi sonu sınama
                  cmp   al, ctrl-z
                  jz    son1
                  inc   dizin1
                  ret
  son1:           mov   gösterge, 0ffh
                  mov   al, kapat
                  out   güdüm+5, al
                  ret
  t-hizmet-1      endp
;.....................................................

  t-hizmet-2      proc  near
; girilen damganın ilgili dizgiye konması
                  mov gösterge, 0
                  lea   bx, dizgi2
                  mov   si, dizin2
                  in    al, giriş+10
                  mov   [bx][si], al
                  push  ax
; damga yankılama
  dön2:           in    al, durum+10
                  test  al, gönd-hazır
                  jz    dön2
                  pop   ax
                  out   çıkış+10, al
; dizgi sonu sınama
                  cmp   al, ctrl-z
                  jz    son2
                  inc   dizin2
                  ret
  son2:           mov   gösterge, 0ffh
                  mov   al, kapat
                  out   güdüm+10, al
                  ret
  t-hizmet-2      endp
;.....................................................

  hata-sınama     proc  near
                  in    al, dx
                  test  al, almaya-hazır
                  jz    dön-hata
                  test  al, eşlik-hatası
                  jnz   dön-hata
                  test  al, taşma-hatası
                  jnz   dön-hata
                  mov   dl, 0ffh
                  ret
```

Çizim 2.27. Üç Terminal için Kesilmeli Giriş/Çıkış Programlama Örneği-1 (devam)

```
dön-hata:       inc   dx     ; RRDY'nin sıfırlanması için
                in    al,dx  ; boş okuma
                mov   dl, 0
                ret
hata-sınama     endp

kod             ends
                end   program
```

Çizim 2.27. Üç Terminal için Kesilmeli Giriş/Çıkış Programlama Örneği-1 (devam)

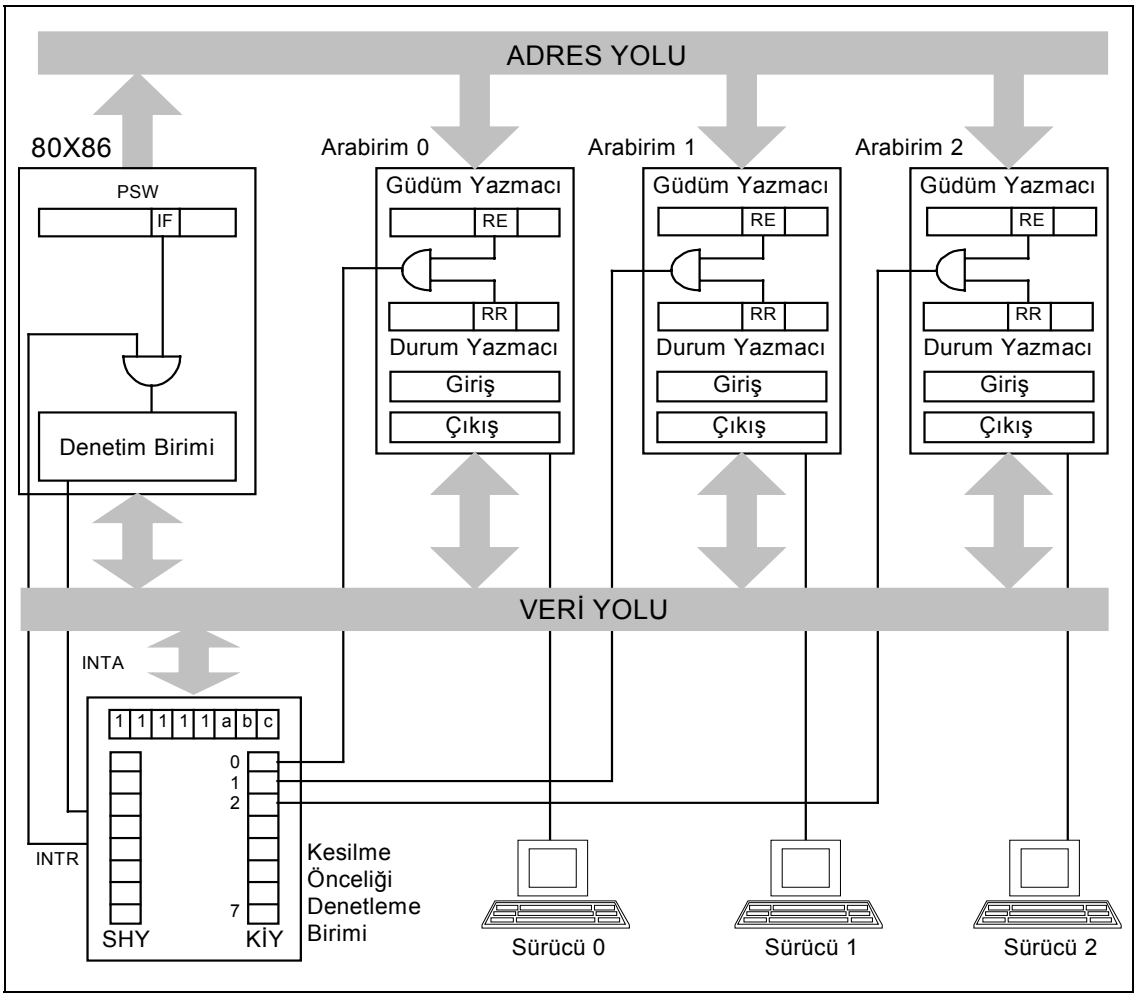
Kesilmeli Giriş/Çıkış Programlama Örneği-1'deki `t-hizmet-0`, `t-hizmet-1` ve `t-hizmet-2` adlı altyordamların herbirinde, aynı işlemler değişik arabirimler için aynı biçimde gerçekleştirilmektedir. Bu yordamlar içinde değişen sadece giriş/çıkış kapı adresleridir. Bu durumda doğal olarak, üç ayrı yordam yerine tüm terminal birimleri için ortak bir hizmet yordamı öngörmek ve değişen kapı adreslerini bu yordama parametre olarak aktarmak akla gelebilir. Ancak verilen örneklerde, açıklanan yöntem ve düzeneklerin kolayca algılanabilmesi için programların okunurluğu ön plana alınmış ve hizmet yordamlarının ayrı ayrı yazılması tercih edilmiştir.

2.4.2. Kesilme Önceliği Denetleme Biriminin Kullanımı

Kesilme yordamları içinde, hizmet kesimlerinin kesilmelere açık olarak işletilmesi arabirimler arası önceliklerin gözetilmesini olanaklı kılar. Ancak bu olanak sözkonusu kesimin başında ve sonunda yerel maskelerin sıfırlanma ve kurulma işlemlerinin ele alınma zorunluluğunu doğurur. Arabirimler arası önceliklerin yazılım yoluyla ele alınması, özellikle arabirim sayısının çok olduğu sistemlerde kesilme yordamlarının işletim sürelerinin artmasına ve sistem başarımının düşmesine neden olur. Bu sakıncanın yanı sıra maskeleme işlevlerinin çok sayıda kesilme yordamının içine serpiştirilmiş olması sistem yazılımlarının karmaşıklaşması ve okunurluklarının olumsuz yönde etkilenmesi sonucunu da doğurur.

Bilindiği gibi kesilme önceliği denetleme birimi adlı özel donanım, arabirimler arası önceliklerin donanım yoluyla ele alınmasına olanak sağlayarak kesilme yordamlarını bu işlevi gerçekleştirme yükünden kurtarır. Çizim 2.28'de, terminal arabirimlerinin kesilme istemlerini doğrudan işleyiciye göndermek yerine, örnek bir kesilme önceliği denetleme birimine gönderdikleri sistemin ilke çizimi yer almıştır. Söz konusu öncelik denetleme biriminin, Çizim 2.20'de verilen öncelik denetleme biriminde olduğu üzere, 8 kesilme istem girişi ile birer baytlık bir kesilme istem yazmacı, bir süren hizmet yazmacı ve bir kesilme kimliği yazmacı bulunduğu düşünülmüştür. Bu birimin programlanır (okunur-yazılır) tek yazmacının süren hizmet yazmacı olduğu ve bu yazmacın giriş/çıkış evreni içinde, (`süren-hiz` sözcüğüyle simgelenen) adresinin `B0H` olduğu varsayılmıştır. Kesilme önceliği denetleme birimi, adına kesilme istediği arabirimin kesilme kimliğini bir baytlık kesilme kimliği yazmacının en küçük ağırlıklı son üç biti üzerinden kodlamaktadır. Kodlamada yararlanılmayan 5 bit, 1 değerini taşımaktadır. Bu

bağlamda, INTA imi alındığında sıfırıncı istem girişine bağlı arabirim için `F8H`, birinci istem girişine bağlı arabirim için `F9H`, ikinci istem girişine bağlı arabirim için ise `FAH` değerleri, kesilme kimliği olarak işleyiciye yollanmaktadır.



Çizim 2.28. Kesilmeli G/Ç Programlama Örneği-2 için Sistem İlke Çizimi

Bilindiği üzere, kesilme önceliği denetleme biriminin kullanılması sayesinde kesilme yordamları maske sıfırlama ve maske günleme amacını taşıyan kesimlerden arınmaktadır. Ancak bu yordamlardan geri dönüşten hemen önce, kesilme yordamına sapılırken kurulan, ilgili süren hizmet yazmaç bitinin sıfırlanması gerekmektedir. Kesilme önceliği denetleme biriminin kullanıldığı durumda, `ctrl-z` damgasına rastlanana değin üç değişik terminal biriminden girilen damgaları ilgili `dizgi` alanına aktaran sistem programı, Çizim 2.27'de verilenle, `t-kes-0, t-kes-1, t-kes-2` adlı kesilme yordamlarının dışında aynıdır. Bu yordamlardan, yukarıda belirtildiği gibi yerel maskeleri günleyen kesimler çıkarılmış ancak `iret` komutlarından hemen önce, süren hizmet yazmacı bitlerini sıfırlayan komutlar eklenmiştir. Bu nedenle Çizim 2.29'da, `t-kes-0, t-kes-1, t-kes-2` adlı kesilme yordamlarının yeni görünümlerinin verilmesiyle yetinilmiştir.

```
  t-kes-0:          push  ax              ;Bağlam saklama
                    push  bx
                    push  dx
                    push  si
                    xor   dx, dx          ;Hizmet
                    mov   dl, durum
                    call  hata-sınama
                    cmp   dl, 0
                    jz    son0
                    call  t-hizmet-0
  son0:             in    al, süren-hiz   ;süren hizmet
                    and   al, 11111110B   ;yazmacı 0ıncı bit
                    out   süren-hiz, al   ;sıfırlama
                    pop   si              ;Bağlam günleme
                    pop   dx
                    pop   bx
                    pop   ax
                    iret
;.....................................................

  t-kes-1:          push  ax              ;Bağlam saklama
                    push  bx
                    push  dx
                    push  si
                    sti
                    xor   dx, dx          ;Hizmet
                    mov   dl, durum+5
                    call  hata-sınama
                    cmp   dl, 0
                    jz    son1
                    call  t-hizmet-1
  son1:             cli
                    in    al, süren-hiz   ;süren hizmet
                    and   al, 11111101B   ;yazmacı 1inci bit
                    out   süren-hiz, al   ;sıfırlama
                    pop   si              ;Bağlam günleme
                    pop   dx
                    pop   bx
                    pop   ax
                    iret
;.....................................................

  t-kes-2:          push  ax              ;Bağlam saklama
                    push  bx
                    push  dx
                    push  si
                    sti
```

Çizim 2.29. Kesilmeli G/Ç Prog. Örneği-2 Kesilme Yordamlarının Görünümü

```
                xor   dx, dx          ;Hizmet
                mov   dl, durum+10
                call  hata-sınama
                cmp   dl, 0
                jz    son2
                call  t-hizmet-2
son2:           cli
                in    al, süren-hiz   ;süren hizmet
                and   al, 11111011B   ;yazmacı 2nci bit
                out   süren-hiz, al   ;sıfırlama
                pop   si              ;Bağlam günleme
                pop   dx
                pop   bx
                pop   ax
                iret
```

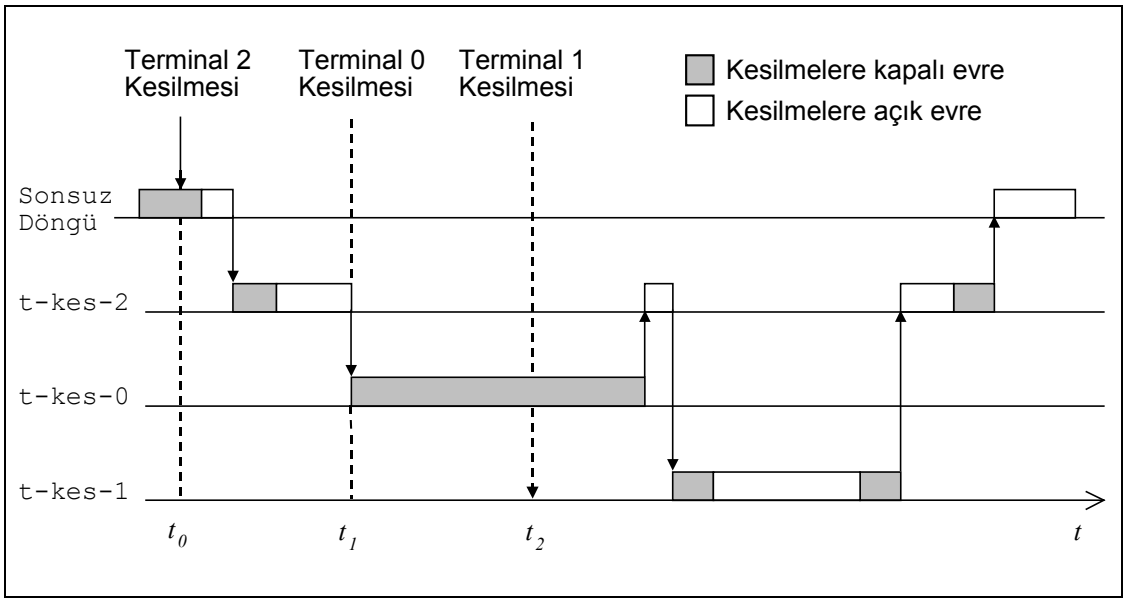
Çizim 2.29. Kesilmeli G/Ç Prog. Örneği-2 Kesilme Yordamlarının Görünümü (devam)

Kesilme önceliklerinin gerek kesilme yordamlarının içinden yazılım yoluyla, gerekse kesilme önceliği denetleme birimi tarafından donanım yoluyla denetlendiği durumlarda kesilme yordamlarının nasıl ve hangi sırada işletime girdiği Çizim 2.30'da örneklenmiştir. Bunun için, alma işlem izinlerinin açıldığı andan başlayarak arabirimlerin kesilme istemlerinin geliş sırası Çizim 2.30'da verilen zaman ekseni üzerinde gösterilmiştir. Buna göre kesilme istemlerinden ilki *2* numaralı terminalden t_0 anında gelmektedir. Bunu t_1 anında *0* numaralı terminalden ve t_2 anında da *1* numaralı terminalden gelen istemler izlemektedir. Zaman ekseni üzerinde *1* ve *2* numaralı kesilme yordamlarının işletimleri üç dilim üzerinden gösterilmiştir. Bu dilimlerden ilki `sti` komutuna kadar olan kesilmelere kapalı kesimi, ikincisi, kesilmelere açık işletilen `hizmet` kesimini, üçüncüsü ise `cli` komutundan başlayarak `iret` komutuna kadar olan ve yine kesilmelere kapalı olarak işletilen son kesimi temsil etmektedir.

Bu çizime göre *2* numaralı arabirimden gelen kesilme istemi ile, `t-kes-2` adlı kesilme yordamına sapılmaktadır. Bununla ilgili `hizmet` kesimi işletilirken *0* numaralı arabirimin istemi ortaya çıkmaktadır. Daha öncelikli bir arabirim olması dolayısıyla `t-kes-2` kesilerek `t-kes-0` adlı kesilme yordamına sapılmaktadır. Bu yordam kesilmelere kapalı işletildiğinden, bu arada *1* numaralı arabirimden gelen kesilme istemi, ancak `t-kes-2` adlı yordama geri dönüldükten sonra göz önüne alınabilmektedir. Zira sistemin kesilmelere açılması, `t-kes-0` yordamı `iret` komutunun, `PSW` yazmacını, IF'nin kurulu olduğu eski değerle günlemesi sonucu gerçekleşmektedir.

Bir numaralı terminalden gelen kesilme istemi, `t-kes-2` içinde dönülen komutun işletimi sırasında algılanabilmektedir. `t-kes-2`'nin bir komutu işletildikten sonra yeniden kesilerek `t-kes-1`'e sapılmaktadır. `t-kes-1` işletildikten sonra `t-kes-2`'ye

geri dönülmekte, bu yordam tamamlanınca da, bu işlemlerle koşut çalıştırılan programların benzetimini yapmaya yarayan bitmeyen döngü komutunun işletimine geçilmektedir.



Çizim 2.30. Kesilme Yordamlarının Önceliklerine göre İç içe İşletilmesi Örneği

2.4.3. Disket Birimi Kesilmeli G/Ç Programlama Örneği

Kesilmeli giriş/çıkış programlama konusunda verilecek üçüncü örnekte, arabirim görünümü Çizim 2.31'de verilen disket sürücü üzerinde saklanan bir öbek verinin ana belleğe aktarılması söz konusu edilecektir. Sözkonusu öbeğin 512 bayt uzunluğunda bir sektörden oluştuğu varsayılacaktır. Örnek disket arabiriminin, bir önceki örnekte açıklanan kesilme önceliği denetleme biriminin *1* numaralı girişine bağlı olduğu; RDY, SC, ZR bitleriyle simgelenen durumlarda kesilme ürettiği; arabirim düzeyinde yer alan yazmaçlardan:

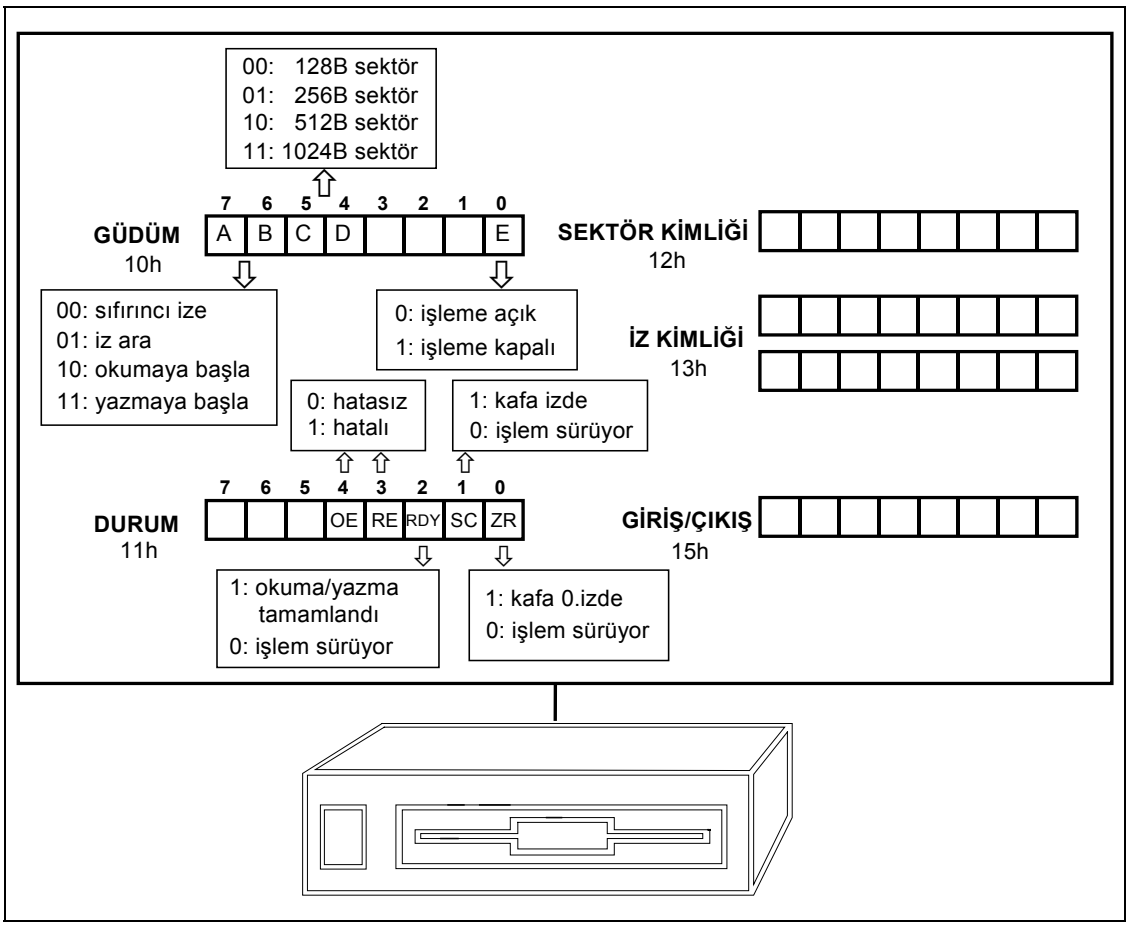
Güdüm	yazmacının	`10H`
Durum	yazmacının	`11H`
Sektör Kimliği	yazmacının	`12H`
İz Kimliği	yazmacının	`13H`
Giriş/Çıkış	yazmacının	`15H`

adreslerine atandığı varsayılacaktır. Okunacak disk öbeğinin {`İZ,SEKTÖR`} ikilisi ile tanımlı olduğu, ana bellekte `disk-yastığı` adlı yastık alanına aktarılacağı düşünülecektir.

{`İZ,SEKTÖR`} ikilisi ile tanımlı öbeğin ana bellekte `disk-yastığı` adlı yastık alanına aktarılabilmesi için:

- önce disket okuma-yazma kafası `İZ`'le simgelenen iz üzerine taşınacak,
- sonra da `SEKTÖR`'le simgelenen sektör baytlarının okuma komutu işletilecektir.

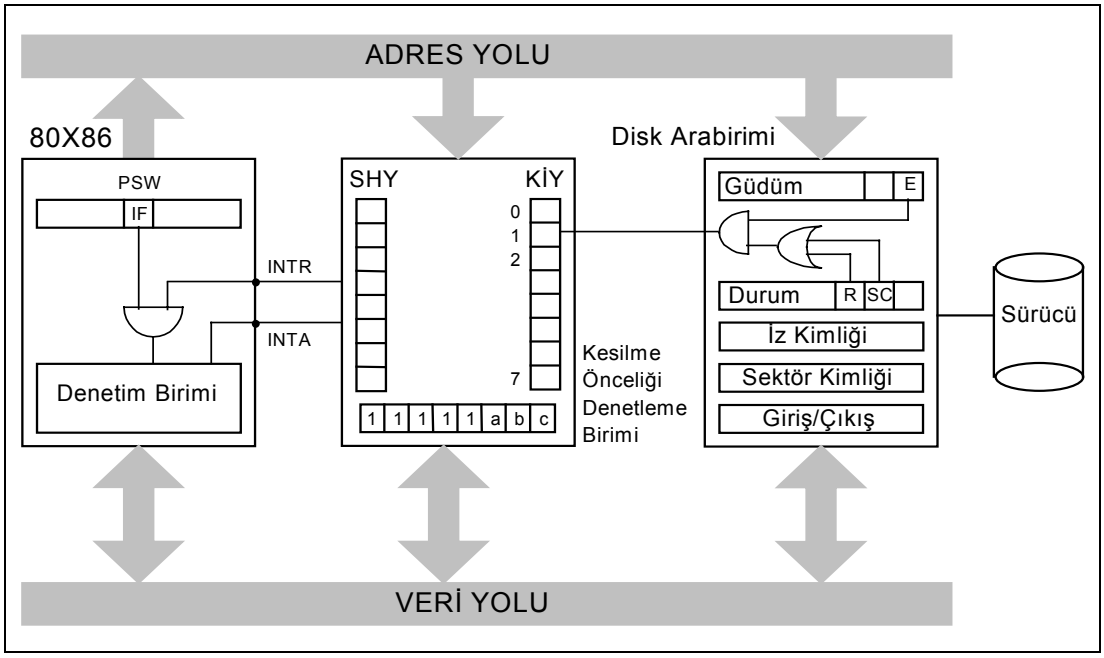
Bunun için disket arabirimi iz kimliği yazmacına `İZ` değeri, sektör kimliği yazmacına da `SEKTÖR` değeri yazılacaktır. Bu işlemin sonrasında güdüm yazmacına, okuma-yazma kafasının ilgili ize götürülmesini sağlayan `0110XXX1` güdüm değeri yazılacaktır. Okuma-yazma kafası ilgili ize ulaştığında arabirim, SC durum bitine bağlı olarak kesilme önceliği denetleme biriminin *1* numaralı girişine kesilme istemi gönderecektir. Bu istemle ilgili kesilme yordamı içinde disket arabirimi güdüm yazmacına, bu kez `1010XXX1` değeri yazılarak `SEKTÖR` baytlarının okunması başlatılacaktır. `SEKTÖR` baytları `giriş/çıkış` yastığında okunmaya hazır oldukça, bu kez RDY biti üzerinden, kesilme önceliği denetleme biriminin *1* numaralı girişine kesilme istemleri gönderilecek ve `giriş/çıkış` yastık içeriklerinin, ardarda `disk-yastığı` adlı alana aktarılması sağlanacaktır. Bu işlemlerle ilgili örnek sistem programı Çizim 2.33'te, ilgili sistem ilke çizimi de Çizim 2.32'de verilmiştir. Çizim 2.8'de verilen disket arabirimi görünümü Çizim 2.31'de, izleme kolaylığı açısından yinelenmiştir.



Çizim 2.31. Örnek Disket Arabirimi Görünümü

Çizim 2.33'te verilen program önbelirleme ve `disk-kes` adlı kesilme yordamından oluşmaktadır. Önbelirlemeler kesiminde, kesim yazmaçları ve kesilme vektörü günleme işlemleri tamamlandıktan sonra disket arabirimi iz ve sektör kimliği yazmaç içerikleri {`İZ,SEKTÖR`} ikilisi ile günlenip okuma-yazma kafasının `İZ`'e taşınma güdümü

çalıştırılmaktadır. Bunun sonrasında sistem kesilmelere açılarak 512 baytlık disk sektörünün (öbeğinin), kesilmeler yoluyla, bayt bayt, ana belleğe aktarılması sağlanmaktadır.



Çizim 2.32. Disket Birimi Kesilmeli G/Ç Programlama Örneği Sistem İlke Çizimi

`disk-kes` adlı kesilme yordamına iki durumda sapılmaktadır. Bu durumlardan ilki, SC bitiyle simgelenen, disket okuma-yazma kafasının iz kimliği yazmacı içinde verilen ize ulaşması durumudur. İkinci durum ise RDY bitinin kurulması durumudur. SC biti okuma-yazma kafasının ilgili ize taşınması için çalıştırılan `izegit` disk güdümü sonrasında bir kez kurulurken RDY biti, öbeği oluşturan baytların herbiri için, 512 kez kurulacaktır. Bu nedenle bir öbeğin sürücüden ana belleğe aktarımı için `disk-kes` yordamı toplam 513 kez çağrılacaktır. SC biti okuma-yazma kafasının ilgili ize taşınması için çalıştırılan disk güdümünün gerçekleşmesi sonucu kurulacak, bunun üreteceği kesilme istemi ile sapılan kesilme yordamında, bu durum sınanarak okuma güdüm kodunun arabirime yazılması gerçekleşecektir. Arabirimin çalışma ilkesi gereği SC biti, arabirime yeni bir güdüm değeri yazılana değin kurulu kalacağından `okumayabaşla` güdümünün arabirime yazılması ile sıfırlanacaktır. Buna dayalı olarak, `disk-kes` yordamına ikinci sapıştan başlayarak hep `yineoku` etiketi ile başlayan kesim işletilecektir.

```
; Disket Birimi için Kesilmeli G/Ç Programlama Örneği
  veri              segment
  diskgüdüm         equ   10h
  diskdurum         equ   11h
  sektörkimliği     equ   12h
  izkimliği         equ   13h
  diskg/ç           equ   15h
  iz                equ   ...
  sektör            equ   ...
  disktür           equ   0f9h
  sc                equ   00000010B
  rdy               equ   00000100B
  izegit            equ   01100001B
  okumayabaşla      equ   10100001B
  kapalı            equ   11111110B
  süren-hiz         equ   0b0h
  diskyastığı       db    512 dup(?)
  dizin             dw    0
  veri              ends
  yığıt             segment stack
                    db    100 dup(?)
  yığıt-başı        equ this word
  yığıt             ends
  kod               segment
                    assume cs:kod, ds:veri, ss:yığıt
  program:          cli
; Kesim Yazmaçlarının günlenmesi
                    mov   ax, veri
                    mov   ds, ax
                    mov   ax, yığıt
                    mov   ss, ax
                    mov   sp, offset yığıt-başı
; Kesilme vektörünün günlenmesi
                    xor   ax, ax
                    mov   es, ax
                    mov   ax, cs
                    mov   bx, disktür*4
                    mov   word ptr es:[bx], offset disk-kes
                    mov   es:[bx+2], ax
; Okuma-yazma kafasının İZ'e taşınması
                    mov   ax, iz
                    out   izkimliği, ax
                    mov   al, sektör
                    out   sektörkimliği, al
                    mov   al, izegit
                    out   diskgüdüm, al
                    mov   dizin,0
                    sti
  dön:              jmp   dön             ; Benzetim
```

Çizim 2.33. Disket Birimi Kesilmeli G/Ç Programlama Örneği

```
disk-kes:
                push  ax
                push  si
                sti
                in    al, diskdurum
                test  al, sc
                jnz   okumabaşlat
                test  al, rdy
                jnz   yineoku
                jmp   son
okumabaşlat:                              ;SC kesilmesi
                mov   al, okumayabaşla
                out   diskgüdüm, al
                jmp   son
yineoku:                                  ;RDY kesilmesi
                in    al, diskg/ç
                mov   si, dizin
                mov   diskyastığı[si], al
                inc   si
                cmp   si, 512
                jz    kapat
                mov   dizin,si
                jmp   son
kapat:                                    ;işlem tamam
                in    al, diskgüdüm
                and   al, kapalı
                out   diskgüdüm, al
son:            cli
                in    al, süren-hiz       ;süren hizmet
                and   al, 11111101B       ;yazmacı 1. bit
                out   süren-hiz, al       ;sıfırlama
                pop   si
                pop   ax
                iret
kod             ends
                end   program
```

Çizim 2.33. Disket Birimi Kesilmeli G/Ç Programlama Örneği (devam)

Kesilmeli programlama yönteminde, arabirimlerin rasgele anlarda ortaya çıkan hizmet istemleri, kesilme yordamları aracılığıyla ele alınır. Arabirimler çoğu kez arabirim giriş/çıkış yastıkları ile ana bellek arası aktarım gereksinimi nedeniyle kesilme istemi üretirler. Arabirim ile ana bellek arasında veri aktarımı, kesilme yordamları içindeki bellek erişim ve giriş/çıkış komutlarının işletimiyle, başka bir deyişle ana işlem birimi aracılığıyla gerçekleşir. Özellikle, disk ve mıknatıslı şerit birimleri gibi, sürücüden arabirime verilerin öbek tabanlı olarak aktığı giriş/çıkış birimleri için arabirim ana

bellek arası veri aktarımlarının, bizzat ana işlem birimi kullanılarak yapılması, bu sonuncunun kullanım verimliliğini olumsuz yönde etkiler. Bu nedenle, sistem donanımları içinde, sözkonusu bu aktarımları, gerektiğinde ana işlem birimi yerine yapacak, Doğrudan Bellek Erişim Denetleme Birimi olarak adlandırılan yardımcı işleyiciler öngörülür. Bu durumda, Doğrudan Bellek Erişim Düzeneği ve Doğrudan Bellek Erişimli Giriş/Çıkış Programlamadan söz edilir. İzleyen kesimde bu düzenek ve ilgili giriş/çıkış programlama yöntemi açıklanacaktır.

2.5. Doğrudan Bellek Erişim Düzeneği

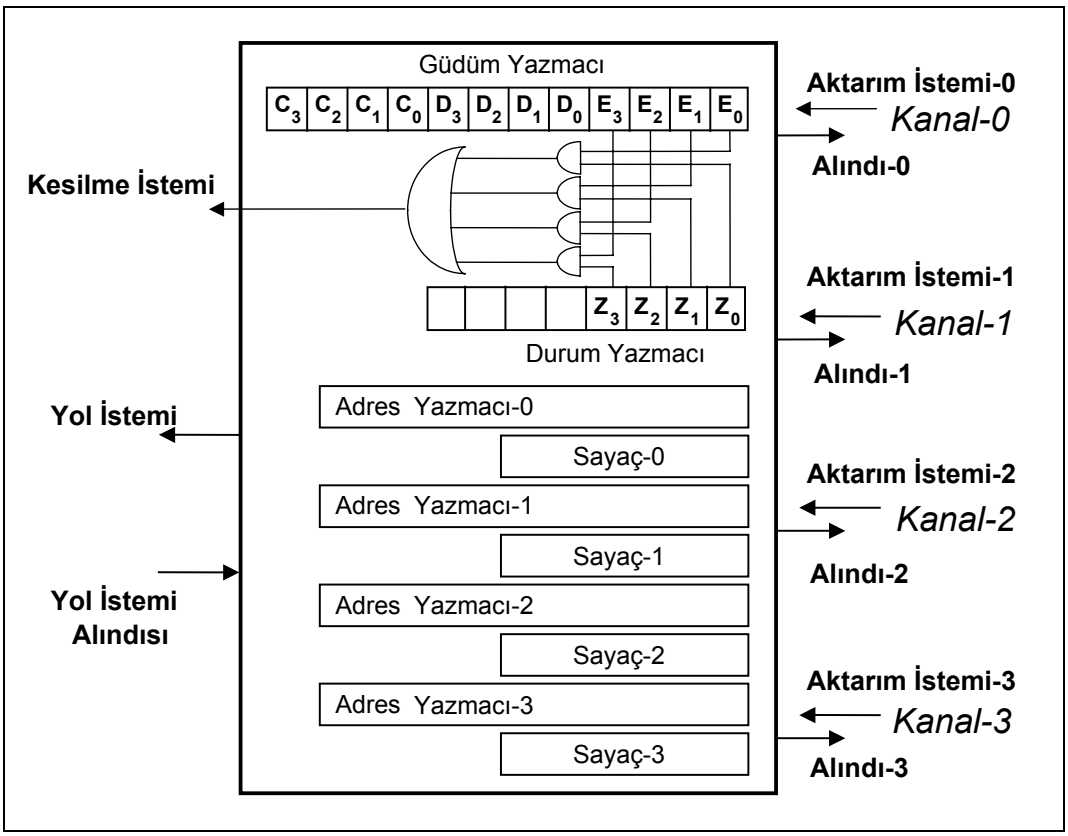
Giriş/çıkış arabirimleri ile ana bellek arasındaki veri aktarımını, ana işlem birimini dışarıda tutarak gerçekleştirmede doğrudan bellek erişim düzeneğinden yararlanılır. Bu düzenek çerçevesinde arabirimlerden ana belleğe, ana bellekten arabirimlere veriler, doğrudan bellek erişim denetleme birimi olarak adlandırılan bir birimin yardımıyla aktarılır. Doğrudan bellek erişim denetleme birimi, birlikte çalıştığı ana işlem biriminin bellek erişimiyle ilgili işlevlerini yüklenebilen yardımcı bir işleyicidir. Doğrudan bellek erişim denetleme biriminin yer aldığı sistemlerde ana belleğe, ana işlem biriminin yanı sıra bu birimin erişimi de söz konusu olur. Birden çok işleyicinin aynı ana bellek birimine koşut erişimi kaynak paylaşımı sorununu ortaya çıkarır. Bu sorun, çoğu kez ana işlem biriminin efendi konumunda bulunduğu, efendi-köle ilişkisi içinde çözülür. Doğrudan bellek erişim denetleme birimi ana işlem biriminin onayı olmadan ana belleğe erişemez. Ana işlem birimi donanımı içinde yer alan bu onay düzeneği, doğrudan bellek erişim düzeneğinin bir parçasıdır. Bu düzenek, ana belleğe erişmede yararlanılan ve kullanılagelen donanım modeline göre biricik olduğu varsayılan adres ve veri yollarının kullanımını düzenler. İki ayrı işleyicinin, ana belleğe, aynı anda, aynı yolları kullanarak erişmesinin mümkün olmadığı açıktır.

2.5.1. Doğrudan Bellek Erişim Denetleme Birimi

Doğrudan bellek erişim düzeneğinin kullanıldığı sistemlerde arabirimler, bağlı oldukları sürücülerden ana belleğe ya da ana bellekten bu sürücülere veri aktarımı söz konusu olduğunda bu aktarımla ilgili istemlerini, ana işlem birimi yerine doğrudan bellek erişim denetleme birimine yaparlar. Aktarım istemini alan denetleme birimi, bu istemi yerine getirebilmek için, ana işlem biriminden adres ve veri yollarının kullanım hakkını ister. Bu istem ana işlem biriminin, yol kullanım istemi girişi üzerinden yapılır. Ana işlem birimi, doğrudan bellek erişim denetleme biriminin yol kullanım isteminin bağlandığı bu girişini, kesilme istemi girişlerinde olduğu gibi, komut işletiminin belirli bir periyodunda sınar. Giriş üzerinde istemi simgeleyen bir gerilim düzeyi saptarsa adres ve veri yolları ile okuma, yazma gibi bellek denetim hatlarıyla ilişkisini keser. Bu durumu, erişim istemini yapan denetleme birimine, yol kullanım istemi alındısı çıkışını kurarak belirtir. Bu alındı ile doğrudan bellek erişim denetleme birimi, arabirim - ana bellek arasında, doğrudan gerçekleşecek aktarımın denetimini ele geçirir. Bu aşamadan sonra denetleme birimi, kullanım hakkını elde ettiği adres yolu üstüne, aktarıma konu olan bellek sözcük adresini koyarak bir bellek erişim döngüsü başlatır. Bunun yanı sıra, adresleme süreci başlatılan ana bellek sözcüğün içeriği, aktarım isteminde bulunan

arabirim giriş/çıkış yastığından sağlanacağından bu arabirime aktarım istemi alındısını yollar. Bu alındı imi ile, aktarıma konu olan verinin arabirim tarafından, ya giriş/çıkış yastığından veri yoluna (yazma) ya da veri yolundan bu yastığa (okuma) aktarılması sağlanır.

Doğrudan bellek erişim denetleme birimi, ana bellekte erişilen sözcük adreslerini, adres yazmacı olarak nitelenen, programlanır bir yazmaç içinde tutar. Aktarım işlemlerinin başında bu yazmaç ana bellek aktarım alanı başlangıç adresini içerir. Aktarılan bayt (sözcük) sayısı bir sayaçla denetlenir. Gerçekleştirilen her doğrudan bellek erişim döngüsü için adres yazmacı otomatik olarak bir artırılırken sayaç içeriği bir eksiltilir. Aktarım işlemlerinin başında ana bellek aktarım alan boyu ile günlenen sayaç içeriği sıfırlandığında bu durum, denetleme birimince, ana işlem birimini, kesilme girişleri üzerinden uyarmada kullanılır. Bu uyarı ile adres yazmacı ile sayacın yeni bir aktarım için, yeni değerlerle günlenmesi sağlanır.



Çizim 2.34. Örnek bir Doğrudan Bellek Erişim Denetleme Birimi Görünümü

Yukarıda açıklanan çalışma ilkesi gereğince doğrudan bellek erişim denetleme birimi, öncelikle bir adres yazmacı ile bir sayaç içermek durumundadır. Çoğu kez doğrudan bellek erişim denetleme birimleri aynı anda birden çok arabirime doğrudan bellek erişim hizmeti sunacak biçimde üretilirler. Bu tür denetleme birimlerinin birden çok aktarım istem girişi ve istem alındısı çifti bulunur. İstem-alındı çiftleri genellikle

doğrudan bellek erişim kanalı olarak adlandırılır. Bu durumda sözkonusu denetleme biriminin herbir kanal için bir adres yazmacı ve sayaç çifti bulunmak zorundadır. Adres yazmaçları ana bellekteki fiziksel adresleri tutarlar. 16 bit uzunluğundaki bir adres yazmacı 64 KB'lık bir bellek kesimini adreslemede yeterli olur. 512 baytlık bir tutanağın, doğrudan bellek erişim yöntemiyle aktarımının yapılabilmesi için, 9 bitlik bir sayaç yeterlidir.

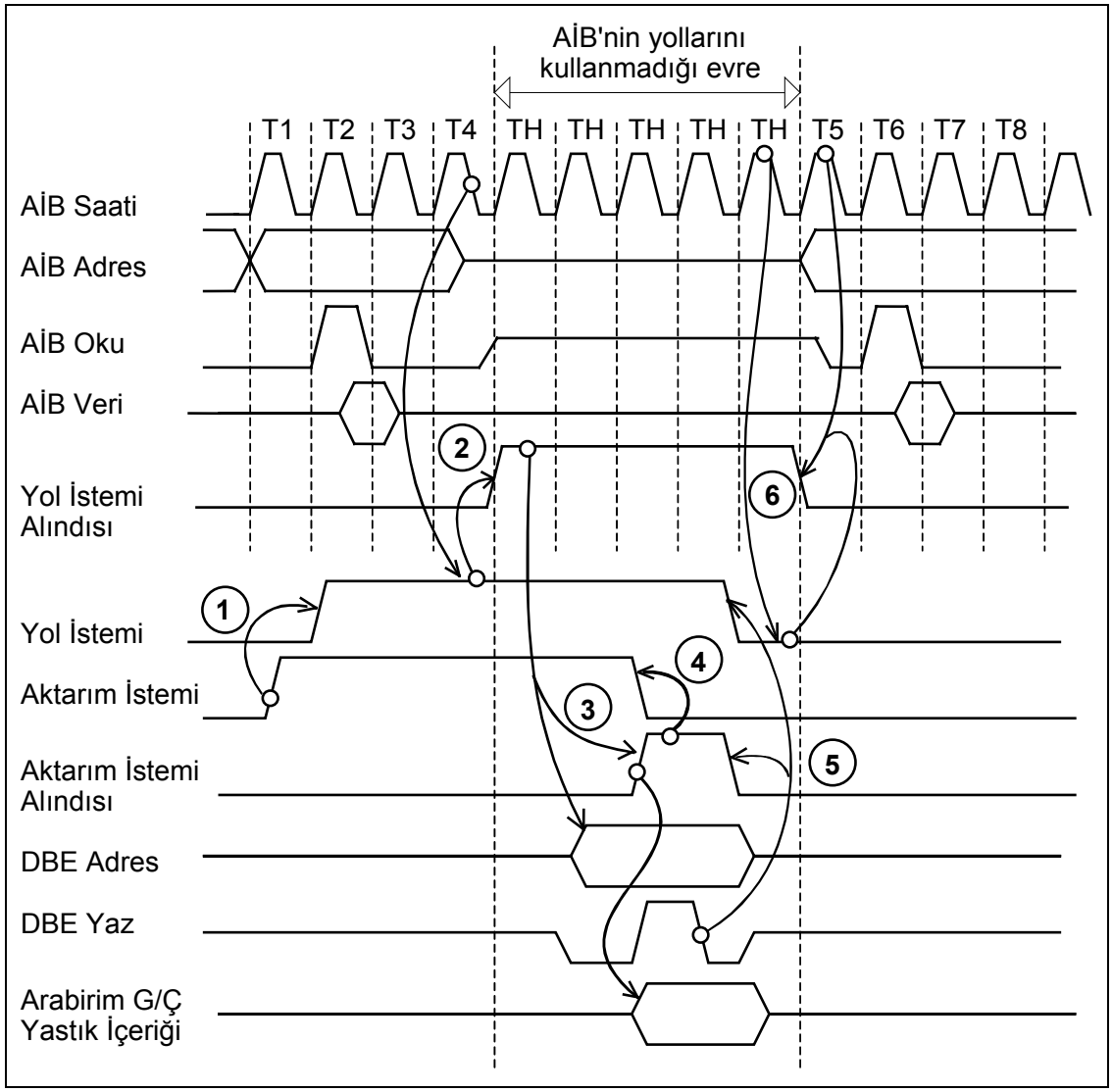
Arabirimler yönünden bakıldığında, doğrudan bellek erişim amacıyla başvurulan yardımcı bir işleyici olarak görünen doğrudan bellek erişim denetleme birimi, efendi-köle ilişkisi gereği, ana işlem birimi için, arabirim benzeri, bir dizi giriş/çıkış kapısından oluşan bir birimdir. Bu giriş/çıkış kapılarının bir kesimi, yukarıda belirtilen adres yazmaçları ve sayaçlardan oluşur. Bunların yanı sıra, denetleme biriminin, ana işlem birimi tarafından denetlenen güdüm ve durum yazmaçları da bulunur. Doğrudan bellek erişim denetleme birimi, arabirimler için işleyici, ana işlem birimi için ise bir arabirim görünümündedir (Çizim 2.34).

Doğrudan bellek erişim denetleme birimi güdüm yazmaçları, birimin her kanalı için işlem izin, yön (okuma/yazma), sürekli/kesikli aktarım bitleri gibi güdüm bitleri içerirler. İşlem izin biti ilgili kanal üzerinden gelen istemlerin gözönüne alınıp alınmamasına olanak verir. Yön biti aktarımın arabirimden ana belleğe mi, yoksa ana bellekten arabirime doğru mu yapılacağını belirlemede kullanılır. Sürekli / kesikli aktarım biti ise aktarımın, bayt (sözcük) ya da öbek tabanlı bir aktarım olacağını belirler. Bayt tabanlı aktarımda arabirim aktarılacak her bayt için bir aktarım istemi üretir. Denetleme birimi de her istem için bir bellek erişim döngüsü kurar ve aktarılan baytlar arasında, ana işlem birimine yolladığı yol kullanım istem imini çeker. Öbek tabanlı aktarımda ise arabirim, aktarımın başında bir kez istemde bulunur. Denetleme birimi ilgili sayaç içeriği sıfırlanana değin, öbek uzunluğu kadar bellek erişim döngüsünü, ara vermeksizin ardarda gerçekleştirir. Aktarılan baytlar arasında ana işlem birimine yollanan yol kullanım istem imi, aktarım sonlanana değin çekilmez.

Durum yazmaçları, çoğu kez kanallarla ilgili sayaçların sıfırlanması durumunda kurulan durum bitlerinden oluşur. Bu bitlerle simgelenen "sayaç sıfırlandı" durumu aynı zamanda, kesilme istem imi biçiminde kullanılmak üzere denetleme biriminin dışına taşınır. Daha önce de belirtildiği üzere, bu uyarı ile, ilgili adres yazmacı ve sayacın yeni aktarım değerleri ile günlenmesi sağlanır.

Çizim 2.35'te verilen zaman çizeneğinde, Çizim 2.36'da verilen sistem yapısına uygun olarak sürücüden arabirime ulaşan bir baytın, bayt tabanlı aktarım modunda ana belleğe yazılması açıklanmıştır. Bu amaçla, giriş/çıkış yastığı okunmaya hazır olduğunda arabirimce kurulan aktarım istemi iminin, doğrudan bellek erişim denetleme birimi tarafından yol istemi imine dönüştürülmesi **1** numaralı geçişle gösterilmiştir. İşleyicinin yol istemi girişini T4 numaralı periyotta sınadığı, izleyen TH adlı periyottan başlayarak yol istemi alındısını kurduğu, **2** numaralı geçişle belirtilmiştir. Zaman çizeneği üzerinde çizimi ağırlaştırmamak amacıyla açıkça belirtilmemiş olmasına karşın, TH periyotlarının hepsinde, yol istemi girişinin ana işlem birimince sınandığı, bu giriş üzerinde istem bulunması durumunda izleyen periyotta ana işlem biriminin adres, veri,

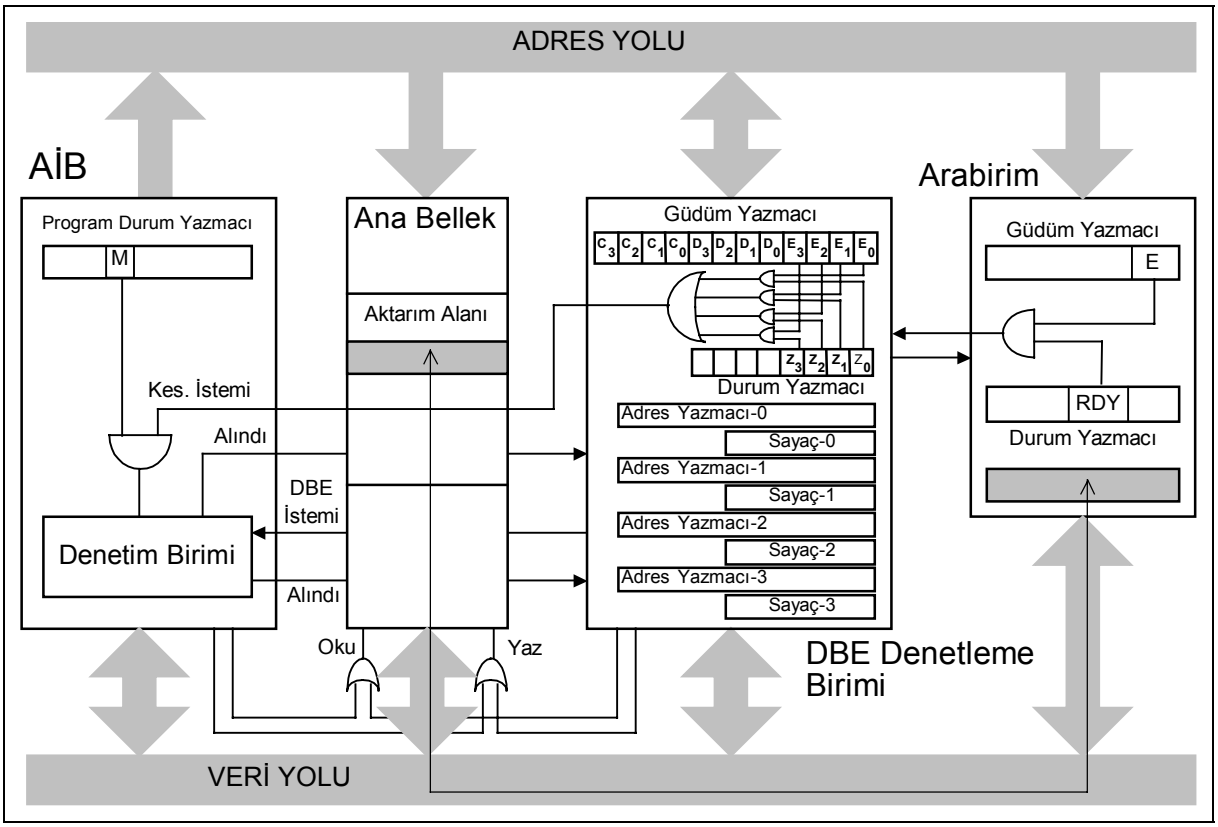
oku, yaz hatları ile ilişkisini kesik tuttuğu varsayılmalıdır. Yol istemi alındısının kurulmasından başlayarak doğrudan bellek erişim denetleme biriminin bir yandan arabirime, giriş/çıkış yastık içeriğini veri yolu üstüne aktarması için aktarım istemi alındısı yolladığı; diğer yandan da, adres yolu üstüne, yazma yapılacak sözcük adresini koyduğu, **3** numaralı geçişle gösterilmiştir.



Çizim 2.35. Doğrudan Bellek Erişimi ile Arabirime Veri Aktarma Zaman Çizeneği

3 numaralı geçişle başlayan bellek erişim döngüsü zaman çizeneği, doğrudan bellek erişim denetleme birimi tarafından gerçekleştirilen belleğe yazma döngüsünü simgelemektedir. Bu döngüyle ilgili imler ile yol istemi, aktarım istemi ve aktarım istemi alındısı imleri ana işlem birimi saat iminden bağımsız olduklarından, saat periyotlarını birbirinden ayıran ve çizeneğin üst kesiminde yer alan kesikli çizgiler bu kesime kadar indirilmemiştir.

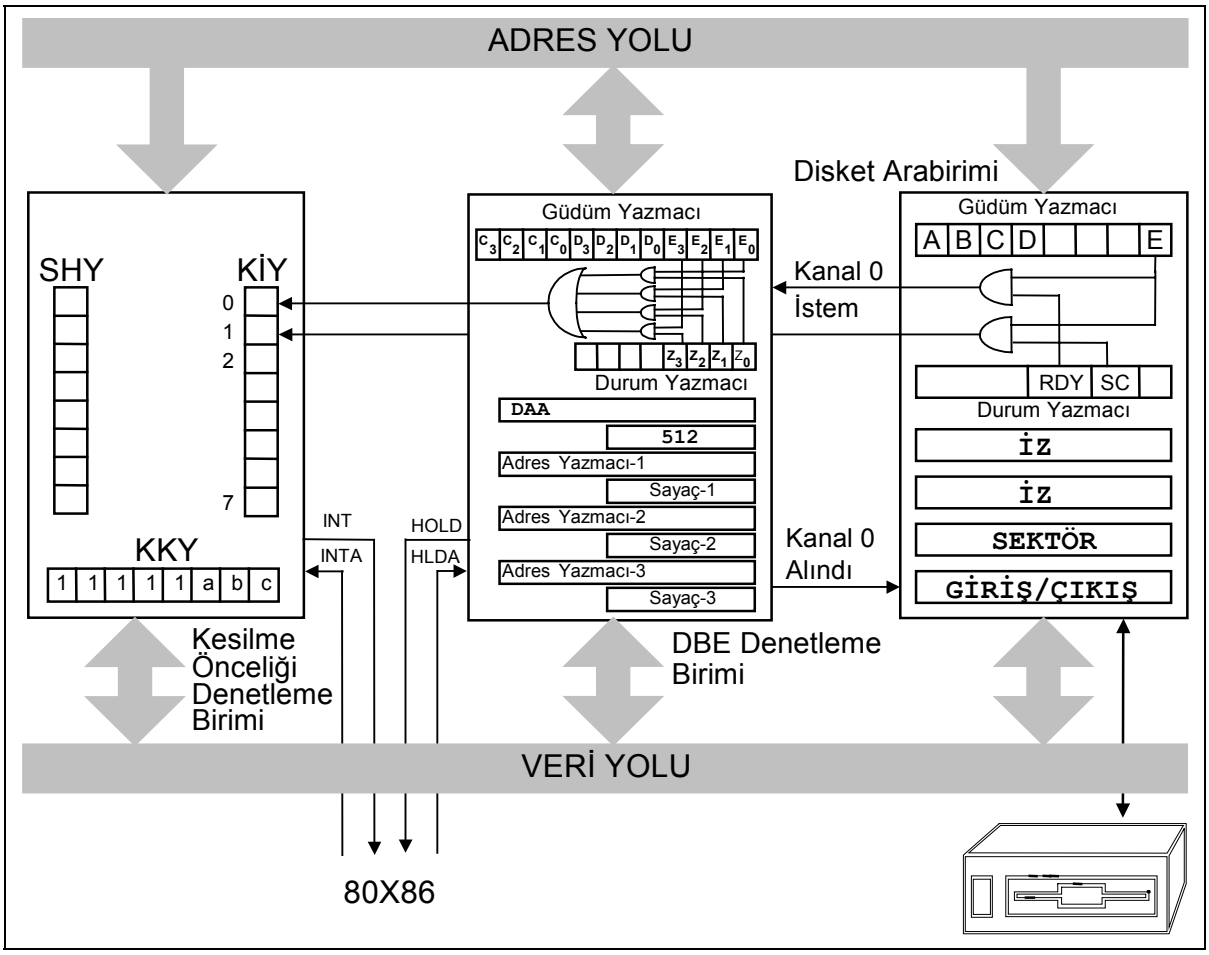
Aktarım istemi alındısının doğrudan bellek erişim denetleme birimi tarafından kurulması üzerine aktarım istemi iminin, arabirim tarafından sıfırlanması **4** numaralı geçişlerle gösterilmiştir. Bellek yazma döngüsünün tamamlanması sonucu aktarım istemi alındısının ve yol istemi iminin, doğrudan bellek erişim denetleme birimi tarafından sıfırlanması **5** numaralı geçişlerle gösterilmiştir. **6** numaralı geçişte, ana işlem biriminin, komut işletimini, sınayageldiği yol istemi girişi üzerinde sıfır düzeyini bulduğu son TH periyodunu izleyen periyottan başlayarak kaldığı noktadan sürdürmesi simgelenmiştir.



Çizim 2.36. Doğrudan Bellek Erişim Denetleme Biriminin Konumu

Çizim 2.36'da doğrudan bellek erişim denetleme biriminin bir bilgisayar sistemi içindeki konumuna ilişkin örnek bir çizim verilmiştir. Bu çizim üzerinde yer alan doğrudan bellek erişim denetleme birimi, Çizim 2.34'te verilen 4 kanallı denetleme biriminin aynısıdır. Doğrudan bellek erişim denetleme birimi içeren bilgisayar sistemleri içinde ana bellek, hem ana işlem birimi hem de denetleme birimi tarafından kullanılabildiğinden, her iki birimden ayrı ayrı gelen oku ve yaz denetim imleri, ana belleğe "ya da" mantığıyla uygulanmıştır. Bunun yanı sıra doğrudan bellek erişim denetleme birimi, hem yardımcı bir işleyici hem de (ana işlem birimi için) bir arabirim gibi işlev görebildiğinden, bu birime bağlı adres yolu iki yönlü olarak gösterilmiştir.

80X86 türü işleyicilerde doğrudan bellek erişim denetleme düzeneği, HOLD ve HLDA *(HoLD Acknowledge)* adlı giriş ve çıkışı içerir. HOLD girişi işleyiciye adres ve veri yolları ile bellek denetim imlerinin taşındığı hatların kullanım isteminin yapıldığı giriştir. Bu giriş işleyici tarafından sınandığında istem belirlenirse istemin belirlendiği anda gerçekleştirilmekte olan bellek erişim döngüsü tamamlanır tamamlanmaz adres ve veri yolları ile RD, WR gibi oku, yaz denetim imi çıkışları ve diğer bellek denetim im çıkışları üzerinde yüksek empedans konumuna geçilir. Çıkışlar üzerinde yüksek empedans konumuna geçiş işleyicinin bu hatları başka birimlerin kullanımına açması anlamına gelir.



Çizim 2.37. DBE Denetleme Düzeneği Kullanım Örneği için Sistem İlke Çizimi

Çıkışlar üzerinde yüksek empedans konumuna geçmeye koşut olarak HLDA imi kurularak, HOLD (yol kullanım) isteminde bulunan birime alındı gönderilir. HOLD girişi üzerinde istem imi varlığını sürdürdüğü sürece ilgili yol ve denetim hatları üzerinde yüksek empedans konumundan çıkılmaz. İşleyici HLDA imini kurduğu periyottan başlayarak HOLD girişini her saat periyodunda sınar. Bu giriş üzerinde HOLD isteminin sıfırlandığını belirlediği ilk periyodu izleyen periyottan başlayarak HLDA çıkışını sıfırlar. Bunu izleyen periyottan başlayarak da komut işletimini kaldığı yerden sürdürür.

2.5.2. DBE Denetleme Düzeneği Kullanım Örneği

Doğrudan bellek erişim denetleme düzeneğine ilişkin genel bilgiler verildikten sonra şimdi, kesilme düzeneği incelenirken örneklenen, bir öbeklik verinin örnek disket biriminden ana belleğe aktarımı, bu kez, doğrudan bellek erişim denetleme birimi ile ele alınacaktır. {İZ,SEKTÖR} ikilisiyle tanımlı 512 baytlık öbeğin, doğrudan bellek erişim yöntemiyle disket sürücüden ana belleğe aktarılabilmesi için:

- sürücü okuma-yazma kafasının ilgili İZ üstüne götürülmesi;
- öbeği oluşturan baytlar, disket arabirimi giriş/çıkış yastığında okunmaya hazır oldukça, RDY biti aracılığıyla, doğrudan bellek erişim denetleme biriminden ardarda aktarım isteminde bulunularak aktarımların yapılması;
- aktarımın tamamlanmasından sonra ilgili aktarım kanalının kapatılması

işlemlerinin gerçekleşmesi gereklidir. Bu işlemlerden sürücü okuma-yazma kafasının İZ üstüne taşınması disket arabirimi güdüm yazmacına ilgili güdüm kodunun (01100001) yazılması ile gerçekleşir. Bu işlem, diğer önbelirleme işlemlerinin de yerine getirildiği ana program içinde başlatılır. Sürücü okuma-yazma kafasının İZ'e ulaşması, arabirim tarafından SC bitinin kurulmasını ve ilgili disk kesilme yordamına sapılmasını sağlar. Bu yordam içinde, arabirim güdüm yazmacına ilgili güdüm kodu (10100001) yazılarak sürücüden SEKTÖR (öbek) okuma işlemi başlatılır ve RDY bitinin bağlı olduğu doğrudan bellek erişim denetleme biriminin *0* numaralı kanalı işlemlere açılarak geri dönülür. Aktarım işlemlerinin tümü, ana işlem birimi devreye girmeden (herhangi bir yordam çalıştırılmadan), doğrudan bellek erişim denetleme birimi tarafından gerçekleştirilir. Bunun sonrasında, Z0 bitinin ürettiği istem ile sapılan kesilme yordamı içinde doğrudan bellek erişim denetleme birimi kapatılarak işlemler son bulur. Bu durumda (İZ,SEKTÖR) ikilisiyle tanımlı öbeğin, doğrudan bellek erişim düzeneği kullanılarak disket sürücüden ana belleğe aktarılmasında, herbiri birer kez çağrılan iki kesilme yordamı ile ilgili ana yordam kesiminin işletimi yeterli olur.

Bu bağlamda, örnek disket arabiriminin, RDY durumunda, görünümü Çizim 2.34'te verilen 4 kanallı doğrudan bellek erişim denetleme biriminin sıfırıncı kanalından aktarım isteminde bulunduğu; denetleme biriminin sayaç sıfırlanması (Z0) durumunda, bir önceki örnekte kullanılan kesilme önceliği denetleme biriminin sıfır numaralı; disk arabiriminin ise SC durumunda *1* numaralı girişten kesilme isteminde bulundukları varsayılacaktır. Doğrudan bellek erişim denetleme birimi öbek tabanlı aktarım moduna ve belleğe yazma yönüne kurulacaktır. Bu bilgi ve varsayımların ışığında sistem ilke çizimi, Çizim 2.37'de verildiği biçimde olacaktır. İlke çizimi verilen bu sistemle uyumlu olarak, (İZ,SEKTÖR) ikilisiyle tanımlı öbeğin, doğrudan bellek erişim düzeneği kullanılarak disket sürücüden ana belleğe aktarılmasını sağlayan örnek sistem programı Çizim 2.38'de yer almıştır.

```
; Disket Birimi DBE ile G/Ç Programlama Örneği
  veri              segment
  diskgüdüm         equ  10h
  diskdurum         equ  11h
  sektörkimliği     equ  12h
  izkimliği         equ  13h
  diskg/ç           equ  15h
  iz                equ  ...
  sektör            equ  ...
  dbegüdüm          equ  20h
  dbeadres0         equ  30h
  dbesayaç0         equ  32h
  dbetür            equ  0f8h
  disktür           equ  0f9h
  izegit            equ  01100001B
  okumayabaşla      equ  10100001B
  sıfırıncıbit      equ  11111110B
  birincibit        equ  11111101B
  açık              equ  0000000000000001B
  dbekanal0         equ  0000000100010000B
  kapalı            equ  1111111111111110B
  sürenhiz          equ  0b0h
  daa               db   512 dup(?)
  daaboyu           dw   512
  veri              ends
  yığıt             segment stack
                    db   100 dup(?)
  yığıt-başı        equ this word
  yığıt             ends
  kod               segment
                    assume cs:kod, ds:veri, ss:yığıt
  program:          cli
; Kesim Yazmaçlarının günlenmesi
                    mov  ax, veri
                    mov  ds, ax
                    mov  ax, yığıt
                    mov  ss, ax
                    mov  sp, offset yığıt-başı
; kesilme vektörünün günlenmesi
                    xor  ax, ax
                    mov  es, ax
                    mov  ax, cs
                    mov  bx, disktür*4
                    mov  word ptr es:[bx], offset disk-kes
                    mov  es:[bx+2], ax
                    mov  bx, dbetür*4
                    mov  word ptr es:[bx], offset dbe-kes
                    mov  es:[bx+2], ax
```

Çizim 2.38. Disket Birimi DBE ile G/Ç Programlama Örneği

```
; Disket Okuma-yazma kafasının İZ'e taşınması
                  mov   ax, iz
                  out   izkimliği, ax
                  mov   al, sektör
                  out   sektörkimliği, al
                  mov   al, izegit
                  out   diskgüdüm, al
; dbe kanal 0'ın kurulması
                  mov   ax, dbekanal0
                  out   dbegüdüm, ax
                  mov   ax, offset daa
                  out   dbeadres0, ax
                  mov   ax, daaboyu
                  out   dbesayaç0, ax
                  sti
; Diğer işlemler
  dön:            jmp   dön                ; Benzetim
.....................................................
  disk-kes:
                  push  ax
                  sti
; dbe kanal0'ın işlemlere açılması
                  in    ax, dbegüdüm
                  or    ax, açık
                  out   dbegüdüm, ax
; disket birimine okumaya başla komutunun yollanması
                  mov   al, okumayabaşla
                  out   diskgüdüm, al
                  cli
                  in    al, sürenhiz       ;süren hizmet
                  and   al, birincibit     ;yazmacı 1. bit
                  out   sürenhiz, al       ;sıfırlama
                  pop   ax
                  iret
  dbe-kes:
                  push  ax
                  in    ax, dbegüdüm       ;DBE birimini
                  and   ax, kapalı         ;işlemlere
                  out   dbegüdüm, ax       ;kapama
                  in    al, sürenhiz       ;süren hizmet
                  and   al, sıfırıncıbit   ;yazmacı 0. bit
                  out   sürenhiz, al       ;sıfırlama
                  pop   ax
                  iret

  kod             ends
                  end   program
```

Çizim 2.38. Disket Birimi DBE ile G/Ç Programlama Örneği (devam)

Çizim 2.38'de verilen programın, Çizim 2.27, Çizim 2.29 ve Çizim 2.33'de verilen program örnekleriyle karşılaştırılması sonucunda, kesilme önceliği denetleme birimi ile doğrudan bellek erişim denetleme biriminin, arabirim - ana bellek arası veri aktarımlarında ana işlem birimine düşen yükün hafiflemesini; bunun sonucunda sistem programlarının yalınlaşarak kısalmasını sağladıkları kolayca görülebilir. Ayrıca doğrudan bellek erişim denetleme biriminin kullanımıyla, arabirim - ana bellek arası veri aktarımlarının, ana işlem biriminin doğrudan denetimiyle yapılan aktarımlara göre daha hızlı gerçekleştirilebildiği de unutulmamalıdır.

2.6. Giriş / Çıkış Kanalları - Giriş / Çıkış İşleyicileri

Bilgisayar sistemlerinde, önceleri ana işlem birimi tarafından, yazılım yoluyla yerine getirilen işlevler, teknolojik gelişmeye paralel olarak, ana işlem birimiyle işbirliği yapan çok sayıda özel donanım tarafından yerine getirilir olmuştur. Giriş/çıkış sistem donanımlarının teknolojik gelişimi bu konuda verilebilecek en anlamlı örneği oluşturur. Seçmeli programlama yönteminden başlayarak doğrudan bellek erişim denetleme düzeneğini kullanan yönteme gelinceğe değin izlenen çizgide, giriş/çıkış işlevlerinin yerine getirilmesinde ağırlığın nasıl ana işlem biriminden özelleşmiş donanım birimlerine kaydığını açıkça görmek olanaklıdır.

Bu bağlamda, sözkonusu gelişim adımlarından ilki giriş/çıkış sürücülerinin denetiminin ana işlem biriminden arabirimlere geçmesidir. Bilindiği üzere giriş/çıkış arabirimleri ana işlem birimi tarafından bir dizi giriş/çıkış kapısı olarak algılanan ve güdüm yazmaçlarının programlanması yoluyla bağlı oldukları sürücüler üzerinde işlem yapılabilen birimlerdir. Giriş/çıkış sürücülerinin denetimi, tümüyle arabirimlerinin yükümlülüğündedir. Gelişim adımlarından ikincisi arabirimlerin programlanmasında, kesilme uyarı düzeneği ve bu düzenekle ilgili öncelik denetleme birimlerinin kullanımıdır. Kesilme uyarı düzeneği ve öncelik denetleme birimi aracılığıyla, giriş/çıkış sürücüleri ile sistem arası veri aktarım işlemlerinin ana işlem birimine yüklediği yük oldukça azalmakta ve bu yolla bu birimin kullanım verimliliği önemli ölçüde artırılabilmektedir. Tek iş düzeninden çok iş düzenine geçiş kesilme düzeneği ile olanaklı olmuştur. Doğrudan bellek erişim denetleme birimlerinin kullanımı ana işlem birimini, arabirim - ana bellek arası veri aktarımlarının sorumluluğundan da kurtarmıştır. Bu yolla ana işlem birimi, giriş/çıkış işlemlerinin başında ve sonunda devreye girmekle yetinen bir birim kimliği kazanmıştır.

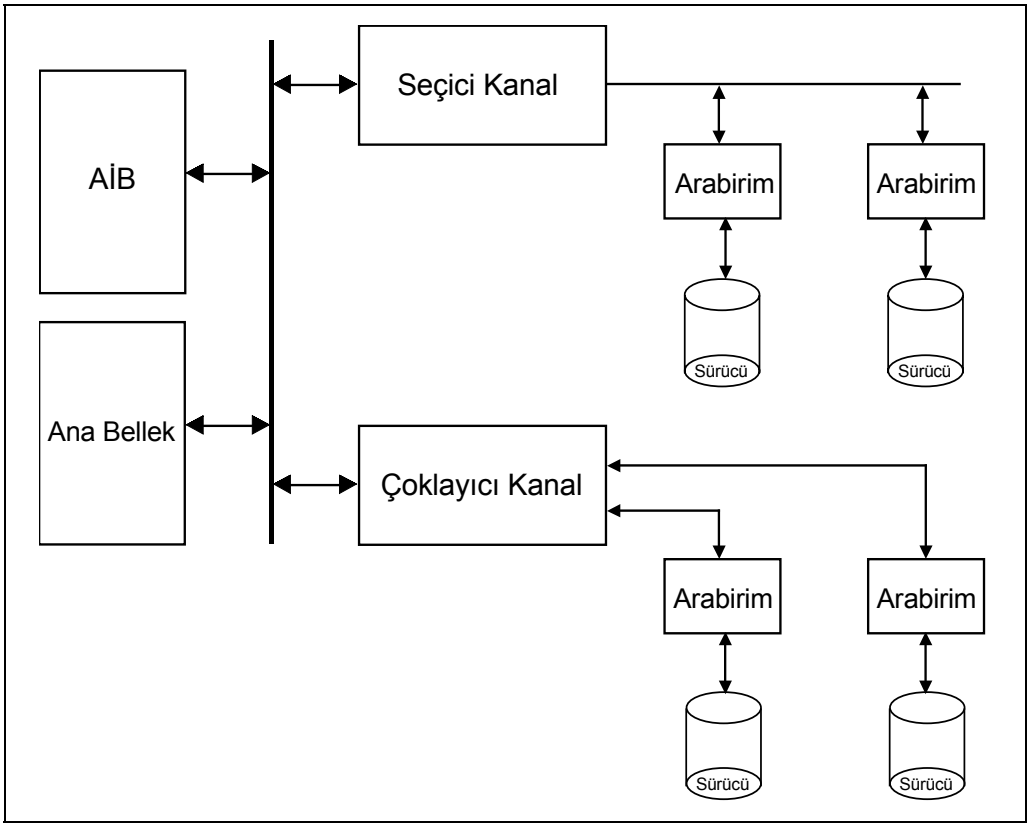
Ana işlem biriminin giriş/çıkış işlemlerinden tümüyle soyutlanması:

- giriş/çıkış işleyicileri ve
- giriş/çıkış kanalları

adlı birimlerin kullanılmasıyla gerçekleşir. Giriş/çıkış işleyicisinden söz edildiğinde, genelde, bilgisayar sistemi içinde kimi giriş/çıkış arabirimlerinin bağlı olduğu, yerel bir bellekle donatılmış, ana işlem birimi ile eşgüdüm içinde ancak bağımsız çalışabilen, genel amaçlı yardımcı bir işleyici akla gelir. Bir bilgisayar sisteminde terminal sürücüler, örneğin bir mikroişleyicinin denetimine bırakılmış terminal arabirimlerine bağlı olarak çalışabilirler. Terminal sürücülerinden yapılan veri girişleri, damga yankılama işlemleri gibi işlemler bu mikroişleyicinin yükümlülüğünde gerçekleşir.

Yerel bellekte saklanan veriler *enter, new line. return* gibi adlarla anılan damgaların girilmesi ile doğrudan, mikroişleyici yerel belleğinden sistem ana belleğine aktarılır. Ana işlem birimi ancak bu aşamada uyarılır.

Giriş/çıkış kanalları, çoğu kez doğrudan bellek erişim denetleme birimlerinin gelişmiş türleridir. Giriş/çıkış kanalları ana bellekte tutulan özel giriş/çıkış programlarını, ana işlem biriminin istemi üzerine işletebilen özel amaçlı yardımcı işleyicilerdir. Ana bellekte tutulan özel giriş/çıkış programları; aktarım yapılacak sürücüler, ana bellek aktarım alanları, öncelik, hata durumunda izlenecek yol gibi tanımları içerirler. Kanallar giriş/çıkış işlemlerini, başka bir deyişle ana bellekten sürücülere, sürücülerden ana belleğe veri aktarımlarını bu programları, ana işlem biriminin istemi üzerine çalıştırarak gerçekleştirirler. Yukarıda tanımlanan giriş/çıkış işleyicileri de, yerel bellekleri ile ana bellek arasındaki veri aktarımlarında kanallardan yararlanabilirler.



Çizim 2.39. Giriş/Çıkış Kanal Türleri

Verilen bu açıklamalarda giriş/çıkış işleyicileri ile giriş/çıkış kanalları tanım olarak ayrıştırılmıştır. Ancak yardımcı işleyici nitelikleri dolayısıyla, kanallar da, giriş/çıkış işleyicileri olarak nitelenebilmektedir. Kanal, doğrudan bellek erişim denetleme biriminin gelişmiş bir biçimi olarak düşünüldüğünden daha çok, disk, mıknatıslı şerit gibi hızlı giriş/çıkış birimlerinden veri aktarımında gerekli olan bir donanım birimi olarak algılanmalıdır. Ancak günümüz bilgisayar sistemlerinde kanallar, ana bellek ile tüm giriş/çıkış birimleri arasındaki aktarımlardan sorumlu donanım katmanını

oluştururlar. Giriş/çıkış işleyicileri, burada verilen tanımları itibarıyla, ana işlem biriminden sürücüye uzanan donanımsal sıradüzen içinde kanaldan sonra gelen birleşenlerdir.

Giriş/çıkış kanallarından yaygın olarak kullanılanları iki türe ayrılır. Bunlar:

- Seçici Kanal
- Çoklayıcı Kanal

türleridir. Birden çok hızlı giriş/çıkış arabirimine hizmet verebilen seçici kanallar bu hizmeti, aynı anda tek bir arabirime sağlayabilirler. Kanal, hizmet vereceği arabirimi bunların istem durumunu sınayarak seçer. Bir arabirimle ilgili aktarım işlemleri tümüyle tamamlanmadan diğer arabirimlerin istemleri, dolayısıyla işlemleri ele alınamaz.

Çoklayıcı kanallar aynı anda birden çok arabirime hizmet verebilen kanallardır. Bu kanallar kendi içlerinde:

- bayt çoklayıcı ve
- öbek çoklayıcı

kanallar olarak ikiye ayrılır. Bayt çoklayıcı kanal, damga tabanlı birden çok arabirime aynı anda damga aktarabilir. Damga tabanlı 3 değişik arabirime aktarılacak $a_1a_2a_3a_4a_5a_6a_7a_8a_9$; $b_1b_2b_3b_4b_5b_6b_7b_8b_9$; $c_1c_2c_3c_4c_5c_6c_7c_8c_9$ damga dizgileri kanaldan $a_1a_2b_1c_1a_3c_2b_2$..... sırasında, karma biçimde akabilir. Ancak her damga ilgili arabirime ya da arabirim yastık alanına anahtarlanır. Öbek çoklayıcı kanallar ise bu çalışma ilkesini öbek tabanlı arabirimlere, öbekler için uygular. Çizim 2.39'da seçici ve çoklayıcı kanalların sistem içindeki yerleri ve arabirimlere göre konumları gösterilmiştir.

3. BÖLÜM

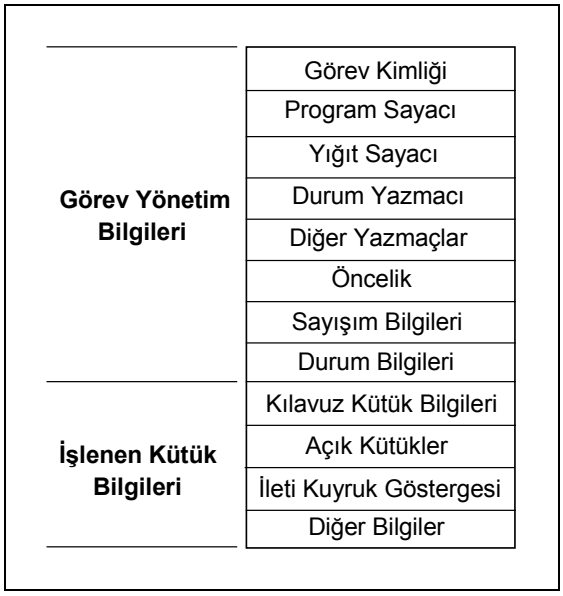
İ Ş L E T İ M S İ S T E M L E R İ

GÖREV YÖNETİMİ

Bilgisayar sistemlerinin verimli kullanımı, ana işlem birimi, ana bellek ve giriş/çıkış birimleri gibi kaynakların, programlar arasında paylaşılmasını gerektirir. Ana işlem biriminin paylaşılması, programların işletimlerinin birlikte sürdürülmesi yoluyla sağlanır. Ana işlem biriminin bir programın işletimini, ileride sürdürmek üzere, bırakıp diğer bir programın işletimine geçmesi anahtarlanma olarak adlandırılır. Ana işlem biriminin paylaşımı, bu durumda, değişik programların bu birime belirli bir sıra ile anahtarlanması yoluyla gerçekleştirilir. Programların ana işlem birimine hangi sıra ve kurallar çerçevesinde anahtarlanacağı, ana işlem biriminin yönetimi kapsamında ele alınır. Programlar işletimleri sırasında görev olarak adlandırılırlar. Ana işlem biriminin yönetimine, paylaşılan kaynak yerine, bu kaynağı paylaşan görevler yönünden bakılarak görev yönetimi de denir.

Programların, çalışabilmek için ana işlem birimine anahtarlanmaları zorunludur. Ancak bunun yanı sıra ana belleğe yüklenmeleri, işletimleri sırasında gereksinim duyacakları giriş/çıkış türü kaynakları, diğer programlarla yarışarak elde etmeleri de gerekir. Programlar, ana işlem birimi dışındaki, ana bellek, giriş/çıkış birimleri gibi kaynakları, yine ana işlem birimi aracılığıyla tüketebilirler. Programlarının, ana belleğe yüklenebilmeleri, giriş/çıkış birimlerinden okuma-yazma yapabilmeleri, işletimleri sırasında gereksinim duyabilecekleri ek bellek alanlarını elde edebilmeleri, işletim sistemi içinde yer alan, ilgili yönetici ve sürücü görevlerin ana işlem birimine anahtarlanarak çalıştırılmaları sonucu gerçekleşebilir. Buradan, bir bilgisayar sisteminde paylaşılan kaynaklar arasında en önemli kaynağın ana işlem birimi olduğu söylenir.

İzleyen kesimde, işletim sistemlerinde ana işlem biriminin yönetiminin nasıl ele alındığı görev yönetimi adı altında incelenecektir. Bu amaçla, önce çok önemli bir kavram olarak ortaya çıkan görev kavramı açıklanacak ve örneklenecektir. Bunu izleyen kesimde görevlerin işletim aşamaları ya da bulundukları durumlar, durumlar arası geçişleri sağlayan sistem komutları ya da sistem çağrıları açıklanacaktır. Bundan sonra görev yönetimi, görev yönetiminin diğer işletim sistemi yönetim kesimleriyle ilişkileri ve görev yönetiminin gerçekleştiriminde kullanılan algoritmalar incelenecektir.



Çizim 3.1. Görev İskeleti Örneği

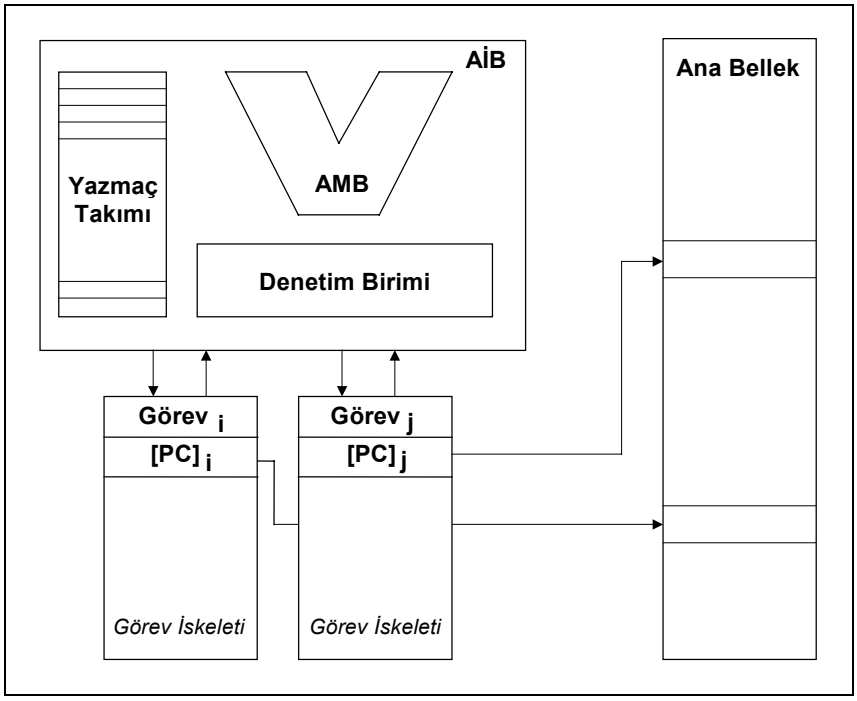
3.1. Görevin Tanımı

Görev bir programın işletimi sırasında aldığı addır. İşletilen her program için en az bir görev tanımlanır. Çok kullanıcılı bilgisayar sistemlerinde yer alan metin düzenleyici gibi, aynı anda birden çok kullanıcı tarafından yararlanılan programların her kullanıcı için ayrı ayrı sürdürülen her bir işletimine ayrı bir görev karşılık gelir. Program, içerdiği komutlar yönünden biricik iken değişik kullanıcılar için, değişik veri takımları ile yapılan birden çok işletim, başka bir deyişle birden çok görev bulunabilir. Görev, program kavramına işletim boyutunu da katan daha geniş bir kavramdır. Program durgun komut dizisini tanımlarken görev bu komut dizisinin devingen işletimine karşılık gelir.

Bilindiği üzere ana işlem biriminin verimli kullanımı birden çok işin (programın), koşut işletimiyle sağlanır. Ana işlem biriminin, koşut işletimi sağlamak üzere bir işletimi bırakıp diğer bir işletime geçmesi belirli önlemler alınmadan yapılamaz. Yarım bırakılan bir işletimin, tutarlı bir biçimde, kalınan noktadan sürdürülebilmesi, işletimin bırakıldığı andaki durum bilgilerinin saklanması yoluyla sağlanır. Bu nedenle, her görev için işletim sistemi tarafından bir veri yapısı tutulur. Bu veri yapısı, örneğin işletimin hangi komuttan başlayarak sürdürüleceği bilgisini de içeren ana işlem birimi yazmaç

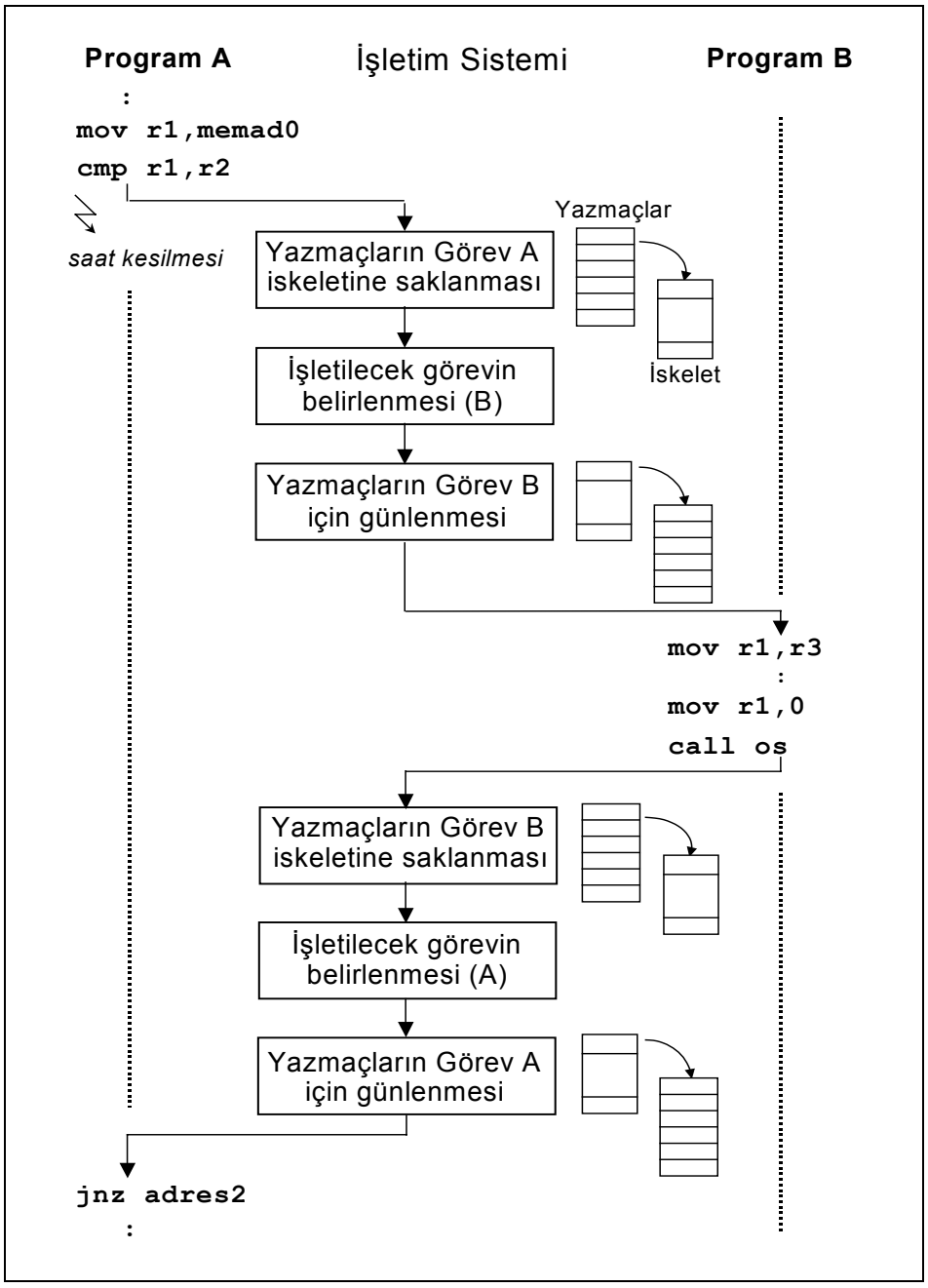
içeriklerini, varsa kullanılan kütüklerle ilgili (kılavuz kütük, açık kütükler gibi) kimi bilgileri içermek durumundadır. Bu yapı, görev denetim öbeği, görev iskeleti gibi adlarla anılır. İzleyen kesimde bu adlandırmalardan iskelet adlandırması kullanılacaktır.

Görev iskeleti, genelde görev kimliği başta olmak üzere, görevin program sayacı, yığıt sayacı, diğer ana işlem birimi yazmaç içeriklerini, durum, öncelik, sayışım ve bellek atama bilgilerini, işlenen kütükler ve bunlarla ilgili kimi gösterge bilgilerini tutar (Çizim 3.1). Görev iskeleti, programların sırayla ana işlem birimine anahtarlanarak çalışmaları sırasında işletim bütünlüğünün korunabilmesi, başka bir deyişle bir çalışmanın diğer bir çalışmayı bozmaması için gerekli tüm bilgileri eksiksiz içermek zorundadır.



Çizim 3.2. Görevlerin Ana İşlem Birimine Anahtarlanması

Birden çok görevin eşanlı işletildiği bir bilgisayar sisteminde, G_i adlı görev işletilirken, G_j adlı görev, sırası gelip ana işlem birimine anahtarlanacağı zaman, önce, o anki ana işlem birimi yazmaç içerikleri, bu yazmaçlar arasında yer alan ve "görev yazmacı" olarak adlandırılan yazmacın, ana bellekte gösterdiği (işletimi kesilen) görev iskeletine saklanır. Bunun sonrasında görev yazmacı, G_j adlı görevin iskeletini gösterecek biçimde, işletim sistemi (görev yönetici) tarafından günlenir. Ana işlem birimi yazmaçları görev yazmacının gösterdiği iskeletteki değerlerle günlenip yeni görevin işletimi başlatılır. Görev yazmacının günlenmesi durumunda, o anki ana işlem birimi yazmaç içeriklerinin, saklanması ve yeni göreve ilişkin değerlerle günlenmesi genellikle ana işlem birimi tarafından otomatik olarak (herhangi bir işletim yordamı çalıştırılmaksızın) gerçekleşir. Bu işleme "görev anahtarlama" işlemi denir.



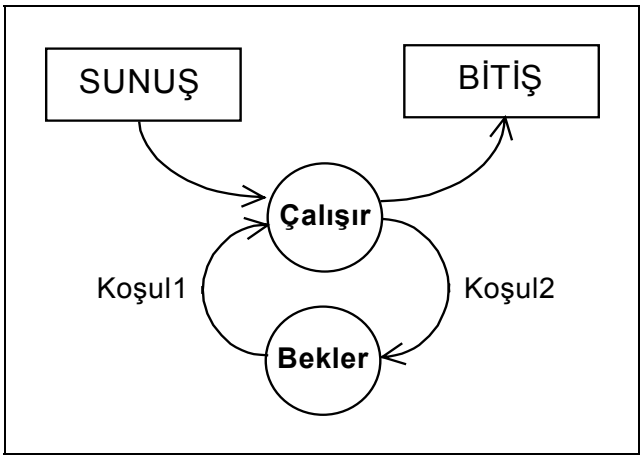
Çizim 3.3. Görev - İşletim Sistemi Etkileşimi

Bu tanımla ilişkili olarak, bir de "bağlam anahtarlama" işlemi vardır. Herhangi bir altyordamın çağrılması ya da kesilme yordamına sapış gibi durumlarda da, yordamlar arası işletim bütünlüğünün korunması amacıyla, görev yazmacı dışındaki ana işlem birimi yazmaçlarının bir kesiminin saklanması ve günlenmesi de sözkonusu olabilir. Bu durumda bağlam anahtarlama işleminden söz edilir. Görev anahtarlama, bağlam anahtarlamaya göre daha geniş, dolayısıyla zaman yönünden daha pahalı bir işlemdir. Aynı görev içinde, görev anahtarlama olmaksızın, yordam çağırma, kesilme yordamına

sapma gibi nedenlerle bir dizi bağlam anahtarlama işlemi gerçekleşebilir. Buradan, görev anahtarlama ve bağlam anahtarlama kavramları arasında ayrım gözetmenin zorunlu olduğu görülür.

3.2. Görevlerin İşletim Süresince Bulunduğu Durumlar

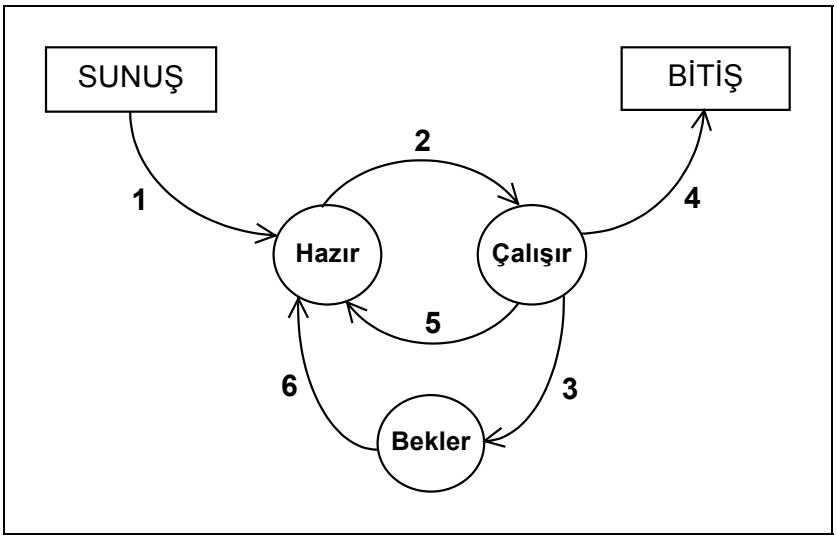
Görev bir programın işletimine verilen addır. Bir program, işletimi sonlanana değin değişik işletim evrelerinden geçer. Bunlar, ana işlem biriminin kullanımı yönünden kabaca çalışma ve bekleme evreleridir. Bu evrelere durum adı verilir. Buradan görevlerin, herhangi bir anda çalışır ve bekler durumlarında bulunduğu söylenir (Çizim 3.4).



Çizim 3.4. Yalın Görev Durum Çizeneği

Görevlerin, işletimleri sırasında bulunduğu durumlar, durum çizenekleri ile gösterilir. Durumların herbiri, bu çizenekler üzerinde bir çember ile simgelenir. Görevin bir durumdan diğer bir duruma geçmesi oklar ile ifade edilir ve geçiş olarak adlandırılır. Bir durumdan diğer bir duruma geçebilmenin gerektirdiği koşul, bu geçişi temsil eden okun üstüne yazılabilir. Görevlerin işletimleri süresince bulunabildikleri tüm durumlar ile bu durumlar arası geçişlerden oluşan çizimlere görev durum çizeneği denir. Çizim 3.4'te gösterilen görev durum çizeneğinde yer alan çalışır durumu, görevin ana işlem birimini kullanıyor olmasını, bekler durumu ise görevin ana işlem birimini yeniden kullanabilmek üzere, başlattığı bir işlemin (örneğin diskten okuma işleminin) sonlanmasını beklemesi anlamına gelmektedir.

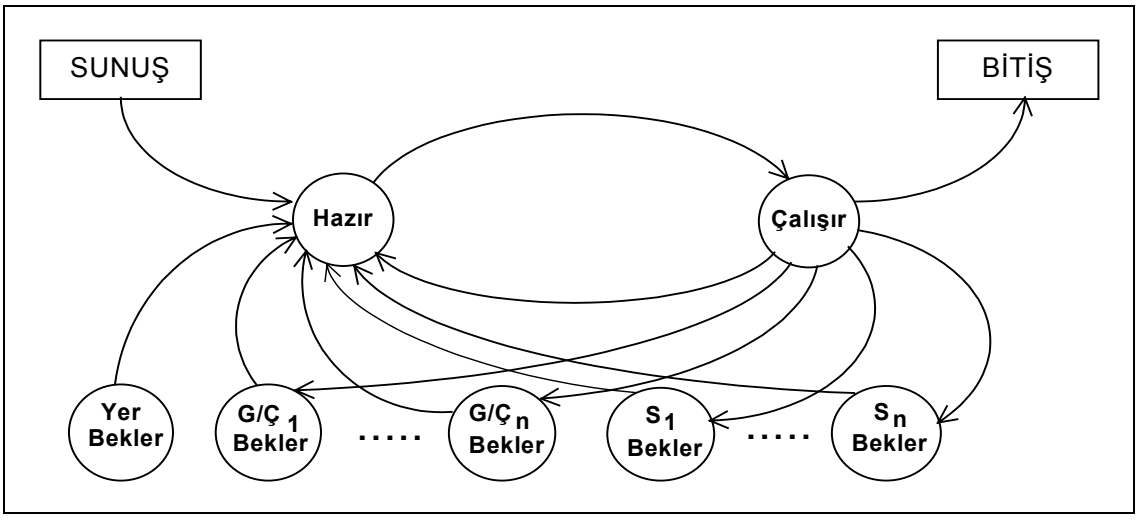
Oluşumunu beklediği bir koşulun gerçekleşmesi, bir göreve, ana işlem birimine anahtarlanarak çalışabilir ya da ana işlem birimine anahtarlanmak üzere seçilebilir görev özelliğini kazandırır. Ana işlem birimini kullanabilir duruma gelen görevler kısaca hazır görevler olarak anılırlar. Bu durumda, Çizim 3.4'te bekler olarak tanımlanan görevler, daha ayrıntılı bir biçimde bir koşulun oluşmasını bekleyen görevler ve ana işlem birimine anahtarlanmak üzere hazır bekleyen görevler olarak ayrıştırılabilirler (Çizim 3.5).



Çizim 3.5. Görev Durum Çizeneği

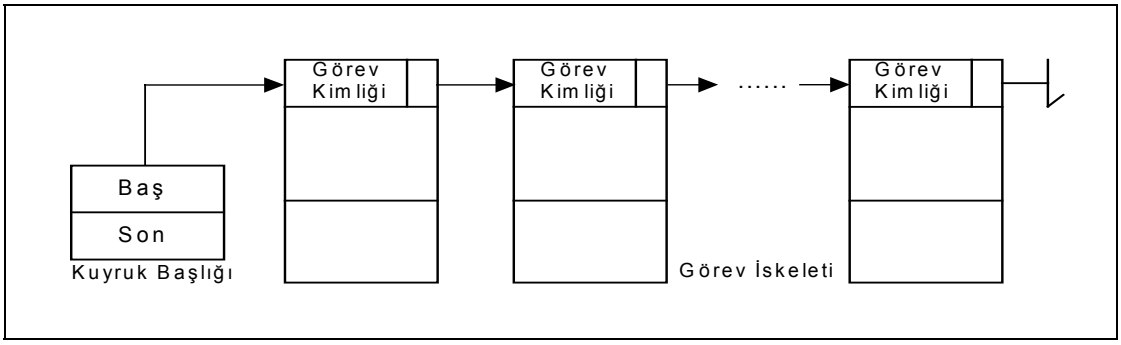
Programlar ya da daha genel olarak işler, bilgisayar sistemine çalıştırılmak üzere sunulduklarında, iskeletleri çatılarak görev ya da görevlere dönüştürülürler. Tanımları yapılan bu görevler hazır görev sınıfına girerler **(1)**. Hazır görevlerden biri, Görev Yönetimi kesiminde ayrıntılı olarak açıklanacak kıstaslara dayanılarak, Görev Yönetici diye adlandırılan özel bir sistem görevi aracılığıyla ana işlem birimine anahtarlanır ve çalışır göreve dönüşür **(2)**. Çalışmakta olan bir görev değişik nedenlerle bu özelliğini yitirebilir. Çalışırken giriş/çıkış işlemi başlatan bir görev, bu işlem tümüyle sonlanmadan işletimine devam edemeyeceğinden bekler görev durumuna geçer ve ana işlem birimini kaybeder (ana işlem birimini bırakır) **(3)**. Bunun gibi, Birlikte Çalışan Görevler bölümünde incelenecek *p, wait, down* gibi adlarla anılan, semafor olarak adlandırılan özel değişkenler üzerinde işlem yapan zamanuyumlama işleçlerini çalıştıran görevler de, ana işlem birimini bırakıp bekler durumuna geçebilirler **(3)**. Çalışmakta olan bir görev, işletiminin bitmesi durumunda ana işlem biriminin yanı sıra genelde sistem içindeki tanımını ve varlığını da yitirir **(4)**. Görevler ana işlem birimini kendilerinden kaynaklanmayan nedenlerle de bırakmak zorunda kalabilirler. Etkileşimli işlemin uygulandığı bir bilgisayar sisteminde görevlerin ana işlem birimini belirli süreler içinde, sırayla kullanmaları istenir. Ana işlem birimine anahtarlanan bir görev, kendine ayrılan süre sona erdiğinde, yeniden hazır görev durumuna getirilir **(5)**. Giriş/çıkış istemi, zamanuyumlama gibi nedenlerle bekler duruma geçen görevler, istemin yerine gelmesi, zamanuyumlamanın gerçekleşmesi *v, signal, up* gibi adlarla anılan, semafor olarak adlandırılan özel değişkenler üzerinde işlem yapan zamanuyumlama işleçlerinin çalıştırılması) gibi hallerde de yine çalışmaya hazır görev durumuna dönerler **(6)**. Çizim 3.5'te verilen görev durum çizeneğinde yer alan durum ve geçişlerle ilgili açıklamalardan da anlaşılacağı üzere, bekler görevlerle ilgili olarak, beklemenin niteliğine göre bekler durumunun giriş/çıkış bekler, zamanuyumlama bekler gibi çok sayıda alt duruma ayrıştırılabileceği görülür. Bu ayrıştırma, gerektiğinde bilgisayar sisteminde yer alan değişik giriş/çıkış sürücülerinin, zamanuyumlama değişkenlerinin (semaforların) adları ile anılan durumlara kadar indirilebilir.

Bunun yanı sıra kimi görevler, işletimleri sonlanmadığı halde, önceliklerinin düşüklüğü gereği, Ana Belleğin Yönetimi adlı bölümde açıklanacağı üzere, bellekte işgal ettikleri alanları başka görevlerin program kodlarına yer açmak üzere yitirirler. Örneğin görüntü bellek yönetiminin uygulanmadığı bir sistemde, giriş/çıkış isteminde bulunarak ana işlem birimini kullanmayı sürdüremeyecek düşük öncelikli bir görev ana bellekte gereksiz yer işgal etmemesi amacıyla bellekten çıkarılır. Bu göreve ilişkin bellekten çıkarılan alanlar, işletime devam edileceği zaman ana belleğe yeniden yüklenmek üzere diskte saklanırlar. İşletilebilmek için ana işlem biriminin yanı sıra, bir de ana bellekte yer açılmasını bekleyen bu tür görevler için de özel bir bekler durumu tanımlanabilir. Bunlara benzer ayrıntıların durum çizeneğine yansıtılması durumunda Çizim 3.6'daki daha ayrıntılı görev durum çizeneği elde edilir. Çizim üzerinde Yer Bekler adıyla yer alan bu duruma görevler, çoğunlukla bekler (G/Ç, Zamanuyumlama bekler) durumlarından geçebilirler. Bellekte kendilerine yeniden yer sağlanan görevler, bu arada bekler durumları da son bulmuşsa hazır görev niteliklerini yeniden kazanırlar. Çizim 3.6'da, Yer Bekler durumundan hazır görev durumuna geçiş belirtilmiş ancak bu duruma gelişi simgeleyen geçişler, çizeneği karmaşıklaştırmamak için gösterilmemiştir.



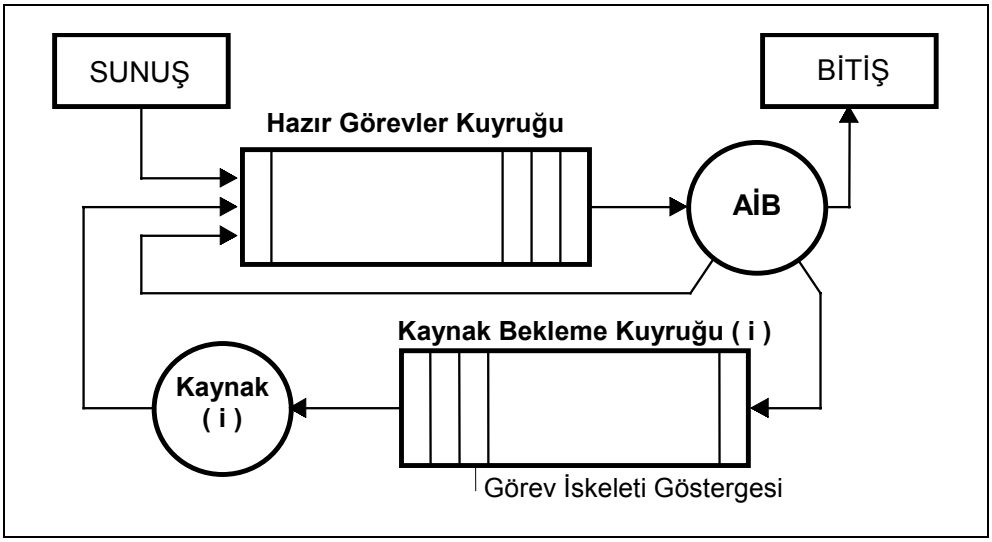
Çizim 3.6. Ayrıntılı Görev Durum Çizeneği

Görevlerin işletimleri boyunca bulunabildikleri evreleri belirtmek için kullanılan durum çizeneği güçlü bir gösterim yöntemidir. Ancak varolan tek gösterim yöntemi değildir. Görevler, işletimleri sonlanana değin ana işlem birimi, ana bellek ve giriş/çıkış birimlerinden oluşan donanımsal kaynakları diğer görevlerle paylaşmak zorundadırlar. Bir bilgisayar sisteminde yer alan kaynak sayısı işletilen görev sayısından çok daha az olduğundan görevler bu kaynakları kullanırken (örneğin ana işlem birimine anahtarlanabilmek, diskte okuma - yazma yapabilmek amacıyla) belirli öncelik kıstaslarına göre sıralanmak zorundadırlar. Görevlerin kaynakları kullanmak üzere kurdukları sıralara kuyruklar denir (Çizim 3.7). Bu kuyruklar gerçekleştirim açısından bakıldığında, genellikle görev iskeletlerinden oluşan bağlı listelerdir. İşletim sistemince, görevler tarafından paylaşılan kaynakların herbiri için; ana işlem birimi kuyruğu, giriş/çıkış birimi kuyruğu, semafor kuyruğu gibi bir kuyruk öngörülür (Çizim 3.8).



Çizim 3.7. Görev Kuyruklarının Gerçekleştirimi

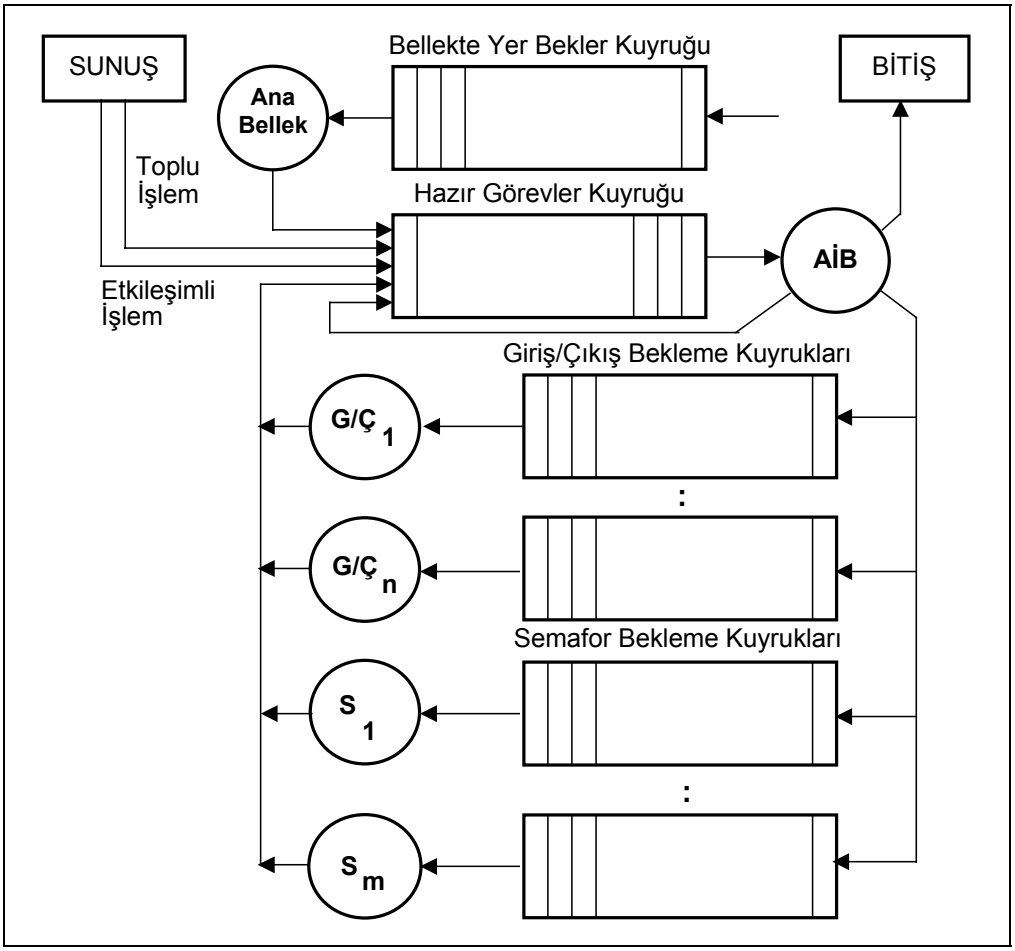
Görevleri, işletimlerinin değişik evrelerine ilişkin olarak, bu kuyruk yapıları içinde kuyruk çizenekleriyle göstermek de olanaklıdır. Bu yapıldığında ana işlem birimi kuyruğu, görev durum çizeneğinde hazır olarak anılan görevleri tuttuğundan hazır görevler kuyruğunu, giriş/çıkış birimi kuyruğu ise giriş/çıkış bekler durumuna ilişkin görevleri kapsadığından giriş/çıkış bekler kuyruğunu oluşturur. Buradan, görev işletim evrelerinin temsiline ilişkin görev durum çizeneği ile görev kuyruk çizeneğinin özdeş gösterim biçimleri olduğu söylenebilir.



Çizim 3.8. Kaynak Paylaşımının Kuyruklarla Gösterimi

Kuyruk çizeneklerinde paylaşılan kaynaklar bir çemberle, bu kaynakları kullanmak üzere bekleyen görev kuyrukları ise bir dikdörtgenle simgelenir. Çizim 3.8'den de anlaşılacağı gibi, ana işlem birimi üzerinde çalışırken giriş/çıkış işlemi başlatan bir görev, (görev iskeleti göstergesi kullanılarak) giriş/çıkış işleminin yapılacağı birimin bekleme kuyruğuna (örneğin disk okuma-yazma kuyruğuna) eklenir. İşlemi sonlanan görev ise bu kuyruktan silinerek hazır görevler kuyruğuna aktarılır. Buradan bir görevin durum değiştirmesi (Örneğin giriş/çıkış bekler durumundan hazır durumuna geçmesi) ile kuyruk değiştirmesi (örneğin giriş/çıkış bekler kuyruğundan hazır görevler kuyruğuna aktarılması) arasında herhangi bir ayrım olmadığı söylenebilir.

Çizim 3.6'da verilen ayrıntılı görev durum çizeneğine özdeş kuyruk çizeneği Çizim 3.9'da verilmiştir. Görevlerle ilgili işletim komutları, sistem çağrıları ve görev yönetimi incelenirken, daha çok bu gösterim biçimi kullanılacaktır. Çizim 3.9'da, yukarıda açıklandığı üzere paylaşılan kaynaklar bekleme kuyruklarıyla birlikte gösterilmiştir.

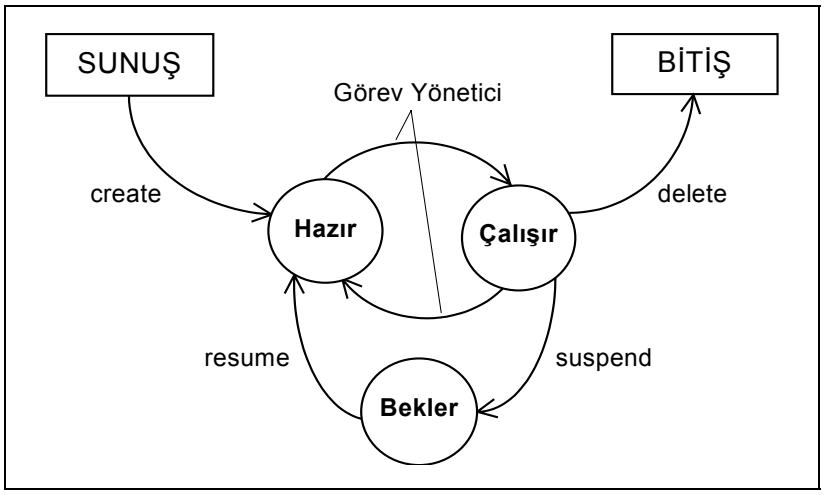


Çizim 3.9. Ayrıntılı Görev Kuyruk Çizeneği

3.3. Görevlerle İlgili Sistem Çağrıları

Sistem çağrıları, gerek derleyiciler gerekse sistem programcıları tarafından, işletim sistemine ilişkin yordamları çağırmak ve bu yolla hizmet almak amacıyla programlara yerleştirilen özel komutlardır. Görevlerle ilgili, işletim sistemi çekirdek katmanında yer alan kimi hizmet yordamlarının üst katmanlardan çağrılması özel sistem çağrıları aracılığıyla olur. Bunlar, hem uygulama programları hem de işletim sisteminin kendisi tarafından kullanılırlar. Sistemden sisteme değişiklik gösteriyor olmalarına karşın, görevlerle ilgili sistem çağrılarına örnek oluşturabilen bir alt kümeyi tüm sistemlerde bulmak olanaklıdır. Görevlerin yaratılması, sonlandırılması; işletimlerinin başlatılması, geciktirilmesi ya da kesilmesi; öznitelik bilgilerinin günlenmesi ve sorgulanması gibi amaçlarla kullanılan bu sistem çağrıları *create, delete (kill), suspend, resume,*

delay, change-priority, change-attributes gibi İngilizce adlarla anılırlar (Çizim 3.10). Bu çağrılar, birebir işletim sistemi komutlarına karşı geldiklerinde komut olarak da anılırlar[11].



Çizim 3.10. Sistem Çağrıları ve Görev Durum Çizeneği

Görevlerle ilgili sistem çağrı ya da komutları genelde argümanlarla kullanılır. Görev kimliği argümanlardan birini oluşturur. Görev kimliğinin yanı sıra, çağrılan işleve aktarılmak üzere süre, öznitelik, öncelik gibi başka argümanlar da bulunabilir. Örneğin yeni bir görevin yaratılmasında kullanılan *create (görev-kimliği, öznitelik-alanı)* komutu, yaratılacak görevin kimliği ile bu göreve ilişkin iskeletin günlenmesinde kullanılacak özel bilgilere (öznitelik bilgilerine) göstergeyi argüman olarak taşımak durumundadır. *create* komutunu çalıştıran görev (başka bir deyişle ata görev), bu işlevi yerine getirecek işletim sistemi kesiminden, yaratılacak görevin (oğul görevin) kimliğini alır. İlgili işletim sistemi çekirdek katman kesimi, boş iskeletler listesinden bir iskeleti, yeni bir görev kimliğiyle ilişkilendirir. İskelet içeriği öncelik ve ayrıcalık düzeyi, bellek boyu, yığıt sığası, erişim hakları, çalıştırılacak program gibi öznitelik bilgileri ve sistemce belirlenen diğer ek bilgilerle günlenerek hazır görevler kuyruğuna bağlanır. Böylece *create* komutunun gereği yerine getirilmiş olur. Bunun gibi *delete (görev-kimliği)* komutu da kimliği verilen görevin iskeletini boş iskeletler listesine ekleyerek görevin sistem içindeki varlığına son vermek için kullanılır.

suspend(görev-kimliği,kuyruk-kimliği) ve *resume(görev-kimliği,kuyruk-kimliği)* sistem çağrıları, sırasıyla, kimliği verilen görevi, gene kimliği verilen bekler türü bir

[11] Bilindiği gibi, bilgisayar sistemlerimde ana bellek, işletim sistemi ve kullanıcı programları tarafından paylaşılır. Kullanıcı programlarının işletim sistemine ayrılan kesime, giriş / çıkış işlemleri gibi kimi istisnalar dışında erişim yapmaları, işletim bütünlüğünü korumak için engellenir. İşletim sistemine, istisnalar dışında, daha geniş haklarla erişim yapabilen programcılar sistem programcılarıdır. Sistem programcıları diğer programcıların yararlandığı işletim ortamını kurmaya yarayan işlemleri yerine getiren, bu nedenle de işletim sistemi üzerinde günleme yapabilen programcılardır. Programcılar arasında yapılan *uygulama programcısı / sistem programcısı* ayrımı, daha geniş bir bağlamda, *sıradan* ve *ayrıcalıklı kullanıcılar* olarak, kullanıcılar arasında yapılır.

kuyruğa bağlamak ve bağlı bulunduğu bekler türü kuyruktan hazır görevler kuyruğuna aktarmak amacıyla kullanılan çağrılardır. *suspend*, giriş/çıkış isteminde bulunma, görevler arası zamanuyumlama gibi durumlarda görevin kendisi tarafından çalıştırılır. *suspend* çağrısıyla işletimi durdurulan bir görevin yeniden işletime alınabilmesi, o görevle ilgili olarak *resume* çağrısının çalıştırılmasını gerektirir. *suspend* çağrısının tersine *resume* çağrısı, ilgili görevin dışındaki görevler tarafından çalıştırılabilir.

delay(süre) komutu *suspend* komutuna benzer bir işleve sahiptir. Bu komut çalıştırıldığında ilgili görevin, argüman olarak verilen süre kadar bekler durumunda kalması sağlanır. Bu yolla, görevin işletimi belirli bir süre geciktirilir. Zaman aşımlarının denetimi, gerçek zamanlı uygulamalarda zamana bağımlı etkinliklerin pogramlanması gibi uygulamalar bu komutun kullanım örnekleri arasında sayılabilir. *UNIX* işletim sisteminde, *delay* komutu olarak `sleep(n)` komutu kullanılmaktadır. `n`, genelde saniye türünden beklenecek süreyi belirlemektedir.

change-priority(görev-kimliği), change-attributes(görev-kimliği,öznitelik-alanı) gibi komutlar, kimliği verilen görevlerin öncelik ve öznitelik bilgilerinin gözlenmesi ve günlenmesi amacıyla kullanılırlar. *change-priority* komutu, ilgili görevin işletim önceliğini gözlemenin yanı sıra günlemeye de olanak sağlayan bir komuttur. Bu komut gerek sistem işletmeni gerekse işletim sistemince, kendiliğinden, örneğin işletimi aşırı gecikmiş görevlerin işletimlerini, dönem dönem hızlandırmak amacıyla da kullanılır.

Görevlerin değişik durumlar ya da kuyruklar arasında geçişlerini sağlamak amacıyla gerek işletim sisteminin kendisi, gerekse işletmen ya da sistem programcıları tarafından kullanılmak üzere öngörülen ve yukarıda sıralanan sistem çağrılarının yanı sıra, işletim sistemlerine özgü başka çağrılar da bulunur. *UNIX* işletim sisteminde *fork* ve *exec* diye adlandırılan özel sistem çağrıları bunlara bir örnektir. Genel olarak *fork* çağrısı (ya da komutu), içinde bulunduğu programı, bu komutu izleyen satırdan başlayıp *join* adlı diğer özel bir komut satırına kadar iki parçaya ayırmak ve parçaların işletimini, biri *fork* komutu tarafından yaratılan oğul görev, diğeri de *fork* komutunu çalıştıran ata görev aracılığıyla, koşut olarak yürütmek için kullanılır. Bu komutlar, birlikte çalışan görevlerin programlanmasına olanak veren programlama dillerinde yer alan (*fork-join, cobegin-coend* gibi) üst düzey komutlara altyapı sağlamayı amaçlar. Ancak *UNIX* işletim sistemi kapsamında *fork* komutu, yeni görevler yaratmak amacıyla, yukarıda açıklanan *create* komutunun yerine kullanılabilen tek komuttur.

UNIX işletim sisteminde yeni bir görev yaratmak amacıyla `fork` ve `exec` sistem çağrı ikilisi kullanılmaktadır. `fork()` sistem çağrısı, bu çağrıyı işleten görevle (kimlik bilgileri dışında) aynı görev iskeletine sahip bir diğer görevi yaratıp hazır görev durumuna getirmek için kullanılmaktadır. Bu bağlamda, `fork()` sistem çağrısını işleten görev, ata, atanın bir kopyası olarak yaratılan görev de oğul görev olarak tanımlanmaktadır. Bu durumda `fork()` sistem çağrısını izleyen program satırları (komutları), biri ata biri de oğul olmak üzere, koşut iki görev tarafından işletilir. Bu nedenle `fork()` sistem çağrısının üreteceği geri dönüş değeri her iki görevce de ele alınır. `fork()` sistem çağrısı ata ve oğul görevlere değişik değerler geri döndürür. Ata

göreve yaratılan oğul görevin görev kimliği *(PID)*, oğul göreve ise 0 değeri döndürülür. Oğul görevin ata görevin görünümünden çıkarak özelleşmesi geri dönüş değerleri arasındaki bu ayrım sayesinde olur. Oğul görev, örneğin, `execlp` adlı bir sistem çağrısını kullanarak görev iskeletinde, işletilen programı temsil eden *instuction segment* ve işlenen verileri temsil eden *user data segment* kesimlerini günler. `execlp` adlı sistem çağrısının görünümü aşağıda verilmiştir:

```
int execlp(char* path, char* program, char* arg1, char* arg2,
           ...., char* argn, char* NULL)
```

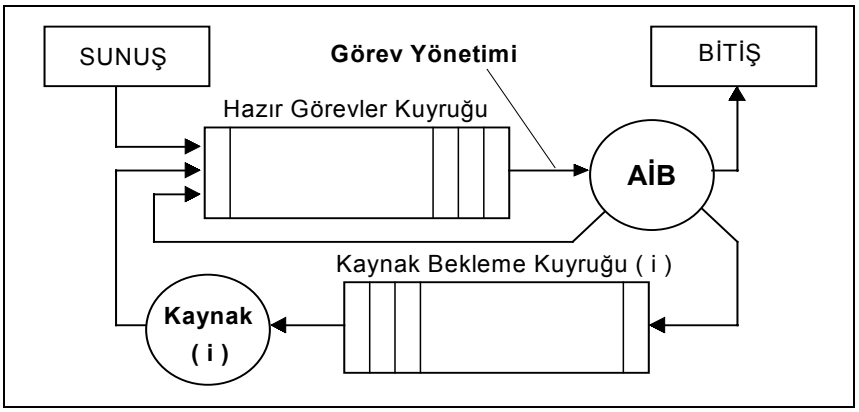
Burada `path` oğul görevin işleteceği amaç `program`'ın yer aldığı kılavuz kütüğü, `arg1, arg2, ...., argn` ise `program`'ın beklediği argümanları göstermektedir. `program`'ın beklediği argümanlar, C programlama dilinde, `main` adlı ana yordamın `argv[]` olarak anılan argüman dizisi üzerinden aktarılmaktadır. Her uygulamada argüman sayısı değişik olabileceğinden, satırın 0 damgası ile sonlanması gerekmektedir.

Aşağıda `fork` ve `exec` komut ikilisini kullanan 3 program örneklenmiştir. Bu programlardan `prog1` `prog2`'yi, `prog2` ise `prog3`'ü yaratıp başlatmaktadır. `prog1` `fork()` komutu ile önce kendisinin kopyasını yaratmakta ve `if` komutu ile `fork()`'un döndürdüğü değeri sınamaktadır. Kendisi bu değeri sıfırdan büyük bulacağından işletimine `for` döngüsüyle devam etmektedir. `fork()` komutuyla yaratılan kopya oğul görev ise `fork()`'un döndürdüğü değeri 0 bulduğundan `execlp("./prog2","prog2","3",NULL)` komutu ile, `prog2` komutlarını `argv[]` üzerinden aktarılan argümanlarla çalıştırmaktadır. Aynı açıklama `prog2` ve `prog3` program ikilisi için de geçerlidir. Bu durumda `prog1` 4 kez `ana`, `prog2` 3 kez `kız`, `prog3` ise 4 kez `torun` yazdırıp sonlanmaktadır.

```
/*  cc -o prog1 program1.c komutuyla derlenen program1.c  */
      main()
      {
      int i;
      if(fork()==0)
        execlp("./prog2","prog2","3",NULL);
      for(i=0; i<4; i++)
        printf("ana\n");
      }
/* cc -o prog2 program2.c komutuyla derlenen program2.c */
      main(argc,argv)
      int argc;       /* iletilen toplam argüman sayısı */
      char *argv[];   /* execlp ile iletilen argümanlar */
      {
      int i,son;
      son=atoi(argv[1]);     /* convert char to integer */
      if(fork()==0)
        execlp("./prog3","prog3","2",NULL);
      for(i=0; i<son; i++)
        printf("kız\n");
      }
```

```
/* cc -o prog3 program3.c komutuyla derlenen program3.c */
       main(argc,argv)
       int argc;
       char *argv[];
       {
       int i,son;
       son=atoi(argv[1]);
       for(i=0; i<son; i++)
         printf("torun\n");
       }
```

UNIX'te her yeni görev değişik bir programı çalıştırıyor olmakla birlikte ata görevden kalan açık kütükleri, *UNIX* kuralları gereği sahiplendirmeyi sürdürmektedir.



Çizim 3.11. Görev Yönetiminin Konumu

3.4. Görev Yönetimi

Bir bilgisayar sisteminde görev, işletim sisteminin, ana işlem birimi yönetimi açısından taban aldığı varlıktır. Sistemden hizmet almak amacıyla tanımlanan işler, sunuş aşamasında bir ya da daha çok göreve dönüştürülürler. Bu aşamadan sonra sistemde bir dizi görevin bulunduğu, bunların sistem kaynaklarını kullanabilmek için değişik işletim evrelerinden geçerek birbirleriyle yarıştıkları söylenebilir. Görevler arasında yer alan bu yarışın kuralları görev yönetimi kapsamında belirlenir. Daha önceden de belirtildiği üzere sistem kaynaklarının görevler tarafından kullanılabilmesi, ana işlem birimine anahtarlanarak çalışabilmeyi gerektirir. Bu nedenle görev yönetimi ana işlem biriminin yönetimi demektir. Görev durum çizeneğinde yer alan hazır görev durumundan çalışır görev durumuna geçişler ya da bu gösterime özdeş görev kuyruk çizeneğinde, hazır görevler kuyruğundan ana işlem birimine anahtarlanma işlemleri görev yönetimi kapsamında ele alınır (Çizim 3.11). Görev yönetiminden, görev yönetici olarak adlandırılan ve işletim sistemi çekirdek katmanında yer alan özel bir yordam sorumludur. Görev yönetici:

- Görevlerin giriş/çıkış ve zamanuyumlama istemlerinde bulunmaları, sonlanmaları gibi, görevlerin ana işlem birimini bırakmasını gerektiren durumlarda,

- Yeni bir görevin sisteme sunulması, giriş/çıkış ve zamanuyumlama istemlerinin yerine getirilmesi gibi, hazır görevler kuyruğuna yeni bir görevin eklenmesini gerektiren olaylar sonucunda, ya da,
- Etkileşimli işlem bağlamında, gerçek zaman saati vuruları sonucunda

ana işlem birimine anahtarlanarak kendisinden sonra işletilecek görevin belirlenmesini sağlar.

Görev yöneticiden sonra ana işlem birimini kullanacak görev, ya görev yöneticinin çalışabilmesi için işletimi kesilen görev ya da (kavramsal olarak) hazır görevler kuyruğunun başında yer alan diğer bir görev olacaktır. Örneğin bir giriş/çıkış işlemi başlatarak bekler duruma düşmüş bir görev, bu işlemin sona ermesi sonrasında ilgili bekler görevler kuyruğundan hazır görevler kuyruğuna bağlanacaktır. Hazır görevler kuyruğuna yeni bir görevin bağlanmış olması, bu aşamadan sonra hangi görevin ana işlem birimini kullanacağı değerlendirmesinin yeniden yapılmasını gerektirecektir. Bu nedenle, yeni bir görevin sisteme sunulması (*create*), giriş/çıkış ya da zamanuyumlama işlemlerinin sonlanması (*resume*), işletim için göreve ayrılan sürenin dolması (gerçek zaman saati vurusu) gibi olaylar sonrasında ortaya çıkan uyarılarla görev yönetici, ana işlem birimine anahtarlanacak, ilgili görevin hazır görevler kuyruğuna bağlanmasını ve bunun sonrasında da ana işlem birimini kullanacak görevin belirlenmesini sağlayacaktır.

Yukarıda anılan nedenlerin yanı sıra, bir görevin işletiminin son bulması, durdurulması (*suspend*), önceliğinin değiştirilmesi (*change-priority*) gibi, hazır görevler kuyruğuna yeni bir görev eklemeyi gerektirmeyen nedenlerle de görev yöneticinin işletilmesi ve işletime alınacak yeni görevin belirlenmesi gereklidir. Buradan, görev yöneticinin, ana işlem birimine çok sık anahtarlanan bir sistem görevi olduğu söylenebilir. Ana işlem birimine çok sık anahtarlanan bir görev olması nedeniyle, görev yöneticinin, sistem işletim hızını düşürmemek için küçük boyutlu olması gerekir.

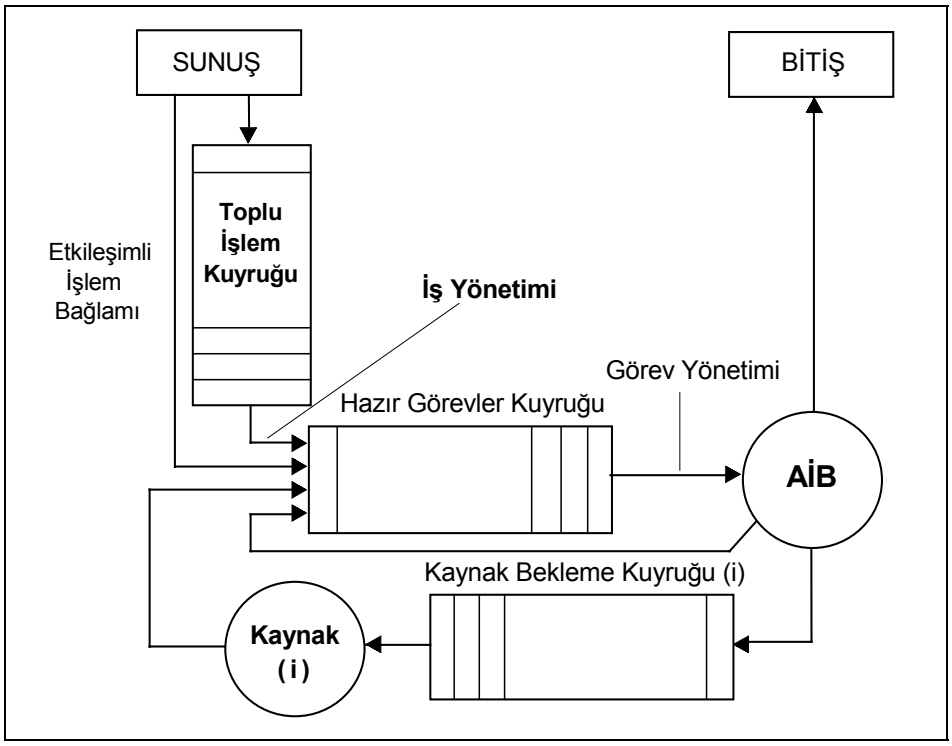
Çok sık gündeme gelmesi nedeniyle kimi zaman, görev yönetimi, kısa dönemli planlama olarak da anılır. Bu bağlamda orta dönemli planlama ve uzun dönemli planlama kesimleri de vardır. Görev yönetici, özetle, hazır görevler kuyruğunun düzenlenmesi ve düzenleme sonucu kuyruk başında yer alan görevin ana işlem birimine anahtarlanmasını gerçekleştiren işletim sistemi kesimi olarak düşünülebilir. Görev yönetici hazır görevler kuyruğunun düzenlenmesinden sorumlu olmakla birlikte bu kuyruğa eklenecek görevleri belirleyen bir kesim değildir. Hazır görevler kuyruğuna yeni görevlerin eklenmesi işletim sisteminin diğer yönetim kesimleri tarafından sağlanır.

İş Yönetimi

Kullanıcılar, almayı istedikleri hizmetleri iş tanımlarına dönüştürerek ya etkileşimli ya da toplu işlem düzenlerinde bilgisayar sistemine sunarlar. Sunulan işler, görev tanımları yapılarak görevlere dönüştürülürler. Oluşturulan bu görevler hazır görevler kuyruğuna bağlanarak işletilmek üzere görev yöneticinin yönetimine bırakılırlar. Toplu işlem, dengesiz kaynak talebinde bulunan, örneğin salt ana işlem birimini kullanan

ya da giriş/çıkış kanallarını aşırı yükleyen düşük öncelikli işlerin ele alındığı bir işletim düzenidir. Bu düzen, daha çok, öncelikli işlere ilişkin etkinliklerin azaldığı işletim dönemlerinde, sistem kaynaklarının (ana işlem biriminin, ana belleğin, giriş/çıkış kanallarının) kullanılır tutulmasını gözetir.

Sisteme toplu işlem düzeninde sunulan işler, toplu işlem kuyruğuna bağlanırlar. Bu kuyrukta bekleyen işlerin görevlere dönüştürülüp hazır görevler kuyruğuna bağlanmaları iş yönetimi kapsamında ele alınır. Toplu işlem kuyruğunda işletilmek üzere bekleyen işlerin ele alınarak görevlere dönüştürülmeleri çok sık yapılan bir işlem değildir. Bu nedenle, toplu işlem kuyruğunda bekleyen işlerin görev tanımlarının yapılarak hazır görevler kuyruğuna bağlanmalarını gerçekleştiren iş yöneticiye, kısa dönemli ile tezat oluşturacak biçimde uzun dönemli planlama kesimi de denir.

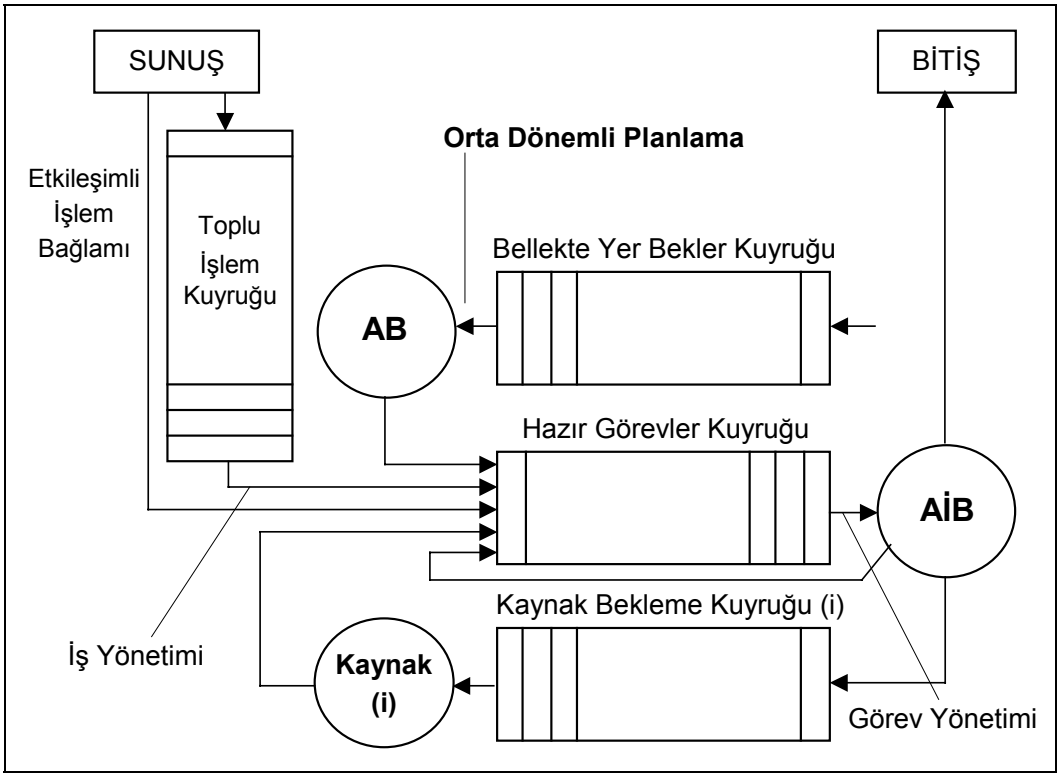


Çizim 3.12. İş Yönetiminin Konumu

Bilgisayar sistemine işler, toplu işlemin yanı sıra etkileşimli işlem bağlamında da sunulurlar. Ancak sisteme etkileşimli işlem düzeninde sunulan işler, beklemeksizin doğrudan görevlere dönüşüp hazır görevler kuyruğuna eklenirler. Bu nedenle sisteme sunulan işlerin tümünü kapsayan iş yönetimi adlandırmasını, salt toplu işlem kuyruğunu ele alan uzun dönemli planlama olarak düşünmek yanlış olmaz. Bu durumda iş yönetici, dönem dönem, örneğin ana işlem birimi kullanım oranı belirli bir düzeyin altına düştüğünde çalışan ve toplu işlem kuyruğunda bekleyen işleri görevlere dönüştürerek hazır görevler kuyruğuna bağlayan bir kesim olarak tanımlanabilir. Nasıl görev yönetici hazır görevler kuyruğunun düzenlenmesinden ve gerektiğinde bu kuyruğun başında yer alan görevin ana işlem birimine anahtarlanmasından sorumlu ise, iş yönetici de toplu

işlem kuyruğunun (ya da kuyruklarının) düzenlenmesinden ve dönem dönem bu kuyruğun başındaki işleri görevlere dönüştürerek hazır görevler kuyruğuna aktarmaktan sorumlu bir işletim sistemi kesimi olarak düşünülebilir (Çizim 3.12).

İş yönetimi ana işlem birimi kullanımı ağırlıklı işlerle giriş/çıkış kullanımı ağırlıklı işlerin dengeli bir biçimde işletilebilmelerini gözetir. Zira sistem kaynaklarının verimli kullanımı buna bağlıdır. Bu amaçla iş yönetici, ana işlem birimi kullanım oranını bir ölçüt olarak alıp bu oran düştüğünde hazır görevler kuyruğundaki görev sayısını artırır; bunun tersi olduğu durumlarda ise bu kuyruktaki görev sayısını sınırlı tutmaya çalışır. Buradan iş yöneticinin, görev yönetimi girdilerini düzenleyen bir kesim olduğu söylenebilir. Yukarıda da belirtildiği üzere, bu düzenlemeyi sık sık yapma gereği bulunmadığı açıktır. Bu nedenle iş yönetici sıkça işletilen bir kesim değildir. Ana işlem birimine az sıklıkta anahtarlanıyor olması, göreli çok sayıda kıstası ele alan, karmaşık algoritmalara dayalı olarak gerçekleştirilebilmesine izin verir. Görev yönetici yalın ve kısa bir yordam iken iş yönetici, genellikle, göreli karmaşık ve uzun bir işletim sistemi kesimi olarak ortaya çıkar.



Çizim 3.13. Orta Dönemli Planlamanın Konumu

Orta Dönemli Planlama

Hazır görevler kuyruğuna görevler, ya sunuş kesiminden ya da bekleme kuyruklarından ulaşırlar. Sunuş kesiminden gelen görevler, etkileşimli işlem sunuş katmanı (kabuk katman) ya da toplu işlem kuyruğundan gelen görevlerdir. Hazır görevler kuyruğuna

girdi üreten bekleme kuyrukları, değişik giriş/çıkış sürücülerine erişim ve zamanuyumlama değişkenlerine (semaforlara) ilişkin bekleme kuyrukları ile bellekte yer bekler görevler kuyruğudur. (Çizim 3.9). Görevlerin giriş/çıkış ve semafor kuyruklarından hazır görevler kuyruğuna geçişleri, giriş/çıkış işlemlerinin sonlanması, zamanuyumlamanın gerçekleşmesi gibi nedenlerle ortaya çıkan uyarılar sonucu gerçekleşir. Başka bir deyişle, bu geçişler iş ve görev yönetiminden bağımsız, kendiliğinden gerçekleşen geçişlerdir.

Bekleme kuyrukları kapsamında düşünülen bellekte yer bekler kuyruğu, bu yönden diğer bekleme kuyruklarından ayrılır. Bellekte yer bekler kuyruğundan bir görevin hazır görevler kuyruğuna bağlanması, toplu işlem kuyruğundan olduğu gibi, değişik kıstaslara dayalı değerlendirmeleri yapacak ve aktarılacak görevi belirleyecek bir kesimin çalışmasını gerektirir. Bu kesimin gerçekleştirdiği yönetim, görev ve iş yönetimlerinin adlandırılmasında olduğu gibi, işletim sıklığı göz önüne alınarak orta dönemli planlama diye anılır.

Çalışan bir görevin bitmesi sonucu ana bellekte yer açılması, hazır görevler kuyruğunda bekleyen görev sayısının belirli bir sınırın altına düşmesi gibi durumlarda ana işlem birimine anahtarlanan bu kesim, bellekte yer bekleyen görevler kuyruğundan bir göreve, ana bellek yönetim kesimiyle eşgüdüm içerisinde bellekte yer bulunmasını sağlayarak hazır görevler kuyruğuna bağlanmasını gerçekleştirir. Bu durumda orta dönemli planlama kesimi, dönem dönem işletime girerek bellekte yer bekler kuyruğundan hazır görevler kuyruğuna girdi sağlayan bir kesim olarak düşünülür (Çizim 3.13). Nasıl iş yönetici ana işlem birimi ile giriş/çıkış birimlerinin dengeli kullanımını kıstas alarak hazır görevler kuyruğuna girdi oluşturuyorsa, orta dönemli planlama kesiminin de, ana işlem birimi ile ana belleğin birlikte, dengeli kullanımını sağlamak üzere girdi oluşturan ve bu yolla görev yönetimini destekleyen bir kesim olduğu söylenebilir.

3.5. Yönetim Algoritmaları

İşletim sistemlerinin kullandığı yönetim algoritmaları, genel hatlarıyla hem uzun dönemli (iş yönetimi), hem kısa dönemli (görev yönetimi), hem de orta dönemli planlamaların tümü için kullanılabilir. Yukarıda da açıklandığı üzere, iş yönetimi toplu işlem kuyruğunun, görev yönetimi hazır görevler kuyruğunun, orta dönemli planlama kesimi ise bellekte yer bekler kuyruğunun düzenlenmesinden sorumludur. Bunlar sorumlu oldukları kuyruklara yeni bir iş ya da görev eklenmesi gerektiğinde ana işlem birimine anahtarlanan, kuyruklarda yer alan görev kimliklerini, belirli kıstaslara göre sıralı yapıda tutan ve kuyruk başında yer alan işin ya da görevin, yönetim kesiminin niteliğine göre ya hazır görevler kuyruğuna aktarılmasını ya da ana işlem birimine anahtarlanmasını gerçekleştiren sistem görevleri olarak düşünülebilirler. Yönettikleri kuyruklar farklı olmakla birlikte yerine getirdikleri yönetim ya da düzenleme işlevleri aynıdır. Bu nedenle, gerek iş yönetimi, gerek görev yönetimi ve gerekse orta dönemli planlama kesimlerinin kullanabildiği algoritmalar da kabaca aynıdır. İzleyen kesimde bu algoritmalar, daha çok görev yönetimi çerçevesinde açıklanacaktır. Ancak iş yönetimi

ya da orta dönemli planlamanın gerektirdiği özel durumlar söz konusu olduğunda ilgili ayrıntılar ayrıca açıklanacaktır[12].

Bilindiği üzere, işletim sisteminin görev yönetimi kesimi, ana işlem biriminin, sistemde tanımlı görevler arasında paylaştırılmasından sorumludur. Bu paylaştırma yapılırken temel amaç sistem başarımının yükseltilmesidir. Sistem başarımı dendiğinde, bir yandan sistemden yararlanan kullanıcılara verilen hizmetin niteliğini belirleyen (toplu işlemde) iş tamamlanma süresi, (etkileşimli işlemde) yanıt süresi gibi kıstaslar göz önününde tutulurken diğer yandan, bu kıstaslarla kimi zaman çelişebilen sistem kaynaklarının verimli kullanımı akla gelir. Sistem başarımını gözeten görev yönetimi kesimi de, bu bağlamda:

- Ana işlem biriminin verimli kullanımı
- Birim sürede sonlandırılan iş sayısı
- Ortalama iş tamamlanma süresi
- Bekleme süresi
- Yanıt süresi

gibi, hem kullanıcılara verilen hizmetin niteliğine, hem de bilgisayar sisteminin verimli kullanımına ilişkin kıstasları gözeten algoritmalara dayalı olarak gerçekleştirilir.

- Ana işlem biriminin verimli kullanılmasından, bu kaynağın mümkün olduğunca çalışır durumda tutulması anlaşılır. Ana işlem birimi, çalıştığı sürelerin dışında boştadır. Boşta olan ana işlem birimi komut işletimi yapmayan (ana beleğe erişmeyen) ancak bulunduğu boşta durumundan çıkmasını sağlayacak yeniden başlatma, kesilme gibi kimi dış uyarıları bekleyen bir ana işlem birimidir. Ana işlem biriminin çalışıyor olması makina komutları işletiyor olması demektir. Bu bağlamda, işletilen komutların hizmete dönük, yararlı iş üretip üretmedikleri gözetilen bir husus değildir. Örneğin bir koşulun oluşması (örneğin klavyede bir tuşa basılması) beklenirken işletilen sınama döngü komutları, hizmet üretmeyen ancak ana işlem birimini çalışır tutan komutlardır. Bununla birlikte ana işlem biriminin büyük oranlarda çalışır tutulması, hizmete dönük yararlı iş üretebilmenin de (yeterli olmamakla birlikte) gerekli bir ön koşuludur.

- Ana işlem biriminin çalışır durumda tutuluyor olması, yukarıda da belirtildiği gibi, tek başına, üretilen hizmetin tam bir ölçüsü olamayacağından birim sürede tamamlanan ortalama iş sayısı sistem başarımının daha dolaysız bir ölçüsüdür. Yönetim yazılımlarının kullandığı algoritmalar, bu parametreye dayalı olarak öngörülebilirler.

---

[12] Bilindiği gibi, bilgisayar sistemine programlar, genelde ya toplu işlem ya da etkileşimli işlem bağlamında sunulur. Günümüzde bilgisayar hizmetleri, işlem ve saklama sığaları gelişmiş, tekil ya da bir ağ içinde kümelenen kişisel bilgisayar sistemlerinden alınır bir biçime gelmiştir. Bunun doğal sonucu olarak bu bilgisayar sistemlerinin kullandığı işletim sistemleri de salt etkileşimli işlem gözetilerek tasarlanmıştır. Bu durumda, çok kullanıcılı bilgisayar sistemlerinde anlamlı olabilen iş yönetimi işletim sisteminin bir parçası olma özelliğini kaybetmiş, görev yönetimi ön plana çıkmıştır. Bu nedenle, tarihsel süreç içerisinde, daha çok iş yönetimi için geliştirilen yönetim algoritmaları, günümüzde görev yönetimi için kullanılmaktadır.

- Birim sürede tamamlanan iş sayısı yerine, işlerin ortalama tamamlanma süreleri de sistem başarımının bir ölçüsüdür. İş tamamlanma süresi, işlerin sisteme sunuluşlarından bitişlerine (sonlanmalarına) kadar geçen sürelerin ortalamasıdır. Bir işin tamamlanma süresi, o işin sunuş kuyruğunda (örneğin toplu işlem kuyruğunda) bekleme, görev tanımlarının yapılmasından sonra işletime alınma, dönüşümlü olarak giriş/çıkış, zamanuyumlama kuyruklarında bekleme ve işletilme sürelerinin toplamıdır.

- İş tamamlanma süresi işletim ve bekleme sürelerinin toplamıdır. Bu toplamın yanı sıra, sistem kaynaklarını diğer işlerle (ya da görevlerle) paylaşmaktan kaynaklanan bekleme süresinin kendisi de, yönetimin dayanabileceği bir kıstas ve sistem başarım ölçüsüdür. Bekleme süresi, işlerin ya da görevlerin, ana işlem birimi kullanımı dışında kaynak bekleyerek geçirdikleri sürelerin ortalamasıdır.

- Etkileşimli işlem ortamında işletilen iş ve görevlerin sonlanma hızı, terminali başında sonuç bekleyen kullanıcıya hemen yansıyan bir durumdur. Bu bağlamda kullanıcının, çalıştırdığı komutlara aldığı yanıtların süreleri de verilen hizmetin niteliğinin bir ölçüsüdür. Bu sürelerin ortalaması, kısaca yanıt süresi olarak anılır. Yanıt süresinin ölçümünde genellikle aritmetik ortalama kullanılır. Ancak yanıt sürelerinin standart sapması da kullanılabilen diğer anlamlı bir ölçümdür. Standart sapmanın küçük tutulmaya çalışıldığı sistemlerde, yanıt sürelerinin işletimden işletime büyük değişiklikler göstermediği, yanıt süreleri açısından öngörülebilir bir sistem görünümü sağlanabilmektedir.

Görev yönetimi kapsamında, bu kıstaslardan bir ya da birkaçını birlikte gözeten değişik yönetim algoritmaları kullanılır. Bunlardan kimileri:

- İlk gelen önce (*First Come First Served*)
- En kısa işletim süresi kalan önce (*Shortest Remaining Time First*)
- Öncelik tabanlı (*Priority Based*)
- Zaman dilimli (*Time Sliced, Round-Robin*)
- Çok kuyruklu (*Multi-level queues*)

gibi adlarla anılırlar. Bu algoritmaları, işletilmekte olan bir görevin işletimini, bu görevin iradesi dışında kesen ya da kesmeyen algoritmalar olmak üzere iki değişik sınıfta ele almak olanaklıdır. Bilindiği gibi görev yönetici, ya giriş/çıkış isteminde bulunma, sonlanma ve zamanuyumlama gereksinimleri gibi işletilmekte olan görevlerin kendilerinden kaynaklanan nedenlerle ya da hazır görevler kuyruğuna yeni bir görevin bağlanmasını gerektiren durumlarda ana işlem birimine anahtarlanmaktadır. Hazır görevler kuyruğuna yeni bir görevin bağlanması, bileşimi değişen hazır görevler kuyruğundan hangi görevin çalıştırılacağının yeniden belirlenmesini gerektirir. Eğer bu belirleme, o anda işletilmekte olan görevi de içerecek biçimde yapılırsa uygulanan algoritmanın kesen (*preemptive*) algoritma olduğu söylenir. Başka bir deyişle, bir görev, uygulanan yönetim gereği bir kez ana işlem birimine anahtarlandıktan sonra giriş/çıkış, zamanuyumlama gereksinimleri gibi, kendisinden kaynaklanan nedenler

dışında (kendi istemi dışında) da ana işlem birimini bırakmak zorunda kalıyorsa uygulananan yönetim algoritması kesen algoritma olarak adlandırılmaktadır. Doğal olarak görevlerin işletimlerinin kendi istemleri dışında kesilmediği durumlarda da kesmeyen (*non-preemptive*) algoritmalardan söz edilmektedir.

3.5.1. İlk Gelen Önce Algoritması *(First Come First Served)*

İlk gelen önce algoritması, adından da anlaşılacağı üzere, görevlerin eş öncelikli olarak ele alındığı ve görevlerin, hazır görevler kuyruğuna geliş sırasında işletildiği, çok yalın bir yönetim algoritmasıdır. Bir görevin, sisteme ilk kez sunulma, başlatılan bir giriş/çıkış işleminin sonlanması, zamanuyumlamanın gerçekleşmesi gibi herhangi bir nedenle hazır görevler kuyruğuna bağlanması gerektiğinde, bu algoritma gereğince, kuyruğa sonuncu öge olarak bağlanır. İşletilmekte olan görevin, kendi istemiyle ana işlem birimini bırakması sonucu, kuyruk başında yer alan görev ana işlem birimine anahtarlanır. İlk gelen önce algoritması, görevlerin eş öncelikli olması nedeniyle kesen bir algoritma değildir.

Gerçekleştirimi çok yalın ve kısa bir biçimde yapılabilen bu algoritma, yönetimini üstlendiği tüm görevleri, niteliklerini gözetmeksizin aynı öncelikle ele alması nedeniyle, genelde yüksek başarım sağlayan bir algoritma değildir. İlk gelen önce algoritması, iş yönetimi kapsamında da aynı ad ve ilkeye dayalı olarak kullanılır. Bu bağlamda işlerin, toplu işlem kuyruğundan alınıp görevlere dönüştürülerek hazır görevler kuyruğuna aktarılmaları, sisteme sunuluş sırasında gerçekleştirilir.

3.5.2. En Kısa İşletim Süresi Kalan Önce (*Shortest Remaining Time First*)

En kısa işletim süresi kalan önce algoritmasında, görevlerin ana işlem birimine anahtarlanmasında göz önüne alınan kıstas, görevlerin sonlanabilmeleri için arda kalan işletim süreleridir. En kısa işletim süresi kalan göreve öncelik sağlanarak bir an önce sistemden çıkması ve bu yolla en kısa ortalama bekleme süresinin elde edilmesi amaçlanır. En kısa işletim süresi kalan önce algoritması, var olan diğer yönetim algoritmaları içinde, kuramsal olarak, ortalama bekleme süresi yönünden en iyi sonucu veren algoritmadır. Ancak kalan işletim süresi ölçülebilen bir değer değildir. Bu değerin bir biçimde kestirilmesi gereklidir. Bu kestirim iş yönetimi kapsamında sağlıklı bir biçimde yapılabilir. İşler bilgisayar sistemine sunulurken, gereksedikleri ana işlem birimi zamanı (t_{max}) bilgisi, genelde kullanıcılardan alınır. Bu bilgi, algoritmanın iş yönetimi çerçevesinde kullanımı için yeterlidir. Zira her işin, işletiminin herhangi bir aşamasında ne kadar süreyle ($\sum t_i$) ana işlem biriminden yararlandığı, sayışımın bir gereği olarak bilinir. Bu iki değer arasındaki ayrım (t_{max}- $\sum t_i$), işin kalan işletim süresi olarak kullanılır. Aynı bilginin görev yönetimi çerçevesinde de kullanılması düşünülebilir. Ancak bu süre, ilgili işe ilişkin görev ya da görevlerin ana işlem biriminden gereksediği toplam süredir. Bu sürenin, görev(ler) tarafından kaç anahtarlanma sonucu tüketileceği, her anahtarlanmada hangi görevin ana işlem birimini ne kadar süre işgal edeceği, bu bilgiye dayanılarak kestirilemez. Sözkonusu algoritma görev yönetimi için kullanılacağı zaman, görevlerin herbiri için, her anahtarlanmada,

ana işlem birimini ne kadar süre işgal edeceklerinin başka bir yolla kestirilmesi gerekir. Bu amaçla üstel (eksponansiyel) ortalama olarak anılan bir kestirim yöntemi kullanılabilir. Bu yöntemde, bir görevin gelecek işletiminde harcayacağı ana işlem birimi süresi; o görevin bir önceki işletiminde harcadığı süre ile bir önceki kestirimden (tarihçesinden) yararlanılarak, aşağıdaki formül aracılığıyla hesaplanır:

$$\tau_{n+1} = \alpha t_n + (1 - \alpha)\ \tau_n$$

τ : kestirilen işletim süresi;
t : gerçekleşen işletim süresi

Bu formül içinde yer alan α değişmezi {0-1} aralığında değer alan bir sayıdır. α değişmezinin 0 değerini alması durumunda $\tau_{n+1} = \tau_n$ olacağından geçmişte gerçekleşen işletim süreleri göz önüne alınmaksızın, aynı görev için hep aynı kestirim değerinin kullanıldığı durum ortaya çıkar. α değişmezinin 1 değerini alması durumunda ise, bu kez $\tau_{n+1} = t_n$ olacağından gelecek işletim süresinin kestirimi olarak, bir önce gerçekleşen işletim süresi alınmış olacaktır. Yukarıda verilen formülün açılmış biçimi aşağıdaki gibidir:

$$\tau_{n+1} = \Sigma\ \alpha (1-\alpha)^i\ t_{n-i} \quad (0 \le i \le N) \quad ya\ da$$
$$\tau_{n+1} = \alpha t_n + \alpha (1-\alpha)\ t_{n-i} + \ldots + \alpha (1-\alpha)^k\ t_{n-k} + \ldots$$

Bu açılımdan da anlaşılacağı üzere α değişmezinin {0-1} aralığındaki değerleri alması durumunda, gelecek işletim süresinin kestirim değeri olarak, geçmiş işletim sürelerinin $\alpha(1-\alpha)^k$ çarpımı ile dengelenen bir ortalaması kullanılır.

Görevlerin bu ya da buna benzer yöntemlerle belirlenen gelecek işletim süreleri taban alınarak ana işlem birimine anahtarlanmaları gerçekleştirilir. Örneğin herhangi bir görev hazır görevler kuyruğuna bağlanacağı zaman, işletime alındığında ana işlem birimini ne kadar süre işgal edeceği hesaplanır. Elde edilen kestirim değerine göre hazır görevler kuyruğunda ilgili yere eklenir. Bu algoritma ile, hazır görevler kuyruğunun, görevlerin, gelecek işletim sürelerine ilişkin kestirimlere göre sıralı bir düzende tutulabileceği söylenir.

En kısa işletim süresi kalan önce algoritması, görev yönetimi kapsamında hem kesen hem de kesmeyen algoritma olarak gerçekleştirilebilir. Bu algoritma, iş yönetimi (toplu işlem kuyruğunun düzenlenmesi) için kullanıldığında en kısa iş önce (*Shortest Job First*) olarak adlandırılmaktadır.

3.5.3. Öncelik Tabanlı Algoritma *(Priority based)*

Öncelik tabanlı algoritma, yukarıda açıklanan algoritmanın genelleştirilmiş biçimidir. Bu algoritma için her görevin bir önceliği bulunur. Bu öncelik değeri görev iskeleti içinde, öncelik alt alanında tutulur. Görevler genelde, sisteme sunuluşları sırasında *0*'dan *N*'ye kadar değişen bir öncelik değeri alırlar. Örneğin görev iskeleti içinde öncelik alt alanı bir bayttan oluşuyorsa görevler 0'la 255 değerleri arasında bir önceliğe sahip olabilirler. Bu bağlamda yüksek öncelikli ya da düşük öncelikli görevlerden söz edilir.

Görev öncelikleri, ana işlem birimi kullanım süresi, ana bellek gereksinimi, giriş/çıkış kanal kullanım sıklığı gibi görevin sistem kaynaklarına ilişkin ölçülebilir taleplerine, ait olunan kullanıcı grubunun özelliklerine, sisteme sunuluş (etkileşimli - toplu işlem) biçimine dayalı olarak belirlenir.

Bu algoritmaya göre, ana işlem birimine yeni bir görev anahtarlanacağı zaman en yüksek önceliğe sahip görev seçilir. Bunun için hazır görevler kuyruğunun görev önceliği sırasında tutulması ve hep kuyruk başındaki görevin ana işlem birimine anahtarlanması bir gerçekleştirim yöntemi olarak düşünülebilir. Bu yaklaşım anlaması kolay, mantıklı bir yaklaşımdır. Ancak görevlerin, hazır görevler kuyruğa, geliş sırasında eklenmesi ve en öncelikli görevin, ana işlem birimine anahtarlama yapma aşamasında, kuyruğun taranarak saptanması da bir diğer gerçekleştirim biçimi olabilir. Buradaki tercihi, doğal olarak sıralı tutulan bir kuyruğa yeni bir öge ekleme ile kuyruk tarama maliyetlerin oranı belirleyecektir. Öncelik tabanlı görev yönetim algoritması, hem kesen hem de kesmeyen algoritma olarak gerçekleştirilebilmektedir.

Bir bilgisayar sisteminde çalışan tüm görevler, ana işlem birimine, her zaman hazır görevler kuyruğu üzerinden anahtarlanmazlar. Kimi görevler, (kesilmeler gibi) donanımdan kaynaklanan uyarılar sonucunda, görev yöneticinin denetimi dışında da ana işlem birimine anahtarlanabilirler. Ana işlem birimine bu biçimde anahtarlanan görevler, yerine getirdikleri hizmetler yönünden, çoğunlukla zamana bağımlı, gecikme tanımayan özel sistem görevleridir. Öncelik tabanlı görev yönetimi, uyarılara bağlı olarak ana işlem birimine anahtarlanan görevler için uyarı tabanlı *(event driven)* görev yönetimi olarak adlandırılır. Uyarı tabanlı görev yönetiminde görev önceliği, görevlerin bağlı oldukları uyarıların ana işlem birimi tarafından ele alınma önceliğiyle belirlenir. Uyarı tabanlı görev yönetimi, daha çok süreç denetimi gibi gerçek zamanlı uygulamaların yürütüldüğü sistemlerde kullanılır.

Görevlerin öncelikleri, sunuş aşamasında bir kez belirlendikten sonra, tüm işletimleri boyunca, genellikle bir daha değişmez. Ancak özellikle çok yüklü sistemlerde, yüksek öncelikli işlerin, sistemde hiç eksik olmaması yüzünden düşük öncelikli işlerin işletimleri, aşırı boyutlarda gecikebilir. Bu sakıncayı ortadan kaldırmak üzere öncelik yaşlanması (*priority aging*) olarak anılan bir yöntem kullanılır. Bu yöntemle, kimi istisnalar dışında, sistemde çalışan tüm görevlerin öncelikleri, işletim sistemi tarafından, belirli sıklıkta (örneğin her yarım saatte bir) *modulo N* bir artırılır. *(i)* öncelik değerine sahip bir görevin *(i+1)* öncelik değerine sahip bir görevden daha öncelikli olduğu varsayılarak, öncelik değeri *N*'ye ulaşan bir görevin, bu yöntemle, bir sonraki adımda 0 değeri ile en öncelikli göreve dönüşmesi gerçekleşir. Böylece düşük öncelikli görevlere de yüksek öncelik kazanarak işletilebilme şansı sağlanmış olur.

3.5.4. Zaman Dilimli Algoritma *(Time Sliced - Round Robin)*

Zaman dilimli görev yönetim algoritmasıyla, hazır görevler kuyruğunda bekleyen görevler, eşit uzunluktaki zaman dilimleri içinde ana işlem birimine, sırayla anahtarlanır. Örneğin, her 5 milisaniyede bir gelen saat uyarılarıyla ana işlem birimine anahtarlanan görev yönetici, çalışmakta olan görevi, hazır görevler kuyruğunun sonuna

ekler. Kuyruk başındaki görevi de, kendisinden sonra çalışmak üzere ana işlem birimine anahtarlar. Görevlerin işletimi, giriş/çıkış ve zamanuyumlama istemi gibi nedenlerle kendilerine ayrılan zaman dilimi dolmadan sonlanabilir. Bu durumda da, yine görev yönetici ana işlem birimine anahtarlanır. Zaman aralığı sayacını sıfırlayarak kuyruk başındaki görevi, yeni bir zaman dilimi için çalıştırır.

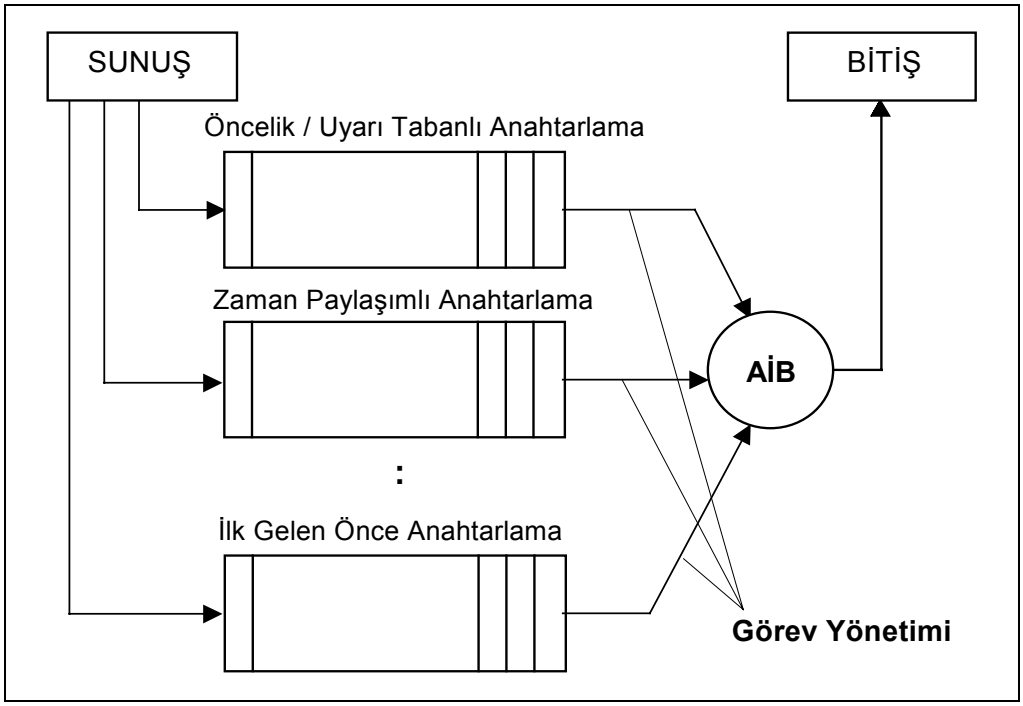
Zaman dilimli görev yönetim algoritması, çoğunlukla, etkileşimli işlemin uygulandığı sistemlerde kullanılır. Etkileşimli işlem kapsamında işletilecek işler, görev tanımları yapılarak hazır görevler kuyruğuna bağlanırlar. Görevlere ayrılan zaman aralığının t milisaniye, sistemde aynı anda işletime alınan görev sayısının da n olduğu varsayılırsa bu sistemde her görevin $t \, x \, n$ milisaniye sıklığında işletileceği söylenebilir. Yukarıda verilen örnekte olduğu gibi, görevlerin 5 milisaniye süreyle ana işlem birimine anahtarlandığı düşünülürse, sistemde aynı anda 20 görevin çalıştığı bir durumda her göreve en geç 100 milisaniyede bir (1/10 saniyede bir) sıra geleceği söylenebilir. Zaman dilimli yönetim algoritması uygulanırken kullanılan zaman aralığının (diliminin) seçimi önemlidir. Eğer bu süre çok uzun tutulursa görevlere sıra gelme sıklığı ve etkileşimli işlem başarımı düşer. Bu sürenin çok kısa tutulması durumunda ise görev anahtarlama işlemlerine harcanan sürenin görev işletimine ayrılan süreye oranı artarak ana işlem birimi kullanım verimliliğinin düşmesine neden olunur.

Zaman dilimli görev yönetim algoritması, öncelik tabanlı görev yönetimi çerçevesinde düşünüldüğünde eş öncelikli bir algoritma olarak değerlendirilebilir. Zira görevler, başkaca hiçbir kıstas göz önüne alınmaksızın, sırayla, eşit zaman aralıklarında ana işlem birimini kullanmaktadır. Zaman dilimli görev yönetimi, şimdiye değin açıklanan diğer algoritmalardan farklı olarak, doğası gereği, salt, kesen bir algoritmadır.

3.5.5. Çok Kuyruklu Algoritma *(Multi-level Queues)*

Şimdiye değin incelenen görev yönetim algoritmalarında, görevlerin, ana işlem birimine anahtarlanabilmek için tek bir hazır görevler kuyruğu üstünde beklediği ve bu kuyruk üzerinde bekleyen tüm görevlere aynı yönetim algoritmasının uygulandığı varsayılmıştır. Bunun yanı sıra, sözkonusu yönetim algoritmalarının herbiri için, belirgin bir işletim ortamı bulunduğu söylenmiştir. Örneğin en kısa işletim süresi kalan önce algoritmasının, görevlerin sonlanması açısından en kısa bekleme süresini sağlayan bir algoritma olduğu belirtilirken, zaman dilimli yönetim algoritmasından, etkileşimli işlem ortamının kurulmasında yararlanılan bir algoritma olarak söz edilmiştir. Bir bilgisayar sisteminde işletime alınan görevler çok değişik niteliklerde olabilirler. Bu görevlerden kimileri, örneğin etkileşimli işlem ortamında program geliştiren kullanıcı görevleri olabilirken, kimileri de toplu işlem kuyruğundan sisteme sunulmuş ve sistem kaynaklarını boş dönemlerinde çalışır tutmayı amaçlayan işlerle ilgili görevler olabilir. Bunun yanı sıra işletilen görevler arasında hiç bekletilmeden ana işlem birimine anahtarlanması gereken çok öncelikli sistem görevleri de bulunur. Bu görevlerin hepsini aynı hazır görevler kuyruğuna bağlamak ve bunların tümüne aynı yönetim algoritmasını uygulamak tüm görevler için uygun bir yol olmayabilir. Örneğin en kısa işletim süresi kalan önce algoritmasının tüm görevlere uygulanması, etkileşimli işlem ortamını tümüyle yozlaştırabilir.

Bu nedenle, değişik yönetim algoritmalarının olumlu yanlarını biraraya getirmek ve sakıncalarını azaltmak üzere çok kuyruklu algoritma kullanılır. Çok kuyruklu algoritmada ana işlem birimine anahtarlanmayı bekleyen görevler, ortak bir hazır görevler kuyruğu yerine, görevin türüne göre ayrı kuyruklara bağlanırlar. Ortaya çıkan birden çok kuyruğun herbiri için, yukarıda açıklanan algoritmalardan uygun olanı kullanılır (Çizim 3.14). Bu durumda, örneğin, sistemde etkileşimli işlem ortamında çalışmak üzere tanımlanan görevler, zaman dilimli algoritmanın uygulandığı kuyruk üzerinden, toplu işlem kapsamında sunulan işlerle ilgili görevler ise ilk gelen önce ya da en kısa işletim süresi kalan önce algoritmalarının uygulandığı diğer kuyruklar üzerinden işletime alınırlar. Bunun gibi, öncelik tabanlı algoritmanın yönetim başarımını yükselttiği görevler için de, bu algoritmanın uygulandığı ayrı bir kuyruk öngörülür.



Çizim 3.14. Çok Kuyruklu Görev Yönetimi

Değişik kuyruklara bağlı olmalarına karşın, görevlerin, anahtarlanmayı bekledikleri ana işlem birimi sayısı kısıtlıdır. Bu nedenle ana işlem birimine yeni bir görev anahtarlanacağı zaman bu görevin hangi kuyruktan seçileceğinin de belirlenmesi gereklidir. Bu bağlamda izlenebilecek yollardan biri, hazır görev kuyruklarına çeşitli önceliklerin verilmesidir. Bu durumda bir görevin ana işlem birimine anahtarlanabilmesi için hem bulunduğu kuyruğun başında yer alması hem de bulunduğu kuyruktan daha öncelikli kuyrukların boş olması gereklidir. Bu yöntemin sakıncası, öncelik tabanlı algoritma açıklanırken belirtildiği üzere, çok yüklü sistemlerde, düşük öncelikli kuyruklarda yer alan görevlerin aşırı derecede gecikmesidir. İzlenebilir diğer bir yol, değişik hazır görev kuyrukları arasında, zaman dilimli yönetim algoritmasını uygulamaktır. Bu yöntemle, sistemde bulunan değişik kuyruklara, öncelikleriyle uyumlu sayıda zaman dilimi verilir. Eğer sistemde, örneğin 3 değişik kuyruk varsa, bu

kuyruklardan en önceliklisine örneğin 5, onu izleyene 3, son kuyruğa da 1 zaman dilimi ayrılır. Kuyruklara ayrılan zaman dilimleri içinde, kuyruk içi sırada, işletilebilen sayıda görev işletilir. Bu yolla kuyruk ve görevler arası öncelikler korunurken çok düşük öncelikli işlere de, dönem dönem işletim hakkı tanınarak aşırı gecikmeleri önlenir.

Çok kuyruklu görev yönetim algoritmasını da kesen algoritmalar çerçevesinde düşünmek gereklidir. Zira bu algoritma, buraya değin bilinen kesen ya da kesmeyen algoritmaların bir karmasıdır. Çok kuyruklu görev yönetim algoritması, genel amaçlı büyük boy bilgisayar sistemlerinde kullanılan bir yönetim algoritmasıdır.

3.6. İşletim Dizileri *(Threads)*

Görev bir programın işletimine verilen addır. Kimi programlar, yerine getirdikleri işlev açısından koşut işletilebilecek kesimler içerirler. Bu kesimlerin tek bir program içinde yer almaları ardarda işletilmelerini zorunlu kılar. Koşut işletilebilir her kesim için ayrı bir program öngörüp bunları koşut işletmek ve bu yolla işletimi hızlandırmak ve verilen hizmetin niteliğini artırmak mümkündür. Ancak ortak bir amaca (işleve) dönük değişik kesimlerin ayrı programlar tarafından ele alınması, bu kesimlere ana bellekte işletim sistemi tarafından ayrı adres (kod ve veri) alanları ayrılması sonucunu doğurur ve sözkonusu kesimlerin ortak verileri paylaşmalarını zorlaştırır. Bu paylaşım, çoğunlukla bir programın adres evreninden diğer bir programın adres evrenine veri aktarma yoluyla gerçekleşir. Örneğin *UNIX* işletim sisteminde koşut çalışan iki ayrı programın (görevin) veri alış verişi, Birlikte Çalışan Görevler konu başlığı altında ayrıntılı olarak açıklanacağı üzere, genelde bir adres evreninden diğer bir adres evrenine veri aktarımı yapan *pipe* ve *fifo* gibi kimi görevlerarası veri iletişim araçları (sistem çağrıları) ile, işletim sistemi ve sistem kaynakları yönünden pahalı bir biçimde mümkün olabilmektedir.

Bu durum, aslında, işletim sisteminin her görevi tek bir işletim dizisi gibi görmesinden kaynaklanmaktadır. Tek bir program içinde birden çok koşut çalıştırılabilir işletim dizisi öngörülebilmesi durumunda bu dizilerin aynı adres evrenini paylaşması ve bu yolla veri alışverişinin kolaylaştırılması sağlanır. Bir program içinde yer alan koşut çalıştırılabilir kesimler işletim dizileri *(threads)* olarak adlandırılır. Bu bağlamda görev, ilgili işletim dizilerinin çalıştığı ortak bir işletim ortamı olarak düşünülür. İşletim sistemi, ana bellek, giriş/çıkış (kütükler) gibi sistem kaynaklarını görevler düzeyinde atar ve o düzeyde izler. Bir görev içinde tanımlı işletim dizileri sözkonusu bu kaynakları paylaşırlar. Ancak işletim dizisi olarak tanımlanmış her kesimin kendine özgü bir program sayacı bulunur. Bu yolla ana işlem birim(ler)ine ayrı ayrı anahtarlanarak koşut bir biçimde işletilebilirler. Bir göreve ilişkin işletim dizilerinin ana işlem birimine anahtarlanmaları "görev anahtarlama"ya göre daha dar kapsamlı dolayısıyla daha hızlı ve daha ucuz bir yol olan "bağlam anahtarlama" yoluyla gerçekleşir. İşletim dizileri, aynı görevler gibi, işletimleri sırasında hazır, bekler, çalışır gibi durumlarda bulunurlar.

İşletim dizilerinin yönetimi genelde, kullanıcı evreninde ya da işletim sistemi evreninde olmak üzere iki değişik biçimde gerçekleşir. Eğer yönetim kullanıcı evreninde ele alınıyorsa işletim sistemi işletim dizilerinin varlığından habersizdir. Bu durumda

kullanıcı evreninde çalışan bir yönetim paketi bir görev içinde tanımlanmış değişik işletim dizilerinin yönetimini üstlenir. Bir işletim dizisi herhangi bir nedenle işletimi bıraktığında, aynı görev içinde tanımlı hangi işletim dizisinin işletime alınacağını, ilgili görevle birlikte işletimde bulunan sözkonusu yönetim paketi belirler. Aynı yönetim paketi birden çok işletim dizisi içeren programlar tasarlanırken, programcılara işletim dizisi tanımlama, çalıştırma, durdurma, zamanuyumlama gibi işlevleri de sağlarlar. Başka bir deyişle bu tür programlar tasarlanırken ilgili paketin tanım ve işlevleri kullanılır. *POSIX P-Threads* ve *Mach C-Threads*, günümüzde bu tür işletim dizisi yönetim paketlerine verilebilecek iki örnektir.

Kimi modern işletim sistemleri, görevlerin yanı sıra görevler altında yer alan işletim dizilerinin yönetimini de üstlenir. Çok işletim dizili işlem *(multi-threading),* bu durumda işletim sisteminin bu amaçla sağladığı sistem çağrıları kullanılarak gerçekleştirilir. *Windows NT* böyle bir işletim sistemine verilebilecek en belirgin örnektir.

Çok işletim dizili işlemin, kullanıcı evreni yönetim paketleriyle mi yoksa işletim sisteminin sağladığı araçlarla mı ele alınması gerektiği halen tartışılan bir konudur. Her iki yaklaşımın da kendine özgü üstünlükleri ve sakıncaları sözkonusudur. Bununla birlikte *UNIX* gibi çok yaygın kullanımı olan işletim sistemlerinde çok işletim dizili işlemi, yukarıda anılan kimi yönetim paketleriyle gerçekleştirmenin dışında da başkaca bir seçenek mevcut değildir.

4. BÖLÜM

İŞLETİM SİSTEMLERİ

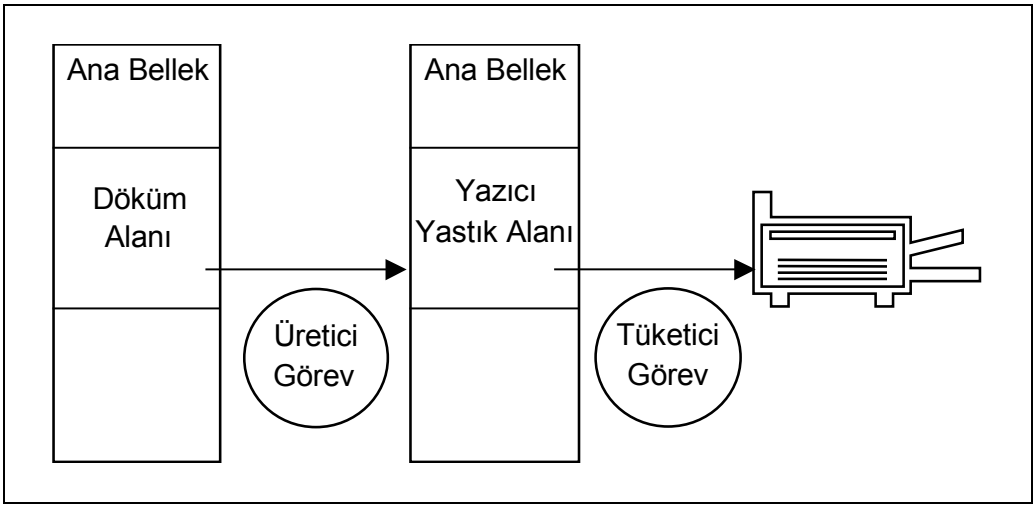
BİRLİKTE ÇALIŞAN GÖREVLER

Birlikte çalışan görevler zaman içinde koşut işletilen görevlerdir. İki görevin koşut işletimi, bunlardan birinin işletimi tümüyle sonlanmadan diğerinin de işletime alınması durumunda sözkonusu olur. Görev işletiminin koşutluğu gerçek ya da görüntü koşutluk olabilir. Birden çok işleyici içeren sistemlerde, eğer iki görev, herbiri bir işleyiciye atanmış biçimde işletiliyorsa buradaki koşutluğa gerçek koşutluk denir. Tek ana işlem birimi içeren sistemlerde iki görevin koşut işletimi, bu görevlerin, zaman içinde ana işlem birimini sırayla kullanmaları yoluyla sağlanır. Buradaki koşutluk, doğal olarak görüntü koşutluktur. Zira bu durumda, mutlak anlamda, her iki görevin de aynı anda işletiliyor olması sözkonusu değildir. Bir sistemde birlikte işletime alınan görev sayısı genelde, sistemde bulunan ana işlem birimi sayısından çok daha büyüktür. Bu nedenle koşut işletim, daha çok görüntü koşutluk kapsamında yürütülür. İster gerçek, ister görüntü olsun koşut işletim, çok önemli işletim bütünlüğü sorunlarını beraberinde getirir. İzleyen kesimde bu sorunların neler olduğu ve bunların nasıl aşılacağı incelenecektir. Bu bağlamda, önce, koşut işletilen görevler arası etkileşimden söz edilecek ve bunun yarattığı sorunlar irdelenecektir. Bu sorunların aşılmasına yarayan görevler arası alt düzey ve üst düzey zamanuyumlama araçları incelenecektir. Alt düzey araçlar kapsamında semaforlar ve bunlar üzerinde işlem yapan ilkel zamanuyumlama işleçleri açıklanacaktır. `send` ve `receive` ileti aktarım komutlarından, uzaktan yordam

çağırma düzeneğinden ve *monitor* kavramından üst düzey zamanuyumlama araçları kapsamında söz edilecektir. Son olarak, görevler arası zamanuyumlamanın yol açtığı görevler arası kilitlenme sorunu irdelenecek ve bu sorunu aşmak için kullanılan yaklaşımlar açıklanacaktır. Bu bağlamda verilen açıklamalar görev kavramı taban alınarak veriliyor olmakla birlikte görevlerin altında yer alan işletim dizileri için de, tümüyle geçerlidir.

4.1. Koşut İşlem ve Görevler arası Etkileşim

Birlikte çalışan ya da işletimleri koşut sürdürülen görevler birbirleriyle etkileşebilirler. Bu etkileşme, görevlerin, işletimleri sırasında kimi kaynakları ortak kullanmalarından kaynaklanır. Çıktı üreten bir kullanıcı görevi ile bu çıktıları yazıcıdan döken sistem görevi birlikte çalışan iki görevdir. Kullanıcı görevinin ürettiği çıktıların yazıcı sistem görevi tarafından dökülebilmesi, ilgili çıktıların bir görevden diğer göreve aktarılmasını gerektirir. Bu aktarım her iki görevin ortak erişebildiği bir ana bellek yastık alanı üzerinden gerçekleşebilir. Kullanıcı görevi, dökümü alınacak çıktıyı, ortak yastık alanıyla aynı boyda tutanaklar biçiminde, sırayla bu yastık alanına yazar. Yazıcı sistem görevi de, bu alandan okuduğu tutanakların yazıcıdan dökülmesini sağlar. Ortak yastık alanı üzerinden veri alış-verişi yapan görevlerden yastık alanına yazma yapan görev üretici, bu alandan okuma yapan görev ise tüketici olarak nitelenir. Yukarıdaki örnekte, bu nedenle, kullanıcı görevi üretici, yazıcı sistem görevi ise tüketici görev konumundadır (Çizim 4.1).



Çizim 4.1. Birlikte Çalışan İki Görevin Gösterimi

Ortak bir yastık alan üzerinden veri alış-verişinde bulunan üretici ve tüketici görevler birbirlerinin işletimini etkileyen görevlerdir. Bu bağlamda, yukarıda verilen örnekte yazıcı sistem görevinin işletilebilmesi, kullanıcı görevinin dökülecek bir tutanağı ortak yastık alan üzerinde hazır etmesine bağlıdır. Bunun gibi kullanıcı görevinin, yeni bir tutanağı yastık alanına yazabilmesi, bir önce yazılan tutanağın yazıcı görev tarafından okunup dökülmüş olmasını gerektirir. Birlikte çalışan görevlerin aralarındaki etkileşim, çoğu kez kaynak paylaşımından doğar. Kullanıcı ve yazıcı sistem görevleri, çıktı

tutanaklarının yazılıp okunduğu ana bellek yastık alanını paylaşmaktadırlar. Paylaşılan kaynak ortak kaynak olarak adlandırılır. Birlikte çalışan görevler için ortak kaynaklar bir değişken, bir ana bellek yastık alanı, bir disk tutanağı, tümüyle bir kütük ya da bir giriş/çıkış birimi olabilir.

Bilgisayar sistem kaynaklarına, nitelikleri ne olursa olsun değişik sınıflandırmalar uygulanır. Bu sınıflandırmalardan biri, kaynakları:

- bölüşülür kaynaklar,
- bölüşülmez kaynaklar

olmak üzere ikiye ayıran sınıflandırmadır. Bölüşülür kaynak, bir görev tarafından kullanımı tümüyle tamamlanmadan diğer görevler tarafından da kullanılabilen (tüketilebilen) kaynaktır. Bu bağlamda ana işlem birimi, ana bellek, disk birimi gibi donanımsal kaynaklar bölüşülür kaynaklardır. Örneğin ana işlem birimi, birden çok görev tarafından sırayla paylaşılabilen; bir görevin bu birimle ilgili kullanımı sonlanmadan bir diğer görevin kullanımına da sunulabilen bir kaynaktır. Bunun gibi, ana bellek de, birden çok amaç programı aynı anda saklayabilen bölüşülür bir kaynaktır. Birden çok programın, aynı anda ana bellekte saklanıyor olması programların işletimini etkilemez. Bunun tersine yazıcılar bölüşülür kaynaklar değildir. Zira bir görevle ilgili döküm yapılırken bu döküm kesilerek diğer bir döküme geçilemez. Başka bir deyişle, yazıcılar bir görev tarafından kullanılmaya başlandığında, bu kullanım tümüyle sonlanana değin diğer görevlerin kullanımına sunulamaz.

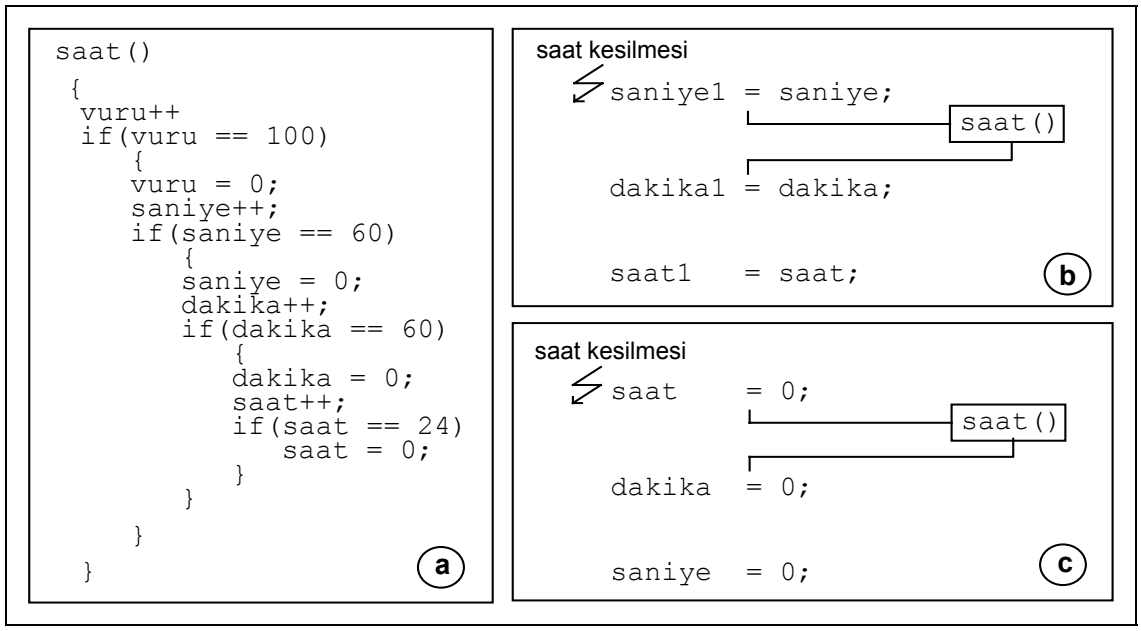
Birlikte çalışan görevlerce paylaşılan değişken, yastık, tutanak, kütük, sürücü gibi kaynakların bölüşülür olup olmama özellikleri bu kaynaklar üzerinde yapılan işlemin türüne de bağlıdır. Kaynaklar üzerinde yapılan işlem türü ikidir. Bunlar okuma ve yazma işlemleridir. Birlikte çalışan iki görev, ortak kaynağa, tüm olasılıklar gözönüne alındığında:

- okuma - okuma,
- okuma - yazma,
- yazma - okuma,
- yazma - yazma

işlem çiftlerini gerçekleştirmek üzere erişirler. Bu işlem çiftlerinden okuma-okuma çifti için kaynak her zaman bölüşülür özellik gösterir. Ancak diğer işlem çiftleri için kaynaklar bölüşülür değildir.

Bölüşülmez kaynaklar kritik kaynaklar olarak adlandırılır. Kritik kaynakların görevler tarafından paylaşılması önlem alınmadan yapılamaz. Kaynağa erişmeden önce kaynağın kullanım durumunun sınanması ve diğer görevlerin o anda kaynak üzerinde işlem yapmadıklarından emin olunması gereklidir. Bunun yapılmaması durumunda birlikte çalışan görevlerin işletim bütünlüğü korunamaz. Bunu örneklemek üzere sistem gerçek zaman saati kesilmelerini işleyen `saat` adlı bir görev ile günün saatini görüntülemeye yarayan `göster` ve saati kurmada yararlanılan `öndeğerata` adlı üç değişik görev düşünülmüştür. Bu görevlerden `saat` görevi bir kesilme görevi olup her 10 ms'de bir oluşan ve sistem kesilme girişine bağlı saat vurularını sayarak `saniye`, `dakika` ve

`saat` adlı, her üç göreve ortak değişkenleri günlemektedir. `göster` adlı görev bu ortak değişkenlerin içeriklerini, yerel değişkenlerinin içine aktararak görüntüleme işlemlerini gerçekleştirmektedir. `öndeğerata` görevi ise `saniye, dakika, saat` değişkenlerine öndeğer atamak (bu değerleri kurmak) için kullanılmaktadır. `saat` adlı görevin algoritmik görünümü Çizim 4.2-a'daki gibidir.



Çizim 4.2. Birlikte Çalışan Görevler ve İşletim Bütünlüğü

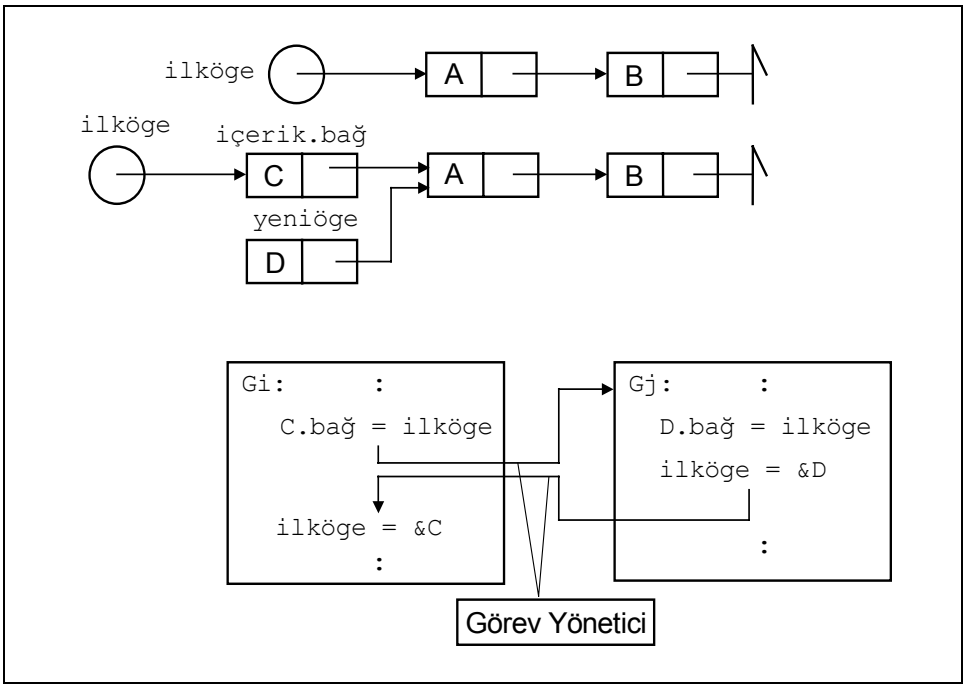
`saat`, `göster` ve `öndeğerata` adlı görevlerin, herhangi bir önlem alınmadan birlikte işletimi sözkonusu olduğunda:

- Saat `07:05:59` iken ve `göster` adlı görev yerel değişkenlerini `saniye, dakika, saat` genel değişkenleriyle günlerken, Çizim 4.2-b'de gösterildiği biçimde saat vurusu kesilme ürettiğinde; `07:06:00` yerine `07:06:59` görüntülenir.
- `saat:dakika:saniye` değerleri `3:59:59` iken ve `ilkdeğerata` adlı görev `saat, dakika` ve `saniye` değişkenlerini sıfırlarken, Çizim 4.2-c'de gösterildiği anda `saat` görevi işletime alınırsa `00:00:00` öndeğeri yerine `01:00:00` öndeğeri atanmış olur.

Bunun gibi, tek bağlı liste yapısını koşut olarak günleyen iki değişik görevin bu listeye, hiçbir önlem alınmadan öge eklemesi durumunda da işletim bütünlüğü kolayca bozulabilir. Sözkonusu tek bağlı listeye, {`yenioge.içerik; yenioge.bağ`} yapısındaki `yenioge`,

```
yenioge.bağ = ilköge;
ilköge   = &yenioge;
```

algoritmasıyla eklenebilir. `ilköge` liste başını gösteren değişkendir. Bu listeye `C` ve `D` adlı yeni ögelerin G_i ve G_j adlı görevler tarafından koşut olarak eklendiği; G_i görevi bu eklemeyi yaparken Görev Yönetici tarafından işletimi kesilerek G_j 'nin işletime alındığı varsayılırsa G_j tarafından listeye eklenen `D` adlı ögenin listede görülemeyeceği durum ortaya çıkar (Çizim 4.3).



Çizim 4.3. Tek Bağlı bir Liste üzerinde Görevlerin Koşut Günleme Yapması

Verilen her iki örnekten de kolayca anlaşılacağı üzere, birlikte çalışan görevlerin kritik kaynaklara erişimi rasgele yapılamaz. Birden çok görev kritik kaynağı paylaşmak istediğinde, aralarında yalnız birinin kaynağı sahiplenmesine yol açacak önlemlerin alınması, bu görevlerin işletim bütünlüğünün korunması için gereklidir. Bunun için bir görevin kritik kaynak üzerinde işlem yaptığı bir anda diğer görevlerin de bu tür istemleri sözkonusu olursa, bu istemlerin, kaynak serbest kalana dek ertelenmelesi gereklidir. Kritik kaynağa erişmeden önce, kaynağın kullanım durumunun sınanması, kaynak kullanılıyor ise amaçlanan erişimin ertelenmesi görevler arası zamanuyumlama olarak bilinir.

Kritik kaynağa erişim yapan program kesimleri kritik kesimler olarak adlandırılır. Kritik kaynağa aynı anda en çok bir görevin erişebilmesi kritik kaynağa erişim yapan görevlerden en çok birinin kritik kesimi içinde bulunmasının sağlanmasıyla gerçekleşebilir. Kritik kesimi içinde işlem yapan bir görevin diğer görevlerin aynı kaynağa erişim yapan kritik kesimlerine girmelerini engellemesi gerekir. Bu engelleme karşılıklı dışlama olarak tanımlanır. Başka bir deyişle görevler arası zamanuyumlama, görevler arası karşılıklı dışlama yoluyla gerçekleşir. Karşılıklı dışlama görevlerin kritik kesimlerine giriş ve çıkışlarında alınan önlemler yoluyla gerçekleştirilir.

Birlikte çalışan görevlerin işletim bütünlüğünün korunması için salt karşılıklı dışlamanın gerçekleştirilmesi yeterli olmaz. Karşılıklı dışlamanın sağlanmasının yanı sıra karşılıklı tıkanmanın da engellenmesi gerekir. Karşılıklı tıkanma, görevlerin, birbirlerinin işletimlerini karşılıklı olarak, sürekli engellemelerine verilen addır.

İşletim bütünlüğünün korunmasına yönelik önlemlerin yanı sıra sistem başarımını gözeten kıstaslar da göz önüne alındığında görevler arası zamanuyumlamada uyulması gereken kurallar ortaya çıkar. Bu kurallar özetle, aşağıdaki gibi sıralanabilir:

1. Birlikte çalışan görevlerden en çok biri, aynı anda kritik kesimine girebilmelidir (karşılıklı dışlamanın sağlanması).
2. Aynı anda ortak kaynağa erişmek isteyen görevlerden en az biri, sonlu bir süre için kritik kesimine girebilmelidir (karşılıklı tıkanmanın engellenmesi).
3. Kritik kesiminin dışındaki görevler ortak kaynağa erişmek isteyen görevleri engellememelidir.
4. Görevlerin göreli hız ve öncelikleri ile sistemde yer alan ana işlem birimi sayısı hakkında herhangi bir varsayım yapılmamalıdır.

Bu kurallar ışığında görevler arası zamanuyumlamanın ele alınışı izleyen kesimde açıklanmıştır.

4.2. Görevler arası Zamanuyumlama

Görevler arası zamanuyumlamanın gerçekleştirilmesinde kullanılan yöntemler, alt düzey ve üst düzey zamanuyumlama yöntemleri olarak iki kategoride düşünülür. Alt düzey zamanuyumlama yöntemlerinden kimileri hiçbir özel donanım desteği gerektirmezken kimileri de, özel makina komutları ya da işletim sistemi çağrılarının kullanımını gerektirirler. Üst düzey olarak nitelenen zamanuyumlama yöntemleri ise, genellikle alt düzey araçlar üzerine kurulurlar.

4.2.1. Özel Donanım Desteği Gerektirmeyen Yöntemler

a. Kesilme Düzeneğinin Kullanımı

Görevler arası zamanuyumlamayı sağlamak üzere akla ilk gelen yol kesilme düzeneğinin kullanılmasıdır. Karşılıklı dışlamayı sağlamak üzere, bir görev kritik kesimine girerken sistemi kesilmelere kapayabilir. Bu yolla ana işlem birimi, bu görev tarafından, kritik kesim çıkışında yeniden kesilmelere açılıncaya kadar, görev yönetici adlı sistem görevi de dahil başka görevlere anahtarlanamaz. Böylece bu görevin birlikte çalıştığı görevlerin ana işlem birimine anahtarlanarak kritik kesimlerine girmeleri engellenmiş olur. Çizim 4.2'de verilen `göster`, `öndeğerata` ve `saat` adlı birlikte çalışan görevler arası karşılıklı dışlamanın sağlanması bu yolla gerçekleştirilmek istenirse `göster` ve `öndeğerata` adlı görevler içinde, ortak değişkenlere erişim yapan komut dizilerinin, `disable-interrupt (di)`, `enable-interrupt (ei)` komutlarıyla ayraç arasına alınması; `saat` adlı görevin ise, tümüyle kesilmelere kapalı olarak işletilmesi gereklidir.

```
di;                    di;
saniye1=saniye;        saat=0;
dakika1=dakika;        dakika=0;
saat1=saat;            saniye=0;
ei;                    ei;
```

`göster` ve `öndeğerata` adlı yordamlarda sistemin, zamanuyumlama amacıyla kesilmelere kapatıldığı süreler üç aktarım komutunun işletim süresi kadardır. Bu komutların herbirinin işletim süresinin yüz nanosaniye olduğu varsayılırsa sistemin kesilmelere kapalı kaldığı süreler yarım mikrosaniyeyi geçmez. Salt saat kesilmeleri yönünden bakıldığında, her 10 milisaniyede bir gelen saat vuruları için bu durum bir sakınca oluşturmaz.

Ancak, birlikte çalışan görevler arası zamanuyumlamanın gerçekleştirilmesinde kesilme düzeneğinin kullanılması genelleştirilebilir bir yöntem değildir. Sistemin zamanuyumlama amacıyla ortak kaynağa erişim yapan komut dizileri için kesilmelere kapatılması, sistemin kesilmelere kestirilemez uzunlukta kapalı kalmasına ve işletim bütünlüğünü diğer yönlerden zedeleyecek sakıncaların doğmasına neden olabilir. Yukarıdaki örnekte yarım mikrosaniye olarak belirlenen sistemin kesilmelere kapalı kalma süresinin daha uzun olması, saat vurularının da, 10 milisaniye yerine 10 mikrosaniyede bir geldiği varsayıldığında kimi saat vurularının sayılamadığı sakıncalı durumun ortaya çıkacağı kolayca görülebilir. Bu da, sistemin, sayışım yönünden işletim bütünlüğünü tümüyle bozan bir durumdur.

Kritik kesimlerin kesilmelere kapalı olarak işletilmesi ana işlem biriminin tüm dış uyarılara kapatılması anlamına gelir. Bunun yapılması, bu uyarılar aracılığıyla ana işlem birimine anahtarlanarak çalışacak ve sistemin kesilmelere kapatılmasını gerektiren kritik kaynakla ilgisi olmayan görevlerin de işletimlerini engeller. Bu da sistemin genel başarımını olumsuz yönde etkiler. Bu durumda, zamanuyumlamayı kurmak amacıyla kesilme düzeneği ile oynamak, köktenci ve kaba bir yöntem olarak ortaya çıkar.

Bu yöntemin en önemli sakıncası, kesilme düzeneğinin denetimini sistemde çalışan çok sayıda göreve, özellikle çekirdek katman dışındaki görevlere de dağıtmasıdır. Kesilme düzeneğinin tutarlı yönetimi güvenli bir işletim için vazgeçilmez bir gerekliliktir. Sistemin kesilmelere açılıp kapatılması gibi temel bir aracı her düzeyden (ayrıcalıklı ve sıradan) görevin inisyatifine bırakmak, kesilmelerin yönetimini bu görevlerle paylaşmak anlamına gelir. Sistemde birlikte çalışan görevleri her düzeyden kullanıcının tanımlayabileceği düşünüldüğünde bunun, güvenli işletim ve koruma ilkeleri yönünden çok sakıncalı bir yaklaşım olduğu kolayca görülür. Bu gerekçelere dayalı olarak kesilme düzeneğinin görevler arası zamanuyumlamada kullanılması, geçerli bir yol olarak benimsenemez.

b. Algoritmik Yaklaşım

Birlikte çalışan görevlerin kritik kesimlere giriş ve çıkışlarında zamanuyumlamanın, görev algoritmalarına katılan sınamalar yolu ile gerçekleştirilmesi akla gelen diğer bir yoldur. Bu bağlamda, aynı kritik kaynağa erişen görevler programlanırken, örneğin,

kaynağın kullanım sırasını belirleyen bir gösterge (sıra göstergesi) öngörülür. Ortak kaynağa erişmek isteyen bir görev kritik kesimine girişte bu sıra göstergesini (değişkenini) sınar. Eğer gösterge sıranın sınayan görevde olduğunu gösteriyorsa kritik kesime girilip ortak kaynağa erişilir. Bunun tersi olduğu sürece, sürekli sınamalarla, sıranın gelmesi beklenir. Kritik kesimine giren görev, çıkışta sıra göstergesini, kendisini izleyen görevin sırasını gösterecek biçimde günler. Açıklanan bu algoritma Çizim 4.4'te, birlikte çalışan iki görev için örneklenmiştir.

```
          GÖREV 1                              GÖREV 2

             .                                    .
while(sıra!=1); /*bekle*/          while(sıra!=2); /*bekle*/
kritik-kesim-1();                  kritik-kesim-2();
sıra=2;                            sıra=1;
             .                                    .
```

Çizim 4.4. Sıra Değişkeni ile Zamanuyumlama

Görevler arası zamanuyumlamada, uyulması gereken kurallar yönünden incelendiğinde, sıra değişkenini kullanan bu algoritmanın, karşılıklı dışlama ve karşılıklı tıkanma diye tanımlanan, işletim bütünlüğüyle ilgili kuralların gereğini yerine getirdiği ancak, yukarıda, üçüncü sırada verilen kuralı karşılamadığı görülmektedir. Zira bu algoritma ile görevler kritik kesimlerine "bir o bir bu" diye nitelenebilen değişmez bir sırada erişebilmektedirler. Bir görev kritik kesiminden çıkışta erişim sırasını diğer görev ya da görevlere bırakmakta, bu görevler kritik kesimlerine girmedikleri sürece bir daha kritik kesimine girme hakkı kullanamamaktadır. Bu da, bir görevin kritik kesiminin dışında olmasına karşın diğer görevlerin kritik kesimlerine girmelerini engelleyen bir durum yaratmaktadır. Bir tutanaklık yastık alanı üzerinden iletişim kuran üretici ve tüketici iki görev örneğinde olduğu üzere yürütülen uygulamanın türü, değişmez sırada işletimi gerektirmiyorsa, bu durum, bu yaklaşım için önemli bir sakıncayı oluşturur.

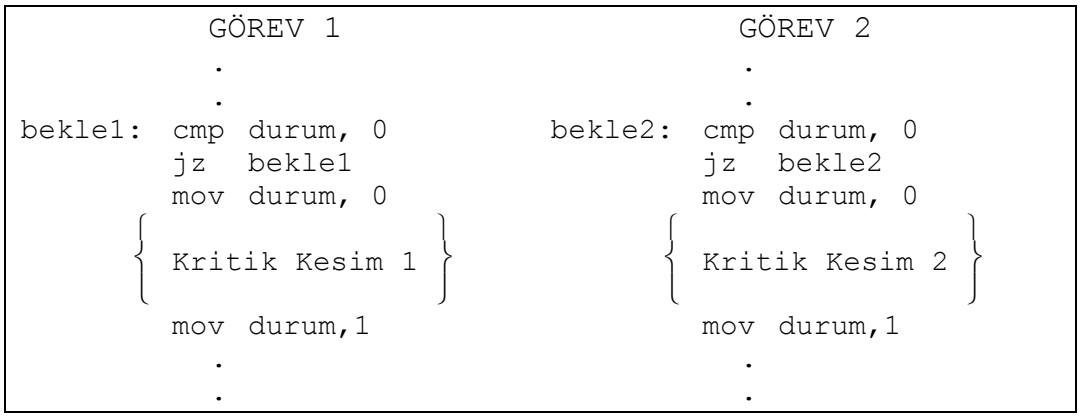
```
        GÖREV 1                              GÖREV 2

           .                                    .
  while(durum==0);/*bekle*/          while(durum==0);/*bekle*/
  durum=0;                           durum=0;
  kritik-kesim-1();                  kritik-kesim-2();
  durum=1;                           durum=1;
           .                                    .
```

Çizim 4.5. Çalışmayan Zamanuyumlama Algoritması

Görevleri ortak kaynağa değişmez bir sırada erişmeye zorlayan bu yaklaşımın aşılması amacıyla, sıra değişkeni yerine kaynağın kullanım durumunu gösteren ikili bir değişkenin kullanılması düşünülebilir. Bu durumda, sözkonusu değişkenin sıfır değerini içermesi kaynağın kullanılmakta olduğunu, bir değerini içermesi ise kaynağın serbest olduğunu simgeler. Ortak kaynağa erişmek isteyen görevler kritik kesimlerine girmeden önce bu değişkeni sınarlar. Değişken içeriğinin sıfır bulunması durumunda,

kritik kesim girişinde bekleme sürdürülür. Değişken içeriğinin bir bulunması durumunda ise içerik sıfırlanarak kritik kesime girilir. Kritik kesim çıkışında değişken içeriği kurularak (birlenerek) diğer görevlerin kaynağa erişebilmesi sağlanır. Çizim 4.5'te sözkonusu bu algoritma, iki görev için örneklenmiştir.

İlk bakışta sıra değişkenini kullanan algoritmaya benzeyen ve görevler arası zamanuyumlamayı sağlayacağı sanılan bu yöntem, ne yazık ki gerçekleştirime ilişkin nedenlerle karşılıklı dışlamayı sağlayamaz. C dili benzeri algoritmik bir dille, Çizim 4.5' te verilen algoritmanın, *ASM86* simgesel diline dönüştürülmüş (ya da derlenmiş) biçimi düşünüldüğünde Çizim 4.6'da verilen örnek elde edilir. Bu örnekte `durum` değişkeninin sınanması `cmp` komutu ile yerine getirilmektedir. `cmp durum, 0` komutu işletilirken `durum` değişkeninin içeriğinin sıfır olması (program durum yazmacı içinde sıfır bitinin kurulu olması) durumunda, kritik kesime girmek yerine sınamayı yeniden yapmak, başka bir deyişle, beklemeyi sürdürmek üzere karşılaştırma komutuna geri dönülmektedir.



Çizim 4.6. Durum Değişkenini kullanan Algoritmanın Gerçekleştirimi

`durum` değişkeninin içeriği bir (kaynak kullanımı serbest) iken, `GÖREV 1`'in işletimi `cmp` komutunun işletiminden hemen sonra kesilip `GÖREV 2`'nin ana işlem birimine anahtarlandığı düşünülürse, bu son görevin de, durum değişkeni içinde bir değerini bularak kritik kesimine hiç beklemeden gireceği söylenebilir. Bu kez, `GÖREV 2`'nin işletiminin kritik kesimi içinde kesilip `GÖREV 1`'in yeniden işletime alındığı düşünülecek olursa, bu görevin, `jz bekle1` komutundan başlayarak işletileceği görülür. `jz bekle1` koşullu sapma komutu bir önceki işletimde, durum değişken içeriği bir iken kurulan ve `GÖREV 1` ana işlem birimine anahtarlandığında program durum yazmacını günleyen sıfır durum bitine dayalı olarak uygulanır. Böylece `GÖREV 1` için de, `bekle1` konumuna sapılmadan kritik kesime girileceği görülür. Bu, her iki görevin de kritik kesimlerini birlikte işletmeleri (ortak kaynağa eşanlı erişmeleri), başka bir deyişle karşılıklı dışlama kuralının ihlali anlamına gelir. Bu gerekçeye dayalı olarak Çizim 4.5 ve Çizim 4.6'da verilen zamanuyumlama algoritmalarının geçerli olmadığı söylenir. Durum değişkenini sınama ve sıfırlama işlemlerinin tek bir adımda yapılamamasından kaynaklanan bu duruma, *(test-and-set)* "sına-ve-kur" türü komutlar açıklanırken yeniden geri dönülecektir.

Çizim 4.7. Görevler arası Zamanuyumlamada *Dekker* Algoritması

`sıra` adlı değişkeni kullanan algoritmanın birlikte çalışan görevleri değişmez sırada işletime zorlaması, `durum` adlı değişkeni kullanan algoritmanın ise, ilk elde aranan karşılıklı dışlamayı bile sağlayamaması birçok kişiyi yazılımsal yolla görevler arası zamanuyumlamayı, yukarıda verilen dört kurala da uyarak sağlayacak algoritma arayışına itmiştir. Bunlardan *Dekker*, ilk kez matematiksel doğrulukta bir algoritmayı öneren matematikçi olmuştur. *Dekker*'in önerdiği algoritma, birlikte çalışan iki görev için Çizim 4.7'de verilmiştir. Bu algoritmaya göre birlikte çalışan iki görevin kritik kesimlerine eşanlı olarak girmeleri, `koşul1` ya da `koşul2`'nin, ilgili görevin kritik kesimine girmesinden önce mutlaka `false` olmasından dolayı olanaksızdır. Karşılıklı tıkanma, `sıra`'nın kritik kesime girmezden önceki karar bölümünde içerik değiştirmemesi sayesinde önlenmektedir. Çizim 4.7'de verilen birlikte çalışan görevler işletime alınmadan önce `koşul1` ve `koşul2`'nin `false` değerine kurulması gereklidir.

Yukarıda açıklanan ve algoritmik diye adlandırılan yaklaşımla, karşılıklı dışlama ve karşılıklı tıkanma gibi yalın sayılabilecek kavramsal kuralları uygulamaya sokmada karmaşık, anlaşılması pek de kolay olan bir çözüme ulaşılmıştır. Bu zorlama, daha çok eldeki makina komutların yetersizliğinden kaynaklanmıştır. Çizim 4.5'te verilen ve çalışmayan algoritmayı çalışır kılmanın yollarını aramak, *Dekker* Algoritmasıyla ulaşılan zorlama ve karmaşıklığı ortadan kaldıracaktır. Daha önce de belirtildiği gibi

Çizim 4.5'te verilen algoritmanın doğru çalışmaması, sınama ve sıfırlama işlemlerinin tek bir adımda yapılamamasından kaynaklanmaktadır. Anılan bu işlemleri tek bir adımda yapmayı sağlayan komutların, makina komutları arasına katılarak kullanılması, izleyen kesimde açıklanan ilk yöntem kapsamında ele alınacaktır.

4.2.2. Donanım Desteği Gerektiren Alt Düzey Araçlar

a. test-and-set türü Komutların Kullanımı

Görevlerin kritik kesimlerine, ortak bir ikili değişkeni sınayarak girmelerini ilke olarak benimseyen ve zamanuyumlama sorununa yalın bir biçimde yaklaşan yöntemin (Çizim 4.5) doğru çalışabilmesi, bu ikili değişkeni sınama ve gerektiğinde kaynağın kullanımda olduğunu belirten değerle günleme işlemlerinin bölünmez bir biçimde, tek adımda yapılabilmesine bağlıdır. Çizim 4.6'da, simgesel dil ile verilen algoritmada sınama ve sıfırlama işlemleri:

```
bekle-i: cmp  durum, 0
         jz   bekle-i
         mov  durum, 0
```

komut dizisi ile gerçekleşmektedir. Bu komut dizisinin bölünmezliği, bunları içeren görevin işletiminin bu komutlar arasında kesilememesi anlamına gelmektedir. Ana işlem biriminin bir görevden alınıp diğer bir göreve atanması kesilme uyarılarına dayalı olarak yapıldığından anılan bölünmezlik ana işlem biriminin bu komutlar işletilirken kesilememesi olarak da yorumlanabilir. Sistemin, bu komutlar işletilirken kesilmelere kapatılması bölünmezlik sorununu çözebilir. Ancak kesilme düzeneğinin denetimini sıradan görevlere de bırakmak anlamına gelen bu çözüm yolu, daha önce de belirtildiği üzere benimsenen bir yol değildir. Bunun yerine, zamanuyumlama değişkenleri üzerindeki sınama ve günleme işlemlerini tek bir adımda yerine getirecek özel komutlar, sistem tasarımı aşamasında makina komutları arasına katılır. Bu komutlar (*test-and-set),* "sına-ve-kur" türü komutlar olarak adlandırılır.

test-and-set türü komutlar genelde iki işlenen içerirler. Bu işlenenlerden biri (`durum`), çoğu kez ortak kaynağın kullanım durumunu gösteren ve birlikte çalışan görevlerin hepsinin paylaştığı genel bir ana bellek değişkenidir. Diğer işlenen ise (`reg`) göreve özel bir yazmaç olabilir.

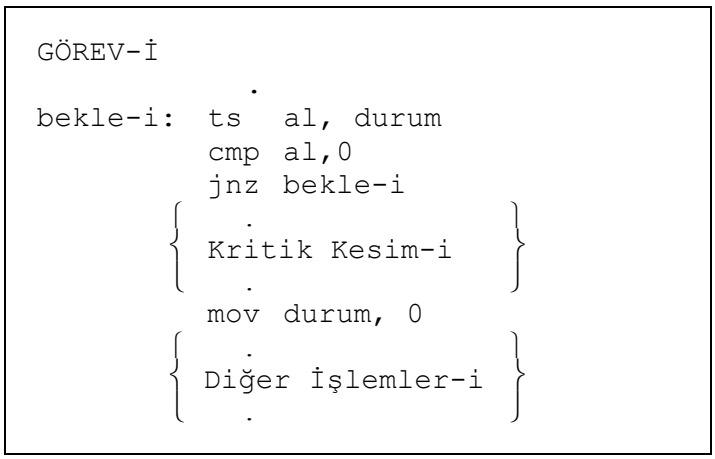
```
ts   reg, durum
```

`ts` komutu `durum` adlı değişkenin içeriğini `reg` adlı yazmaca aktarırken, aynı zamanda bu değişkenin içeriğini sıfırdan farklı bir değere kurar. Bu komut sonrasında `reg` adlı yazmaç, `durum` değişkenin içeriğini taşırken `durum` değişkeni de, bir önceki içeriği ne olursa olsun sıfırdan farklı bir değer taşır. `ts al,durum` türü bir makina komutunun *80X86* türü işleyicilerin makina komutları arasında yer aldığı varsayılırsa[13], kullanımı Çizim 4.8'de verilen biçimde örneklenebilir.

---

[13] *80X86* türü işleyicilerde `ts reg,durum` komutu bulunmamaktadır. Bu varsayım salt bu işleyicilerin simgesel dilini kullanarak *test-and-set* komutlarının kullanımını örneklemek için yapılmıştır.

Bu örnekte bir görev kritik kesimine girmeden önce `ts al,durum` komutu ile ortak kaynağın kullanımda olup olmadığını belirten `durum` adlı gösterge içeriğini, göreve özel `al` yazmacına aktarmakta, eşanlı olarak `durum` değişkeninin içeriğini de, örneğin `ffh` değerine kurmaktadır. Bu komut sonrasında `al` yazmacı sıfır değeriyle karşılaştırılarak ortak kaynağın kullanılıp kullanılmadığı sınanmaktadır. `al` içinde sıfır değerinin bulunması kaynağın kullanılmadığını göstermektedir. Bu durumda eğer `al` içinde sıfırdan farklı bir değer bulunursa, sınamayı yinelemek (beklemek) üzere `ts al,durum` komutuna geri dönülmekte, sıfır bulunması durumunda ise kritik kesime girilmektedir. Ortak kaynağın serbest olduğu bir anda, kritik kesimlerine girmek üzere `durum` değişkenini eşanlı olarak sınayan görevlerden sınamayı ilk başlatan görev `durum` içeriğini sıfır bulmakta ve bölünmez olarak hemen bunu sıfırdan farklı bir değere günlemektedir. Böylece `durum` değişkeni üzerinde eşanlı sınama yürüten diğer görevlerin kritik kesimlerine girmeleri önlenmektedir. Kritik kesim çıkışında `durum` değişkeninin içeriğinin, klasik `mov` türü bir komutla sıfırlanması, ortak kaynağı serbest bırakmak için yeterli olmaktadır.



Çizim 4.8. `test-and-set` Komutunun Kullanımı

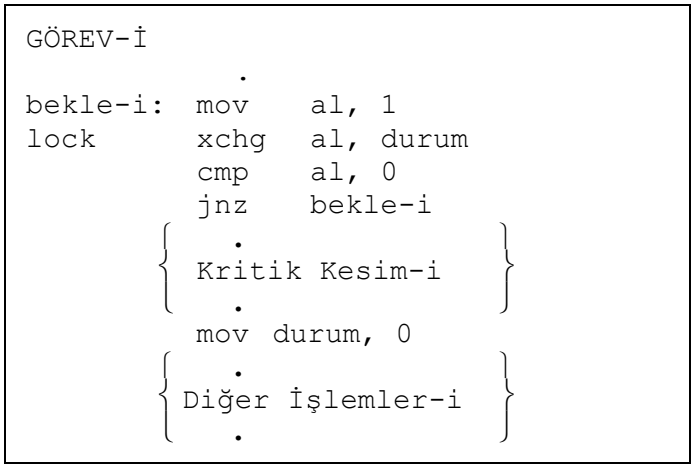
`ts` türü komutlar ilk kez *IBM System/360* serisi bilgisayar sistemlerinde, gerek işletim sistemi geliştirmeye gerekse çok işleyicili sistem mimarilerine destek verebilmek amacıyla kullanılmıştır. Günümüzde hemen hemen tüm işleyicilerde `ts` türü komutlara ya aynı adla ya da başka adlarla yer verilmektedir. Bunlardan örneğin *Motorola 680X0* serisi işleyicilerin `tas (test-and-set)`, *Intel 80X86* serisi işleyicilerin ise `xchg (exchange)` komutları bulunmaktadır.

`xchg dest,src` komutu, `ts` komutu örneklenirken belirtildiği gibi, iki işleneni olan bir komuttur. Bu işlenenlerden biri mutlaka bir yazmaç, diğeri ise herhangi bir değişken ya da başka bir yazmaç olabilmektedir. Komut işletildiğinde:

```
temp <-- dest
dest <-- src
src  <-- temp
```

işlemlerini bölünmez biçimde gerçekleştirmektedir. `temp`, işleyicinin programlanır yazmaçları arasında yer almayan ancak geçici olarak değer saklamaya yarayan bir

yazmacını simgelemektedir. `xchg dest,src` komutunun, birlikte çalışan görevlerin kritik kesimlerine girişini denetlemede kullanımı Çizim 4.9'da örneklenmiştir. Bu örnekten `xchg` komutu ile `ts` komutu arasındaki tek ayrımın, `ts` komutunun ortak değişkeni (`durum`'u) kendiliğinden, sıfırdan farklı bir değere kurması olduğu görülür. `xchg` komutu ise `durum`'u, değiş tokuş edilen yazmacın içeriği ile günlemektedir. Buradan her iki komutun da özdeş işlemler gerçekleştirdikleri kolayca anlaşılır.



Çizim 4.9. `xchg` Komutunun Kullanımı

`ts` ve `xchg` komutlarının durum adlı genel değişken ile kullanımı, tek ana işlem birimi içeren sistemlerde görevler arası zamanuyumlamayı gerçekleştirmede yeterlidir. Ancak görevler birden çok ana işlem biriminin bulunduğu çok işleyicili bir sistemde çalışıyorsa, bu komutların örneklerde verilen biçimde kullanımları, karşılıklı dışlamayı sağlamada yetersiz kalır. Çok işleyicili sistemlerde çoğu kez bellek, tüm işleyicilerin erişebildiği ortak bellek ve işleyicilere özel yerel belleklerden oluşan düzendedir. İşleyicilerin ortak belleğe erişimleri, adres ve veri hatları ile kimi ilgili denetim imlerinden oluşan ortak bellek yolunun kullanım istemlerini ele alan bir hakem donanımın *(bus arbitration logic)* denetiminde koşut biçimde gerçekleşir. Bir işleyici ortak belleğe erişirken, (aynı doğrudan bellek erişim denetleme birimiyle olduğu üzere) diğer bir işleyici bunun erişimini bölerek aynı bellek sözcüğüne koşut olarak erişebilir. Bu durumda, ortak değişken üzerinde işlem yapan `test-and-set` komutunun işletim bütünlüğü bozulur. `durum` adlı ortak değişkenin sıfır değerini içerdiği bir aşamada bir görev, `xchg` komutunu işleterek bu değişkenin içeriğini `temp` içine aldıktan hemen sonra, bu görevin anahtarlı olduğu işleyicinin bellek erişimi bölünerek, başka bir işleyici, başka bir görev için, yine `xchg al,durum` komutunu işletmek üzere aynı durum değişkenine erişirse, değişik işleyiciler üzerinde çalışan her iki görev de `durum` içinde sıfır değerini okumuş olur. Bu, her iki görevin de kritik kesimlerine birlikte girmeleri, başka bir deyişle karşılıklı dışlama kuralının ihlali demektir.

Bu nedene dayalı olarak, çok işleyicili sistemlerde ortak bellek üzerinde koşut erişilen zamanuyumlama değişkenlerinin eşanlı (içiçe) erişimlere karşı korunması gereklidir. Çok işleyicili sistemlerde kullanılabilecek işleyicilerde, bu amaca dönük kimi araçlar

bulunur. *80X86* serisi işleyicilerde, `lock` olarak adlandırılan ve komutların başına önek olarak konulan sözde komut bu amaçla kullanılır. `lock` önekini alan komutun işletimi sırasında, bu komutu işleten işleyicinin bellek erişim döngüsünün bölünmesine izin verilmez. `lock` sözde komutu simgesel derleyici tarafından, `F0H` kodunun, ilgili komut kodunun önüne yerleştirilmesini sağlar. `F0H` özel kodu, bu kodu işleten işleyicinin, izleyen komutun işletimi boyunca diğer işleyicilere yol kullanım hakkı tanımasını engeller. Çizim 4.9'da verilen örnekte, `xchg al,durum` komutunun önüne `lock` öneki konarak bu komutun işletimi boyunca, ilgili işleyicinin bellek erişim döngüsünün bölünmesi, dolayısıyla karşılıklı dışlama kuralının ihlali riski ortadan kaldırılmıştır.

80386'dan başlayarak *Intel 80X86* işleyicilerinde `xchg` komutu, 8086'dan farklı olarak `lock` öneki konsun ya da konmasın hep bellek erişim döngüsü bölünmeyecek biçimde işletilmektedir. Bu nedenle, bu işleyicilerde `lock` sözde komutu bulunmakla birlikte `xchg` komutu önek gerektirmemektedir. Sözkonusu önek, bellek erişim döngüsünün kilitlenmesini gerektirebilen `bt (bit test), bts, btr, btc, add, xor, inc, dec` gibi diğer komutlarla kullanılmaktadır.

Motorola 680X0 serisi işleyicilerde, genel değişkenleri sınama ve kurmayı tek adımda gerçekleştiren `tas (test-and-set)` komutu bulunmaktadır. Bu komut da, aynı `xchg` komutunda olduğu gibi, işleyici yol kullanım izinlerine kapalı olarak işletilmekte ve önek gerektirmemektedir.

`ts` komutları çoğu kez, bellek erişiminin kilitlenmesini kendiliğinden yerine getiren komutlar olarak `tsl (test and set lock)` bellenir simgesiyle de anılmaktadırlar. `tas, xchg` gibi komutların yanı sıra, işletim ve bellek erişim bölünmezliği sağlanmış örneğin `increment memory` gibi, görevler arası zamanuyumlamayı kurmada yararlanılabilen diğer başka komutlar da bulunur. *IBM System/370* serisi bilgisayar sistemleriyle ilk kez kullanılan `compare-and-swap` komutu da, bu konuda verilebilen diğer bir örneği oluşturur.

b. İlkel Semafor İşleçlerinin Kullanımı

1960 yıllarda, birlikte çalışan görevler arası zamanuyumlamayı gerçekleştirecek algoritma arayışlarının tatmin edici sonuçlar vermemesi[14] bu zamanuyumlamayı sağlamada kullanılacak kimi ilkel işleçlerin kuramsal olarak tanımlanmasını gerekli kılmıştır. İlk kez 1965 yılında *Dijkstra*, iki ya da daha çok görev arasında karşılıklı dışlamayı sağlayacak, semafor[15] *(S)* adlı, özel, artı tamsayı türündeki değişkenler üzerinde işlem yapan *P (S)* ve *V (S)* adlı ilkel işleçleri tanımlamıştır. Bu tanıma göre *P(S)*; bölünmez (atomik) biçimde *S* değişkenini sınayan eğer *S* sıfırdan büyükse içeriğini bir eksilten, değilse beklemeyi sağlayan bir işleç tanımı içerir. *V(S)* ise *S*

---

[14] Bu yolla ulaşılan *Dekker* Algoritması matematiksel yönden tam doğru olmakla birlikte en çok iki görevin zamanuyumlamasında kullanılabilmekte, yalın bir biçimde ikiden çok göreve genelleştirilememektedir. Bu önemli kısıtlamadan dolayı pratikte uygulanamamıştır.

[15] *Semaphore*, Latince kökenli dillerde deniz feneri anlamına gelmektedir.

değişkenini bir artırmayı sağlayan bir işleç olarak tanımlanır. Bunların ilkel işleçler olmaları daha alt düzey işlemlerle ifade edilemedikleri anlamına gelir. İlkel işleçler doğal olarak bölünemezler. *P(S) ve V(S)*'nin algoritmik tanımları aşağıda verilmiştir:

```
P (S)                                V (S)
int *S;                              int *S;
 {while(*S <= 0); /*bekle/           {(*S)++; }
 (*S)--; }
```

P (S) ve V (S) işleçlerinin görevler arası zamanuyumlamada kullanımı, kritik kesimlerin bu işleçlerle ayraç arasına alınması yoluyla olur:

```
.
P (S);
{Kritik Kesim }
V (S);
.
```

Bir görev kritik kesimine girmeden önce *P(S)* işlecini çalıştırır. *S*'nin içerdiği değere göre ya kritik kesimine girer ya da bu işleç üzerinde bekler. Kritik kesim çıkışında mutlaka *V(S)* işlecinin çalıştırılması gerekir. Bu tanımlara göre *S*'nin salt 0 ve 1 değerlerini alabildiği görülür. Salt 0 (meşgul) ve 1 (serbest) değerlerini almasına izin verilen semaforlara ikili semaforlar denir.

Yukarıda verilen tanımlara bakılarak *P(S)* işlecini, kullanım durumu *S* adlı ortak değişken ile simgelenen ortak kaynak serbest kalıncaya kadar beklemeyi sağlayan; *V(S)* işlecini ise, *S* değişkenini, ortak kaynağın kullanılır olduğunu belirleyecek biçimde günleyen işleçler olarak yorumlamak mümkündür. Bu yoruma dayalı olarak, *P(S)* ve *V(S)* işleçlerinin gerçekleştirimi söz konusu olduğunda, bunun *test-and-set* türü komutlarla yerine getirilebileceği görülür. Bu bağlamda Çizim 4.8 ya da 4.9'da verilen örneklerin, *P(S)* ve *V(S)* işleçlerinin birer gerçekleştirimi olduğu söylenebilir.

Bu örneklerde, herhangi bir görev kritik kesimine girmeden önce semaforu sınamakta, ortak kaynağın meşgul olduğu durumlarda bu sınamaları yineleyerek kaynağın boşalmasını beklemektedir. Bu durumda, erişeceği kritik kaynak meşgul olan bir görev, sırası gelip ana işlem birimine anahtarlandığında kendisine ayrılan süre boyunca gerçekleşmeyecek bir koşulun oluşmasını bekleyerek, gereksiz yere ana işlem birimini meşgul edecektir. Zira kritik kaynak, onu kullanırken ana işlem birimini yitiren görevin tekrar ana işlem birimine anahtarlanarak çalışması sonucu serbest kalacaktır.

Sistem başarımını olumsuz yönde etkileyecek bu durumun ortadan kaldırılması için zaten kuramsal tanımları itibarıyla işletim sistemi işlevleri olarak düşünülen *P(S)* ve *V(S)* işleçlerinin gerçekleştiriminde, işletim sistemiyle etkileşimi taban alan bir yaklaşımın kullanılması gereklidir. Bu nedenle erişeceği kritik kaynak meşgul olan bir görevin, *P(S)* işlecini çalıştırması durumunda, gereksiz sınamalarla zaman öldürmek yerine bu kaynakla ilgili bir zamanuyumlama kuyruğuna bağlanarak ana işlem birimini

serbest bırakması daha uygun bir gerçekleştirim biçimi olacaktır. Bunun sonucunda *V(S)* işlecinin de, *S*'yi bir artırmanın yanı sıra *S*'ye ilişkin kaynağa erişebilmek üzere bekleyen bir görevi, ilgili zamanuyumlama kuyruğundan hazır görevler kuyruğuna aktarmayı sağlaması gerekecektir. Bu ilkeler çerçevesinde *P(S)* ve *V(S)* işleçlerinin uygulamaya dönük tanımları Çizim 4.10' da, *ASM86* simgesel dili ile uygulama örnekleri de Çizim 4.11' de verilmiştir.

```
P (S)
int *S;
 {
   if(*S >0) {(*S)--;}
   else
   ⎡Görevi S'ye ilişkin ⎤
   ⎨Bekleme kuyruğuna   ⎬
   ⎣bağla               ⎦

 }
```

```
V (S)
int *S;
 {
   (*S)++;
   if(S-kuyruğu!=boş)
    {
     (*S)--;
    ⎡S'ye ilişkin Bekleme  ⎤
    ⎨kuyruğundan bir görevi⎬
    ⎣Hazır kuyruğuna bağla ⎦
    }
 }
```

Çizim 4.10. P ve V İşleçlerinin Uygulamaya Dönük Tanımları

Çizim 4.11'de verilen uygulama örneğinde, *P (S)* ve *V (S)* işleçleri birer makro biçiminde ele alınmıştır. `durum` ile korunan ortak kaynağın meşgul olduğu bir sırada `P(durum)` makrosunu çalıştıran görevin `suspend` adlı diğer özel bir makro ile, `durum` semaforuna ilişkin kuyruğa bağlanacağı; `V(durum)` makrosunu çalıştıran görevin ise, `durum` semaforuna ilişkin kuyruğun boş olmaması durumunda, kuyruk başındaki görevi hazır görevler kuyruğuna `resume` makrosu ile aktaracağı düşünülmüştür. Bu bağlamda `int 1ch` yazılım kesilmesinin, işletim sistemi için, zamanuyumlama kuyruklarıyla ilgili hizmetlere atandığı; bu kesilmenin `al` yazmacı içinde aktarılan:

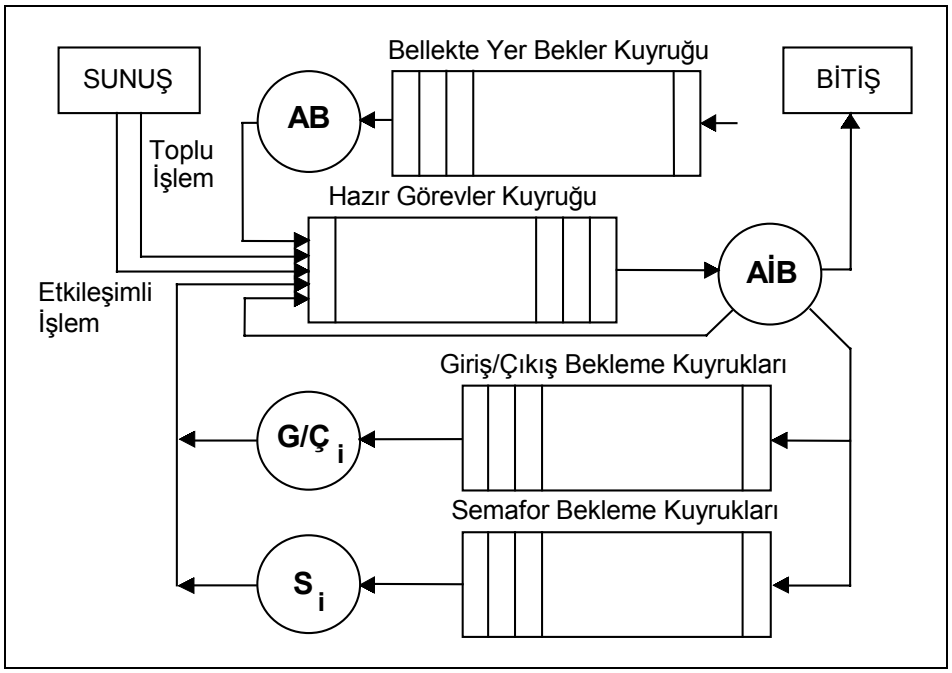
- 05 parametresiyle, kimliği verilen görevi, yine kimliği verilen bekleme kuyruğuna bağlamayı sağladığı;
- 06 parametresiyle, kimliği verilen görevi, yine kimliği verilen bekleme kuyruğundan hazır görevler kuyruğuna aktarmayı sağladığı;
- 07 parametresiyle, kimliği verilen bekleme kuyruğunun başındaki görev kimliğini `al` yazmacı içinde döndürdüğü varsayılmıştır.

```
P           macro  durum,durum-kuyruğu
local yine,eriş
yine:       mov   al, 0
lock        xchg  al, durum
            cmp   al, 0
            jnz   eriş
            suspend(görev-kimliği,durum-kuyruğu)
            jmp   yine
eriş:
P           endm
V           macro  durum,durum-kuyruğu
local son
            mov   durum,1
            mov   dl, durum-kuyruğu
            mov   al, 07h
            int   01ch      ; al <-- kuyruk başındaki görev kimliği
            or    al,al     ; al = 0 ise kuyruk boş demektir.
            jz    son
            mov   kimlik, al
            resume(kimlik, durum-kuyruğu)
son:
V           endm
suspend     macro görev-kimliği,kuyruk-kimliği
            mov   dl, görev-kimliği
            mov   dh, kuyruk-kimliği
            mov   al, 05h
            int   01ch
suspend     endm
resume      macro görev-kimliği,kuyruk-kimliği
            mov   dl, görev-kimliği
            mov   dh, kuyruk-kimliği
            mov   al, 06h
            int   01ch
resume      endm
```

Çizim 4.11. P ve V İşleçlerinin Gerçekleştirim Örneği

`P` adlı makro içinde, `suspend(görev-kimliği,durum-kuyruğu)` makrosunun hemen ardından `jmp yine` komutu ile makro başına dönülmektedir. `suspend` makrosunu çalıştırarak işletimini durduran bir görevin, bir sonraki işletiminde işleteceği ilk komut, bu makroyu izleyen komut olacaktır. Bu konuma `jmp yine` komutunun konmaması durumunda, `resume` makrosu ile hazır görevler kuyruğuna bağlanarak işletime alınan bir görevin, semaforu sınamadan kritik kesimine hemen gireceği görülür. Görevlerin ana işlem birimine anahtarlanma sırası konusunda herhangi bir varsayım yapma olanağı bulunmadığından bu durum, daha önce işletime alınarak kritik kesimini işleten diğer bir görevle eşanlı olarak ortak kaynağa erişmeye, dolayısıyla karşılıklı dışlama kuralının ihlaline neden olabilir. Bu nedenle görevleri kritik kesimlerini işletmeden hemen önce semaforu sınamaya zorlamak gerekmektedir. Bu amaçla `jmp`

`yine` komutu ile makro başına dönülmektedir. Bu zorlama, Çizim 4.10'da, `V (S)` içinde, semafor kuyruğu boş değilse, kuyruktan bir görevin hazır görev kuyruğuna bağlanmasından önce `S`'nin yeniden eksiltilmesi yoluyla sağlanmaktadır.



Çizim 4.12. Ayrıntılı Görev Kuyruk Çizeneği

Görevlerin ortak bir kaynağa erişirken, zamanuyumlama amacıyla bekledikleri kuyruklar, bu kaynağı korumada yararlanılan semafora bağlı bekleme kuyrukları olarak tanımlanır. Ancak bu kuyrukların, giriş/çıkış işlemlerinin sonlanmasının beklendiği giriş/çıkış bekleme kuyrukları ile karıştırılmaması gerekir. Anımsanacağı üzere, görev yönetimi incelenirken bunlar, giriş/çıkış bekleme (G/Çi) ve *(Si)* semafor (zamanuyumlama) bekleme kuyrukları olarak ayrıştırılmıştır. Bu bağlamda, ortak kaynağa koşut biçimde erişecek görevler, aralarında zamanuyumlama sağlamak üzere kullanacakları ortak değişkeni *(S)* sıradan bir değişken olarak belirleyemezler. Bu değişkenin, görevleri programlamada kullanılan programlama dili derleyicisi tarafından bilinen özel tür (semafor türü) bir değişken olması gereklidir. Bu yolla, görevler derlenirken, amaç programlar içine, işletim aşamasında bu özel değişkenle ilgili zamanuyumlama bekleme kuyruğunun yaratılmasını gerçekleştirecek sistem çağrı(lar)ının konması sağlanır. Buradan, ortak kaynağın kullanım durumunu simgeleyen değişkenin, aynı `integer, real, boolean` gibi özel bir adla anılan ve sistemce tanınan bir türde olması gerektiği daha kolay anlaşılacaktır. Birlikte çalışan görevleri programlamaya olanak veren *Concurrent C, Concurrent Pascal, Ada, Modula* gibi programlama dillerinde, bu nedenle, genelde *semaphore* diye anılan özel bir değişken türü bulunmaktadır.

Birlikte çalışan görevlerin içine, sıradan programcılar tarafından yerleştirilen semaforların herbiri, işletim aşamasında zamanuyumlama kuyruğu tanımlarına dönüşmek durumundadır. Bu semaforların herbiri için ayrı bir bekleme kuyruğunun tanımlanması, sistemde aynı anda var olan (Çizim 4.12' de S_i olarak adlandırılan) kuyruk sayısını aşırı bir biçimde artırabilir. Bu, işletim sisteminin, görev yönetimi kapsamında kuyruk yönetim yükünün aşırı artması anlamına gelir. Gerçekleştirim yönünden bakıldığında, kimi durumlarda, semaforların herbiri için ayrı bir kuyruk öngörmek yerine semaforları (dolayısıyla kritik kaynakları) belirli kıstaslara göre gruplayarak bu gruplar için, tek bir semafor, dolayısıyla tek bir kuyruk öngörmek de düşünülebilir. Doğal olarak bu durum, birlikte çalışan görevler yönünden, işletim hızının ve sistem başarımının düşmesi sonucunu doğuracaktır. Zira bir görev kendi kritik kaynağı serbest olmasına karşın, ortak bir semaforla korunan kendi grubundaki başka bir kaynağın kullanılıyor olması nedeniyle bekleme durumunda kalabilecektir. Bu sorun, yönetim yükü ve işletim hızı kıstasları arasında bir uzlaşma noktası bulunarak işletim sisteminin gerçekleştirimi aşamasında çözülecek bir sorundur.

```
P (S)
int *S;
 {
   (*S)--;
   if(*S < 0)
   ⎧Görevi S'ye bağlı ⎫
   ⎨bekleme kuyruğuna ⎬
   ⎩bağla             ⎭

 }
```

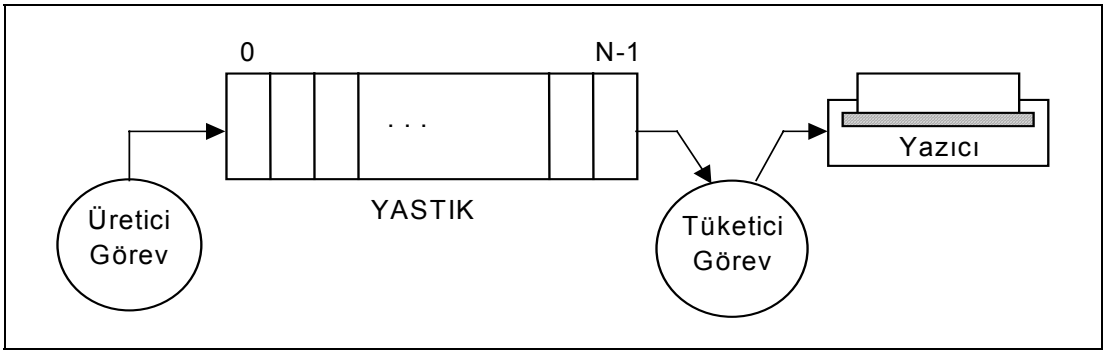
```
V (S)
int *S;
 {
  (*S)++;
  if(*S =<0)
    ⎧S'ye bağlı Bekleme    ⎫
    ⎨kuyruğundan bir görevi⎬
    ⎩Hazır kuyruğuna bağla ⎭
 }
```

Çizim 4.13. Sayan Semaforlar üzerinde işlem yapan P ve V İşleçleri

Şimdiye kadar verilen kuramsal semafor tanımları ve gerçekleştirim örnekleri, ikili olarak nitelenen semaforları sözkonusu etmiştir. *Dijkstra* tarafından artı tamsayılar evreninde tanımlanan semafor işleçleri, eksi tamsayıları da kapsayacak işleçler biçimine dönüştürüldüğünde, ilgili semafor üzerinde bekleyen görev sayısını (kuyruk boyunu) belirleme olanağı da bulunur. Eksi tamsayı değerleri de alabilen semaforlar sayan semaforlar ya da genel semaforlar olarak adlandırılır. Sayan semaforlar üzerinde işlem yapan *P* ve *V* işleçlerinin tanımları Çizim 4.13'te verilmiştir.

Bir bilgisayar sisteminde birlikte çalışan görevler paylaştıkları ortak kaynaklara, ya okuma ya da yazma amacıyla erişirler. Daha önce de belirtildiği üzere, ortak kaynağa yazma yapan görev üretici görev, kaynaktan okuma yapan görev de tüketici görev olarak tanımlanır. Bu durumda, ortak bir yastık üzerinden iletişimde bulunan görevlerin üretici-tüketici ilişkisi içinde bulundukları söylenir. Ürettiği çıktıları (damga dizgilerini), N tutanaktan oluşan bir yastık alanı üzerine yazan bir kullanıcı görevi ile, bu çıktıları yazıcıdan döken bir yazıcı görevi üretici-tüketici görevlere örnek olarak verilebilirler. Sayan semaforların kullanımını örneklemek üzere, birlikte çalışan bu görevlerin programlanması ele alınmıştır. Bu bağlamda, kullanıcı görevinin ürettiği

çıktıları, tutanak tutanak, yastık alanındaki boş tutanaklara yazdığı, yazıcı görevinin ise çıktı tutanaklarını, geliş sırasında bu yastık alanından alarak yazıcıdan dökeceği varsayılmıştır. Kullanıcı görevinin yastık alanı dolu olduğunda, en az bir tutanaklık yer açılıncaya kadar beklemesi; yazıcı görevinin ise, yastık alanı tümüyle boş olduğu sürece hiçbir işlem yapmadan beklemesi gerekmektedir. Kullanıcı ve yazıcı görevlerin ortak yastık alanına tümüyle zamanuyumsuz (birbirlerinden habersiz) biçimde eriştikleri düşünülecektir. Herhangi bir anda yastık alanında bulunan boş (kullanılır) tutanak sayısı `(e)`, döküm için bekleyen dolu tutanak sayısı ise `(f)` olarak adlandırılacaktır. Tutanak yazma ve tutanak okuma işlemleri için erişilen yastık alanının tümü, kritik kaynağı oluşturacaktır. Bu durumda birlikte çalışan kullanıcı ve yazıcı (üretici ve tüketici) görevler arası etkileşim ve bu görevlerin programlanması Çizim 4.14'te verildiği biçimde olacaktır.



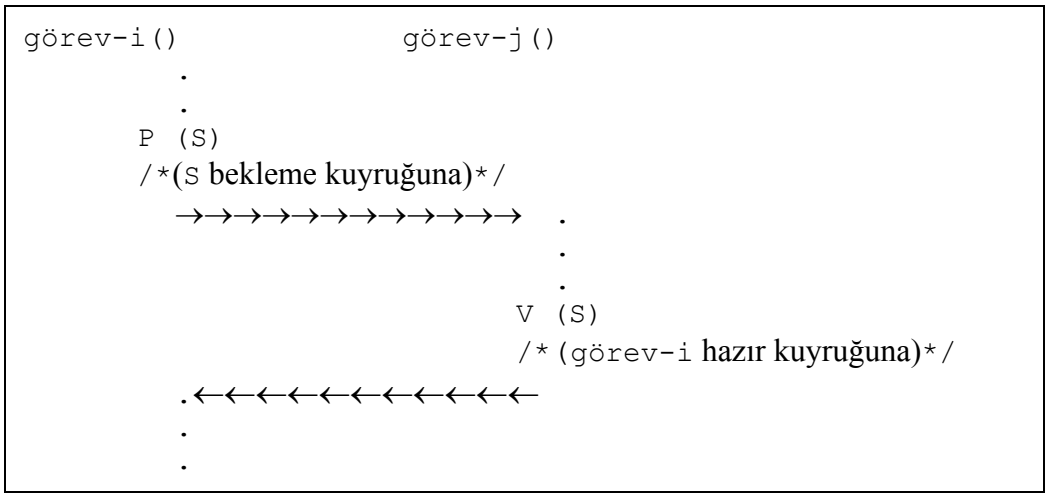

```
int b,e,f;

kullanıcı()
{
char tutanak [80];
   while(true){
   çıktı-üret(tutanak);
   P (e);
   P (b);
   yastığayaz(f,tutanak);
   V (b);
   V (f);
   }
 }
```

```
yazıcı()
{
char tutanak [80];
   while(true){
   P (f);
   P (b);
   yastıkoku(f,tutanak);
   V (b);
   V (e);
   dök(tutanak);
   }
 }
```

Çizim 4.14. Üretici-Tüketici Görevlerin Programlanması

Kullanıcı ve yazıcı görevlerin programlanmasında, ikili bir semafor olan `b` semaforu, ortak yastığa erişimde görevler arası karşılıklı dışlamayı sağlamak amacıyla kullanılmaktadır. Bu amaçla, yastığa erişim yapan `yastığayaz(f,tutanak)` ve `yastıktanoku(f,tutanak)` işlevleri, `P (b)` ve `V (b)` işleçleriyle ayraç arasına alınmaktadır. `e` ve `f` semaforları sayan semaforlardır. Bunlardan `e`, görevler işletime alınmadan önce `N` değerine, `f` ise `0` değerine kurulmaktadır. Böylece `yazıcı` görev,

yastık alanına herhangi bir tutanak yazılmadan ana işlem birimine anahtarlanırsa, P(f) işlecini çalıştırarak yastıkla ilgili bekleme kuyruğuna bağlanacaktır. Bu görevin bekleme kuyruğundan hazır görevler kuyruğuna aktarılması, kullanıcı görevinin V(f) işlecini çalıştırmasıyla gerçekleşecektir. kullanıcı görevi her işletiminde P(e) işleci ile, başlangıçta içeriği N olan e semaforunu bir eksiltecektir. e semaforunun bir artırılması, yazıcı görev tarafından, dökülmek üzere bir tutanağın yastık alanından alınması (okunması) sonrasında, V(e) işleci ile gerçekleşecektir. e' nin içeriği 0 olduğunda, yastıktan en az bir tutanağın dökülmesini beklemek üzere, kullanıcı görevi yastık bekleme kuyruğuna bağlanacaktır. Bu görevin yeniden hazır görevler kuyruğuna aktarılmasını, yazıcı görev içinde yer alan V(e) işleci sağlayacaktır.

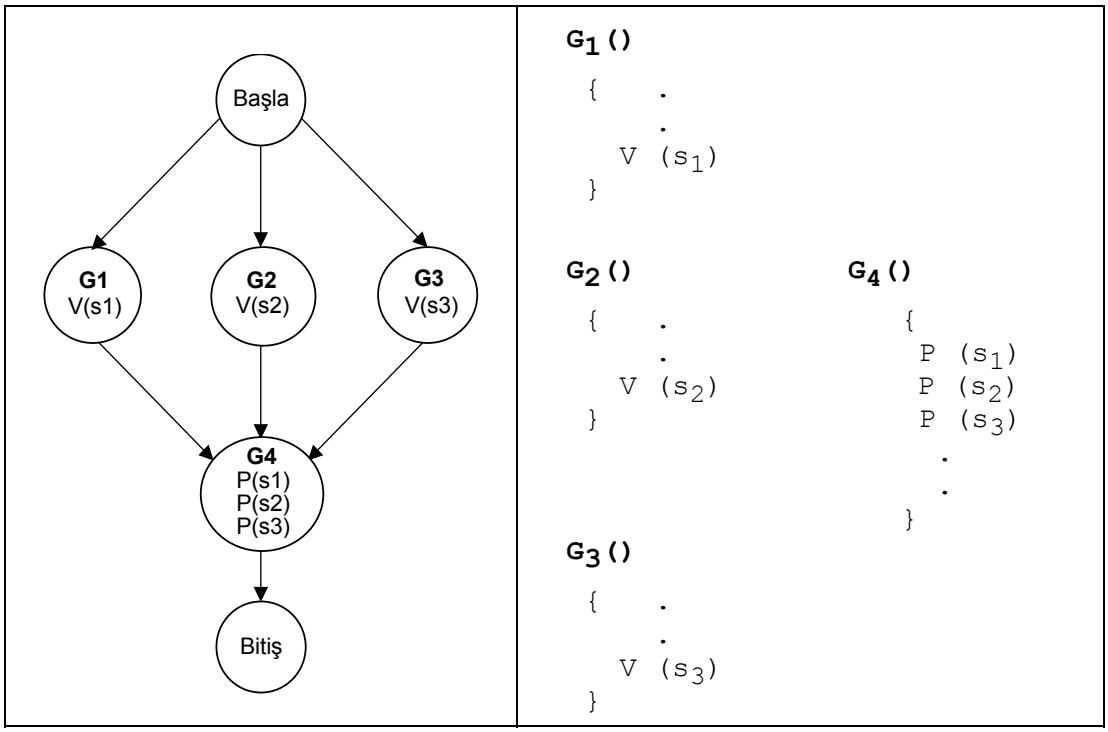


Çizim 4.15. Görevler arası Akış Denetiminin Sağlanması

kullanıcı ve yazıcı görevler içinde P (e) ile V (e), P(f) ile V(f) çapraz biçimde kullanılmıştır. P(e) kullanıcı görevi içinde bulunurken V(e) yazıcı görev içinde yer almaktadır. Bunun gibi P(f) yazıcı görev içinde bulunurken V(f) kullanıcı görevi içinde yer almaktadır. Daha önce P ve V işleçlerinin kullanımı örneklenirken, görevlerin ortak kaynağa erişen kritik kesimlerinin bu işleçlerle ayraç arasına alınarak karşılıklı dışlamanın sağlanmasından söz edilmişti. P ve V işleçlerinin görevler içinde çapraz kullanımı, bu işleçlerin bir diğer kullanım biçimini oluşturur. Bu yolla bir görev diğer bir görevin işletim akışını denetleyebilir (Çizim 4.15).

Birlikte çalışan görevler, kimi zaman, aralarındaki işbirliği gereği yerine getirdikleri işlemleri bozulmayan bir sırada gerçekleştirmek durumunda kalırlar. Örneğin bir iş kapsamında, P_1, P_2 ve P_3 adlı programların K_1, K_2 ve K_3 adlı bağımsız kütükleri üretmesi; bu işlemler tamamlandıktan sonra P_4 adlı diğer bir programın K_1, K_2 ve K_3 adlı kütükleri birleştirerek K_4 adlı sonuç kütüğü elde etmesi istenebilir. Burada P_1, P_2 ve P_3 adlı programlar koşut çalışabilir. Ancak P_4 adlı programın çalışabilmesi, bundan önceki üç programın da sonlanıp ilgili (K_1, K_2 ve K_3 adlı) kütüklerin üretilmiş olmasına bağlıdır. Bu durum Çizim 4.16' da verilen çizge ile gösterilebilir. Bu tür çizgeler, öncekilik çizgeleri olarak adlandırılır. Birlikte çalışan görevlerin, varsa işletim sıraları

öncekilik çizgeleriyle ifade edilir. Bu çizgelerle ifade edilen işletim sırasının korunmasında *P* ve *V* işleçlerinden yararlanılır. Bu, görevler içinde, *P* ve *V* işleçlerinin, yukarıda sözü edilen çapraz kullanımı ile sağlanır. Örneğin P_4 adlı programa ilişkin G_4 adlı görevin, K_1, K_2 ve K_3 adlı kütükleri birleştirerek K_4 adlı sonuç kütüğü elde edebilmesi için P_1, P_2 ve P_3 adlı programlara ilişkin G_1, G_2 ve G_3 adlı görevlerin sonlanmasını beklemesi gerekmektedir. Bu koşulun yerine getirilmesi, G_1, G_2 ve G_3 adlı görevlerin herbirisi ile G_4 görevi arasında s_1, s_2 ve s_3 adlı ikili semaforlara dayalı zamanuyumlamanın sağlanmasını gerektirir. Bu, bir yandan G_4 görevinin başına, $P(s_1)$, $P(s_2)$ ve $P(s_3)$ işleçlerinin ardarda konması; diğer yandan da G_1, G_2 ve G_3 adlı görevlerin sonuna $V(s_1)$, $V(s_2)$ ve $V(s_3)$ işleçlerinin eklenmesi yoluyla sağlanır. Bu yolla G_1, G_2 ve G_3 adlı görevler G_4 görevinin işletime giriş sırasını başka bir deyişle işletim akışını denetim altında tutarlar.



Çizim 4.16. Öncekilik Çizgesi ve İkili Semafor Kullanımı ile Gerçekleştirimi

Yukarıda verilen açıklamalar çerçevesinde alt düzey zamanuyumlama araçları, özde semafor olarak adlandırılan özel bir tür değişken üzerinde işlem yapan, çoğu kez işletim sistemi işlevleri olarak gerçekleştirilen, *P* ve *V* olarak adlandırılan ilkel işleçlerden oluşur. Alt düzeyli bu zamanuyum işleçleri, birlikte çalışan görevlerin yönetimine hem kuramsal hem de uygulanırlık açısından gerekli ve yeterli desteği sağlıyor olmalarına karşın azımsanamayacak tasarım ve programlama yükünü de birlikte getirirler. Görevlerarası zamanuyumu, görevlerin içine *P* ve *V* işleçlerinin serpiştirilmesini gerektirir. Bu durum programların okunurluğu ve bakılırlığını olumsuz yönde etkiler. Bunun yanı sıra bu işleçlerin yanlış kullanımı, kimi zaman sistem kaynaklarının kilitlenmesine, dolayısıyla sistemin başarımının düşmesine ve güvenirliğinin

zedelenrnesine de neden olabilir. Örneğin Çizim 4.14' te verilen `kullanıcı()` görevi içindeki `P` işleçlerinin, `P(e),P(b)` yerine, yanlışlıkla ters sırada, `P(b),P(e)` biçiminde yazıldığı düşünülürse, yastığın tümüyle dolu olduğu bir anda `kullanıcı()` görevinin yastığa yeni bir tutanak daha yazmaya çalıştığı andan başlayarak hem kendini hem de `yazıcı()` görevi, dolayısıyla işletimi kilitleyeceği durum ortaya çıkar. Bu gerekçeye dayalı olarak, görevler arası zamanuyumlamada alt düzey işleçleri doğrudan kullanmak yerine, bunlara dayalı, daha yalın ve sistemli üst düzey araçlardan yararlanmak düşünülür. İzleyen kesimde bu araçlar açıklanacaktır.

4.2.3. Üst Düzey Zamanuyumlama Araçları

Birlikte çalışan görevlerin birbirleriyle olan etkileşimleri, görevler arası iletişim ve görevler arası akış denetimi olmak üzere iki genel grupta düşünülür. Görevler, üretici-tüketici ilişkisi çerçevesinde veri alış-verişinde bulunuyorlarsa bu, görevler arası iletişim kapsamında ele alınır. Görevlerin, birlikte yerine getirdikleri iş kapsamında yapmak zorunda oldukları işbirliği gereği, aralarında hiçbir veri aktarımı sözkonusu olmaksızın, birbirlerinin işletim akışlarını denetlemeleri, görevler arası akış denetimi kapsamında düşünülür. Gerek iletişim, gerekse akış denetimi yönünden etkileşen görevler arasında zamanuyumlama zorunluluğu vardır. Bilgisayar sistemlerinde zamanuyumlama işlevlerinin temel kullanım nedenlerinin başında görevler arası iletişim gelir. Görevler arası iletişimin, alt düzeyli zamanuyumlama işleçleri kullanılarak gerçekleştirilmesi üretici-tüketici görevlerin programlanması kapsamında, daha önce örneklenmişti. Görevler arası iletişimi alt düzey işleçleri doğrudan kullanarak yapmak yerine, bu işleçlerden yararlanılarak gerçekleştirilen daha üst düzey tanımlara dayalı işlevlerle ele almak programlama kolaylığı sağlayan bir yoldur. Bu amaçla, sistem çağrıları arasında çoğu kez `send` ve `receive` olarak adlandırılan işlevler öngörülür. Bu işlevler, görevler arası iletişimin yanı sıra bunun ortaya çıkardığı zamanuyum sorununu da, programcılara yansıtmadan çözerler. Bu işlevlere, üst düzeyli olma özelliğini bu olgu sağlar. İlk elde görevler arası iletişimi gerçekleştiren bu komutlar kimi düzenlemelerle görevler arası işletim akış denetimine de olanak verirler. Bu yolla, alt düzey zamanuyumlama işleçlerinin yerini tümüyle alabilirler. `send` ve `receive` komutları aracılığıyla görevler arasında aktarılan veriler `ileti` diye adlandırılır.

a. İleti Aktarımı, send/receive Komutları

`send` komutu bir görevin diğer bir göreve ileti göndermesi için kullanılır. Görevler `receive` komutunu diğer görevlerden ileti beklemek ve almak için kullanırlar. Bu komutlar:

```
send     (hedef, ileti)
receive  (kaynak, ileti)
```

sözdiziminde, iki parametreli komutlar olarak düşünülebilirler. `hedef` ve `kaynak` adlı parametreler alan ve gönderen görevlerin kimlikleridir. `ileti` ise aktarılan veri öbeğini belirler. Bu şemaya göre `send` komutu, `ileti` adlı veri öbeğinin `hedef` kimlikli alıcı göreve aktarılmasını; `receive` komutu ise `kaynak` kimlikli görevin göndereceği

ileti'nin okunmasını sağlar. İletilerin kaynak kimlikli görevden hedef kimlikli göreve aktarımı, değişik yollarla gerçekleştirilebilir. Bunlar iletinin:

- kaynak görevin bellek alanından, doğrudan hedef görevin bellek alanına aktarılması;
- işletim sistemine ilişkin yastık alanları üzerinden dolaylı taşınması ve
- posta kutuları'nın kullanılması

yollarıdır. İletinin kaynak göreve özgü bellek alanından, doğrudan hedef görev ortamına aktarılması yolu benimsendiğinde send komutunu çalıştıran kaynak görev, hedef görev receive komutunu çalıştırana değin bekleme kuyruğuna konur. receive komutu çalıştırıldığında ileti, işletim sistemi tarafından gönderen ortamından alıcı ortamına doğrudan kopyalanır. Eğer receive komutu send komutundan önce çalıştırılmışsa, bu kez receive komutunu çalıştıran görev, ilgili send komutu çalıştırılana değin bekletilir. Bu yaklaşım, birlikte çalışan görevlerin programlanmasına olanak veren *ADA* programlama dilinde *rendez-vous* (randevu) düzeneği olarak bilinir. Bu düzenek aracılığıyla görevler, veri iletişiminin yanı sıra birbirlerinin işletim akışlarını da denetleyebilirler. Gerçekleştirim kolaylığı sağlıyor olmasına karşın bu yol, özellikle görevler arası veri iletişimi için esnek olmayan, hem gönderen hem de alıcı görevlerin işletimlerini bekle-ilerle biçimiyle yavaşlatan bir yoldur.

Görevlerin, send ve receive komutları üzerinde, sistemli biçimde, aktarım gerçekleşene değin beklemeleri, iletilerin, veri iletişim hizmetini üreten işletim sistemi ortamında yastıklanması yoluyla engellenebilir. Aktarılan iletilerin işletim sistemi tarafından geçici olarak yastıklanması durumunda send komutunu çalıştıran görevin bekleme kuyruğuna bağlanması gerekmez. Görevlere belirli sayıda iletiyi, beklemeden aktarma olanağı tanınır. Bir görev, kendi kimliği ile ilgili yastıklanmış hiçbir ileti bulunmadığı durumlarda receive komutunu çalıştırırsa, doğal olarak bekleme kuyruğuna bağlanır. İletiler, işletim sistemince yastıklanacağından, veri yapıları, sistemce tanınan özel tür'de olmak zorundadır. send ve receive komutları içinde hedef ve kaynak görevlerin kimlikleri, iletinin adresi olmayı bu yöntemde de sürdürür.

İletilerin işletim sistemi ortamında yastıklanacağı veri yapılarının tanımlarını kullanıcılara bırakma esnekliği posta kutusu kullanımıyla sağlanır. Posta kutuları belirli sayıda ve çoğu kez değişmez uzunlukta iletinin yastıklanmasına (saklanmasına) yarayan ve görevler programlanırken bizzat kullanıcı tarafından programlama dili olanakları ile tanımlanan özel tür yapılardır. Görevler iletileri doğrudan birbirlerine göndermek yerine bu posta kutularına aktarırlar. Bu durumda send ve receive komutlarının görünümü:

```
send     (posta-kutusu, ileti)
receive  (posta-kutusu, ileti)
```

biçimine dönüşür. Bir posta kutusu üzerinden iletişim kuracak görevler, doğal olarak bu kutunun adı (kimliği) ve yapısı üzerinde anlaşmak durumundadır. send komutu ile posta kutusuna ileti gönderen görev bunu, posta kutusu doluncaya kadar beklemeden gerçekleştirir. Posta kutusunda yazılacak boş ileti alanı bulunmadığı durumda send

komutunu çalıştıran görev bu posta kutusuna ilişkin bekleme durumuna alınır. `receive` komutu, varsa posta kutusundan bir iletiyi okumaya yarar. Bu kutuya ilişkin, varsa bekleme durumundaki görevi hazır göreve dönüştürür. Posta kutusu boşken `receive` komutu ile ileti okumak isteyen görev de yine, bu kutuya ilişkin bekleme durumuna alınır[16]. Posta kutuları iletişim yönü açısından tek yönlü yapılardır. Posta kutularına dayalı iletişim bire-bir ya da birden-birçoğa biçiminde gerçekleşebilir. Başka bir deyişle, tek bir posta kutusu üzerinden birden çok göreve veri aktarılabilir.

UNIX işletim sisteminde görevler arası veri iletişimi *pipe* diye adlandırılan, *FIFO* yapılı posta kutuları üzerinden yapılabilir. Verilerin posta kutusuna gönderilmesi ve posta kutusundan alınması, sıradan kütükler için de kullanılan yazma ve okuma (`write, read`) komutlarıyla gerçekleştirilir. Görevler *pipe* nitelikli bir posta kutusu yaratmak için `pipe(pfd)` sistem çağrısını kullanırlar. `pfd`, sözkonusu posta kutusunun iki ögeli kütük belirtecidir. `pfd[0]` posta kutusundan okumada, `pfd[1]` ise posta kutusuna yazmada kullanılır. *UNIX* işletim sisteminde *pipe* olarak adlandırılan posta kutularının kimlikleri (adları) bulunmaz. Bu nedenle bu kutular ancak ata-oğul yakınlığındaki görevler tarafından kullanılabilirler. Bu bağlamda, *pipe* üzerinden iletişim yapacak görevlerden biri ata, diğer(ler)i de oğul nitelikli olmak durumundadır. Ata nitelikli görev, `pipe(pfd)` sistem çağrısını kullanarak bir posta kutusu yarattıktan sonra, iletişim kuracağı görev(ler)i, `fork()` sistem çağrısı ile yaratıp, sözkonusu iletişimin yönüyle uyumlu olarak, `pfd[0]` ya da `pfd[1]` belirteçlerinden ilgili olanını, oğul görev(ler)e, `execlp()` sistem çağrısının argümanı olarak aktarmak durumundadır. *pipe* üzerinden yapılan iletişimde görevler arası zamanuyum denetimi, doğal olarak bu aracı sunan işletim sistemince sağlanır. Çizim 4.17'de, *pipe* üzerinden iletişim kuran `ana` ve `kız` adlı iki görev örneklenmiştir.

pipe adlı posta kutuları, *(shell)* komut yorumlama katmanında kullanıcılara, sistem komutlarıyla birlikte kullanılan (kapalı) biçimleriyle yansır. Bir sistem komutunun ürettiği çıktının diğer bir sistem komutunun giriş verilerini oluşturması istendiğinde, bu komutlar, düz bir çizgi ile ayrılmış olarak (`ls|wc` gibi) arka arkaya yazılıp işletime sunulurlar. Buradaki düz çizgi (`|`) posta kutusunu simgeler. Bu yolla `ls` sistem komutu, ürettiği kütük listesini, giriş kütüğü olacak biçimde *pipe* (posta kutusu) üzerinden `wc` komutuna aktarır. Bu bağlamda, *pipe* nitelikli posta kutusunun yaratılması, belirteçlerin `ls` ve wc görevlerine aktarılması, Çizim 4.17'de örneklenen işlemlere benzer biçimde, `ls` ve wc görevlerinin atası konumundaki *shell* görevi tarafından gerçekleştirilir.

pipe nitelikli posta kutularının, salt ata-oğul yakınlığındaki görevler tarafından kullanılabilir olmasının yarattığı kısıtlamayı ortadan kaldırmak üzere, *UNIX System III*' ten başlayarak *named pipe* ya da *FIFO* olarak adlandırılan, kullanıcılar tarafından adlandırılabilen (kimlik verilebilen) klasik posta kutuları da *UNIX* işletim sistemine eklenmiştir. *UNIX* işletim sisteminde *FIFO*, *pipe* ile kütüğün özelliklerini birleştiren

---

[16] Burada *N* tutanaklık bir yastık alanını ortak kaynak olarak kullanan üretici-tüketici görevler incelenirken örneklenen yaklaşım kullanılır. Posta kutusuna erişim açısından `send` komutu üretici, `receive` komutu da tüketici rolü üstlenmektedir.

özel bir kütük yapısıdır. Örneğin, `mknod("iz/ad",S_IFIFO|0666,0)` sistem çağrısı kullanılarak yaratılır. `open(), read(), write(), close()` gibi sistem çağrıları aracılığıyla, `iz/ad` üzerinde anlaşan ve ata-oğul ilişkisi dışındaki görevlerce de kullanılır. Kütüklerden farklı olarak `lseek()` sistem çağrısının, *FIFO* nitelikli özel kütüklerle kullanımı anlamsızdır. Zira, adından da anlaşılacağı üzere, bu özel kütüklerden okuma, yazma sırasında, *first-in-first-out* mantığıyla yapılmak durumundadır. Bu nedenle, *FIFO* içinde yer alan damgalara rasgele erişim olanaksızdır. *FIFO* nitelikli özel kütüklere koşut erişimde görevler arası zamanuyum denetimi, doğal olarak, bunu bir araç olarak sunan işletim sisteminin yükümlülüğündedir.

```
/*  cc -o ana ana.c  komutuyla derlenen ana.c  */
  #include <stdio.h>
  main()
  {
  int i, pfd[2], boy;
  char fdstr[10], msg[80];

    pipe(pfd);
    if(fork()==0){
        close(pfd[1]);  /* kız adlı görev */
        sprintf(fdstr,"%d",pfd[0]);
        execlp("./kız","kız",fdstr,NULL);
        exit(0);
        }
    close(pfd[0]);         /* ana adlı görev */
    boy = getline(msg);    /* klavyeden mesajı oku */
    write(pfd[1],msg,boy);/* posta kutusuna yaz */
    wait();                /* kız'ın sonlanmasını bekle */
  }

/*  cc -o kız kız.c  komutuyla derlenen kız.c  */
  #include <stdio.h>
  main(argc,argv)
  int argc;
  char *argv[];
  {
  int i, fd, nread;
  char s[80];

    fd=atoi(argv[1]);
    nread=read(fd,s,sizeof(s));      /* gelen mesajı oku */
    printf("reading %d bytes: %s\n",nread,s); /* yazdır */
  }
```

Çizim 4.17. *pipe* ile İletişim Kuran Görevler Örneği

Üst düzey zamanuyumlama araçları olmaları itibarıyla bu posta kutularının kullanımı, işletim hızı yönünden yavaştır. İşletim hızının önemli olduğu uygulamalarda, *UNIX* işletim sistemine *System V* ile giren *semaphore, shared memory* gibi alt düzey araçların kullanımına geri dönülür. Zaten genellikle alt düzey araçların hızlı ancak kullanımı karmaşık, üst düzey araçların ise yavaş ancak kullanımı kolay araçlar olduğu bilinen bir gerçektir. Kullanımda sağlanan kolaylığın bedeli hızdan verilen tavizle ödenir.

`send` ve `receive` komutları aracılığıyla yapılan görevler arası veri aktarımları ya işletim sisteminin kullanıcıya saydam yastık alanları ya da kimliği bizzat kullanıcı tarafından belirlenen posta kutuları üzerinden gerçekleşir. Gerek işletim sistemi yastık alanları, gerekse posta kutuları, birden çok görevin bölüştüğü (eşanlı okuma ve yazma yapabildiği) kritik kaynaklardır. Dolayısıyla görevlerin, bu alanlara erişimleri sırasında zamanuyumlanmaları gerekir. Bu zamanuyumlama semaforlar kullanılarak yapılabilir. `send` ve `receive` komutlarına ilişkin sistem çağrıları içinde semaforlara dayalı zamanuyum düzenekleri öngörülür. `send` ve `receive` komutları derlenirken, aktarım işlemlerinin yanı sıra aktarım alanlarına erişimde zamanuyumlamayı da sağlayan sistem çağrıları amaç kodlar arasına yerleştirilir. Bu yolla kullanıcılar, zamanuyumlamayı bizzat gerçekleştirme yükümlülüğünden kurtulmuş olur. Daha önce de belirtildiği gibi, `send` ve `receive` komutlarına üst düzey işlev olma özelliğini veren bu olgudur. Bu bağlamda, Çizim 4.14' te verilen üretici-tüketici görevlerin *P* ve *V* işleçleriyle programlanması örneği, `send` ve `receive` komutlarının kullanımıyla, Çizim 4.18' de yeniden ele alınmıştır.

```
message ileti;

kullanıcı()
{
char tutanak [80];



 while(true){
   çıktı-üret(tutanak);
   receive(yazıcı,ileti);
   yükle(tutanak,ileti);
   send(yazıcı,ileti);
   }
 }
```

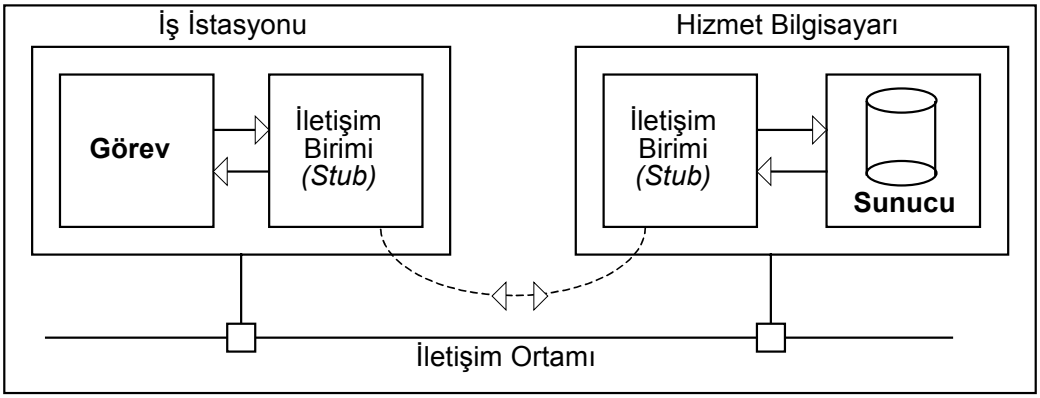
```
yazıcı()
{
 char tutanak [80];
 int i;
 for(i=0;i<N;i++)
   {send(kullanıcı,ileti);}
 while(true){
   receive(kullanıcı,ileti);
   boşalt(tutanak,ileti);
   send(kullanıcı,ileti);
   dök(tutanak);
   }
 }
```

Çizim 4.18. `send` ve `receive` Komutlarının Kullanımı

Bu yeni örnekte, `send` ve `receive` komutlarının, ileti adresi olarak görev kimliklerini kullandıkları; iletilerin *message* olarak, sistemce bilinen, eşit uzunlukta özel tür bir veri yapısında olduğu düşünülmüştür. Bir görevden diğer bir göreve gönderilen iletilerin, alıcı görev, `receive` komutuyla okuyuncaya değin işletim sistemince saklanacağı varsayılmıştır. Çizim 4.14' te N tutanaklık bir yastık alanı kullanıldığından burada da

aynı anda N değişik iletinin bulunması öngörülmüştür. Bu amaçla `yazıcı` görev işletimine, N boş (anlamsız) iletiyi `kullanıcı` görevine göndermekle başlamaktadır. Bu, N adet ileti, doğal olarak işletim sistemince yastıklanmaktadır. `kullanıcı` görevi dökülecek tutanaklar hazır oldukça, `receive` komutu aracılığıyla boş bir iletiyi işletim sisteminden okumakta, `tutanak` ile yükleyerek geri göndermektedir. `yazıcı` görev ise aldığı iletiyi yerel tutanağına boşalttıktan sonra boş bir iletiyi tekrar kullanıcı görevine göndermektedir. Bu durumda işletim sistemince saklanan ileti sayısı hep N olarak kalmakta, böylece N tutanaklık yastıklama sığası değişmez biçimde korunabilmektedir.

`send` ve `receive` komutlarının, görevler arası zamanuyumlamaya getirdiği programlama kolaylığının yanı sıra diğer önemli bir özellikleri daha vardır. Bu komutlar, iletişim komutları olmaları itibarıyla bir ağ içinde bütünleşmiş değişik bilgisayarlar üzerine dağılmış görevler arası iletişim ve zamanuyumlamaya da olanak verirler. Semaforları kullanan alt düzey işleçler ancak, ya tek ana işlem birimi ya da ortak bir ana belleğe sahip çok işleyicili sistemler üzerinde kullanılabilirler. Ağlar içinde bütünleşmiş bilgisayar sistemlerinin ortak bellekleri bulunmadığından semafor adlı özel bellek değişkenlerine dayalı zamanuyumlama düzenekleri, bu ağ içinde yer alan bilgisayar sistemleri üstünde çalışan görevler için kullanılamazlar. Tek bir uygulamanın, bir ağ içinde yer alan bilgisayar sistemlerinden iki ya da daha çoğunu, aynı anda kullanabilmesine olanak veren işlem türüne dağıtılmış işlem adı verilir. `send` ve `receive` komutları, dağıtılmış işlem ortamında kullanılabilen tek zamanuyumlama araçlarıdır.



Çizim 4.19. Uzaktan Yordam Çağırma *(RPC)* Düzeneği

Dağıtılmış işlemin yürütüldüğü bilgisayar ağlarında bir uygulamaya ilişkin veriler birden çok bilgisayar sistemi üstüne dağılmış biçimde saklanabilir. Dağıtılmış işleme olanak veren kimi işletim sistemlerinde, görevler arası iletişim, verilerin (kütüklerin), bulundukları fiziksel konumlardan soyutlanmaları amacıyla kullanılır. Kütüklerin konumlarından soyutlanması, erişilecek kütüğün bulunduğu bilgisayar sistemi hangisi olursa olsun, hep bu kütüğe erişim yapan görevle aynı sistemde yer alıyormuş gibi davranabilmeye olanak verilerek sağlanır. Bu bağlamda bir görev, uzak bir bilgisayar sistemi üstünde bulunan bir kütüğe erişmesi gerektiğinde bunu, sanki kendi ortamındaki

bir diskten yapıyormuş gibi klasik bir `read` komutunu kullanarak yapar. Ancak okuma istemi iletişim birimi (*stub*) adlı bir kesime yönlendirilir. Bu hizmet birimi kütüğün fiziksel olarak bulunduğu bilgisayar sistemindeki iletişim birimi ile iletişim kurarak kütüğün buradaki hizmet birimi (*server*) tarafından okunarak gönderilmesini sağlar. Aldığı kütüğü `read` komutunu işleten göreve yönlendirir. Açıklanan bu düzenek uzaktan yordam çağırma (*remote procedure call*) olarak bilinir. Bu düzeneği kullanan model, istemci sunucu modeli (*client-server model*) olarak adlandırılır (Çizim 4.19).

Bu çerçevede, `read` komutunu çalıştıran görev, bu komutla, normal bir işlev / yordam çağırma işlemini yerine getirmektedir. Görevin bu isteminin yerine getirilebilmesi için, okunacak kütüğün yer aldığı bilgisayar sistemiyle yapılacak gerekli ileti aktarım işlemleri, işletim sisteminin iletişim birimlerince gerçekleştirilmektedir. `send` ve `receive` türü komutları kullanan kesimler bunlardır.

b. *Monitor* Kullanımı

Monitor, birlikte çalışan görevlerin, ortak kaynağa erişim yapan yordamlarının (kritik kesimlerinin) toplandığı kümeye verilen addır. Görevlerin ortak kaynağa erişimlerini denetim altında tutmanın bir yolu da bunların, kritik kaynaklara merkezi bir denetim altında erişmelerini sağlamaktan geçer. Monitor kavramının temelinde bu ilke yatar. Birlikte çalışan görevler programlanırken ortak kaynağa erişim yapacak yordamlar özel bir kümede toplanır. Bu kümenin tüm görevlerce bilinen özel bir adı bulunur. İşletim aşamasında bir görev ortak kaynağa erişmek istediğinde, ilgili yordamı bu kümeden çağırır. Programlama aşamasında bu küme özel bir tür olarak, *monitor* biçiminde tanımlandığından, derleme aşamasında bu kümenin başına, karşılıklı dışlamayı sağlayacak gerekli karşılıklı dışlama komutları derleyici tarafından eklenir. Bu yolla bu kümeden, aynı anda ençok bir görevin yordam çağırmasına izin verilerek ortak kaynağa aynı anda en çok bir görevin erişim yapması sağlanır. Görevlerin monitorda yer alan bir yordamı çağırmaları monitora girmeleri olarak nitelenir. Bir görev monitor içindeki bir yordamın işletimini tümüyle tamamlamadan diğer bir görev monitor içinden yordam çağıramaz. Başka bir deyişle bir görev monitorun içinde iken başka bir görev monitora giremez. Bu amaçla, monitora ilişkin bir bekleme kuyruğu da öngörülür. Yukarıda verilen açıklamalardan da anlaşılacağı üzere monitor bir programlama dili yapısıdır.

Monitorlarla ilgili önemli bir sorun, monitora bir kez girebilmiş bir görevin burada, karşılıklı dışlama gereğinin dışında, başka bir nedenle beklemesi gerektiğinde bu beklemenin, diğer görevleri kilitlemeden nasıl gerçekleşeceği sorunudur. N tutanaklık bir yastık üzerinden iletişim yapan iki görevin, bu yastığa erişim yapan kesimlerinin `üretici-tüketici` adlı bir monitor içine konduğu düşünülebilir. Bu görevlerden üretici olanının, monitora girerek, yastık dolu iken buraya yeni bir tutanak eklemeye kalkması durumunda beklemeye alınması gerekir. Bu görev, gerekli önlemler alınmadan beklemeye alınırsa, hala monitor içinde göründüğünden tüketici görevin monitora girmesi engellenir. Tüketici görev monitora girip yastıktan bir tutanak almadan üretici görevin bekleme durumu son bulamayacağından, kilitlenme olarak bilinen, görevlerin karşılıklı bekleşme durumu ortaya çıkar. Bu önemli sakıncayı ortadan kaldırmak için kimi koşullarda, bekleme durumundaki bir görev, halen monitor içinde iken, diğer bir

görevin de monitora girişinin sağlanması gerekir. Bunun için, monitorda, koşul *(condition)* adlı yeni bir değişken türü ile bunlar üzerinde işlem yapan `wait` ve `signal` adlı komutlar tanımlanır.

```
monitor üretici-tüketici
condition dolu,boş;
int sayaç;
tutanakekle(tutanak)
  char *tutanak;
  {
  if(sayaç == N) {wait(dolu);}
  tutanakyaz(tutanak);
  sayaç++;
  if(sayaç == 1) {signal(boş);}
  }
tutanakal(tutanak)
  char *tutanak;
  {
  if(sayaç == 0) {wait(boş);}
  tutanakoku(tutanak);
  sayaç--;
  if(sayaç == (N-1)) {signal(dolu);}
  }
üretici-tüketici ends

char tutanak[80];
kullanıcı()
  {
    while(true){
    çıktı-üret(tutanak);
    üretici-tüketici.tutanakekle(tutanak);
    }
  }
yazıcı()
  {
    while(true){
    üretici-tüketici.tutanakal(tutanak);
    dök(tutanak);
    }
  }
```

Çizim 4.20. Üretici-Tüketici Görevlerin *Monitor* ile Programlanması

`wait(koşul-değişkeni)` komutu kendisini çalıştıran görevi `koşul değişkeni`'ne bağlı bekleme kuyruğuna koyar. `signal(koşul-değişkeni)` komutu ise `koşul değişkeni`'ne bağlı kuyruktaki görevi hazır görev durumuna getirir. Görevler, monitor içinde bekleme koşulu olarak sınadıkları değişkenleri koşul değişkeni olarak tanımlarlar. Monitor içinde sınanan bir koşul değişkeni bekleme yapmayı gerektiriyorsa `wait` komutu çalıştırılarak bekleme durumuna geçilir. `wait` komutu, aynı zamanda,

varsa, monitora girmeyi bekleyen bir görevin monitora girmesini de sağlar. Bekleme durumundaki bir görevin hazır görev durumuna geçebilmesi, monitora bundan sonra giren görevlerden birinin, aynı koşul değişkenine ilişkin `signal` komutunu çalıştırmasıyla gerçekleşir.

Daha önce hem semaforlar hem de iletişim komutlarının kullanımını örneklemede yararlanılan klasik üretici-tüketici görevler, bu kez monitor yaklaşımını örneklemede kullanılmış ve elde edilen program listesi Çizim 4.20' de verilmiştir. Burada kullanılan dil, C programlama dilini andırıyor olmakla birlikte, tümüyle varsayımsal bir dil olarak düşünülmelidir. N tutanaklık yastık alanına tutanak ekleme ve buradan tutanak alma işlemleri ortak kaynağa erişim yapan kritik kesimler olarak `üretici-tüketici` adlı *monitor* içine konmuştur. Yastık alanının, dolu olduğu bir durumda tutanak eklemek isteyecek üretici görevin `dolu` adlı koşul değişkenine; boş olduğu durumda da, tutanak okumak isteyecek tüketici görevin `boş` adlı koşul değişkenine bağlı bekleme konumlarına, sırasıyla `wait(dolu)` ve `wait(boş)` komutları kullanılarak alınacağı görülmektedir. `sayaç`'ın `1` ve `N-1` değerlerini alması durumlarında ise, `boş` ve `dolu` koşul değişkenlerine bağlı olarak bekleyen görevlerin `signal` komutuyla hazır görevler kuyruğuna bağlanacağı düşünülmüştür. `sayaç` modülo N artırılıp eksiltilmektedir.

`kullanıcı` ve `yazıcı` adlı görevler içinden, monitordaki bir yordam çağırılacağı zaman bunun, `monitor-adı.yordam-adı` komutuyla gerçekleştiği varsayılmıştır. `signal` komutu, halen monitor içinde bulunan bir görevin, monitor içindeki diğer bir görevi hazır duruma sokması için kullanılmaktadır. Aynı anda birden çok görevin monitor içinde bulunmasını önlemek üzere, çoğu kez signal komutunu işleten görevin hemen monitoru terketmesi gözetilir. Bu nedenle, `tutanakekle` ve `tutanakal` monitor yordamları içinde `signal` komutları hep sonuncu komut olarak kullanılmıştır.

Birlikte çalışan görevler yönünden monitor, bölüşülen ortak kaynak için vardır. Bir sistemde birden çok ortak kaynak ve bunların çevresinde kümelenen birden çok birlikte çalışan görev bulunur. Koşut işlem, her görev kümesi için ayrı bir monitor tanımlanarak yürütülür. Monitor yaklaşımını kullanan sistemlerde, zamanuyumlanmaları gereken yordamlar belli kümeler içinde toplandıklarından, semaforlar ya da iletişim komutları kullanılarak gerçekleştirilenlere göre daha kolay anlaşılır ve bakılır yazılımların elde edilmesi sağlanır. Daha önce de belirtildiği üzere, monitor bir programlama dili aracıdır. Bu aracı kullanan sistem programcıları programladıkları görevler arası ayrıntılı zamanuyumlama eklerini öngörme zorunluluğundan kurtulurlar. Zira monitor kullanımına olanak veren programlama dillerinde görevler arası zamanuyumlama işlevi, tanım gereği, derleyici ve sistem yazılımlarının yükümlülüğüne girer.

Monitor kavramını destekleyen üst düzey programlama diline verilebilecek örnek, JAVA'dır. JAVA, programların işletim dizileri içermesine olanak vermekte ve bunların zamanuyumlanmasını monitor yaklaşımıyla ele almaktadır. Bu amaçla, üretici-tüketici nitelikli işletim dizi sınıfları *(classes)* içinde, paylaşılan özel bir nesneye erişim yapan yordamlar *(methods)*:

```
synchronized void yordam_1() {...}
synchronized void yordam_2() {...}
.
```

biçiminde, *synchronized* olarak tanımlanmakta ve bunun sonucunda, paylaşılan nesneyle ilgili *synchronized* nitelikli herhangi bir yordamı en çok bir işletim dizisinin işletmesi (karşılıklı dışlama) sağlanmaktadır. Bir işletim dizisi, paylaşılan nesneyle ilgili *synchronized* nitelikli bir yordamın işletimini tümüyle tamamlamadan diğer bir işletim dizisi de, aynı nesneye ilişkin *synchronized* nitelikli başka bir yordamı işletmeye kalktığında, bu sonuncunun, otomatik olarak bekler duruma geçmesi sözkonusu olmaktadır. Zira JAVA'da, her nesne için bir `kilit` *(lock)* belirteci öngörülmekte ve *synchronized* nitelikli bir yordam bir nesne üzerinde işlem yaparken bu nesne kilitlenerek diğer *synchronized* nitelikli yordamların erişimine kapalı tutulmaktadır. `kilit` belirteci, bu niteliğiyle, paylaşılan nesnenin (zamanuyumlanması gereken) yordam kümesini belirlemekte ve monitor olarak anılmaktadır.

4.3. Görevler arası Kilitlenme

Bilindiği gibi işletim sistemleri kullanıcılara hızlı işletim, kolay ve güvenli kullanım yönlerinden nitelikli bir hizmet sunarken sistem kaynaklarının en verimli düzeyde kullanılmasını amaçlarlar. Yukarıda incelenen koşut işlem, bu genel amaca ulaşmanın temel araçlarından biridir. Gerek çok iş düzeni gerekse çok görevli işlem, koşut işletim yapılmadan gerçekleşemez. Koşut işlem, sistem kaynaklarının görevler arasında paylaşılmasını ve bu paylaşımın eşgüdüm içinde yapılmasını gerektirir. Paylaşım ve eşgüdümün yarattığı görevler arası trafik, kilitlenme olayını ortaya çıkarır.

```
G1()                    G2 ()
     .                       .
     .                       .
     P (DS);                 P(YS);
     P (YS);                 P(DS);
     .                       .
     .                       .
     V (YS);                 V(DS);
     V (DS);                 V(YS);
     .                       .
     .                       .
```

Çizim 4.21. Kilitlenen Görevler

4.3.1. Kilitlenmenin Tanımı

Kilitlenme, bekleme durumundaki görevlerin, hazır görev durumuna geçebilmek için bir diğerinin sağlayacağı koşulu karşılıklı beklemelerine verilen addır. Görevlerin herbirinin bekler olması ve çalışabilmek için bir diğerinin işletilmesinin gerekmesi durumu, görevlerin hiçbir zaman gerçekleşmeyecek bir koşulu beklemeleri sonucunu

doğurur. Örneğin G_1 ve G_2 adlı iki görevin aynı disk birimi üzerinde yer alan kütükleri yazıcıdan dökmede kullanıldığı; sistemde yazıcının YS, disk biriminin de, bir bütün olarak DS adlı semaforlarla korunduğu ve görevlerin kaynakları P (kaynaksemaforu) komutu ile diğer görevlerin kullanımına kapayabildiği; V (kaynaksemaforu) komutu ile de serbest bıraktığı varsayılırsa, bu görevlerin, Çizim 4.21'de verilen biçimde programlanmaları durumunda kilitlenebilecekleri kolayca görülür. Bu çizime göre G_1 görevi *P* işlecini kullanarak önce disk sonra yazıcıyı, G_2 görevi de önce yazıcı sonra diski kullanmak üzere programlanmıştır. G_1 görevinin işletiminin P(DS) komutundan sonra kesilmesi ve G_2 görevinin işletime alınması durumunda bu sonuncunun P(YS) komutunu çalıştırdıktan sonra P(DS) komutu ile DS bekleme kuyruğuna bağlanacağı; G_1 görevinin ise, yeniden işletime alınması sonrasında P(YS) komutu ile YS bekleme kuyruğuna bağlanacağı görülür. Bu durumda G_1 görevi G_2 görevinin V(YS) komutunu, G_2 görevi de G_1 görevinin V(DS) komutunu çalıştırmasını bekler. Her iki görev de bekler durumunda olduğundan bu komutlar hiçbir zaman çalıştırılamazlar. Böylece bu görevler hiçbir zaman oluşmayacak koşulları karşılıklı bekleyerek kilitlenirler. Görevler arası kilitlenmeyi ortaya çıkaran koşullar aşağıda verilen dört ana başlık altında toplanabilir:

a. Karşılıklı Dışlama: Bir kaynağın aynı anda yalnız bir görev tarafından kullanılabilmesi.

b. İstem üzerine kaynak atama: Görevin gereksediği kaynakları, teker teker, işletim aşamasında elde etmesi.

c. Atanan kaynakların, görevler serbest bırakmadıkça geri alınamaması.

d. Döngüsel Bekleme: Bir görevin elinde tuttuğu kaynaklardan bir ya da daha çoğunun, bu görevle aynı döngüsel işletim zinciri içinde yer alan diğer görevlerce de istenmesi.

Bu koşullardan dördünün de aynı anda bulunduğu durumlarda kilitlenme oluşur. Kilitlenmeleri önlemek üzere, bu koşullardan en az birinin oluşmasını engellemek yeterlidir. Ancak sıralanan bu koşullar, çoğu kez ekonomik ve fiziksel kısıtlamalardan kaynaklanır. Örneğin karşılıklı dışlamayı ortadan kaldırmak, hemen hemen tüm donanımsal kaynaklar için olanaksız, görevler arası iletişimde ise bir zorunluluktur. Bunun gibi, görevlerce kullanılan kaynakları gerektiğinde görevlerden geri alan bir kaynak atama yöntemini uygulamak ta, örneğin ana işlem birimi ve ana bellek dışındaki çoğu fiziksel birim için her zaman olanaklı değildir. Ancak koşut işlemin yürütüldüğü işletim sistemlerinde, olanaklar elverdiği ölçüde kilitlenmelere karşı önlemler öngörülür. Bu önlemler, genelde üç değişik yaklaşımla ele alınır. Bunlar:

- Kilitlenmelerin özdevimli olarak yakalanması ve ortadan kaldırılması,
- Kilitlenmelerden korunma,
- Kilitlenmelerden sakınma

olarak adlandırılır. Bu yaklaşımların yanı sıra kilitlenmeler konusunda herhangi bir önlem almamak ta dördüncü bir yaklaşımı oluşturabilir. Bu yaklaşım, kilitlenmelerin az rastlanan bir olgu olduğu, çok ender olarak oluşacak kilitlenmeleri çözmek üzere

görevleri sistemli bir denetim altında tutmanın gereksizliği varsayımına dayanır. Ender de olsa kilitlenmelerin oluşması ve algılanması durumunda, sistem işletmeninin devreye girerek gerekli önlemleri alması öngörülür. Bu önlemler çoğu kez kilitlenmeye neden olan görevlerin yok edilmesi ya da bunlar yakalanamıyorsa tüm sistemin yeniden başlatılmasıdır. *UNIX* işletim sisteminin kullandığı yaklaşım budur.

Kilitlenmeler, sistem kaynaklarının görevler arasında paylaşım zorunluluğundan doğar. Bir bilgisayar sisteminde kaynaklar:

- tüketilir kaynaklar ve
- yeniden kullanılır kaynaklar

olmak üzere kabaca iki grup içinde düşünülebilir. Yeniden kullanılır kaynaklar istem üzerine, aynı anda en çok bir göreve atanan, sınırlı (sonsuz olmayan) bir süre sonunda serbest bırakılan ve başka bir görevin kullanımına verilen kaynaklardır. Ana bellek kesim ve sayfaları, ana işlem birimi, disk birimleri, mıknatıslı şerit birimleri, yazıcılar, veri tabanı kütükleri yeniden kullanılır kaynaklardır. Tüketilir kaynaklar, adlarından da anlaşılacağı üzere işletim aşamasında görevler tarafından yaratılıp kullanılan ve işletim sonunda varlığı son bulan kaynaklardır. Görevler arası iletişimde yararlanılan iletiler, posta kutuları, geçici yastık alanları bu gruba giren kimi kaynak örnekleridir. Bu grup kaynaklarla ilgili kilitlenmeler, tüketilir kaynağı paylaşan belirli sayıda görevle sınırlı kalır. Bu kilitlenmeler çoğu kez zamanuyumlamaya yönelik programlama hatalarından kaynaklanır. Birlikte çalışan görevler programlanırken tüm görevler birlikte düşünülerek tasarım yapılır. Olası kilitlenme durumları tasarım ve gerçekleştirim aşamasında bulunup ayıklanmalıdır. Bu nedenle, işletim sistemlerinde, bu tür kilitlenmelere karşı herhangi bir önlem öngörülmez. Bunların ortadan kaldırılması, programcıların sorumluluğuna bırakılır. Aşağıda açıklanan önlemler, salt yeniden kullanılır kaynaklara yönelik olanlardır.

4.3.2. Kilitlenmelerden Korunma

Kilitlenmelerin oluşma koşulları, dört başlık altında yukarıda belirtilmişti. Kilitlenmelerden korunma, aynı anda bu dört koşuldan en az birinin oluşmasının engellenmesiyle gerçekleşir. Bu, işletime alınan görevlere kimi kısıtlamalar getirilerek sağlanır. Yukarıda sayılan koşullardan karşılıklı dışlama koşulunu ortadan kaldırma olanağı genelde bulunmaz. Zira bir sistemde yer alan kaynakların sayısı, işletilen görev sayısından çok daha az olduğundan, bu kaynakların zorunlu paylaşımı karşılıklı dışlamayı da birlikte getirir. Bu durumda arda kalan diğer üç koşuldan birinin kırılması kilitlenmeye karşı korunmayı sağlayabilir.

Görevlere istem üzerine kaynak atandığı durumlarda belirli sayıda kaynağı elinde tutan bir görev yeni bir kaynak gereksediğinde bunu işletim sisteminden talep eder. Bu yeni istemine olumlu yanıt alıncaya değin halen elinde tuttuğu kaynaklarla birlikte bekler duruma girer. Gereksenen kaynakların, çapraz biçimde, bekler durumdaki görevlerce tutulması kilitlenmeleri oluşturabilir. Görevlere istem üzerine kaynak atamaktan vazgeçilebilirse kilitlenmeyi oluşturan koşullardan biri ortadan kaldırılmış olur. Bunun için, örneğin işlere, sisteme sunuş aşamasında gereksediği tüm kaynakları, iş tanım

dilleri aracılığıyla bildirme zorunluluğu getirilir. Gereksediği kaynakların tümü bir çırpıda sağlanamayan işlere hiçbir kaynak ataması yapılmaz. Başka bir deyişle, tüm kaynakları sağlanana değin işlerin görev tanımları yapılmaz. Görevler işletim aşamasında ek kaynak isteminde bulunamazlar. Bir göreve atanan kaynaklar, bu görevin işletimi sonlanana değin diğer görevlerce kullanılamazlar. Bu kısıtlamaya uyan görevler kilitlenme oluşturmazlar.

Ancak bu kısıtlamaya uymak her zaman mümkün olmayabilir. Zira kimi işler için, gereksenecek kaynakları, sunuş aşamasında belirleme olanağı bulunmayabilir. Bunu yapmaya zorlanan işler (kullanıcılar), işletim aşamasına ilişkin tüm olasılıkları düşünerek kaynak istemleri hakkında abartılı davranırlar. Bunun yanı sıra bu kısıtlama, koşut kaynak kullanım düzeyinin düşmesine de neden olabilir. Bir veri tabanı üzerinde uzunca süre sorgulama yapıp elde edilen kimi sonuçları dökmede kullanılan bir iş ya da görev, işletiminin son aşamasında gereksinim duyacağı bir satır yazıcıyı, bu yönteme göre, işletiminin başında sahiplenerek diğer görevlerin, bu süre içinde, bu kaynağı kullanmasını engeller. Bu durum hem kaynak kullanım verimliliği hem de koşut işlem düzeyini olumsuz yönde etkileyerek sistem başarımının düşmesine neden olur.

Bu önemli sakıncaları ortadan kaldırmak üzere, istem üzerine kaynak atamanın kilitlenmelere yol açması başka bir yolla da engellenebilir. Bu başka yolda görevlere kaynaklar, istem üzerine yine, işletim aşamasında atanır. Ancak yeni kaynak isteminde bulunan bir görev, bu yeni kaynak kendisine atanana değin elinde tuttuğu kaynakları serbest bırakma zorunda bırakılır. Ancak bu yöntemin de uygulanması her zaman mümkün olmayabilir. İşletim sonlanmadan kimi kaynakların, örneğin günlenen kütüklerin geri verilmesi işletim bütünlüğü yönünden mümkün olmayabilir.

Açıklanan bu yaklaşıma benzer bir yaklaşım veri tabanı sistemlerinde kullanılır. Veri tabanı üzerinde günleme yapan görevlerin erişecekleri tutanaklar birer kaynak olarak düşünülür. Görevler, erişecekleri tutanakları baştan belirlerler. Bir görev, gereksediği tutanakları, sırayla kilit altına almaya çalışır. Eğer gereksenen bir tutanak başka bir görev tarafından daha önce kilit altına alınmışsa, o ana kadar kilitlenen tutanaklar serbest bırakarak işlemlere yeniden başlanır. Bir görev, kilitlediği tutanakları her an serbest bırakabilme yolunu açık tutmak için, gereksediği tüm tutanakları kilit altına almadan bunlar üzerinde hiçbir günleme işlemi yapmaz. Günleme işlemi sonunda da, doğal olarak kilitlenen tüm tutanaklar serbest bırakılır. Bu yöntem iki aşamalı kilitleme *(two phase locking)* olarak bilinir.

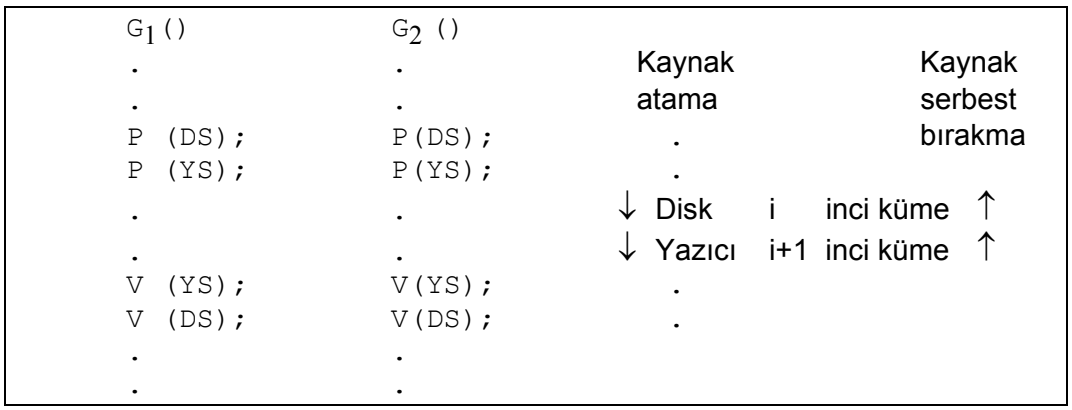
Görevlere atanan kaynakların, görevler istemedikce geri alınamaması kilitlenmeleri oluşturan bir diğer koşuldur. Atanan kaynakların, görevlerin istemleri dışında da sistemce geri alınmasının sağlanması, kilitlenmelerden korunmada akla gelen bir diğer yöntemdir. Ancak bu yöntemi tüm kaynaklar için uygulamaya olanak bulunmaz. Giriş/çıkış sistemi incelenirken açıklandığı üzere, bilgisayar sistemini oluşturan kaynaklar bölüşülür ve bölüşülmez kaynaklar olarak ikiye ayrılır. Ana işlem birimi, ana bellek gibi, görevlerce eşanlı biçimde paylaşılabilen birimler bölüşülür kaynakları; yazıcı, mıknatıslı şerit sürücü gibi, görevlerce ardıl biçimde kullanılması zorunlu olan birimler de bölüşülmez kaynakları oluştururlar. Bölüşülmez kaynakların, kullanımları

tümüyle sonlanmadan bir görevden geri alınarak diğer bir göreve atanmaları yapılamaz. Dolayısıyla bu tür kaynaklarda sözkonusu koşulu kırma olanağı bulunmaz.

Bir görevin elinde tuttuğu kaynaklardan bir ya da daha çoğunun, bu görevle aynı döngüsel işletim zinciri içinde yer alan diğer görevlerce de istenmesi, döngüsel beklemeyi yaratarak kilitlenmelerin oluşmasına kaynaklık eden diğer bir koşuldur. Bu koşulun kırılmasında kullanılan bir yol, sistem kaynaklarını belirli bir sıradüzen içinde atamak ve serbest bırakmaktır. Bunun için kaynaklar belirli kümelere ayrılır ve bu kümeler numaralanarak sıralanır.

- Görevler aynı kümede bulunan kaynaklar için, birden çok atama isteminde bulunmazlarsa;
- *(i)* sırasındaki bir kümeden kaynak sağlamış bir görev, yeni bir kaynak istemi sözkonusu olduğunda, bunu yalnız *(i+1), (i+2), …, (i+n)* gibi daha üst sıralardaki kümelerden sağlarsa;
- *(i)* ninci kümedeki kaynaklar, *(i-1)* inci kümedeki kaynaklardan önce serbest bırakılmak zorunda ise

ilgili sistemde kilitlenme oluşmayacağı kanıtlanabilir. Bu bağlamda, Çizim 4.20'de verilen örnekteki disk biriminin *(i)* inci, yazıcı biriminin ise *(i+1)* inci kümede yer aldığı düşünülürse, kaynakların *(i), (i+1),..* sırasında istenmesi, *(i), (i-1),..* sırasında serbest bırakılması durumunda kilitlenmenin önlenebileceği görülür (Çizim 4.22).



Çizim 4.22. Sıradüzensel Kaynak Atama

4.3.3. Kilitlenmelerden Sakınma

Kilitlenmelerden sakınma, kilitlenmelere neden olan koşulları toptan kırmaya çalışmak yerine, herbir görev için, ayrı, ayrı gerekli ön incelemeleri yapmayı ve kaynak atamalarını buna göre yönlendirmeyi gerektirir. İlke olarak, bir görev, yeni bir kaynak isteminde bulunduğunda bu kaynakla ilgili işletim sistemi kesimi devreye girer. Görevin bu isteminin potansiyel bir kilitlenmeye neden olup olmayacağını inceler. Kaynak ataması, ancak sonuç olumlu ise gerçekleştirilir. Görevlere kaynak atanırken bunun ileride bir kilitlenmeye neden olup olmayacağını belirleyebilen bir algoritma 1965

yılında *Dijkstra* tarafından tanımlanmıştır. Görevlere, kilitlenme oluşturmadan kaynak atayabilme, bir bankerin iflas etmeden borç verebilmesine benzediğinden Banker Algoritması olarak adlandırılmıştır.

Bu algoritmanın çözdüğü sorun, değişmez bir sermayeye sahip olup bunu müşterileri arasında bölüştüren bir bankerin iflas etmeden herkese para sağlaması olarak tanımlanır. Bankerden borç alan müşterilerin herbirinin istedikleri toplam kredi bellidir. Borç alma ve geri ödeme işlemleri, her seferde bir birim üzerinden yürütülmektedir. Banker müşteriye, gereksediği toplam krediyi sınırlı bir süre içinde sağlayacağını, müşteri de tüm gereksinimi karşılandığında borcunu sınırlı bir sürede geri ödeyeceğini taahhüt etmektedir. Sermayesi 10 birim olan ve 3 müşteri ile iş yapan bir bankerin Çizim 4.23'deki durumda bulunduğu varsayıldığında borç istem toplamının 18 birim, kasanın ise 2 birim olduğu görülür.

Müşteri	İstenen Toplam Kredi	Ödünç Alınmış Miktar	Geriye Kalan Miktar
1	4	2	2
2	6	3	3
3	8	3	5

Çizim 4.23. Banker Algoritmasına ilişkin Çizelge

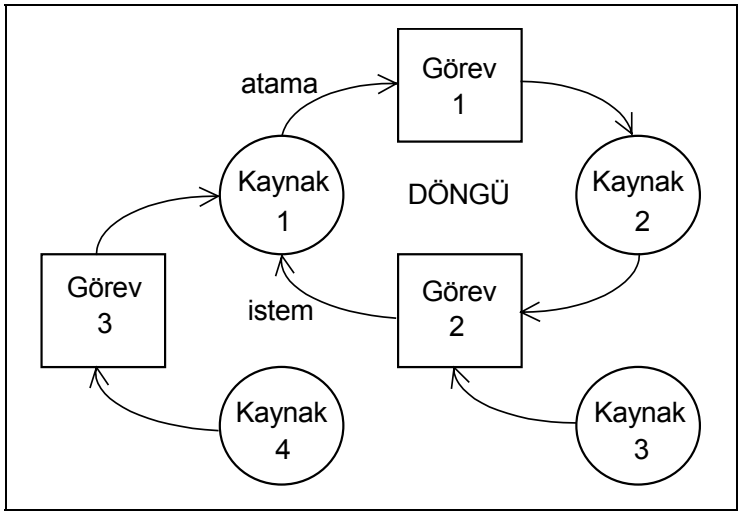
Bu aşamada üçüncü müşterinin 1 birimlik yeni bir borç istemi ile gelmesi durumunda bankerin bu isteme olumlu yanıt verme olanağı bulunmaz. Zira kasasında 1 birim kalacağından, geri ödeme sürecini başlatabilecek, en az borç gereksinimi bulunan birinci müşterisinin istemlerini karşılayamayacak ve kilitlenerek batacaktır. Bu gerekçeye dayalı olarak ödünç verme algoritması, kasada, her zaman en az bir müşterinin geri kalan gereksinimini karşılayacak miktarda para kalmasını gözetecektir. Bu açıklamalardaki banker, işletim sistemi; sermaye, benzer nitelikli kaynaklar bütünü; müşteriler de görevler olarak düşünüldüğünde aynı yaklaşımın görevlere kaynak atamada da kullanılabileceği görülür. Benzer nitelikli kaynaklardan, örneğin salt yazıcılar ya da salt mıknatıslı şerit sürücüler gibi aynı tür kaynaklar anlaşılır. Banker algoritması değişik tür kaynaklarla da kullanılabilecek biçimde genelleştirilebilir. Ancak kuramsal olarak geçerli olan bu algoritmanın, dayandığı kimi varsayımlar nedeniyle uygulamada kullanılması her zaman mümkün olmaz. Örneğin, görevlerden kaynaklara ilişkin toplam gereksinimlerini, işletimlerinin başında belirlemelerini istemek her zaman beklenemez. Bunun yapılamaması durumunda da bu algoritmayı uygulama olanağı bulunmaz.

4.3.4. Kilitlenmelerin Yakalanması ve Ortadan Kaldırılması

Kilitlenmelerle başa çıkmak için uygulanan bir diğer yol da, görevlerin kaynak istemlerine ilişkin herhangi bir kısıtlamaya gitmeden kilitlenmelerin oluşmasını

beklemek, bunlar oluştuğunda belirlenerek ortadan kaldırılmalarını sağlamaktır. Bu yol kullanıldığında, işletim sistemi görevlerin kaynak gereksinimlerini, istem üzerine işletim aşamasında hiçbir kısıtlama getirmeden sağlar. Yalnız görevlere atanan ve bunlar tarafından serbest bırakılan kaynakları izleyerek ortaya çıkan kilitlenmeleri saptamaya çalışır. Bu saptama, değişik yöntemlerle yapılabilir.

Bu yöntemlerden biri, hangi kaynakların hangi görevlere atandığını gösteren kaynak çizgelerini kullanmaya dayanır. Bu çizgeler üzerinde görevler bir kare, kaynaklar da bir çember ile gösterilir. Bir kaynağın bir göreve atanmış olduğu, kaynağı simgeleyen çemberden görevi simgeleyen kareye doğru çizilen bir okla ifade edilir. Bir görevin bir kaynağa ilişkin istemi ise, görev karesinden kaynak çemberi yönünde çizilen bir okla simgelenir. Daha önce belirtildiği gibi, görevler arası döngüsel kaynak bekleme, kilitlenmelerin oluşması için gerekli ve yeterli bir koşulu oluşturur. Döngüsel (çapraz) beklemeler kaynak çizgeleri üzerinde bir döngü olarak ortaya çıkar (Çizim 4.24). Kaynak çizgeleri, matrislerle sayısal biçimde de ifade edilebilirler. Bu ifade biçimi, sözkonusu döngülerin, doğrusal cebir algoritmalarına (matrisler arası işlemlere) dayalı olarak saptanabilmesine olanak verir. Saptanan döngüler içinde yer alan görevler yok edilerek kilitlenmeler çözülür.



Çizim 4.24. Kaynak Atamalarda Döngülerin Oluşması

Görevler arası kilitlenmeler başka yöntemlerle de belirlenebilir. İşletim sistemi içinde bu belirlemeyi yapacak kesim, dönem dönem çalışarak bekleyen görev kuyruklarını tarayabilir. Bu görevlerden, belirli bir sınırın üstünde (örneğin 2 saat) beklemiş görevleri, kilitlenmiş görevler olarak yorumlayıp yok edebilir. Bu yolla, bu görevler, eğer kilitlenmeye neden olan görevler ise, yok edilmeleriyle tuttukları kaynaklar serbest kalacağından varsa kilitlenmeler çözülmüş olur. Daha çok Büyük İskender'in Gordiyon Düğümü'nü çözerken kullandığı yöntemi anımsatan bu yöntem büyük boy bilgisayar sistemlerinde, çoğu kez sistem işletmeni eliyle kullanılan bir yöntemdir.

5. BÖLÜM

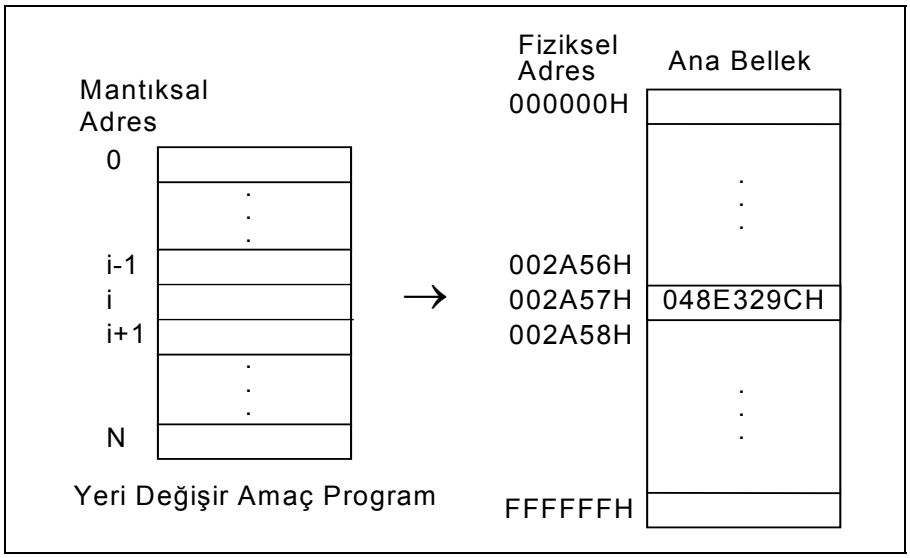
İ Ş L E T İ M S İ S T E M L E R İ

ANA BELLEK YÖNETİMİ

Bilindiği gibi, bir bilgisayar sistemini oluşturan temel birleşenler üç tanedir. Bunlar ana işlem birimi, ana bellek ve giriş/çıkış birimleridir. Daha önceki kesimlerde, hem ana işlem birimi hem de giriş/çıkış birimleri birer kaynak olarak tanımlanmış ve görevler arasında paylaştırılmaları, bu kaynakların yönetimi olarak adlandırılmıştı. Bunun gibi, bir bilgisayar sisteminin en önemli donanımsal kaynaklarından birini oluşturan ana belleğin, görevler arasında paylaştırılması da ana belleğin yönetimi olarak adlandırılır. İşletim sistemi içinde bu işlevi karşılayan kesime ana bellek yöneticisi ya da kısaca bellek yönetici denir. Programların ve işlenen verilerin ana bellekte yer alacakları konumların belirlenmesi, düzenlenmesi, izlenmesi, gereksenen alanların sağlanması, bu alanların dışına taşmaların denetlenmesi gibi işlevler hep bellek yönetimi kapsamında ele alınır.

Bilgisayar sistemlerinde bellekler, ana bellek ve ikincil bellekler olarak iki gruba ayrılır. Ana bellek, aynı zamanda canlı bellek, yarı iletken bellek, *RAM* bellek gibi adlarla da anılır. Programların ve verilerin işlem aşamasında yer aldığı, ana işlem biriminin de dolaysız erişebildiği asıl bellek ana bellektir. İkincil belleklerin, işletim dışı verilerin saklandığı ve korunduğu, yüksek oylumlu bir bellek türü olduğu bilinir. Ana işlem birimi yönünden bakıldığında ana bellek, bir sözcük dizisi gibi görülür. Ana bellekte her sözcüğün bir adresi bir de içeriği bulunur. İzleyen kesimde ana bellek, gerçekleştirim özellikleri gözetilmeksizin, kavramsal olarak, bitişken bir dizi biçiminde düşünülecektir. Genelde sekizin katlarından oluşan sözcük uzunlukları konusunda herhangi bir varsayım yapılmayacaktır.

Ana bellekte bir sözcüğün adresi, bu sözcüğe erişimde, adres yolu üstüne yüklenen konum değeridir. Bu değer, ilgili sözcük içeriğinin ana bellekte bulunduğu fiziksel konumu gösterdiğinden fiziksel adres olarak nitelenir. Ana işlem birimi, doğrudan bellek erişim denetleme birimleri, kanallar, ana bellekteki sözcüklere fiziksel adreslerini kullanarak erişirler (Çizim 5.1).



Çizim 5.1. Fiziksel ve Mantıksal Adres Evrenleri

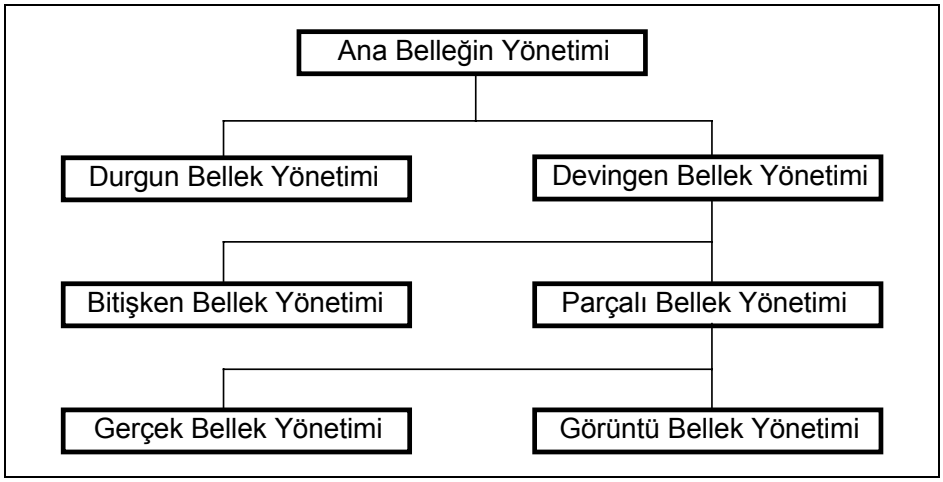
Ana bellek sözcükleri komut kodları ile bu komutların işlenenlerini tutan ögelerdir. Üst düzey programlama dilleri ile yazılan kaynak programlar içinde işlenenler, kullanıcıların özgürce belirledikleri simgesel adlarla anılırlar. Üst düzey programlar derlendiklerinde, yeri değişir amaç programlara dönüşürler. Yeri değişir amaç program, tüm adreslerin program başına (sıfıra) göreli olarak belirlendiği programdır. Kullanıcılar tarafından bitişken kod dizileri olarak düşünülen amaç programlar içinde işlenenler, simgesel adları yerine, program başına göreli adresleriyle tanımlanırlar. Düşünce evreninde gerçekliği olan program içi göreli adresler, fiziksel nitelemesiyle tezat oluşturmak üzere mantıksal adresler olarak adlandırılırlar. Bu bağlamda, ana belleğin fiziksel adres evreni ile programların mantıksal adres evreninden söz edilir. Program içi göreli adreslerin fiziksel adreslere dönüştürülmesi, mantıksal adres evreninden fiziksel adres evrenine geçiş olarak tanımlanır. Bu geçiş, ilgili programa yer sağlanarak ana belleğe yükleme sırasında, dolayısıyla bellek yönetimi kapsamında gerçekleşir.

Tek iş düzeninden çok iş düzenine geçiş ana işlem biriminin işler arasında paylaşılması yoluyla gerçekleşir. Çok iş düzeninin yanı sıra çok görevli işlem de ana işlem biriminin görevlerce bölüşülmesini gerektirir. Gerek çok iş düzeni gerekse çok görevli işlem bilgisayar sistem verimliliğini artırmayı, sistem başarım düzeyini yükseltmeyi amaçlayan düzenlemelerdir. *Von Neumann* türü bilgisayar sistemlerinde ana işlem birimi ana bellekten ayrı düşünülemez. Ana işlem biriminin görevler arasında paylaşımı, ana bellek paylaşılmaksızın gerçekleşemez. Bu nedenle ana belleğin yönetimi sistem başarımını çok yakından etkiler.

İşletim sistemlerinde, ana bellek yönetimi için değişik yöntemler kullanılır. Sistem başarımını yükseltmek için ödenen bedelle orantılı olarak bu yöntemler değişik yetkinlik ve dolayısıyla karmaşıklık düzeyinde olabilirler. En yalınından en karmaşığına doğru bu yöntemler aşağıda sıralanmıştır:

- Tek ve bitişken bölümlü bellek yönetimi
- Değişmez bölümlü bellek yönetimi
- Değişken bölümlü bellek yönetimi
- Yeri değişir bölümlü bellek yönetimi
- Sayfalı bellek yönetimi
- Kesimli bellek yönetimi
- Sayfalı Görüntü bellek yönetimi
- Kesimli Görüntü bellek yönetimi
- Sayfalı-Kesimli Görüntü bellek yönetimi

İzleyen kesimde bu yöntemler sırayla incelenecektir. Bu inceleme yapılırken, her yöntemin çalışma ilkesinin yanı sıra, varsa gereksediği donanım desteği de açıklanacaktır. Böylece ana bellek yönetiminin, özellikle ana işlem birimi donanımına olan izdüşümleri açıklanmış olacaktır. Yukarıda verilen sıralamada, her yöntem, kendinden bir önce gelen yöntemin önemli bir sakıncasını ortadan kaldırmak üzere düşünülmüş ve tanımlanmıştır. Bu nedenle, yöntemler açıklanırken bunların önemli darboğazları da açıklanacak ve bir yöntemden onu izleyen yönteme geçiş bu biçimde gerekçelenecektir.



Çizim 5.2. Ana Belleğin Yönetiminde Kullanılan Yöntem Grupları

Yukarıda sıralanan yöntemlerin çoğu bugün artık pek kullanılmayan yöntemlerdir. Özellikle listede ilk sıralarda bulunanlara, bugün kullanılan gelişmiş yöntemlere nasıl ulaşıldığını açıklayabilmek, bellek yönetiminin ortaya çıkardığı darboğazların nasıl aşıldığını sergileyebilmek için yer verilmiştir. Geçmiş yıllarda bellek yönetimi, hep ana belleğin çok pahalı bir kaynak olduğu ve her ne pahasına olursa olsun çok verimli

kullanılması gerektiği varsayımıyla ele alınmıştır. Günümüzde ana bellek, bilgisayarın ilk yıllarına göre çok ucuzlamış ve kısıtlı bir kaynak olma niteliğini oldukça yitirmiştir. Bu nedenle aşağıda açıklanacak kimi yalın öncü yöntemlere, "kullanım verimliliğinden taviz ver, yönetim yükünden kurtul" felsefesiyle geri dönülebileceğini de düşünmek mümkündür. Ancak ana belleğin, aynı atmosfer gibi, boşluğu hiç sevmediği[17], ne kadar bol olursa olsun hep dolu kalacağı ve hep en karmaşık yöntemlerle yönetileceği varsayımı da çok yanlış değildir. Ana belleğin yönetiminde kullanılan yöntemleri ileride, bir de bu gözle yorumlamak yararlı olacaktır.

Ana belleğin yönetiminde kullanılan yöntemler, çeşitli kıstaslara dayalı olarak gruplandırılır. Programlara atanan alanların, işletim sırasında yer değiştirmesine olanak sağlanıp sağlanamamasına göre bu yöntemler:

- Durgun yöntemler
- Devingen yöntemler

olarak iki ayrı grupta düşünülürler. Bu bağlamda, ilk üç yöntemin birinci gruba diğerlerinin ise ikinci gruba gireceği söylenir. Devingen yöntemler kendi aralarında, programların mantıksal adres evrenlerini bitişken bir bütün ya da sayfa, kesim gibi parçalarla ele almalarına göre iki değişik alt gruba ayrılırlar. Buna göre:

- Bitişken bellek yönetimi
- Parçalı bellek yönetimi

alt yöntem gruplarından söz edilir. İlk dört yöntem bitişken bellek yönetimi grubuna girerken diğerleri parçalı bellek yönetimleri olarak bilinirler. Parçalı bellek yönetimleri de kendi içlerinde:

- Gerçek bellek yönetimi
- Görüntü bellek yönetimi

olarak, iki alt grupta toplanırlar. Gerçek bellek yönetimleri kapsamında bir program, tümü ana belleğe yüklenmeden işletime alınamaz. Hem bu koşulun kırılması, hem de programlara ana belleğin fiziksel boyunu aşma olanağının verilmesi, görüntü bellek yönetimleri ile sağlanır. Buna göre son üç yöntem, adlarından da anlaşılacağı gibi görüntü bellek yönetimi kapsamında yer alırlar (Çizim 5.2). İzleyen kesimde tüm yöntemler herhangi bir gruplandırmaya sokulmadan sıradan açıklanacaktır.

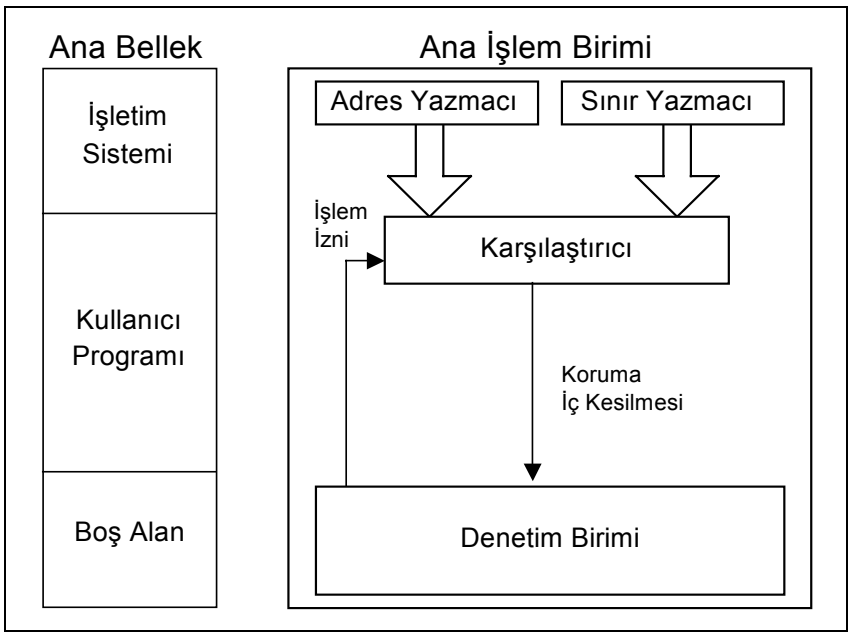
5.1. Tek ve Bitişken Bellek Yönetimi

Tek ve bitişken bellek yönetiminde ana bellek, işletim sisteminin yüklendiği kesim dışında tümüyle tek bir işe atanır. Ana bellekte aynı anda tek bir programın yer alıyor olması tek iş düzenini, doğal bir sonuç olarak ortaya çıkarır. Tek ve bitişken bellek yönetiminin uygulandığı sistemlerde, aslında bellek yönetiminin varlığından da pek söz edilemez. Bu tür sistemlerde, bellek yönetimi çok yalın bir biçimde yerine getirilir.

---

[17] Atmosfer basıncının bilimsel olarak bilinmediği yıllarda, havanın her türlü boşluğu doldurması bu ilkeyle açıklanmaya çalışılmıştır.

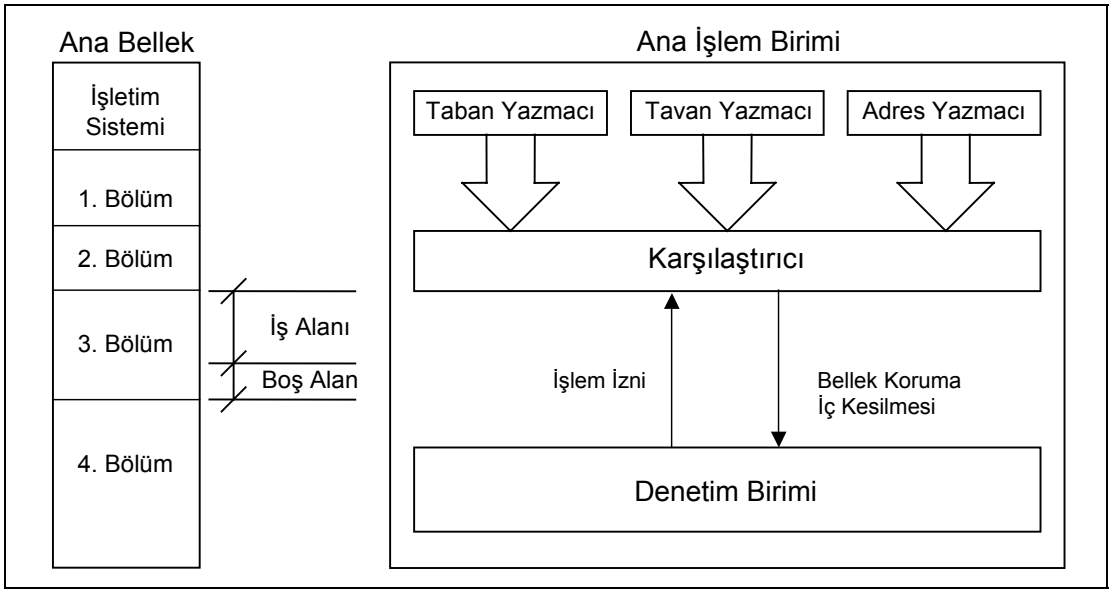
Tek ve bitişken bellek yönetiminde ana bellek, aynı anda, işletim sistemi ile tek bir kullanıcı programı tarafından paylaşılır. Çoğu kez, işletim sisteminin ana belleği paylaştığı kullanıcı programı tarafından bozulmasını önlemek üzere kimi önlemler alınır. Bunu gerçekleştirmek için kullanıcı programına, ana bellekte erişebileceği alanın ataması yapılır. Adresleme süreci içinde de, programın kendine atanan sınırlar içinde kalıp kalmadığı denetlenir. Doğal olarak bu sınır biriciktir ve programla işletim sistemi arasında yer alır. Ana bellek düz bir sözcük dizisi olarak düşünüldüğünde, işletim sistemi, genellikle bu dizinin alt (adres) kesimine yüklenir. Bu durumda kullanıcı programına belirli bir sınır değerinin altında adresleme yapma yasağı getirilir. Bunun için ana işlem birimi içinde sınır yazmacı olarak adlandırılan bir yazmaç öngörülür. Kullanıcı programı işletime alınmadan önce, kullanıcı alanı başlangıç adresi bu yazmaç içine yüklenir. İşletim boyunca sınır yazmacının çıkışları, erişilen sözcüklerin adresleriyle sürekli karşılaştırılarak kullanıcı programının, işletim sistemi alanına erişimi denetim altında tutulur. Böyle bir durum ortaya çıktığında, kesilmeler işlenirken sözü edilen bir iç kesilme ile kullanıcı programının işletimi sonlandırılıp işletim sistemine geri dönülür. Açıklanan bu denetim düzeneği bellek koruma düzeneği olarak bilinir. Doğal olarak bu düzeneğin işletim sistemi çalışırken devre dışı tutulmasını sağlayacak ekler, sınır yazmacıyla birlikte ana işlem birimi içinde öngörülür. Bu düzeneği açıp kapamak, sınır yazmacını günlemek gibi işlemler, sıradan kullanıcılara kapalı ayrıcalıklı komutlarla gerçekleşir (Çizim 5.3).



Çizim 5.3. Tek ve Bitişken Bellek Yönetimi ve Bellek Koruma Düzeneği

Tek ve bitişken bellek yönetimi, tek iş düzeni ve tek görevli işlem ortamlarında kullanılabilen, bu nedenle de, artık kişisel bilgisayar sistemleri için bile yetersiz kalan bir bellek yönetim biçimidir. Aslında burada, sınır taşmalarında işletim sistemine geri dönüşü sağlayan kesilme yordamı ile işletimin başında kullanıcı programlarını ana belleğe yükleyen yordamdan başka, bellek yönetimini ilgilendiren bir işletim sistemi

kesimi de sözkonusu değildir. Bu yönteme göre kullanıcı programları bitişken bir bütün olarak ele alınır ve bu bütünün boyu ana bellekle kısıtlıdır. Başka bir deyişle, derlenmiş ve bağlanmış amaç programların içerdiği sözcük sayısı, ana belleğin fiziksel sığasının altında olmak zorundadır. Bu kısıtlama gerçek bellek yönetimi kapsamına giren diğer tüm bellek yönetimleri için de geçerlidir.



Çizim 5.4. Değişmez Bölümlü Bellek Yönetimi

5.2. Değişmez Bölümlü Bellek Yönetimi

Bu yöntemde ana bellek, işletim sistemi alanı ve kullanıcı alanı olmak üzere salt iki bölüme ayrılmak yerine, işletim sistemi ile birden çok kullanıcı programı arasında paylaşılır. Tek iş düzeninden çok iş düzenine geçiş, ana bellek yönetiminde de tek ve bitişken bellek yönetiminden bölümlü bellek yönetimine geçişi zorlamıştır. Ana işlem biriminin işler arasında paylaştırılması, doğal olarak ana belleğin de, eşanlı olarak işler arasında paylaşılmasını gerektirmektedir.

Değişmez bölümlü bellek yönetiminde ana bellek, işletim sistemi ve kullanıcı alanları olmak üzere bitişken, irili ufaklı, birden çok bölüm olarak düzenlenir. Bir iş, işletime alınmadan önce, ana bellekte kendisine, boyuyla uyumlu bir bölüm atanır. İş, kendisine atanan bu bölümü, işletimi tümüyle sonlanana değin korur. Bölümler, bilgisayar sisteminin işletimi başlamadan önce belirlenen boylarını sistem işletime kapanana değin korurlar. Başka bir deyişle ana belleğin düzeni, işletimin başındaki görünümünü, işletim süresince korur. Bu görünüm, gerektiğinde sistem işletmeni tarafından, işletimin başında yeniden düzenlenebilir. Bu düzenleme bölümlerin boy ve sayıları değiştirilerek yapılır.

Tek ve bitişken bellek yönetiminde olduğu gibi, bu yöntemde de, kullanıcı programlarının işletim sistemine ayrılan alanı bozmalarının engellenmesi gereklidir. Ancak burada bir de, kullanıcı programlarının birbirlerine karşı korunması gerekir. Bu

amaçla, ana işlem biriminde bir taban, bir de tavan yazmacı öngörülür. Bu yazmaçlar, çok iş düzeni çerçevesinde, ana işlem birimini kullanan işin, ana belleğe erişimde aşmaması gereken alt ve üst adres değerlerini tutarlar. Her bellek erişim döngüsünde, erişilen bellek sözcük adresi bu iki değerle karşılaştırılarak taşma denetlenir. Taşma durumunda, bir önceki yöntemde olduğu gibi, ilgili işin işletimi sonlandırılır. Taban ve tavan olarak iki sınır yazmacının kullanılması, bu yönteme çift yönlü koruma olanağı sağlar. Ana işlem biriminin her yeni göreve (işe) anahtarlanışında bu yazmaçlar, işletim sistemi tarafından, ilgili işin taban ve tavan değerleriyle günlenmek zorundadır.

Çizim 5.5. Bölüm Tanım Çizelgesi Örneği

Bu yöntem kapsamında işletim sistemi, ana bellek düzeniyle ilgili bir çizelge tutar (Çizim 5.5). Bölüm tanım çizelgesi olarak adlandırılan bu çizelgede tüm bölümlerin giriş adresleri, boyları ve kullanımda olup olmadıklarını gösteren bir gösterge bulunur. İşletimin başında, gerekirse, bölüm boy ve sayılarının değiştirilmesi için sistem işletmeni tarafından günlenebilen çizelge budur. Çok iş düzeni içinde, bir iş, toplu işlem kuyruğu üzerinden işletime alınacağı zaman önce bu çizelge taranır. İşin gereksediği bellek sığasını karşılayabilen boş bir bölümün bulunup bulunmadığı sınanır. Bulunursa, çizelgede bu bölümle ilgili satır, sunulan işe ilişkin bilgilerle günlenir. Durum göstergesi kullanımda olarak kurulur ve işletim başlatılır. Çizelgede işin sığasını karşılayabilen bölümler kullanımda ise iş, bu bölümlerden biri boşalana dek toplu işlem kuyruğunda bekletilir. Kimi büyük sığalı işler için, hiçbir bölümün, işin sığasını karşılayamadığı durumlar da ortaya çıkabilir. Bu gibi durumlarda, sistem işletmeninin yardımıyla, gelecek açılışta bölümlemenin, sözkonusu işin gereksediği yeri karşılayan en az bir bölümü içerecek biçimde yapılması gerekir.

Yukarıda belirtildiği gibi, bir iş, toplu işlem kuyruğu üzerinden işletime alınacağı zaman bölüm tanım çizelgesi taranarak bölüm ataması yapılır. Bu atama genelde iki yolla gerçekleşir. Bu yollardan biri, işe, tarama sırasında rastlanan ilk boş bölümün atanması yoludur (*first fit*). İkinci yolla ise, işin gerektirdiği sığaya en uygun boydaki bölüm

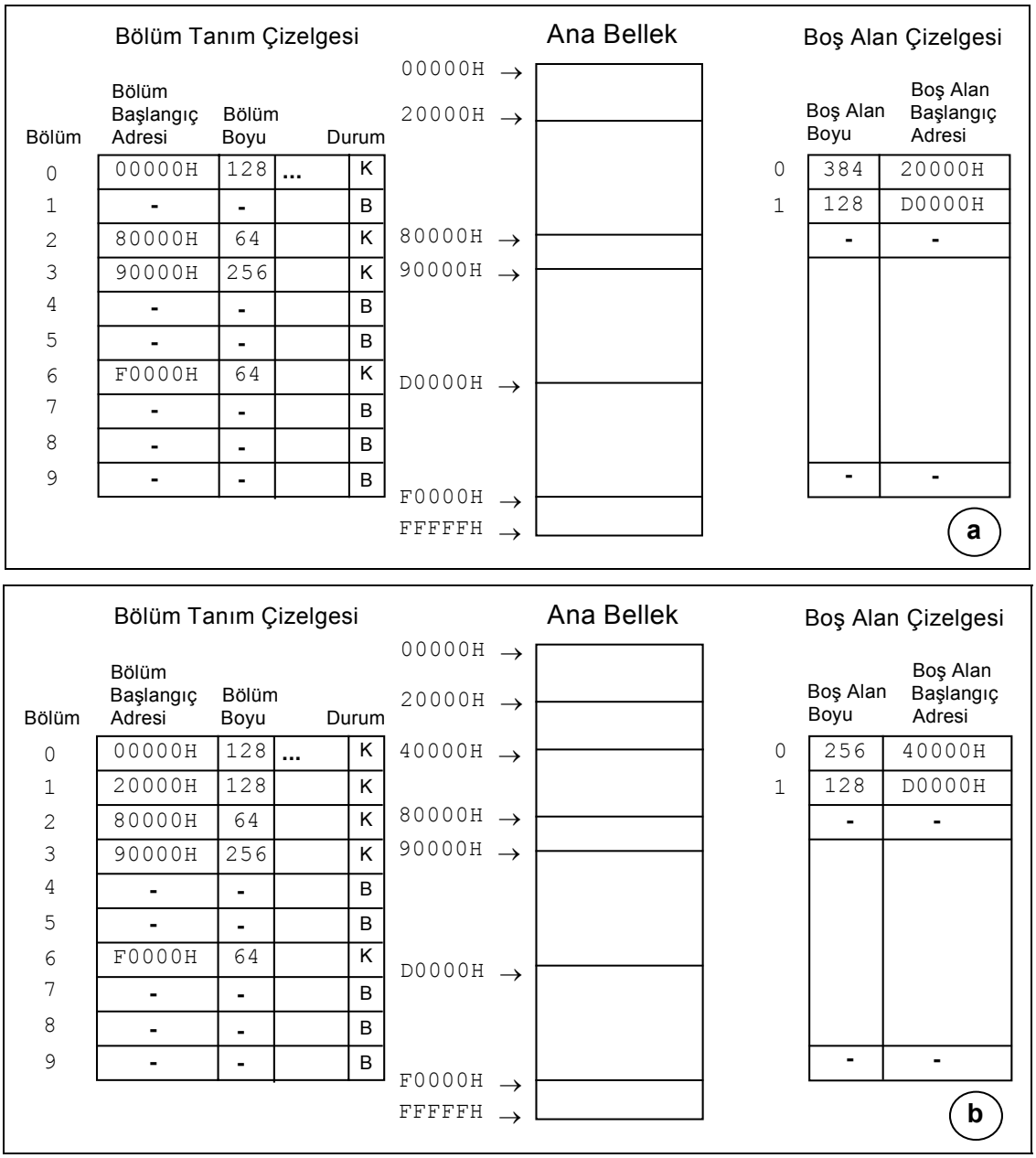
aranır (*best fit*). Bölüm tanım çizelgesi bölüm boylarına göre sıralı biçimde tutulursa, işlere sıradan atama yoluyla da zaten, boylarına en uygun bölüm atanmış olur. Bölüm boylarının, dolayısıyla boy sırasının değişmediği bu yöntemde bu yollardan hangisinin kullanıldığı önemli değildir. Bu seçim, bölüm boylarının işletim aşamasında değiştiği değişken bölümlü bellek yönetiminde önem kazanır.

Değişmez bölümlü bellek yönetiminde tanım çizelgesindeki bölüm (satır) sayısı, çok iş düzeyiyle ilgili olarak belirlenir. Bu sayının, birlikte işletime alınan iş sayısından daha büyük olmasının hiçbir yararı yoktur. Bu nedenle, değişmez bölümlü bellek yönetiminin uygulandığı yalın sistemlerde, ana bellekte öngörülen bölüm sayısının bir düzine kadar olabildiğini düşünmek yanlış olmaz. Bölüm boylarının belirlenmesinde sistemde işletilen program istatistiklerinden yararlanılır. Birlikte işletime alınan işlerin, örneğin yarısı 10 KB ve daha küçük işlerden oluşuyorsa, bölümlerin de yarısı 10 KB olarak tanımlanır. Bu yolla, bir yandan belleğin savrukça kullanımı elverdiğince engellenirken diğer yandan da, bellekte uygun bölüm bulunamama olasılığı azaltılmış olur.

Değişmez bölümlü bellek yönetimi, bellek kullanım düzeyini, tek ve bitişken bellek yönetimine göre yükselten, çok iş düzenine olanak veren bir yöntem olmakla birlikte ana belleği, değişmeyen bir yapıda bölümlere ayıran, işlerden arta kalan bölüm içi kesimlerin yitirilmesine neden olan, tanımlanan bölümlere uymayan işlerin işletilebilmesi için, sistem işletmenini de devreye sokarak işletim dışı yeni düzenlemeler yapılmasını gerektiren ve bu yönüyle çok esnek olmayan bir yöntemdir. Bu yöntemin sayılan sakıncalarını aşabilmek için, bölümlemeyi, işlerin sisteme sunuluş aşamasında, işlerin gerektirdiği boyuttaki bölümlerle, devingen olarak yapabilmek gerekir. Bu, izleyen kesimde, değişken bölümlü bellek yönetimi adıyla, yeni bir yöntem olarak açıklanacaktır. Değişmez bölümlü bellek yönetimi, çok iş düzeninin ilk uygulandığı bilgisayar sistemlerinde kullanılmış bir yöntemdir. Sisteme sunulan işlerin boylarının ve sunuluş sıklıklarının öngörülebildiği yalın, öncü sistemlerde başarıyla kullanılmıştır. Günümüz bilgisayar sistemlerinde kullanılan bir yöntem değildir.

Yukarıda verilen açıklamalar kapsamında, kullanıcı, program, iş ve görev kavramları yerine göre, karışık biçimde kullanılagelmiştir. Ancak bu kavramlar, kavram sıradüzeni içinde, kullanıcı, iş, program, görev sırasında, biri diğerini içeren ama biri diğerinin yerine, her zaman rasgele kullanılamayan kavramlardır. Bilindiği üzere görev bir programın işletimi sırasında aldığı addır. Bir program birden çok görev olarak işletilebilir. Bir iş birden çok programın, sisteme birlikte sunulabildiği bütünün adıdır. Kullanıcı, birden çok işi sisteme sunabilen bir varlıktır. Kullanıcılar bilgisayar sisteminden almak istedikleri hizmetleri işler olarak tanımlayarak sisteme sunarlar. Sunulan bu işler, bir ya da daha çok görev tanımına dönüşerek işletilirler. İşletim aşamasında sözkonusu olan kavram görev kavramı olduğuna göre, ana bellek yönetiminden söz ederken de görevi taban almak gerekir. Örneğin bellek bölümlerinin kullanıcılara, işlere, programlara atanmasından söz etmek yerine bunların görevlere atandığını söylemek daha doğru olur. Ancak bu kavramlar işletim sistemlerinin gelişmesine koşut olarak ortaya çıkmış ve oturmuş kavramlardır. Tek ve bitişken bellek yönetimi ile değişmez bölümlü bellek yönetiminin kullanıldığı yıllar göz önüne alınarak, eski bir yöntemi yeni bir kavrama (görev kavramına) dayandırarak

açıklamanın doğru olmayacağı düşünülmüş ve bu bağlamda daha çok, iş ve program kavramları kullanılmıştır. Ancak, izleyen kesimden başlayarak ana belleğin, daha çok, görevler arasında paylaşımından söz edilecektir.



Çizim 5.6. Değişken Bölümlü Bellek Yönetimi

5.3. Değişken Bölümlü Bellek Yönetimi

Bu yöntemin temel ilkesi, bölümlerin, konum ve boyları itibarıyla, işlerin görevlere dönüştürülüp sisteme sunuluşları aşamasında, devingen olarak yaratılmasıdır. Bu yöntemle bir görev hazır görevler kuyruğuna ilk kez bağlanacağı zaman, gerektirdiği

büyüklükte bir alanın, ana bellekte kullanılmayan boş alanlar içinde bulunması ve yeni bir bölüm olarak bölüm tanım çizelgesine katılması gerçekleştirilir. Görev iskeleti bu bölümle ilgili (bölüm başlangıç adresi, boyu gibi) değerlerle günlenir. Bu yöntemde görevlere atanan bölümlerin yanı sıra, bu bölümler arasında kalan boş alanların da izlenmesi gereklidir. Bu nedenle, bölüm tanım çizelgesinin yanı sıra, bir de boş alan çizelgesi tutulur. Bölüm tanım çizelgesi bölümlere atanan bölümlerle ilgili bilgileri içerirken, boş alan çizelgesi de bölümler arasında kalan boş bitişken alanlara ilişkin konum, boy gibi bilgileri tutar (Çizim 5.6).

Yeni bir görev hazır görevler kuyruğuna bağlanacağı zaman bellek yönetici devreye girer. Boş alan çizelgesi taranarak görevin gerektirdiği büyüklükte boş bir alan bulunmaya çalışılır. Böyle bir alan bulunduğunda, bölüm tanım çizelgesi bu alanın başlangıç adresiyle günlenir. Bu alandan arda kalan kesimin başlangıç adresi ise, boş alan çizelgesindeki yeni boş alan başlangıç adresini oluşturur.

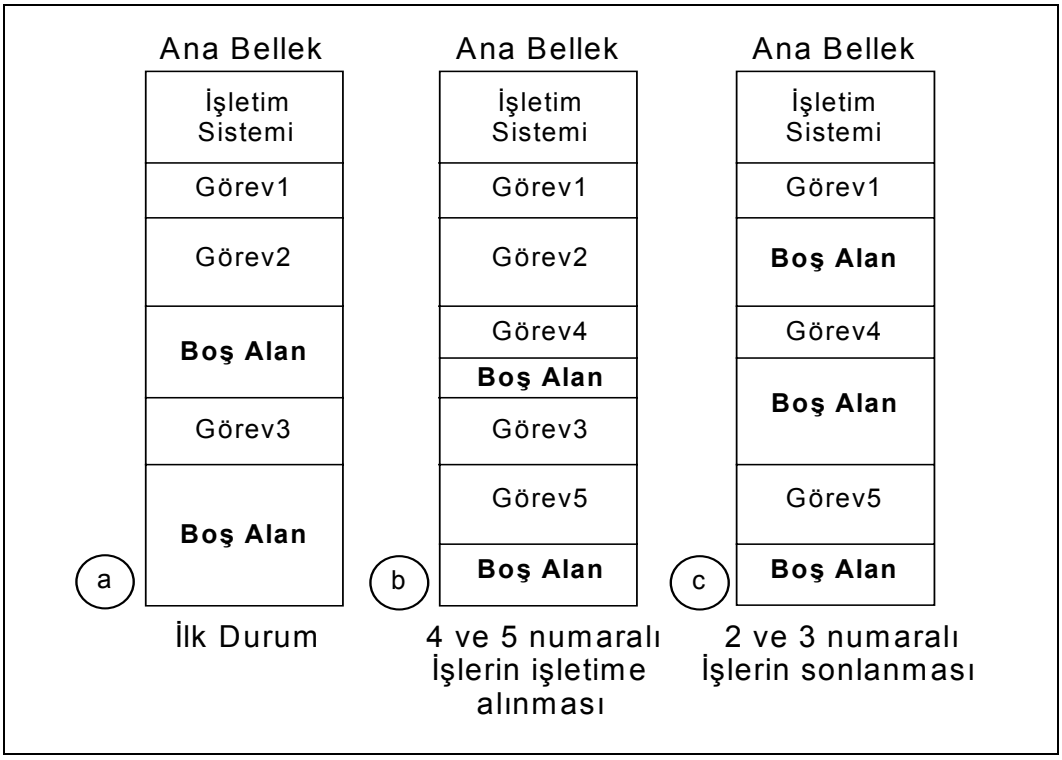
Çizim 5.6.a'da, sistemde 3 değişik görevin çalıştığı örnek bir durum gösterilmiştir. Buna göre toplam fiziksel sığası 1 MB olan ana bellekte, işletim sistemine ilişkin 128 KB' lık bölümün yanı sıra, toplam sığaları (64 + 256 + 64) 384 KB olan, bu 3 göreve ilişkin 3 değişik bölüm yer almıştır. İşletim sistemi ve bölümlerden arda kalan (1024 - 512) 512 KB' lık boş bellek kesimi ise, 384 ve 128 KB'lık iki bitişken alandan oluşmuştur. 0 numaralı bölümün işletim sistemine; 2, 3 ve 6 numaralı bölümlerin ise görevlere atandığı varsayılmıştır. Bu aşamada, 128 KB'lık yere gereksinim gösteren yeni bir görevin sisteme sunulduğu düşünülmüştür. Çizim 5.6.b'de, boş alan çizelgesi taranarak 384 KB'lık boş alanın ilk 128 KB'lık kesiminin, 1 numaralı bölüm olarak bu yeni göreve atandığı; bunun sonucunda, bölüm tanım çizelgesinin boş işaretli 1 numaralı satırı ile boş alan çizelgesinin kullanılan boş alana ilişkin ilk satırının yeni değerlerle günlendiği gösterilmiştir.

Değişken bölümlü bellek yönetiminde yeni bir görev hazır görevler kuyruğuna bağlanacağı zaman, bu görevin gerektirdiği yer, boş alan çizelgesinden seçilirken çeşitli algoritmalar kullanılır. Bunlar genelde 3 tanedir. İlk ikisi, daha önce değişmez bölümlü bellek yönetimi kapsamında açıklanan ve göreve, ana bellek alanı olarak ya ilk uyan ya da en uygun boş alanın atanmasını sağlayan algoritmalardır. Boş alan çizelgesinin düzenleniş biçimi bunlardan hangisinin kullanılacağını belirler. Örneğin bu çizelge boy sırasında tutuluyorsa kullanılan algoritma en uygun alan *(best fit)* algoritmasıdır. Çizim 5.6'da boş alan çizelgesi boy sırasında tutuluyor olarak gösterilmiştir. İlk uyan *(first fit)* algoritmasından en uygun alan algoritmasına geçiş, belleğin parçalanma sorununu hafifletmeyi amaçlar.

Değişken bölümlü bellek yönetimi, bellekteki boş alanların bitişken yapısını parçalama eğilimindedir. İlk uyan alan algoritmasında bu eğilim, daha belirgin bir biçimde ortaya çıkar. En uygun alan algoritması boş bitişken alanları daha verimli kullanır. Ancak bu algoritma da ana bellekte, hiçbir işe yaramayan çok küçük parçalar bırakır. İlk uyan alan algoritması irili ufaklı, en uygun alan algoritması ise ufak taneli parçalanmaya neden olur. Kimi durumlarda görevlere, gereksedikleri yere karşılık en büyük boş alanı atama yaklaşımı da kullanılır. Bu yolla, zaten engellenemeyen parçalanmanın diğer görevler

için kullanılabilir büyüklükte bitişken boş parçalar yaratması amaçlanır. Bu algoritma en büyük alan *(worst fit)* atama algoritması olarak bilinir. Parçalanma sorununa ileride yeniden değinilecektir.

Görevlerin, hatalı çalışma durumunda gerek işletim sistemi ve gerekse diğer görevlere ayrılan bölümleri bozmalarını engelleyebilmek için, bu yöntem de, ana işlem birimi içinde, taban ve tavan adlı sınır yazmaçlarının kullanımını gerektirir. Bir görev ana işlem birimine anahtarlanmadan önce sınır yazmaçlarının, bölüm tanım çizelgesindeki, bu göreve ilişkin taban ve tavan değerleriyle günlenmesi gereklidir. Daha önce açıklanan yönetimlerde olduğu gibi, değişken bölümlü bellek yönetiminde de bir görev, işletiminin başında kendisine atanan bellek alanını, işletiminin sonuna değin korur. Başka bir deyişle, bir görevin, işletiminin başında yüklendiği bölüm sonradan (işletim sırasında) değiştirilemez.



Çizim 5.7. Ana Belleğin Parçalanması

Ana Belleğin Parçalanması Sorunu

Ana belleğin parçalanması, bitişken alanların görevlere atanan bölümlerle, zaman içinde ufalanması olarak tanımlanır. Bu sorun, kullanılan bölümler arasına sıkışmış, işletim için bekleyen görevlerin gereksinimini karşılayamayan boş alanların varlığıyla ortaya çıkar. Belleğin parçalanması sonucu, bellekteki boş alanların toplamı, gereksenen sığaları karşılıyor olmasına karşın yeni görevlere yer sağlanamaz durumlarla karşılaşılır. Görevlere, gereksedikleri belleğin tümüyle ve bitişken bir bütün olarak, işletime sunuş aşamasında sağlanması ve sağlanan bu alanların konumlarının işletim sırasında

değiştirilememesi parçalanma sorununun temel nedenleridir (Çizim 5.7). Parçalanma sorunu değişken bölümlü bellek yönetiminin ortaya çıkardığı bir sorun olmakla birlikte, örneğin değişmez bölümlü bellek yönetiminde, görevlere atanan bölümler içinde, görevlerden arta kalan boş alanlar da parçalanma sorunu kapsamında düşünülür. Bu bağlamda, bölüm içi yararlanılamayan boş alanlar iç parçalanma, bölümler arasında kalan boş alanlar ise dış parçalanma kapsamında düşünülür. İzleyen kesimde tartışılan parçalanma türü dış parçalanmadır.

Çizim 5.7'de parçalanmanın ortaya çıkışı örneklenmiştir. Çizim 5.7-a'da, işletim sisteminin yanı sıra ana belleğin Görev1, Görev2 ve Görev3 olarak adlandırılan üç değişik görev tarafından paylaşıldığı durum gösterilmiştir. Bu aşamada Görev4 ve Görev5'in işletime girdiği; bu amaçla Görev4'e, Görev2 ve Görev3 arasında kalan, Görev5'e ise Görev3' ü izleyen boş alandan yer atandığı varsayılmış ve bu durum Çizim 5.7-b üzerinde gösterilmiştir. Çizim 5.7-c'de, Görev2 ve Görev3'ün sonlandığı ve bunlara ilişkin bölümlerin yeniden boş alanlara dönüştüğü durum verilmiştir. Bu son çizimde, Görev2 ve Görev3 daha işletimdeyken bellek ataması yapılan Görev4 ve Görev5'in, büyük bir bitişken alanı nasıl parçaladıkları örneklenmiştir.

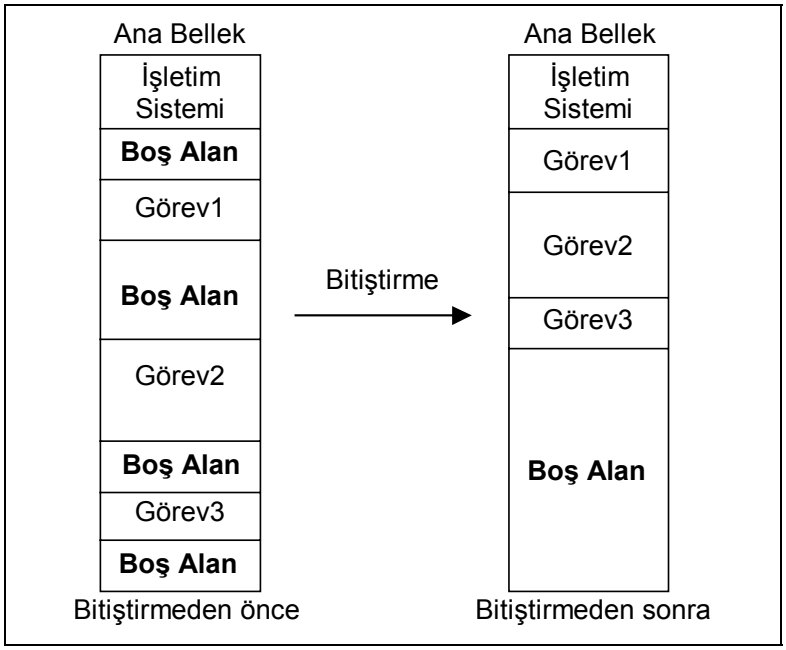
Bu son örnekte, görevlere işletime sunuş aşamasında bellek ataması yapılması ve bir göreve atanan bellek konumunun işletim sırasında değiştirilememesi sonucu bitişkin alanların parçalandığı ve bu nedenle belleğin verimli kullanımının engellendiği görülmektedir. Bu sakıncalı durumun yok edilebilmesi için yukarıda sayılan iki temel nedenden en az birinin ortadan kaldırılması gerekmektedir. Görevlere işletime sunuş aşamasında yer atanması, değişken bölümlü bellek yönetiminin çalışma ilkesidir. Bu ilkeye, değişmez bölümlü bellek yönetiminin, görevlere, konumları ve boyları değişmeyen bölümler atamasından kaynaklanan sakıncayı aşmak için ulaşılmıştır. Bu nedenle, bu ilkeden vazgeçme olanağı yoktur. Bu durumda görevlere ayrılan bölümlerin yerlerinin, işletim sırasında değiştirilebilmesinin yollarını aramak gereklidir. Ancak bu yolla, parçalanmış alanlar, yeniden daha büyük bitişken alanlara dönüştürülür.

Bitiştirme

Tüm bellek alanına dağılmış durumdaki bölümleri, yerlerini değiştirerek yanyana yerleştirme ve bu yolla bölümler arasında kalan boş alanları da yanyana getirerek tek bir bitişken boş alan yaratma işlemine bitiştirme işlemi denir (Çizim 5.8). İlgili olduğu görevin işletimi sürerken bir bölümün yerinin değiştirilmesi (bir yerden diğer bir yere kopyalanması) kolayca ve hiçbir önlem alınmaksızın yapılabilen bir işlem değildir.

Derleyiciler, program başlangıç adresinin sıfır varsayıldığı, yeri değişir olarak nitelenen amaç programlar üretirler. Bu programlar komut kodları ve işlenen adreslerinden oluşur. İşlenen adresleri, konuma duyarlı ya da konuma duyarsız değerler olarak ikiye ayrılırlar. Değişmezler, ana işlem birimi yazmaç numaraları, fiziksel giriş/çıkış kapı adresleri konuma duyarsız işlenen değerleridir. Bellek erişimli komutlarda, bellek sözcük adresleri, konuma duyarlı işlenen adreslerini oluştururlar. Komut kodları konuma duyarsız değerlerdir. Yeri değişir amaç programlardaki konuma duyarsız sözcük içerikleri, bağlama - yükleme aşamasında, oldukları gibi; adres değeri taşıyan sözcükler

ise, içerikleri bölüm başlangıç adresiyle toplanarak ana belleğe yüklenirler. Yükleme sonrasında program başına göreli adresler saltık (bellek başına göreli) adres değerlerine dönüşürler. Derleyiciler, yükleme aşamasında içerikleri dönüştürülecek, adres değeri içeren sözcükleri işaretlerler. Yükleyicinin, hangi sözcükleri doğrudan, hangi sözcükleri ise hesaplama sonrası elde edilen içeriklerle ana belleğe yükleyeceğini belirlemede yararlandığı bu işaretler, yükleme evresinden sonra yok olurlar. Yükleme sonrası hangi ana bellek sözcüğünün konuma duyarlı, hangisinin konuma duyarsız içerik taşıdığını belirlemek olanaksızlaşır. Bir görevin, işletimi sürerken, bir bellek kesiminden diğer bir bellek kesimine kopyalanması, bu nedenle yapılamaz.



Çizim 5.8. Bitiştirme İşlemi

İşletim sırasında görevlerin yerlerinin kolayca değiştirilebilmesi, dolayısıyla bitiştirme işlemlerinin esnek ve hızlı bir biçimde yapılabilmesi için bilgisayar donanımlarına özgün ekler yapmak gerekir. Bellekte saklanan veri türlerini kodlayarak bu kodları, verilerle birlikte ve bunların uzantıları olarak saklamak, bitiştirme işlemlerine altyapı oluşturan bir yöntemdir. Veriler, derleme aşamasında tamsayı, kayan ayrımlı, damga, adres gibi, konuma duyarlı ve konuma duyarsız ayrımının yapılabilmesine olanak verecek biçimde değişik türlere ayrılır. Bu türler belirli sayıda bitle kodlanır. Bu bitler, veri bitlerinin baş ya da son kesimine tür tanım eki (*tag, descriptor*) olarak eklenir. Bu altyapıyı içeren sistemlerde, görevlerin bellekteki yerleri, bitiştirme amacıyla değiştirileceği zaman konuma duyarlı (adres türündeki) sözcüklerin hangileri olduğu bilinir ve bunların içerikleri yeni değerleriyle günlenerek yer değiştirme (bitiştirme) gerçekleştirilir. Bitiştirme işlemlerine ne zaman başvurulacağına, ya işletim sistemi tarafından, bunu gerektiren koşullar oluştuğunda kendiliğinden, ya da sistem işletmeninin inisyatifiyle karar verilir. Bitiştirme işlemleri işletim kesilerek (işletim dışında) yapılan işlemlerdir.

Yukarıda sergilenen yaklaşım, bir çözüm yolu olmakla birlikte önemli sakıncalar içerir. Herşeyden önce bellek sözcüklerinin kimi bitlerini veri türü koduna ayırmayı gerektirdiğinden bellekte yer kaybına neden olur. Bunun yanı sıra, tür bitlerini sınayarak yapılan içerik günleme işlemleri aşırı zaman tüketici işlemlerdir. Bu önemli sakıncalarına rağmen bu yöntem, geçmişte *Burrough's* ve *CDC* marka bilgisayar sistemlerinde kullanılmıştır.

Bilindiği gibi, kullanıcılar programlarını doğrusal bitişken bir dizi biçiminde düşünürler. Bu programları oluşturan komut ve işlenenlerin konumları, ya birbirlerine ya da program başına göreli olarak irdelenir. Program geliştirirken kullanıcıların muhakemelerine temel oluşturan bu algılama biçimi mantıksal olarak nitelenir. Bu bağlamda komutların ve işlenenlerin konumlarının irdelendiği ortam mantıksal adres evreni olarak bilinir. Aynı komut ve işlenenlerin, bir de ana bellekte işgal ettikleri sözcükler itibarıyla (konumları) adresleri bulunur. Ana işlem biriminin ilgili sözcüklere erişimde kullandığı bu adresler fiziksel adreslerdir. Fiziksel adreslerin oluşturduğu evren, fiziksel adres evreni olarak adlandırılır. Program içi göreli adreslerden fiziksel ana bellek sözcük adreslerine geçiş mantıksal adres evreninden fiziksel adres evrenine geçiş olarak nitelenir.

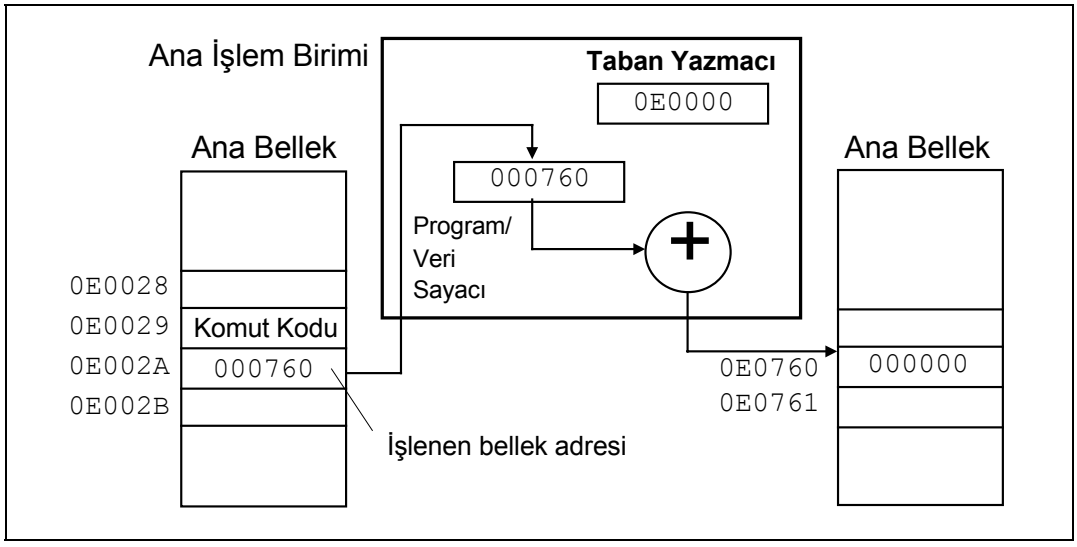
Bu geçiş işlemi, şimdiye kadar sözü edilen bellek yönetimlerinde, bağlama ve yükleme aşamalarında, program işletime alınmadan önce, bir defada, programın tümü için gerçekleştirilir. Bu yönetimler, daha önce de belirtildiği üzere programların fiziksel konumlarının işletim süresince değişmesine izin vermezler. Bundan dolayı durgun bellek yönetimleri olarak adlandırılırlar. Durgun bellek yönetimlerinde ana belleğin parçalanma sorununa, sözcüklere tür bilgileri ekleyerek bitiştirme işlemlerine başvurma gibi zorlama çözümlerin dışında etkin ve tatmin edici bir çözüm yolu bulma olanağı yoktur. Mantıksal adres evreninden fiziksel adres evrenine geçiş işlemlerini, işletimin başında bir kez yapılan bir işlem olmaktan çıkarıp her sözcük için işletim sırasında yapılan devingen bir işlem biçimine sokmak, programların ana bellekteki yerlerinin, devingen biçimde, işletim aşamasında değiştirilebilmesini sağlayarak parçalanma sorununa daha kökten bir çözüm bulma olanağı yaratır. İzleyen kesimden başlayarak açıklanacak tüm bellek yönetimleri, bunu şu ya da bu biçimde gerçekleştiren, devingen nitelikli yönetimlerdir.

5.4. Yerideğişir Bölümlü Bellek Yönetimi

Yerideğişir bölümlü bellek yönetimi kapsamında, mantıksal adreslerden fiziksel adreslere geçişi işletim aşamasına taşıyabilmek amacıyla, ana işlem birimine yerdeğiştirme taban yazmacı olarak adlandırılan özgün bir yazmaç eklenir. Yerideğişir amaç programlar ana bellekte kendilerine atanan bölümlere oldukları gibi hiçbir değişiklik yapılmadan yüklenirler. Başka bir deyişle, konuma duyarlı ögeler yükleme sonrası program başına (sıfıra) göreli adres değerlerini içermeyi sürdürürler. Her bellek erişim döngüsünde, bu adres değerleri, yerdeğiştirme taban yazmacı içeriğine eklenerek erişilen fiziksel adresler elde edilir. Bir görev ana işlem birimine anahtarlanacağı zaman yerdeğiştirme taban yazmacının içeriği, ilgili bölüm başlangıç adresiyle günlenir. Böylece işletim sırasında program başına göreli içerikler, bölüm

başlangıç adresine eklenerek bellek başına göreli saltık değerler elde edilmiş olur. Program başına göreli mantıksal adreslerden fiziksel adreslere geçiş adres dönüştürme işlemi olarak bilinir.

Çizim 5.9'da, ana bellekte 0E0029H ve 0E002AH adreslerinde yer alan bellek erişimli bir komutun bellek işleneninin, program başına göreli adresi 000760H olarak gösterilmiştir. Bu komutu içeren program, ana bellekte 0E0000H adresinden başlayan bir bölüme yüklenmiş olduğundan, işletimi sırasında yerdeğiştirme taban yazmacı da bu değerle günlenmektedir. Sözkonusu bellek erişimli komutun uygula evresinde 000760H göreli adresteki sözcük içeriğine, bu adres değerinin taban yazmaç içeriğiyle toplanması sonucu elde edilen 0E0760H adresiyle erişilmektedir. Sözkonusu iki sözcüklük bellek erişimli komutun algetir evresinde ise, bu iki sözcüğün bellekten okunup sırasıyla komut yazmacı ve veri sayacına aktarılmasında program sayaç içeriği, 000029H ve 00002AH göreli adres değerlerini içerecektir. Zira hangi gerekçe ile yapılırsa yapılsın tüm bellek erişim döngülerinde adresleme süreci hep taban yazmacı üzerinden gerçekleşecektir. Buradan görevlerin program sayaçlarının da, bellek işlenen adresleri gibi hep göreli değerler taşıyacağı görülmektedir. Bu durum programların, ana belleğe, derleme sonrası elde edildikleri biçimde yüklenerek işletilebilmelerini ve işletimleri sırasında, herhangi bir önlem almaya gerek kalmaksızın bir yerden diğer bir yere taşınabilmelerini olanaklı kılmaktadır.



Çizim 5.9. Yer Değiştirme Taban Yazmacının Kullanımı

Taban yazmacına göreli adresleme yöntemi, görevlerin ana bellekteki yerlerini, işletim sırasında esnek bir biçimde değiştirmeye ve bitiştirme işlemlerini daha az zaman kaybıyla yerine getirmeye olanak sağlayarak parçalanma sorununa daha etkin bir çözüm yolu sunar. Ana işlem birimi içinde yerdeğiştirme taban yazmacının yanı sıra bir de sınır *(bound)* yazmacı öngörülür. Sınır yazmacının varlığı, daha önce açıklanan bellek koruma düzeneğine yöneliktir. Adresleme süreci içinde hesaplanan fiziksel adres ilgili görevin aşmaması gereken sınır ya da tavan değeriyle karşılaştırılır. Aşma durumunda

bellek koruma iç kesilmesi üretilerek görevin işletimine son verilir. Sınır yazmacının içeriği, yerdeğiştirme taban yazmacında olduğu gibi, görev anahtarlama sırasında, ilgili göreve atanan bölüm boyu değeriyle günlenir.

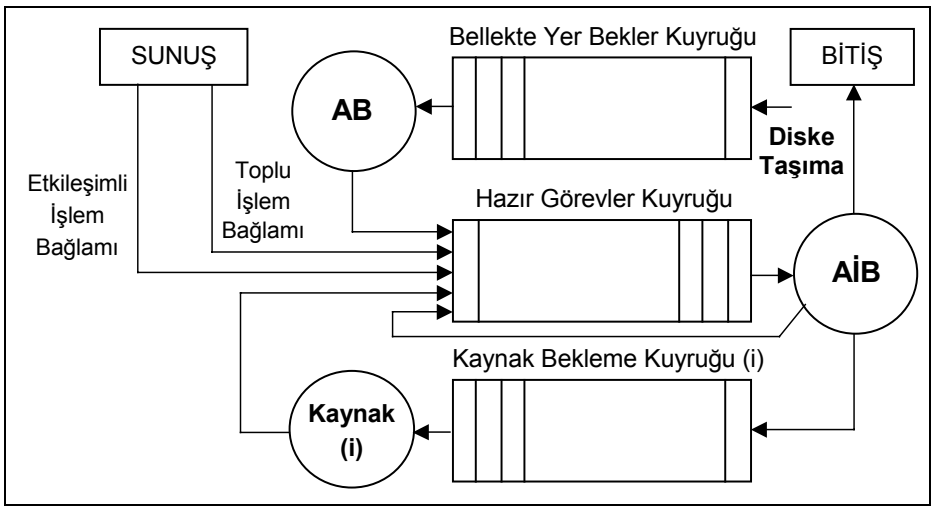
Yerideğişir bölümlü bellek yönetiminde yanıtlanması gereken bir soru da, bitiştirme işlemlerinin ne zaman ya da hangi sıklıkta yapılacağıdır. Bellekte parçalanmayı önlemek üzere bitiştirme işlemi, ya her görevin işletiminin bitiminde ya da işletime sunulan işlere bellekte yer bulunamadığı durumlarda gündeme gelebilir. Bitiştirme işlemlerini her işletim bitişte yinelemek, ana belleği çok verimli kullanmayı amaçlarken işletimi sık sık durdurarak sistem hızının ve kullanılabilirliğinin aşırı düşmesine neden olur. Bitiştirme işlemlerine, ana bellekte yer gereksinimi oluştukça başvurmak daha anlamlıdır. Bu gereksinimin ortaya çıkışı bellek yönetici tarafından saptanır. Bitiştirme işlemi, ya işletim sisteminin inisyatifinde otomatik olarak ya da sistem işletmeni uyarılarak başlatılır.

Diske Taşıma (*Swapping*)

Kimi durumlarda bitiştirme işlemleri de bellekte gerekli büyüklükte boş bitişken alan yaratmak için yeterli olmaz. Sisteme sunulan iş, o an sistemde işletimde olan işlerden daha öncelikli ise, görev tanımlarının yapılarak hemen işletime alınmasını gerektirir. Kimi az öncelikli görevlerin, işletimleri sonradan tamamlanmak üzere geçici olarak diske taşınması boş bellek alanı yaratmak için başvurulan bir yol olabilir. Bu yolla açılan boş bellek alanları yeni görevlerin tanımlanabilmesine ve öncelikli işlerin işletimlerinin bir an önce başlatılabilmesine olanak sağlar. İşletimi tamamlanmamış bir görevin, daha öncelikli görevlere ana bellekte yer açmak üzere, geçici olarak diskte, bu amaçla öngörülen alanlara taşınmasına diske taşıma *(swapping)* denir.

Diske taşıma yönteminin ortaya çıkardığı bir sorun, ana bellekte yer açmak gerektiğinde diske taşınacak görevin nasıl belirleneceğidir. Bu belirleme doğal olarak öncelikle, değişik bekleme kuyrukları üzerinde bekleyen görevler arasından yapılacaktır. Bekleyen görevler arasından, bellek alanı elinden alınacak görevi seçmek için değişik kıstaslardan yararlanılabilir. Görev iskeletinde tutulan görev önceliği bilgisi, örneğin bu amaçla kullanılabilir. Bu durumda, gerekli yeri sağlayabilen en düşük öncelikli görev bellekten çıkarılacak görev olarak seçilir. Öncelik bilgisinin yanı sıra en uzun kaynak bekleme süresi kalan, ana belleği o ana kadar en uzun süre kullanmış olan gibi başka kıstaslara göre de seçim yapılabilir. Diske taşıma işlemi, sistemde yer alan tüm görevlere uygulanmaz. Kimi görevler, yerine getirdikleri hizmetlerin kritikliği nedeniyle, dönem dönem de olsa sistemden uzaklaştırılamazlar. Örneğin kesilme hizmetleriyle ilgili görevler bu kategoride düşünülebilen görevlerdir. Bu bağlamda kimi görevler, yaratılma aşamasında sisteme, diske taşınmaz (bellekte kalıcı) görevler olarak tanımlanır. Diske taşınacak görevler bu görevlerin dışından seçilir. Bir görevin diske taşınmaz olarak tanımlanabilmesi genelde ayrıcalıklı sistem programcılarına sağlanan bir haktır. Sıradan kullanıcıların görevleri, genelde diske taşınır özellik içerir. Zira çok sayıda görevin bellekte kalıcı tanımlanması, diske taşıma işlemlerini aşırı zorlaştırır ve sistem başarımını olumsuz yönde etkiler.

Kaynak beklerken ana bellekte kendisine atanan bölüm geri alınarak diske taşınan bir görev, görev yönetimi incelenirken açıklanan bellekte yer bekler kuyruğuna bağlanır (Çizim 5.10). Bilindiği üzere, bellekte yer bekler kuyruğunda bekleyen görevleri hazır görev kuyruğuna aktarmaktan orta dönemli planlama kesimi sorumludur. Bu durumda bellekte yer bekler kuyruğu, kabaca, diske taşıma kesimi tarafından doldurulan, orta dönemli planlama kesimi tarafından ise boşaltılan bir kuyruk olarak algılanabilir .

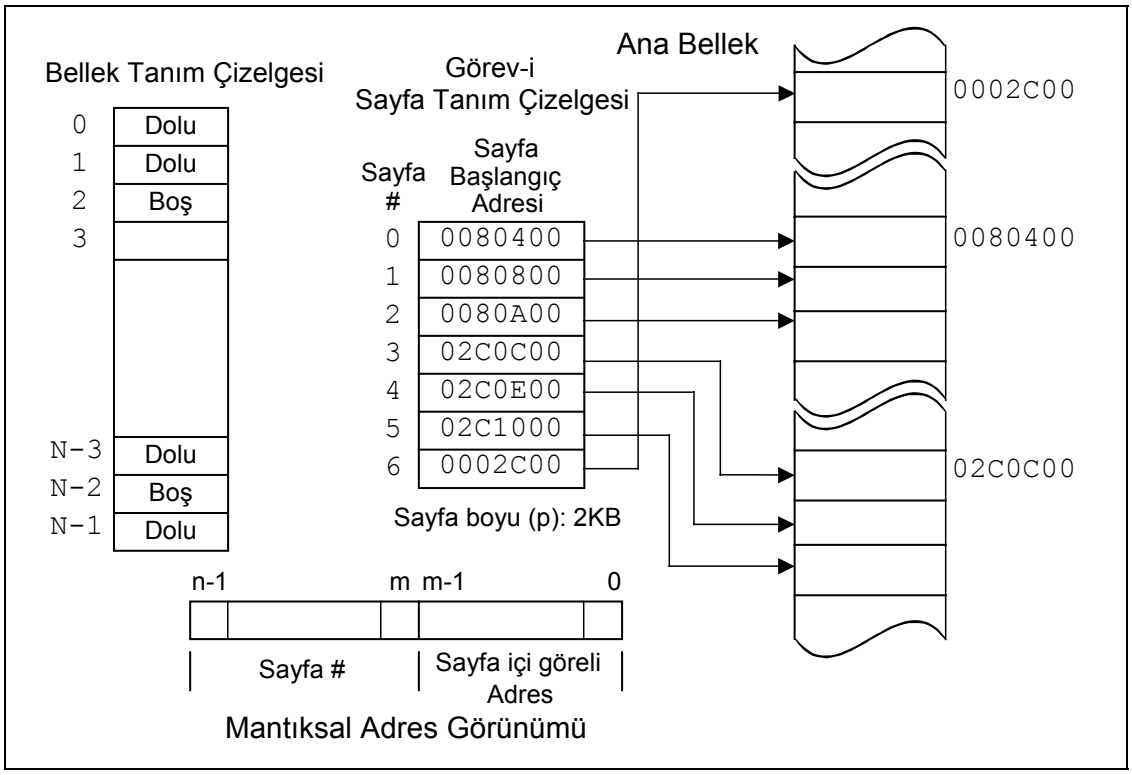


Çizim 5.10. Diske Taşıma ve Bellekte Yer Bekler Kuyruğu

Diske taşıma yöntemi, yerideğişir bölümlü bellek yönetiminin yanı sıra, şimdiye kadar incelenen diğer yönetimler kapsamında da kullanılabilir. Bu bağlamda, örneğin bu yöntemi değişmez bölümlü bellek yönetimiyle birlikte de kullanma olanağı bulunur. Sisteme yeni ve öncelikli bir iş sunulduğunda, bölüm tanım çizelgesinde, bu işin sığabileceği bir bölüm halen kullanılıyor durumda ise o bölümü işgal eden iş geçici olarak diske taşınabilir. Açılan bölüm ise daha öncelikli yeni işe atanarak işletimi hemen başlatılır. Bunun gibi, diske taşıma yöntemini değişken bölümlü bellek yönetimine uygulama olanağı da vardır. Ancak burada bir iş, diske taşıma sonrası yeniden ana belleğe döneceği zaman, kendisine ilk atanan alana taşınmak zorundadır. Zira bir işin işletim süresince yerinin değiştirilmesi bu yönetim kapsamında mümkün değildir. Bu durum, diske taşıma yöntemini, değişken bölümlü bellek yönetimiyle birlikte kullanmayı aşırı derecede güçleştirir. Bu nedenle, diske taşıma yöntemine ilişkin açıklamalara, ilk kez, yerideğişir bölümlü bellek yönetimi kapsamında yer verilmiştir. Yerideğişir bölümlü bellek yönetimiyle anlamlı bir biçimde kullanılabilen bu yöntem bundan sonra açıklanacak diğer bellek yönetimleri ile de kullanılabilecektir. Özellikle görüntü bellek yönetimleri kapsamında, diske taşıma işlemleri, çalışma ilkesinin bir parçasını oluşturacaktır.

Yerideğişir bölümlü bellek yönetimi, bitiştirme ve diske taşıma işlemlerinin hızlı ve esnek bir biçimde yapılmasını sağlar. Böylece ana bellek kullanım verimliliği artar. Ancak bu yöntemle de görevlere bellek alanları bitişken bir bütün olarak atanmak zorundadır. Görevlere gereksedikleri alanları bitişken bir bütün olarak sunabilmek için, dönem dönem parçalanan alanların bitiştirilmesi gerekir. Bitiştirme işlemleri, işletimin kesilmesine neden olan zaman alıcı işlemlerdir. Görevlere gereksedikleri alanları

bitişken bir bütün olarak sunma koşulu kaldırıldığında parçalanmayı önleyen daha etkili çözümler bulunur. İzleyen kesimde bunu gerçekleştiren sayfalı bellek yönetimi açıklanacaktır.



Çizim 5.11. Görev Sayfa Tanım ve Bellek Tanım Çizelgeleri

5.5. Sayfalı Bellek Yönetimi

Sayfalı bellek yönetiminde görevlerin mantıksal adres evrenleri, birbirini izleyen, eşit uzunlukta parçalardan oluşur. Bu parçalar program sayfası olarak adlandırılır. Bu evren içinde adreslerin iki birleşeni bulunur (s,x). Bu birleşenlerden ilki (s) sayfa numarası, ikincisi ise (x) sayfa başına göreli adrestir. Sayfa boyunun p sözcük olduğu varsayılırsa bir sözcüğün program başına göreli adresi (s.p + x) olarak hesaplanır. Mantıksal adres evreninin sayfalı olarak düzenlenmesi fiziksel adres evreninin de aynı biçimde düşünülmesini gerektirir. Bu durumda ana belleğin, p sözcük uzunluğunda N adet sayfadan oluştuğu varsayılır. (p.N) değeri ana belleğin toplam sığasını belirler. Ana belleği oluşturduğu varsayılan sayfalar bellek sayfaları olarak adlandırılır. Sayfalı bellek yönetiminin uygulandığı sistemlerde, amaç programlar içindeki tüm adresler, sayfa numarası ve sayfa içi göreli adresten oluşur biçimde düşünülür. Ana işlem birimi program ve veri sayaçları da iki kesimli yapıdadır.

Sayfalı bellek yönetiminde görevlerin her program sayfasına bir bellek sayfası atanır. Mantıksal adres evreni içinde bitişken olarak yer alan program sayfalarının ana bellekteki karşılıklarının bitişken olma koşulu aranmaz. Bu yolla görevlere, ana bellekte

bitişken alan bulma zorunluluğu ortadan kalkar. Hangi program sayfasının hangi bellek sayfasında bulunduğunu belirleyebilmek amacıyla, her görev için, bir eşleme çizelgesi tutulur. Bu çizelgeler sayfa tanım çizelgesi[18] olarak adlandırılır. Sayfa tanım çizelgeleri, ilgili görevin program sayfalarının yer aldığı bellek sayfası giriş (başlangıç) adreslerini tutarlar. Her görev için ayrı ayrı tutulan bu sayfa tanım çizelgelerinin yanı sıra, bir de, tüm ana bellek için bellek tanım çizelgesi tutulur. Bu çizelge içinde hangi bellek sayfasının dolu, hangisinin boş olduğu bilgisi bulunur. Görevler işletime alınacağı zaman önce bu çizelge taranır. Görevin gereksediği sayıda boş bellek sayfası belirlenir. Görevin sayfa tanım çizelgesi oluşturularak içeriği, belirlenen bu boş sayfa giriş adresleriyle günlenir. Bellek tanım çizelgesinde, kullanılan fiziksel sayfalar dolu olarak işaretlenir.

Çizim 5.11'de, 7 sayfadan oluşan Görev-i 'nin sayfa tanım çizelgesi ile ana bellek tanım çizelgesi örneklenmiştir. Bellek tanım çizelgesi, gerçekleştirim yönünden bit matrisi, bağlaçlı sayfa listesi, bağlaçlı boş sayfa listesi gibi değişik yapılarda tutulur. Çizim üzerinde bu çizelge, gerçekleştirimden bağımsız kavramsal görünümüyle verilmiştir. Görevin mantıksal adres evreninde, ardarda yer alan program sayfalarının ana bellekte, karışık sıradaki fiziksel sayfalara atandığı örneklenmiştir.

Sayfalı bellek yönetiminin uygulandığı sistemlerde, görevlerin mantıksal adres evrenlerinden fiziksel adres evrenine geçişi sağlayacak özgün donanım düzeneklerine gerek duyulur. Anımsanacağı üzere, yerideğişir bölümlü bellek yönetiminde mantıksal adresler program başına göreli adreslerdir. Fiziksel adresler, işletim sırasında, program başına göreli mantıksal adreslerin bölüm başlangıç adresine eklenmesiyle elde edilir. Bölüm başlangıç adresi yerdeğiştirme taban yazmacı içinde tutulur. Yerideğişir bölümlü bellek yönetiminde hem mantıksal hem de fiziksel adres evrenleri bitişken tek bir bölümden oluştuğundan yerdeğiştirme taban yazmacı da biriciktir. Sayfalı bellek yönetiminde mantıksal adresler sayfa başına göreli olduğundan fiziksel adresler, sayfa başına göreli mantıksal adreslerin sayfa başlangıç adresine eklenmesiyle elde edilir. Bunun için hangi sayfa işletimde ise, adres dönüştürme işlemleri o sayfa başlangıç adresini taban almak durumundadır. Bir görev birden çok sayfadan oluşabildiğine göre, ana işlem birimi de, düz bir mantıkla, sayfa sayısı kadar sayfa taban yazmacı bulundurabilir.

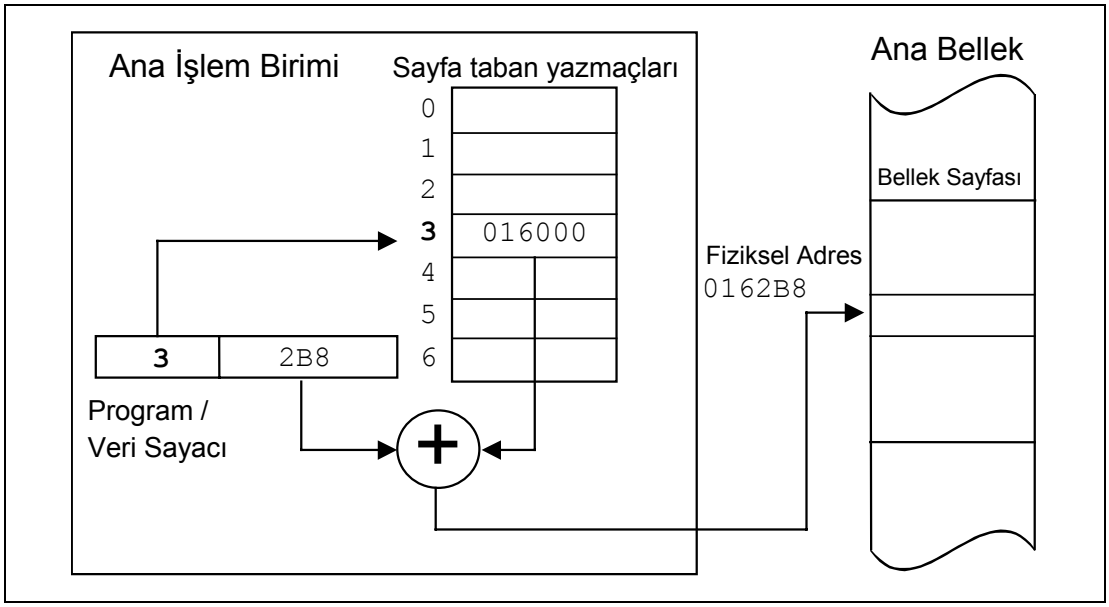
Bu modele göre bir görev ana işlem birimine anahtarlandığında taban yazmaçları, görevin sayfa tanım çizelgesindeki başlangıç adresleriyle günlenir. Adresleme süreci içinde, sayfa numarasının gösterdiği taban yazmaç içeriği sayfa içi göreli adrese eklenerek[19] fiziksel ana bellek adresi elde edilir (Çizim 5.12). Ana işlem birimleri, ekonomik nedenlerle, genelde sayfa sayısı kadar taban yazmacı bulunduramaz. Örneğin,

---

[18] Sayfa Tanım Çizelgesi, İngilizce *Page Description Table (PDT)* karşılığı kullanılmıştır. Ancak bu çizelge için, İngilizce *Dynamic Adres Translation Table (DAT)*, *Page Map Table (PMT)* adlandırmaları da kullanılmaktadır.

[19] Burada sözü edilen ekleme, mutlaka aritmetiksel anlamda bir ekleme (toplama) işlemi olmak zorunda değildir. Kimi uygulamalarda, iki birleşenin yanyana getirilmesi yoluyla da bu ekleme gerçekleştirilebilir. Örneğin sayfa başlangıç adresi, adresin üst kesimini, sayfa içi göreli adres de alt kesimini gösterebilir. Böylece bu iki birleşen yanyana getirildiğinde *(concatenation)* adresin tamamı elde edilir.

sayfa boyunun 4 KB olduğu bir sistemde, 1 MB uzunluğunda bellek alanı gerektiren bir görev için, ana işlem biriminin 250 adet taban yazmacı içermesi gerekir.

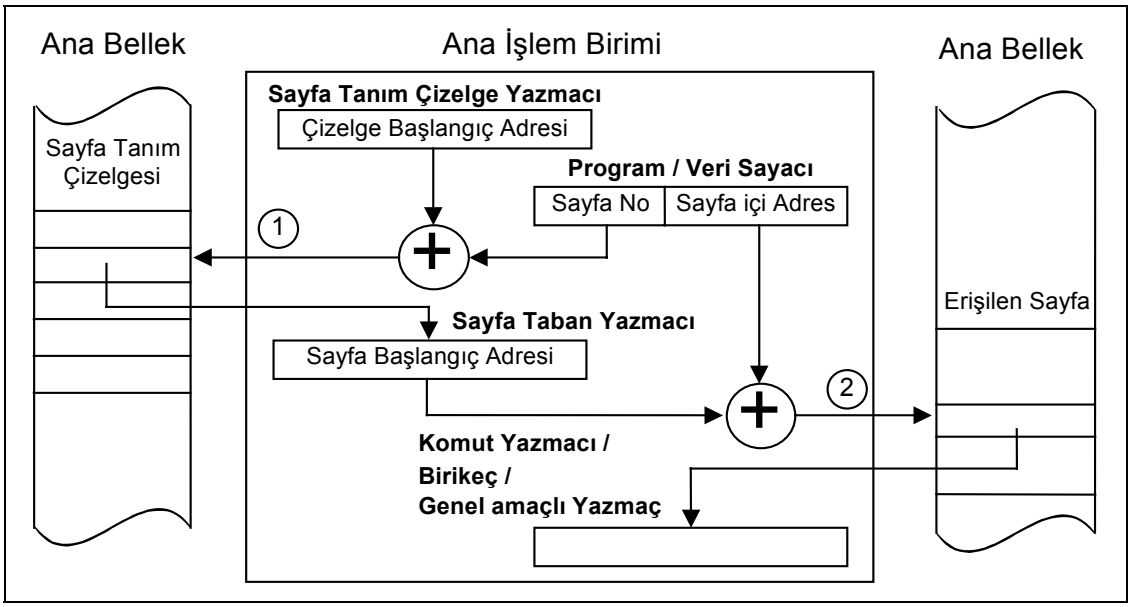


Çizim 5.12. Sayfalı Bellek Yönetimi Adresleme Mantığı

Bu çok pahalı çözüm yerine, salt işletilen sayfa başlangıç adresini tutan tek bir taban yazmacı ile yetinilebilir. Ana işlem biriminde, bu taban yazmacının yanı sıra, bir de sayfa tanım çizelgesi yazmacı olarak adlandırılan bir ikinci yazmaç daha öngörülür. Bir görev ana işlem birimine anahtarlandığında bu ikinci yazmaç içeriği, işletilen görevin sayfa tanım çizelgesi başlangıç adresiyle günlenir. Belleğe erişim iki adımda gerçekleşir. İlk adımda erişilen sözcüğün yer aldığı mantıksal sayfa numarasına karşı gelen bellek sayfa başlangıç adresi, ana bellekte tutulan sayfa tanım çizelgesinden okunarak sayfa taban yazmacına taşınır. İkinci bir adımda da, bu yazmaç içeriği ile sayfa içi göreli adres değeri toplanarak ilgili fiziksel adres elde edilir. Bu ilkeye göre, mantıksal adreslerden fiziksel adreslere geçiş süreci iki bellek erişimi gerektirir. Bu uygulama bellek erişim süresini iki kat artırır (Çizim 5.13). Bilgisayar sistemlerinde bellek erişim süresi, sistem başarımını doğrudan etkileyen çok önemli bir parametredir. Komut işletimi, algetir evresinde mutlaka, uygula evresinde ise çoğu kez ana belleğe erişim gerektirdiğinden bellek erişiminin iki kat uzaması sistem başarımının yaklaşık iki kat düşmesi anlamına gelir. Ana belleğin parçalanmasını önlemek uğruna işletim hızının aşırı düşmesine neden olmak benimsenebilir bir yaklaşım değildir. Bu gerekçeyle, sayfa tanım çizelgesine yapılan ek erişimin süresini kısaltmanın yolları aranır. Bu amaçla sayfa tanım çizelgesi, çok hızlı erişime izin veren özel bir bellek biriminde tutulur. Böylece bu çizelgeye yapılan ek erişimin, toplam erişim süresi içindeki payı azaltılmaya çalışılır.

Çok hızlı erişime olanak veren bellek birimleri ön bellek *(cache)* olarak adlandırılır. Ana belleği oluşturan sıradan bellek birimlerinin (yongalarının) erişim süreleri 50 - 100 nanosaniye arasındadır. Ön bellek birimlerinde ise bu süre 5 - 10 nanosaniyeyi aşmaz. Sayfa tanım çizelgelerinin hızlı bellek birimleri üzerinde saklanması, bu çizelgelere

yapılan ek erişimin, sıradan sayfalara yapılan erişimin onda biri gibi, benimsenebilir bir sürede gerçekleşmesini sağlar. Bu durumda sayfalı bellek yönetiminin bellek erişimine getirdiği artış %10 düzeyinde kalır.



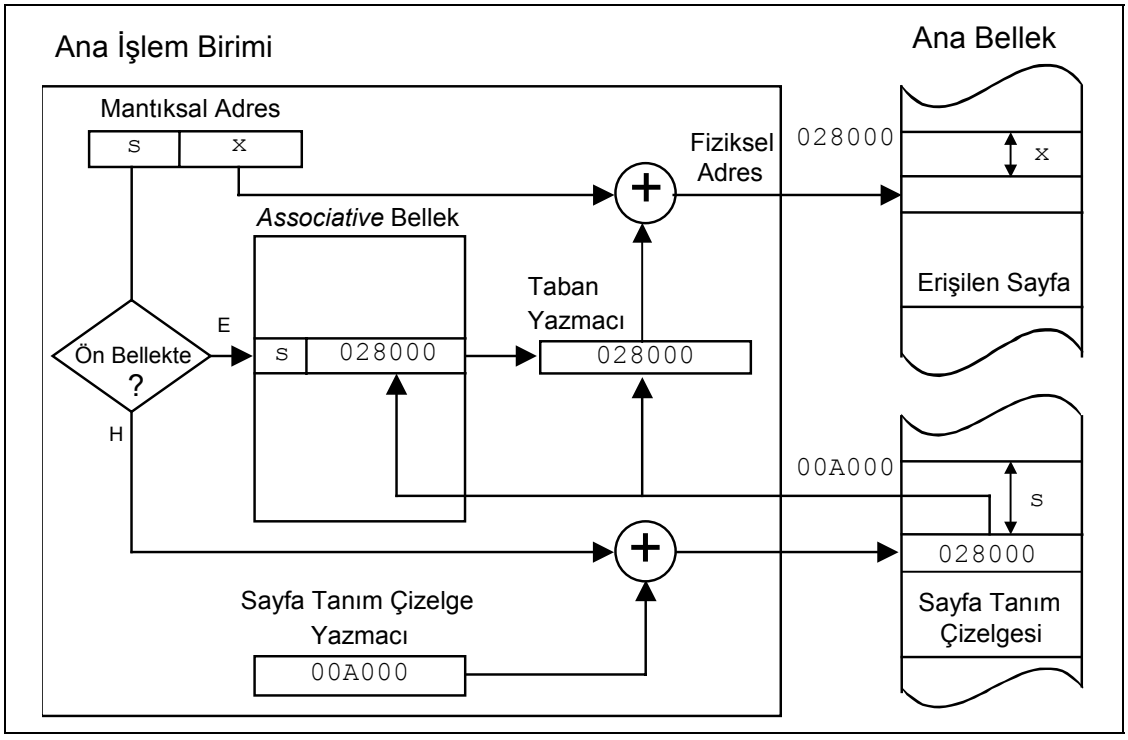
Çizim 5.13. Sayfa Tanım Çizelge Yazmacının Kullanımı

Belleğe erişimde, bire on iyileştirme sağlayan hızlı bellek birimleri aşırı pahalı birimlerdir. Bu nedenle çoğu kez sayfa tanım çizelgesinin tümünü tutacak sığada olmayabilirler. Bu kısıtlama sonucu, sayfa tanım çizelgesinin tümü, yine sıradan bellek kesiminde tutulurken ön belleğin, son kullanılan birkaç sayfa başlangıç adresini içermesiyle yetinilir. Ancak bu durumda, adresleme sürecinin başında erişilen sayfa başlangıç adresinin ön bellekte bulunup bulunmadığının sınanması; sayfa başlangıç adresi ön bellekte değilse, ön belleğe taşınmasının sağlanması gereklidir. Bu olgudan hareketle, sayfalı bellek yönetiminin uygulandığı sistemlerde, belleğe erişimi bu sınamayla aynı anda gerçekleştirebilen *associative* tür bellekler kullanılır[20]. Bu tür belleklerde sözcüklere adresleri ile erişmek yerine doğrudan içerik değerleri ile erişilir. Çalışma ilkesi gereği, bellek erişim döngüsü içinde *associative* bellek birimi, girişlerine uygulanan içeriğin bellekte bulunup bulunmadığını sınar. Bulunuyorsa bununla ilişkili bir değeri veri yolu üstüne aktarır. Aranan içerik bellekte yoksa bir göstergeyi kurar. Bilindiği gibi, klasik belleklerde bir birim, erişim için seçildiği sürece uygulanan adres değerine karşıt gelen sözcük içeriğini veri yolu üstüne yükler. Klasik bellekler için olmayan adres, olmayan içerik gibi kavramlar yoktur.

Sayfalı bellek yönetiminde *associative* bellekler, sayfa tanım çizelgesinde yer alan ve en sık erişilen kimi sayfa başlangıç adreslerini tutmak için kullanılır. Adres dönüştürme süreci içinde, adreslenen mantıksal sayfa numarasına karşıt gelen bellek sayfa başlangıç

[20] Bu tür bellekler, İngilizce *translation look aside buffer (TLB)* olarak da adlandırılır.

adresi önce *associative* ön bellekte aranır. Bunun için sayfa numarası ön belleğe uygulanır. Bu numara bellekte ise, ilişkili olduğu sayfa başlangıç adresi erişim döngüsü sonunda, ön bellekçe sağlanır. Sayfa taban yazmacına aktarılan bu değer sayfa içi göreli adrese eklenerek fiziksel adres elde edilir ve adres dönüştürme süreci son bulur. Uygulanan sayfa numarası ön bellekte değilse, ön bellek erişim döngüsü sonunda, olmayan numara uyarısı üretilir. Bu uyarı ile denetim birimi, adres dönüştürme işlemlerini sayfa tanım çizelgesinin bellekteki kopyasına dayalı olarak gerçekleştirecek döngüyü başlatır. Bu bağlamda elde edilen sayfa başlangıç adresi gelecek erişimlerde kullanılmak üzere ön belleğe de yazılır.



Çizim 5.14. Sayfalı Bellek Yönetimi ve Ön bellek Kullanımı

Sayfalı bellek yönetiminin uygulandığı, adres dönüştürme işlemlerini, yukarıda açıklanan modele göre yürüten bir sistemde ana belleğe ortalama erişim süresi:

$$t_e = t_d + t_m$$

olarak ifade edilebilir. Burada t_d adres dönüştürme amacıyla harcanan ortalama zamanı, t_m ise, adres dönüştürme süreci sonunda elde edilen fiziksel adrese erişim süresini göstermektedir. Adres dönüştürme süresi t_d :

$$t_d = p\ t_c + (1\text{-}p)\ (t_c + t_m) = t_c + (1\text{-}p)\ t_m \quad (0 \le p \le 1)$$

olarak yazılabilir. Burada t_c ön belleğe erişim süresini, p ise sayfa başlangıç adresini ön bellekte bulma olasılığını göstermektedir. Bu durumda:

$$t_e = t_d + t_m = t_c + (2\text{-}p)\ t_m$$

olur. Yukarıda örneklendiği gibi, ana bellek erişim süresi ile ön bellek erişim süresi arasında (7ns ve 70ns gibi) bire on bir oran bulunduğu; erişilen sayfa başlangıç adresini ön bellekte bulma olasılığının ise 0.85 olduğu varsayılırsa:

$$t_e = 0.10\ t_m + (2 - 0.85)\ t_m = 1.25\ t_m$$

olarak bulunur. Buna göre, sayfalı bellek yönetiminin bellek erişim süresine getirdiği artışın, ön bellek kullanımı sayesinde %25'ler düzeyinde tutulabildiği söylenir. Zira ön bellek kullanılmadığı durumlarda t_e değerinin $2.t_m$ olacağı unutulmamalıdır. Sayfalı bellek yönetimi kapsamında, şimdiye değin açıklanan adres dönüştürme düzeneklerinin herbiri için, ek erişim yükünü simgeleyen $(t_e-t_m) / t_m$ oranları hesaplanarak Çizelge 5.1'de topluca, karşılaştırma amacıyla verilmiştir. Bu çizelgede, eder / hız (fiyat / performans) yönünden, sayfa tanım çizelgesinin bir kesimini ön bellekte tutan düzeneğin, sayfalı bellek yönetimi için en iyi adres dönüştürme düzeneği olduğu kolayca görülebilmektedir.

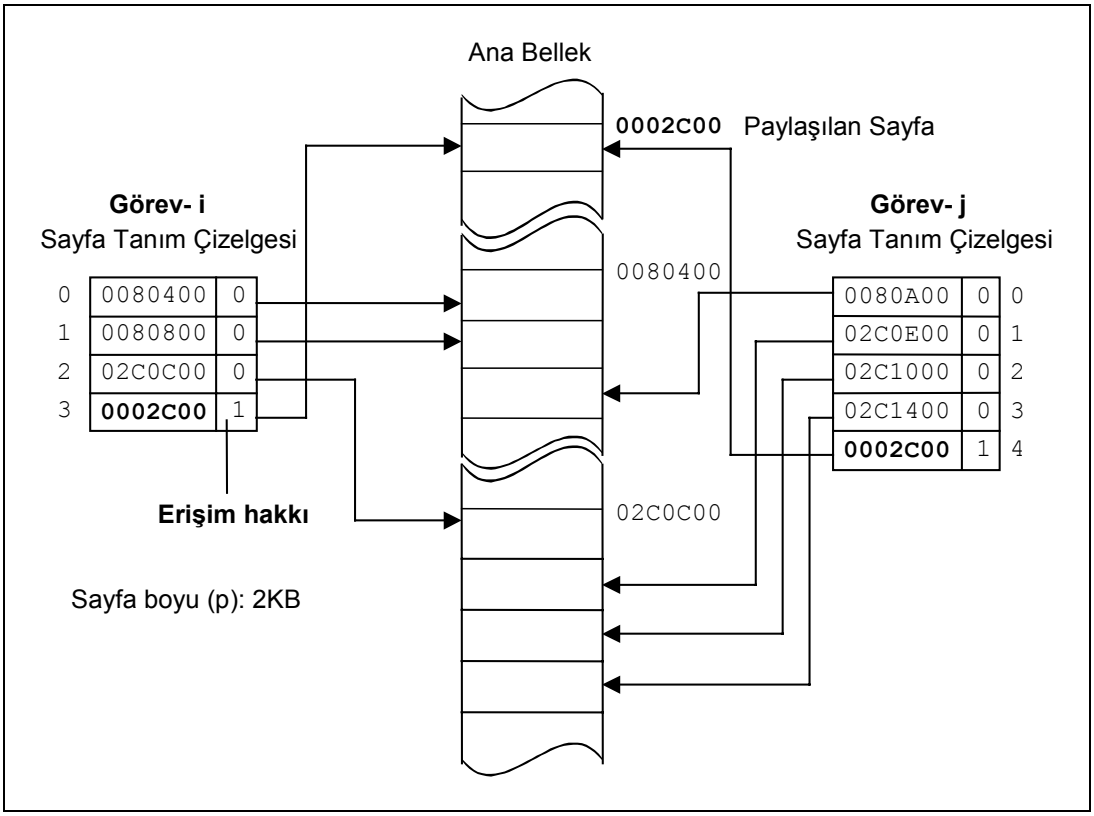
Çizelge 5.1. Adres Dönüştürme Düzeneklerinin Karşılaştırılması

Adres Dönüştürme Düzenek Türü	**$(t_e-t_m) / t_m$**
1. Sayfa Tanım Çizelgesinin AİB Taban Yazmaçlarında	% 0
2. Sayfa Tanım Çizelgesinin tümüyle Ön Bellekte tutulması	% 10
3. Sayfa Tanım Çizelgesinin bir kesiminin Ön Bellekte tutulması	% 25
4. Sayfa Tanım Çizelgesinin tümüyle Ana Bellekte tutulması	% 100

Görevlerin Adres Evreninin Kesişmesi, Sayfa Paylaşımı

Ana belleği paylaşan görevlerin, birbirlerinin alanlarına erişimlerinin engellenmesi işletim bütünlüğü açısından gereklidir. Görevlerin, kendilerine atanan sınırlar içinde kalmalarının denetlenmesi bellek koruma düzeneği ile sağlanır. Etkili bir bellek koruma düzeneği iyi bir bellek yönetimi için aranan önemli bir özelliktir. Ancak görevlerin bellek alanlarının kesiştirilmesi, yordam ve verilerin ortaklaşa kullanılması yoluyla, bellek kullanım verimliliğini artıran bir yol olarak da düşünülebilir. Bu bağlamda, bellek koruma ile alan paylaşım işlevlerini bağdaştırma yolları aranır. Buraya kadar incelenen bölümlü bellek yönetimlerinde, adres evrenlerinin bitişken bir bütün olarak ele alınma zorunluluğundan dolayı, bellek koruma düzeneği ile paylaşımı bağdaştırmanın yalın ve esnek bir yolunu bulma olanağı yoktur. Sayfalı bellek yönetiminde ise, değişik görevlere ilişkin birden çok program sayfasının, aynı fiziksel bellek sayfasıyla eşleştirilmesi mümkündür (Çizim 5.15). Böylece mantıksal adres evrenleri itibarıyla birbirlerinden kopuk olan görevlerin fiziksel adres evrenlerini çakıştırma yolu bulunur. Bu durum, bir yandan ana belleğin kullanım verimliliğini artırırken diğer yandan da koşut işlem olanağını yaratır. Zira aynı fiziksel bellek sayfalarının birden çok görev tarafından paylaşılması, görevlere ortak veri kümeleri üzerinde eşanlı işlem yapma olanağı sağlar. Bellek bölümlerinin kesiştirilemediği bellek yönetimlerinde görevler, ya aynı veri kümesinin değişik kopyaları üzerinde çalışmak ya

da işledikleri veri kümelerini, tümüyle diğer görevlere kapamak zorunda kalırlar. Her iki durumda da, anlamlı ve gerçek bir koşut işlemden söz edilemez.



Çizim 5.15. Sayfalı Bellek Yönetiminde Adres Evrenlerinin Kesişmesi

Bölümlü bellek yönetimlerinde, görevlerin birbirlerinden kopuk fiziksel konumlarda yer alması, bir fiziksel adres evreninin bir diğeriyle kesişmemesi, görevler arası doğal bir koruma düzeneği oluşturur. Sayfalı bellek yönetiminde fiziksel bellek sayfalarının birden çok görevin sayfa tanım çizelgesinde yer alması, bellek koruma açısından, ek önlemlerin alınmasını gerektirir. Bu bağlamda paylaşılan sayfaların erişim hakları belirlenir. Hangi görevin hangi sayfa üzerinde hangi haklara sahip olduğu bilgisi kodlanır. Bu bilgiye dayanılarak görevlerin sayfalara erişimi denetim altında tutulur. Sayfa tanım çizelgelerinde, sayfa başlangıç adresine ek olarak, bir de ilgili sayfaya ilişkin erişim hakkı bilgisi tutulur. Bunlar, okuma - yazma, salt okuma, işletim gibi haklar olabilir. Sayfa başlangıç adresi, sayfa taban yazmacına taşınırken erişim kodu bitleri de ana işlem birimine taşınır. Sayfa içinde, bu kod değeriyle çelişen erişim istemleri sözkonusu olduğunda, bellek koruma iç kesilmesi üretilerek hakkını aşan görevin işletiminin kesilmesi sağlanır.

Görevler arası sayfa paylaşımının zorladığı erişim hakkı denetiminin yanı sıra, sayfalı bellek yönetiminde, yine bellek koruma düzeneği çerçevesinde, adres dönüştürme süreci içinde, sayfa tanım çizelge boyunun aşılması durumunun denetlenmesi de gerekir. Bu amaçla ana işlem biriminde, sayfa tanım çizelgesi ve sayfa taban yazmaçlarının

yanında, bir de, sayfa tanım çizelgesi sınır yazmacı bulunur. Bir görev ana işlem birimine anahtarlandığında bu yazmaç, görevin sınır değeri ile günlenir. Adres dönüştürme işlemleri sırasında sayfa tanım çizelgesine erişimde, çizelge sınırları içinde kalınması denetlenir. Bunun için, çizelgeye erişim amacıyla elde edilen fiziksel adres değeri, sayfa tanım çizelgesi sınır yazmacı içeriğiyle karşılaştırılır. Çizelgenin aşılması durumunda, bellek koruma iç kesilmesi üretilerek görevin işletimine son verilir. Bu koruma düzeneği, doğal olarak sayfa tanım çizelgesinin ana bellekteki kopyasına erişilirken devreye girer. Ön bellekteki sayfa giriş adreslerine erişim bu düzeneğin dışındadır.

Sayfalı bellek yönetimi, ana belleğin parçalanmasını, yerideğişir bölümlü bellek yönetimine göre daha etkin biçimde engeller. Görevlere bitişken alan atama zorunluğunun ortadan kalkması, bitiştirme işlemlerinin de ortadan kalkmasını sağlar. Belleği eşit uzunlukta sayfalar olarak düzenleyen bu yönetimde bellek atama işlemleri yalın işlemler olarak kalır. Bu iyileştirmelerin bedeli olarak, komut işletiminin süresi az da olsa artar. Ana işlem birimine yapılan özgün donanım ekleri sistem maliyetini olumsuz yönde etkiler. Her görev için tutulan sayfa tanım çizelgeleri ile bellek tanım çizelgesi, harcanan bellek alanı yönünden işletim yükünü artırır. Sayfalı bellek yönetimi, görevlere bitişken bellek atama zorunluluğunu ortadan kaldırmasına karşın görev sayfalarının tümünün, işletim öncesinde ana belleğe yüklenmiş olmasını gerektirir.

5.6. Kesimli Bellek Yönetimi

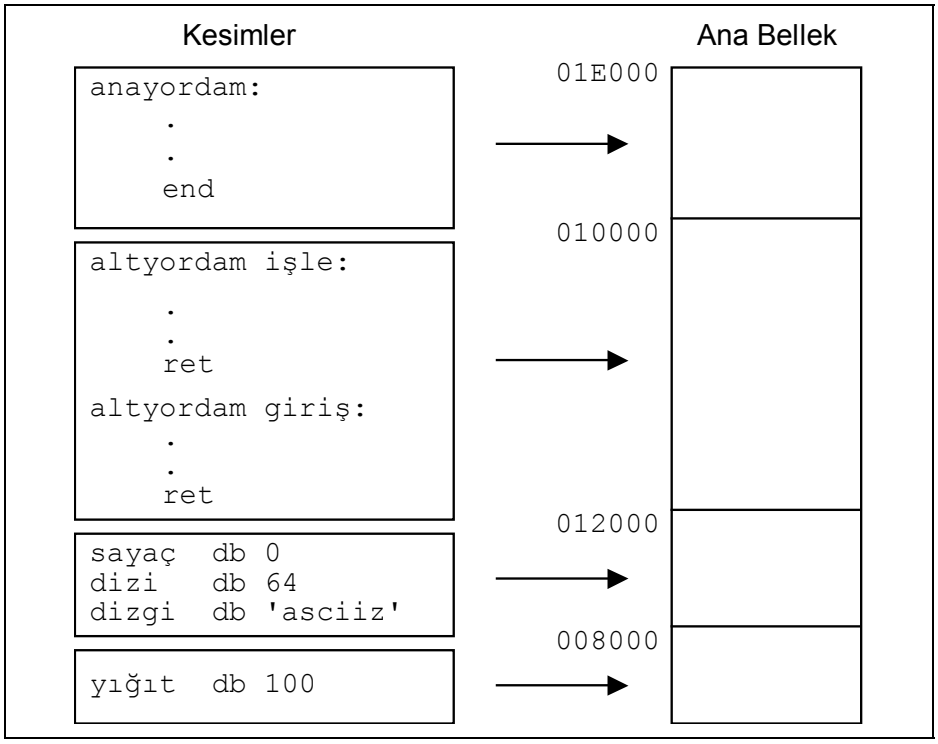
Sayfalı bellek yönetiminde, programların mantıksal adres evrenlerini eşit uzunlukta sayfalara ayırmak ve bu sayfaları ana bellekte bitişken olma zorunluğu bulunmayan fiziksel sayfalarla eşleştirmek ilkesi, görevlere atanacak bitişken bellek parçalarını küçülterek kullanılmayan en küçük parçayı bile değerlendirebilmeyi amaçlar. Görevlere atanacak bitişken bellek parçalarını küçültmenin bir yolu da program adres evrenlerini kesimlere ayırmaktır. Kesimler, programlar içinde, içerikleri yönünden mantıksal bütünlüğü bulunan parçalara verilen addır. Programlar, örneğin, ana yordamın bulunduğu kesim; alt yordamların bulunduğu kesim; değişken, dizi, dizgi gibi, verilerin topluca tutulduğu kesim; yığıt olarak kullanılan kesim gibi değişik türde kesimlerden oluşmuş biçimde düşünülebilir. Programları oluşturan bu bağımsız kesimlerin birbirleriyle bitişken olarak düşünülmeleri de gerekmez. Kesimlerden oluşan programlar içindeki adresler, sayfalama yöntemine benzer biçimde kesim kimliği (numarası) ve kesim içi adres olarak, iki birleşenden oluşur. Kesim içinde adresler kesim başına (sıfıra) görelidir.

Sayfalı ve kesimli bellek yönetimleri birbirlerine çok benzerler. Ancak bu iki yönetim biçimi arasında önemli temel ayrımlar vardır:

- Programların mantıksal adres evrenleri; sayfalı bellek yönetiminde eşit uzunlukta sayfalara, kesimli bellek yönetiminde ise değişken uzunlukta, mantıksal bütünlüğü taban alan kesimlere ayrılır.

- Kesimli bellek yönetiminde kesimler, sistem programcılarının programlarını oluştururken doğrudan kullandıkları, sayılarını ve adlarını özgürce tanımlaya-bildikleri; üst düzey kullanıcıların ise amaç programlarına, derleyiciler tarafından yerleştirilen, bu yolla programların mantıksal adres evrenlerini kesimler biçiminde düşünmeyi gerektiren nesnelerdir. Sayfalı bellek yönetiminde sayfa kavramı, her düzeyden kullanıcıya saydam bir kavramdır. Sayfalama, programın mantıksal adres evrenini mekanik biçimde eşit uzunluktaki parçalara bölen bir işlemdir.

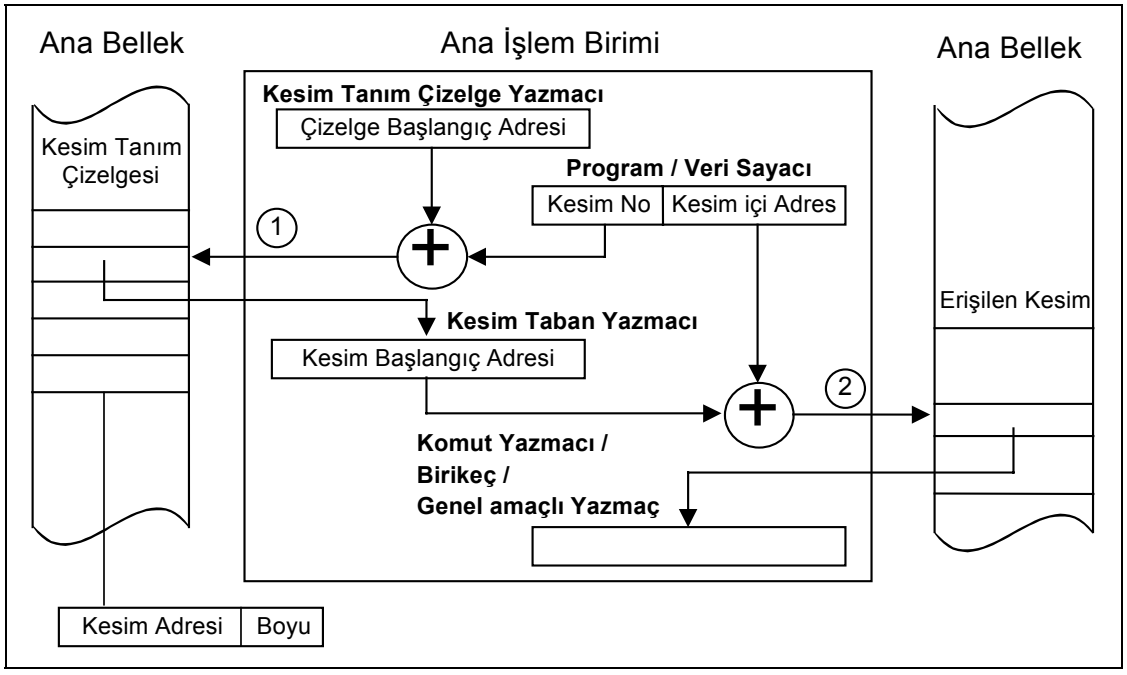
Programların mantıksal adres evrenini oluşturan kesimler ana bellekte, değişken uzunlukta fiziksel kesimlerle eşleştirilir (Çizim 5.16). Bu eşleştirme, kesim - bellek konumu ilişkisini tutan Kesim Tanım Çizelgeleri aracılığıyla gerçekleştirilir. Kesim tanım çizelgesi ilgili olduğu görevin kesimlerinin ana bellekteki başlangıç adreslerini tutar. Kesim uzunlukları, kesimden kesime değişebildiğinden bu çizelgelerde, başlangıç adreslerinin yanı sıra kesim boyu bilgisine de yer verilir.



Çizim 5.16. Kesimler ve Ana Bellekte yer aldıkları Konumlar

Kesimli bellek yönetiminin uygulandığı sistemlerde, adres dönüştürme işlemleri, sayfalı bellek yönetiminde olduğu gibi, ilke olarak üç değişik yolla yapılabilir. Bu yollardan ilki bir görevin içerebileceği tüm kesimlerin başlangıç adreslerini tutacak sayıda kesim taban yazmacının, ana işlem birimi içinde öngörülmesidir. Böylece bir görev ana işlem birimine anahtarlandığında taban yazmaçları görevin kesim tanım çizelgesindeki giriş adresleri ile günlenir. Adres dönüştürme işlemleri, mantıksal adres içindeki kesim numarasının gösterdiği taban yazmacı kullanılarak gerçekleştirilir. Bu yol, sayfalı bellek

yönetimi incelenirken belirtildiği gibi, ekonomik nedenlerle kullanılan bir yol değildir. İkinci yolda ise tek bir taban yazmacı ile yetinilir. Ancak bu taban yazmacının yanında, bir de, işletilen görevin kesim tanım çizelgesi başlangıç adresini tutacak çizelge yazmacı öngörülür. Bir görev ana işlem birimine anahtarlandığında bu çizelge yazmacı, görevin kesim tanım çizelgesi başlangıç adresiyle günlenir. Adres dönüştürme süreci içinde, önce, ana bellekte saklanan bu çizelgeye erişilir. Mantıksal adres içindeki kesim numarasının gösterdiği kesim giriş adresi taban yazmacı içine taşınır. İkinci bir adımda da, taban yazmaç içeriği kesim içi adres değerine eklenerek hedef sözcüğe erişilir. Bu ikinci yol bellek erişim süresini iki katına çıkarır.



Çizim 5.17. Kesim Tanım Çizelgesinin Ana Bellekte Saklanması

Bellek erişim süresinin uzamasını önlemek üzere, açıklanan bu iki yolun dışında, yaygın olarak kullanılan bir üçüncü yol daha vardır. Bu yol, kesim başlangıç adreslerini ana işlem birimi içindeki kesim taban yazmaçlarında tutmayı ilke edinen birinci yola benzer. Ancak ana işlem biriminde, kesimlerin herbiri için ayrı bir taban yazmacı gerektirmez. Programları oluşturan kesimleri, değişik türler altında sınıflandırarak her kesim yerine, her sınıf için bir taban yazmacıyla yetinilmesini sağlar. Bu yolla, adres dönüştürme sürecinde, ana işlem birimi kesim taban yazmaçlarının kullanılması ekonomik uygulanabilirlik kazanır, bellek erişim hızının düşmesi engellenir. Sözkonusu bu üçüncü yolun dayandığı temel gözlem şöyledir: Komut işletimi kapsamında:

- Komut algetir evresinde komut kodu ve işlenen adresine,
- Komut uygula evresinde ise verilere ve yığıta erişim gerçekleştirilir.

Bu erişimler, aynı komut için, hem komut kodunun bulunduğu bellek alanına, hem verilerin yer aldığı alana, hem de yığıt olarak adlandırılan alana yapılabilir. Örneğin `pop memword` ya da `push memword` türü komutlar, işletimleri sırasında:

- hem komut kodunun bulunduğu alana,
- hem `memword` adlı işlenenin bulunduğu alana,
- hem de yığıt başında yer alan sözcük için yığıt alanına

erişimi gerektirirler.

Bu gerekçeye dayalı olarak programlar, komut, veri, yığıt *(code, data, stack)* gibi belirgin türlere giren kesimlerden oluşur biçiminde düzenlenirler. Buna bağlı olarak, her kesim türü (sınıfı) için, ana işlem biriminde, yine komut, veri, yığıt *(code, data, stack)* adlı kesim yazmaçları öngörülür. Bu yazmaçlar içinde, işletilen görevin komut, veri ve yığıt türü kesimlerinin ana bellekte yüklendiği alanların başlangıç adresleri tutulur. Kesimlerin sınıflandırılması, adres dönüştürme işlemlerinde gereksinim duyulacak tüm kesim taban adres değerlerini, aynı anda, ana işlem birimi içinde bulundurma olanağını yaratır. Kesim yazmaçları sayesinde, kesim başlangıç adreslerini okumak üzere, sistemli biçimde ana belleğe ek erişim yapma zorunluğu ortadan kalkar. Kesimleme düzeneğinin bellek erişim hızını yavaşlatması engellenir[21].

Programlar içinde aynı türden birden çok kesim bulunabilir. Bir program, örneğin k adet *code*, m adet *data*, n adet de *stack* türüne giren kod_1,...,kod_k; $veri_1$,...,$veri_m$; $yığıt_1$,...,$yığıt_n$ adlı kesimler içerebilir. Ancak işletim sırasında her tür kesimden yalnız bir tanesinin başlangıç adresi ana işlem birimi içinde tutulur. Komut işletimi sırasında, o anda kesim taban yazmaçları içinde hangi adres değerleri bulunuyorsa onlar kullanılır. İşletim sırasında, kesim değiştirmek gerektiğinde, örneğin, $veri_1$ kesimi işlenirken, $veri_3$ kesimindeki verilerin işletimine geçmek gerektiğinde *data* kesim yazmacının günlenmesi gerekir. Kesim yazmaçlarının günlenmesi, bu yöntemle program geliştirme ya da derleme aşamasında programa katılan, `move, load` türü komutlarla, program içinden gerçekleştirilir. Çizim 5.18'de `yerel-veri` kesimi üzerinde işlem yapılırken `genel-veri` kesimindeki değişkenlere erişilmesi gerektiğinde, `es` kesim yazmacının `genel-veri` başlangıç adresiyle günlenmesinin program içinde nasıl öngörüldüğü örneklenmiştir.

Çizim 5.18'de kesim yazmaçlarının, programlara eklenen komutlarla günlendiği durum örneklenmiştir. *data* ve *stack* kesim yazmaçlarının günlenmesi için geçerli olan bu yöntem, *code* kesim yazmacının günlenmesinde kullanılamaz. Zira kesimler arası sapma ve yordam çağırma komutlarının işletimi, bu günlemenin, code kesim yazmacı için otomatik olarak yapılmasını zorunlu kılar. Bu amaçla, bu tür kesimleme düzeneğini kullanan sistemlerde, sapma ve yordam çağırma komutları, yakın ve uzak adlandırmasıyla, kesim içi ve kesimler arası olmak üzere ikiye ayrılır. Buna bağlı olarak, kesimler arası sapma / çağırma komutları için, derleyiciler, program sayacının yanı sıra kesim yazmacını da günlemeyi sağlayacak ekleri programlara katarlar.

---

[21] Bu bağlamda, Intel 80486 işleyicisinde *code (CS), stack (SS), data (DS), (ES), (FS), (GS)* adlı 6 adet kesim yazmacı bulunur.

Kesim değiştirmede kesim taban yazmaçlarının günlenme sorumluluğunun uygulama programlarına bırakılmış olması, mantıksal adresler içinde kesim numarası tutmayı gereksiz kılar. Bu durumda mantıksal adresler salt kesim içi adreslerden oluşur. Kesim tanım çizelgesi, adres dönüştürme sürecinde, sistemli biçimde erişilen bir çizelge olmaktan çıkıp, kesim yazmaçlarını günlemek gerektiğinde yararlanılan kesim taban adreslerinin tutulduğu bir çizelgeye dönüşür.

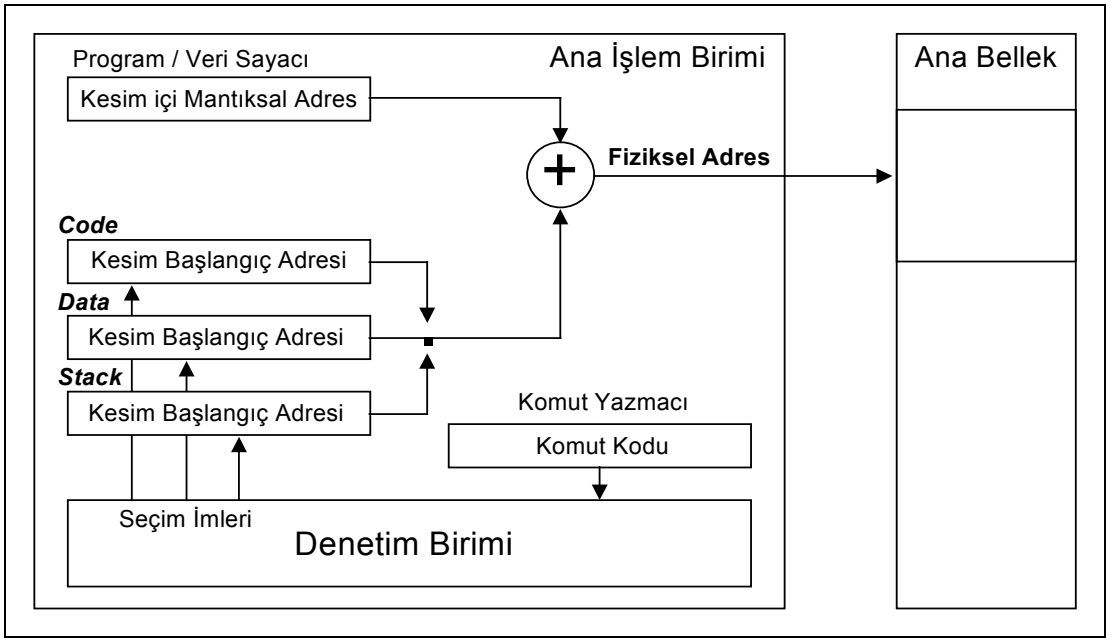
```
yerel-veri   segment
             .
             .
yerel-veri   ends

genel-veri   segment  public
             public   değişken1,değişken2
değişken1    dw       ?
değişken2    dw       ?
             .
             .
genel-veri   ends

kod1         segment
             assume cs:kod1,ds:yerel-veri,es:genel-veri
             .
             .
             mov ax, genel-veri
             mov es, ax
             mov bx, değişken1    ;bu komut es'yi taban alır.
             mov değişken2, bx    ;bu komut es'yi taban alır.
             .
kod1         ends

             end
```

Çizim 5.18. *ASM86*'da Kesim Tanımları ve Kesim Yazmaçlarının Günlenmesi

Mantıksal adresler içinde kesim numarasının tutulmaması, ana işlem birimi içindeki *code, data, stack* gibi birden çok kesim taban yazmacından hangisinin, adres dönüştürme işlemlerine taban alınacağı sorusunu doğurur. Kesimli bellek yönetimini gerçekleştirmek üzere tür taban yazmaçlarını kullanan işleyicilerde, adres dönüştürme işlemlerinde kullanılacak taban yazmaç bilgisi, genelde işletilen komut kodunda gizlidir. Denetim birimi, taban alınacak kesim yazmacını otomatik olarak seçer.

Kullanılacak taban yazmacı bilgisinin komut kodu içinde gömülü olması, istenen kesime özgürce erişimi kısıtlayabilir. Bu sakıncayı ortadan kaldırmak üzere, tersi belirtilmediği sürece, kendiliğinden kullanılacak kesim taban yazmacının kimliğini değiştirme olanakları yaratılır. Bu olanak *(segment override),* kesim zorlama olarak bilinen öneklerle yapılır. Program geliştirme aşamasında, kullanılması istenen kesim kimliği, ilgili değişkenin önüne yerleştirilir. Derleme aşamasında, bu değişkeni işleyen bellek erişimli komut kodunun önüne *(segment override prefix),* kesim zorlama

öneki olarak adlandırılan özel bir ek kod yerleştirilir. Bu, ana işlem birimini, ilgili komutun içine gömülü yazmaç bilgisi ne olursa olsun önekteki yazmacı, taban yazmacı olarak kullanmaya zorlar.

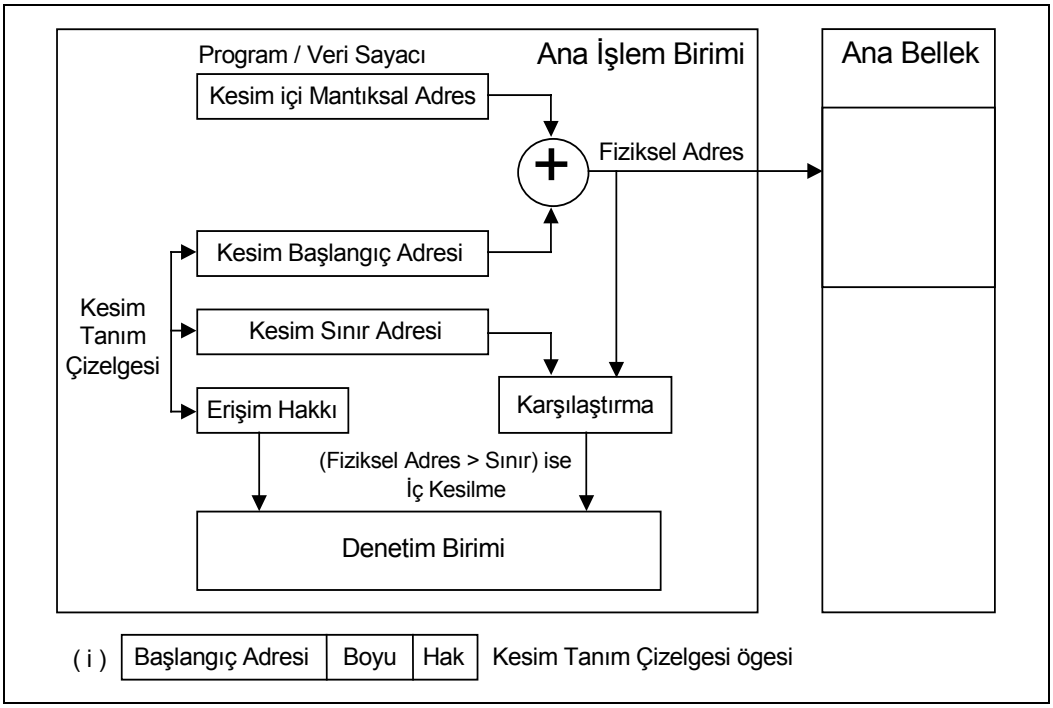


Çizim 5.19. Kesimlerin Türlere Ayrılması

Kesimli bellek yönetiminde, mantıksal adreslerden fiziksel adreslere geçiş, kullanılan adres dönüştürme yöntemi ne olursa olsun, kesim - bellek konumu ilişkisini tutan kesim tanım çizelgeleri aracılığıyla gerçekleştirilir. Kesim tanım çizelgesinde, kesim başlangıç adreslerinin yanı sıra kesim boyu bilgisine de yer verilir. Bu bilgi, ilgili kesim işletilirken, hatalı çalışma sonucu oluşabilecek kesimden taşmaları denetlemek üzere kullanılır. Ana işlem biriminde kesim taban yazmaçlarının yanı sıra, herbir kesim için bir de sınır yazmacı öngörülür. Taban yazmaçları kesim başlangıç adresleriyle, sınır yazmaçları da kesim boyu bilgisiyle günlenir. Adres dönüştürme sırasında elde edilen fiziksel adres, ana işlem birimi içinde yer alan özel bellek koruma düzeneği aracılığıyla, hangi kesime erişiliyorsa o kesimin sınır adresiyle karşılaştırılır. Taşma saptanırsa bellek koruma iç kesilmesi oluşturularak görevin işletimi kesilir ve denetim işletim sistemine bırakılır.

Sayfalı bellek yönetiminde olduğu gibi, kesimli bellek yönetiminde de görevlerin kimi bellek alanlarını paylaşması esnek bir biçimde gerçekleştirilebilir. Ortak kullanılan kesimlere, ilgili tüm görevlerin kesim tanım çizelgesinde yer verilerek paylaşım sağlanır. Görevler ortak kesimlere, genelde aynı haklarla erişmezler. İşletim bütünlüğünün ve güvenli kullanımın sağlanabilmesi için, ortaklaşa kullanılan kesimlere yazma amacıyla erişimlerin kısıtlanması gereklidir. Bu amaçla kesim tanım çizelgelerinde kesim başlangıç adresi ve kesim boyu bilgisinin yanı sıra, bir de erişim hakkı bilgisine yer verilir. Kesimlere erişim hakkı; okuma, okuma - yazma, işletim biçiminde kodlanır. Belleğe, izin verilenin dışında bir amaçla erişim söz konusu edildiğinde ilgili görevin işletimi, koruma amacıyla sonlandırılır.

Kesimli bellek yönetiminde koruma düzeneği bir yandan kesimden taşmaları diğer yandan da paylaşılan kesimlerin erişim haklarının denetlenmesini sağlar. Bu koruma düzeneğinin dışında, kesimlerin türlere ayrılarak, bu türlerin herbiri için ayrı bir kesim taban yazmacının kullanılması da korumaya katkı veren bir yapıyı oluşturur. Zira bu yapı sayesinde örneğin veri ya da yığıt kesimindeki değerlerin, hatalı işletim sonucu komut kodu olarak işletilmesi engellenmiş olur. Bunun gibi kod kesimine erişimlerde yazma yasaklanarak komut kodlarının bozulması ve olası bozulmaların işletim bütünlüğüne olumsuz etkileri engellenebilir. Başka bir anlatımla, kesimlerin türlerine göre kümelenmesi, bunlar üzerinde yürütülen işlemlerin, türleriyle uyumlu olup olmadığı konusunda denetim yapabilme olanakları yaratır.



Çizim 5.20. Kesimli Bellek Yönetiminde Bellek Koruma Düzeneği

Görevlerin kesimler biçiminde düzenlenmesi, bunlara ana bellekte tümüyle bitişken alan atama zorunluğunu ortadan kaldırmakla birlikte, (dış) parçalanma sorununu sayfalı bellek yönetiminin etkililiği düzeyinde çözemez. Kesimli bellek yönetimiyle de, dönem dönem bitiştirme işlemlerine başvurma gereği bulunur. Kimi kesimlerin birden çok görev tarafından paylaşılabilmesi bu kesimlerin başlangıç adreslerinin birden çok kesim tanım çizelgesinde yer almasıyla sağlanır. Bu kesimlerin yerlerinin değiştirilmesi durumunda birden çok çizelgenin günlenmesi gereği ortaya çıkar. Bu durum bitiştirme işlemlerinin karmaşıklaşmasına ve uzamasına neden olabilir[22]. Bunun yanı sıra

[22] Bu sakıncadan dolayı, kesimli bellek yönetimine altyapı sağlayan kimi sistemlerde kesim tanım çizelgeleri genel *(global)* ve yerel *(local)* olarak ikiye ayrılır. Paylaşılan kesimler genel tanım çizelgelerinde saklanır. Böylece paylaşılan kesim başlangıç adreslerinin günlenme işlemleri yalınlaştırılmış olur.

görevlerin, dolayısıyla ana belleğin değişik uzunlukta kesimler biçiminde düzenlenmesi bellek atama yöntemlerini karmaşıklaştırabilir. Kesimli bellek yönetimi, bu sakıncalarına karşın, aşağıda açıklanacak görüntü bellek düzeniyle, en yaygın kullanılan birkaç bellek yönetiminden birini oluşturur.

Görüntü Ana Bellek Düzeni

Şimdiye değin incelenen bellek yönetimlerinin uygulandığı sistemlerde, görevlerin mantıksal adres evrenlerinin boyu ana belleğin fiziksel sığası ile sınırlı kalır. Her bir ögesi 64 bit üzerinden kodlanan 300 x 300 boyutlu, tek bir matrisin bile 9.10^4 x 8, yaklaşık 1 MB'lık bir ana bellek alanı gerektirmesi bunun ne kadar kısıtlayıcı olduğunu açıkça gösterir. Görevlerin mantıksal adres evrenlerinin ana belleğin fiziksel sığası ile sınırlı olduğu sistemlerde bu kısıtlamayı aşmanın sorumluluğu programcılara bırakılır. Bellek kısıtlaması nedeniyle bir seferde işletilemeyen programlar ve veriler, daha küçük parçalara ayrılarak adım adım, ayrı işletimlerle ele alınırlar. Bu zorlama, doğal olarak özlenen bir işletim biçimi değildir. Tüm gerçek bellek yönetimlerine ortak, bu önemli sınırlamanın aşılabilmesi, bir görevin gereksediği toplam bellek alanının, işletiminin başında atanması zorunluğunun ortadan kaldırılması ile gerçekleşir.

Bir programın işletimi zaman içinde incelendiğinde işlenen komut ve verilerin belirli bölgeler içinde kaldığı görülür. Genel amaçlı bir uygulama programının özel bir işletimde, gereksediği toplam bellek alanının çok kısıtlı bir kesimini kullanması sıkça karşılaşılan bir durumdur. Bu durum programı oluşturan parçaların birbirlerini dışlamalarından kaynaklanmaktadır. Kullanıcının girdiği seçeneklerle işletimi yönlendirilen bir paket programda, sözkonusu edilmeyen seçeneklere ilişkin program kesimlerinin ana bellekte bulunmasının hiçbir gereği yoktur. Bu gözlemin sonucunda bir programın işletimini başlatabilmek için kimi parçalarının ana belleğe yüklenmiş olmasının yeterli olabileceği görülür.

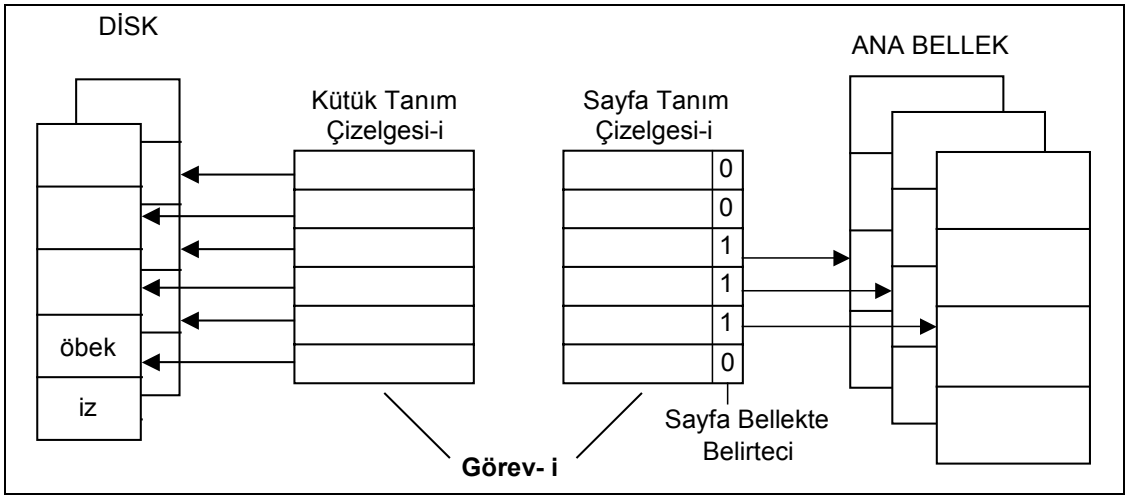
Bu bağlamda, görüntü bellek yönetiminin uygulandığı sistemlerde, işletime alınan görevlere, tüm adres evrenlerini karşılayacak sığada ana bellek alanı yerine disk alanı atanır. Görüntü bellek düzeni sayfalı, kesimli ve bu ikisinin karması kesimli-sayfalı bellek yönetimleri ile birlikte kullanılır. Görev işletimi başlatılırken ilk kesimin ya da ilk sayfaların ana bellekte bulunmasıyla yetinilir. Ancak, her an, gereksinim duyulup da ana bellekte bulunmayan kesim ya da sayfaların buraya nasıl taşınacağı sorusunun yanıtlanması gereklidir. Bu soru, aşağıda, görüntü bellek düzenini kullanan sayfalı, kesimli ve kesimli - sayfalı bellek yönetimleri çerçevesinde yanıtlanacaktır.

5.7. Sayfalı Görüntü Bellek Yönetimi

Sayfalı bellek yönetiminde amaç programlar, derleyiciler tarafından eşit uzunlukta sayfalar biçiminde hazırlanırlar. Görüntü bellek düzeni çerçevesinde, hazırlanan bu programların ana belleğe yüklenmeye hazır kopyaları, tümüyle diskte oluşturulur. Diskte yer alan sayfalardan bir kesimi ana belleğe yüklenerek işletim başlatılır. Bilindiği üzere, işletim aşamasında görevlerin mantıksal sayfaları ile fiziksel ana bellek sayfaları, sayfa tanım çizelgeleri aracılığıyla eşleştirilir. Sayfa tanım çizelgeleri bir görevin mantıksal sayfalarına karşı gelen ana bellek sayfa başlangıç adreslerini tutarlar. Görüntü

bellek düzeninde program sayfaları diskte de saklandığından işletime alınan her görev için, sayfa tanım çizelgesi gibi, bir de kütük tanım çizelgesi tutulur. Kütük tanım çizelgeleri program sayfalarının diskte saklandığı tutanak adreslerini içerir (Çizim 5.21).

Görüntü bellek düzeninde tüm program sayfalarının ana bellekte bulunma zorunluğu olmadığından sayfa tanım çizelgelerinde, sayfa başlangıç adreslerinin yanına, ilgili sayfanın belleğe yüklü olup olmadığını gösteren bir durum belirtecinin eklenmesi gerekir. Adres dönüştürme aşamasında, erişimine hazırlanılan sayfanın ana bellekte bulunmadığı durumlarda, bu belirteç aracılığıyla, gerekli önlemleri alma olanakları yaratılır. Çoğu kez bir bit uzunluğundaki bu durum belirteci, sayfa bellekte biti olarak bilinir. Sayfa tanım çizelgesinin erişilen sayfayla ilgili satırı, fiziksel adres hesaplama amacıyla ana işlem biriminde sayfa taban yazmacına taşındığında belirteç biti denetim birimince sınanır. Sınama sonucu ilgili sayfanın ana belleğe yüklü olmadığı anlaşılırsa adres dönüştürme işlemleri yarıda kesilerek denetim bellek yöneticiye bırakılır. Bu, bir makina komutunun, işletimin ortasında kesilmesi demektir[23].



Çizim 5.21. Sayfalı Görüntü Bellek Düzeni ve Kütük Tanım Çizelgesi

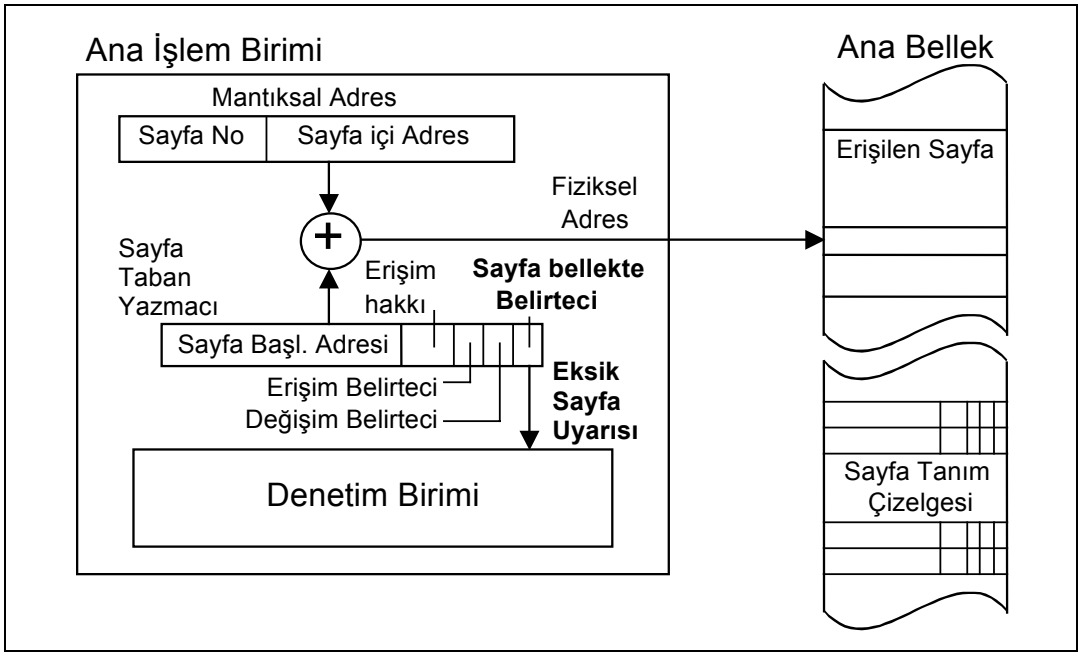
Adres dönüştürme sürecinin ortasında, denetimin donanımdan bellek yönetici yazılıma aktarılmasından sonra gerçekleşmesi gereken işlemler şunlardır:

a. Bellekte göreve atanacak boş bir sayfa aramak.
b. Bellekte kullanılabilir boş bir sayfa yoksa, yer açmak üzere ana bellekten çıkarılacak sayfanın seçimini yapmak.
c. Ana bellekten çıkarılacak sayfada günleme yapılmışsa (diskteki kopyasından farklı ise) sayfayı diskteki yerine yazmak.
d. Erişilmek istenen sayfayı ana belleğe yüklemek.
e. Yarıda kesilen komutun işletimini yeniden başlatmak.

---

[23] Şimdiye değin, işletim akışının denetlenmesi yönünden söz konusu edilegelen iç ve dış donanım kesilmeleri hep komut aralarında gerçekleşen işletim kesilmeleri olmuştur.

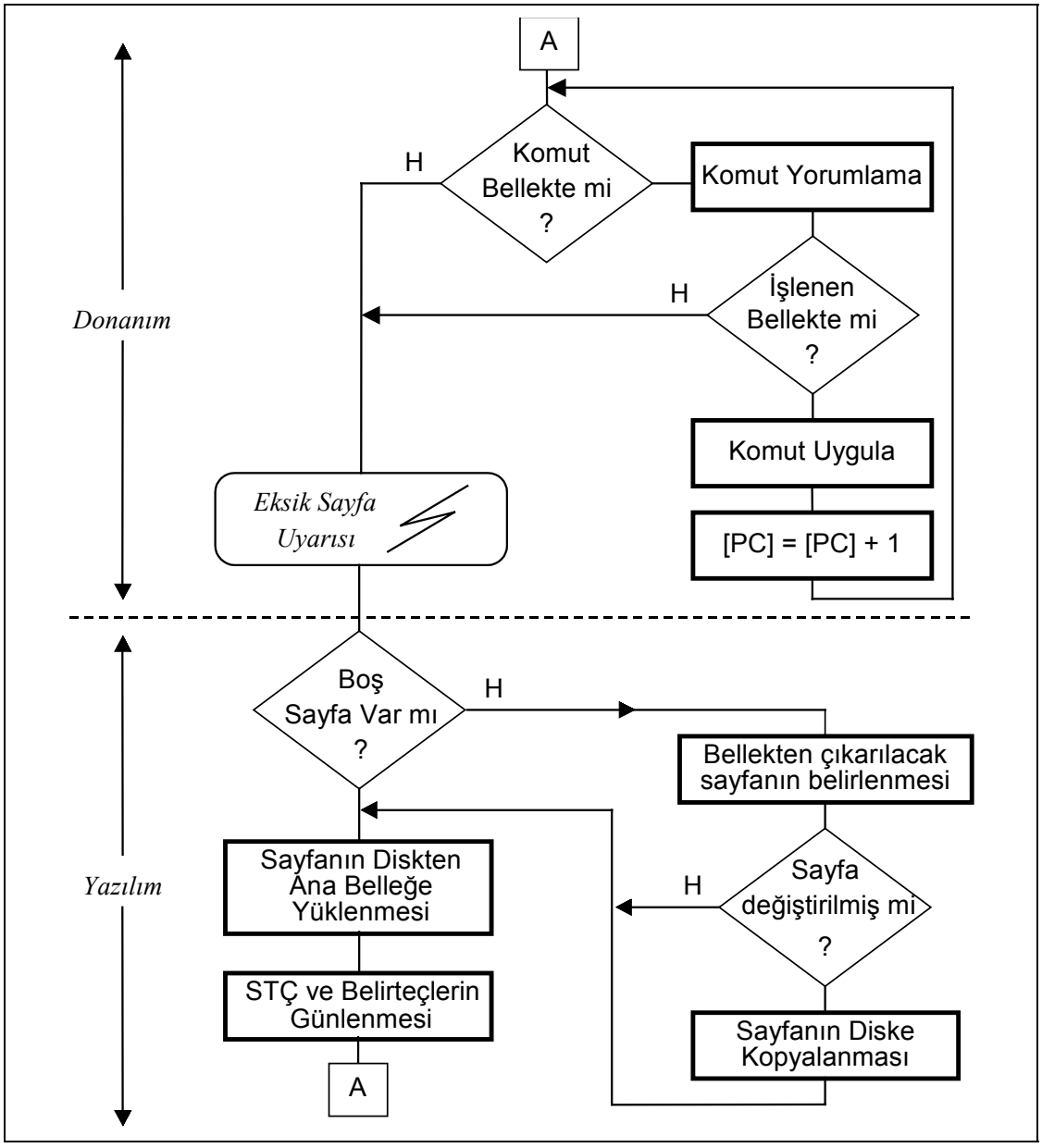
Sıralanan bu işlemlerin de çağrıştıracağı gibi, sayfa tanım çizelgelerinde, sayfa başlangıç adresleriyle birlikte tutulan sayfa bellekte belirtecinin yanı sıra, başka belirteçlerin varlığına da gerek duyulur. Örneğin, yukarıda (c) şıkkında sözü edilen, aynı sayfanın bellekteki ve diskteki kopyaları arasındaki farklılıkların saptanmasına yarayacak ve sayfaya ilk yazma işleminde kurulacak değişim belirteci bunlardan birisidir. Bunun gibi, yer darlığı nedeniyle ana bellekten çıkarılacak sayfayı seçerken yararlanılacak ve sayfaya herhangi bir nedenle erişimde kurulacak erişim belirteci de bu belirteçlerden bir diğerini oluşturacaktır. Bu durumda sayfa tanım çizelgesinin bir satırı, ilgili sayfanın başlangıç adresi, erişim hakkı bitleri ile erişim, değişim ve sayfa bellekte belirteçlerini içerecektir (Çizim 5.22).



Çizim 5.22. Sayfalı Görüntü Bellek Yönetimi

Sayfalı görüntü bellek yönetiminde fiziksel adresi hesaplanmakta olan sözcüğün bulunduğu sayfa ana bellekte değilse denetim donanımdan yazılıma aktarılarak yukarıda sıralanan bir dizi işlem yerine getirilir. Bu işlemlerden ilki bellekte göreve kullanabileceği boş bir sayfa bulmaktır. Bir görev, çoğu kez hiçbir sayfası ana belleğe yüklenmeden işletime alınır. Görev, ana işlem birimine anahtarlanıp ilk komutu işletilirken, eksik sayfa uyarısı üreterek ilk sayfasının ana belleğe yüklenmesini sağlar. Görevin diğer sayfaları da bu biçimde sırayla ana belleğe yüklenirler. Ancak bir göreve ana bellekte ayrılabilecek fiziksel sayfa sayısı, genelde işletimin başında belirlenen sayıyla kısıtlanır. Örneğin bir göreve aynı anda 10 sayfadan fazla bellek sayfası kullanım izni verilmez. Görevin izin verilen sayıda sayfası ana belleğe yüklendikten sonra, yeni sayfalarına, eski sayfaları bellekten çıkarılarak yer açılır. Bu yaklaşım yerel sayfa çıkarma politikası *(local replacement policy)* olarak bilinir. Bu politikanın tersi, genel sayfa çıkarma politikasıdır *(global replacement policy).* Bu son politikayla, görevlere atanacak fiziksel sayfa sayısına bir kısıtlama konmaz. Bunun sonucu olarak,

bir göreve yer açmak gerektiğinde diğer görevlerin sayfalarının da bellekten çıkarılması söz konusu edilir. İzleyen kesimde açıklanan sayfa çıkarma algoritmaları, anlatım yalınlığı ve kolaylığı amacıyla, salt yerel sayfa çıkarma politikası kapsamında ele alınmıştır.



Çizim 5.23. Sayfalı Görüntü Bellek Yönetiminde Adres Dönüştürme Süreci

Bu bağlamda bellekte bulunmayan bir sayfaya erişim söz konusu olduğunda, eksik sayfa uyarısı ile bellek yönetici ana işlem birimine anahtarlanır. İlgili göreve atanacak boş bir sayfa bulunup bulunmadığı sınanır. Boş bir sayfa bulunabilirse bu sayfanın başlangıç adresi, görevin sayfa tanım çizelgesinde, erişilen mantıksal sayfa satırına işlenir. Bu satırdaki, sayfayla ilgili belirteçler de günlenerek yarıda kesilen komutun

işletiminin başına geri dönülür. Bir komutun işletimi ya komuta (algetir evresinde) ya da işlenene erişimde (uygula evresinde) kesilebilir. Her iki durumda da mutlaka komut algetir evresinin başına geri dönülmelidir. Sayfalı görüntü bellek yönetiminde adres dönüştürme sürecini özetleyen ve Çizim 5.23'te verilen akış çizgesinde, bu nedenle, işletimin, hem komuta, hem de işlenene erişimde kesildiği durumların sonrasında hep komut algetir evresinin başına dönüldüğü gösterilmiştir.

Sayfa Çıkarma Algoritmaları

Eksik sayfa uyarısı sonrasında, erişilmek istenen sayfanın taşınacağı boş bir sayfa bulunamaması durumunda bellekten çıkarılacak sayfanın seçimi gündeme gelir. Bu seçim değişik algoritmalara dayalı olarak yürütülür. Bu algoritmaların hepsi, eksik sayfa uyarısı sayısını en aza indirmeyi amaçlar. Ancak bu algoritmaların işletim süreleri de göz önüne alınması gereken diğer bir kıstası oluşturur. Bellekten sayfa çıkarma algoritmaları:

- İlk giren sayfayı çıkarma (*FIFO*)
- En erken erişilmiş sayfayı çıkarma (*Least Recently Used - LRU*)
- En geç erişilecek sayfayı çıkarma (*Optimal*)

algoritmaları olarak bilinir. İlk giren sayfayı çıkarma algoritması, bellekte uzun süre kalan bir sayfanın, gelecek komut işletimlerinde erişilme olasılığının düşük olacağı varsayımına dayanır. Bu algoritmanın uygulandığı sistemlerde, görevin ana belleğe yüklü tüm sayfaları ilk giren ilk çıkar (*FIFO*) türü bir kuyruk yapısında tutulur. Bu kuyruk yeni bir sayfaya erişmek gerektiğinde günlenir. Kuyruğun başında belleğe ilk yüklenen, sonunda da en son yüklenen görev sayfası yer alır. Bellekten bir sayfa çıkarmak gerektiğinde kuyruk başındaki sayfa seçilir. Bu algoritmada, *FIFO* kuyruğu, sadece erişilen sayfa değiştiğinde günlenir. Dolayısıyla işletim yükü düşük, hızlı bir algoritmadır. Ancak bellekte uzun süre kalmış sayfalar, bu algoritmanın varsaydığı gibi, her zaman, gelecek komut işletimlerinde kullanım olasılığı mutlaka en düşük olan sayfaları oluşturmazlar. Zira görevin en sık kullanılan sayfaları da belleğe en erken girmiş sayfalar olabilir. Bu durumda ilk giren sayfayı çıkarma algoritması, sık sık başvurulan sayfaları bellekten çıkararak ana bellek-disk arası sayfa gelgitlerini artıran bir algoritma olarak ortaya çıkar.

Bu algoritmanın bu önemli sakıncasını aşmak üzere erişim belirtecinden yararlanılabilir. Bu amaçla, tüm sayfaların erişim belirteçleri, bellek yönetici tarafından, belirli bir periyodla (örneğin 100 milisaniyede bir) sıfırlanır. Bellekten bir sayfa çıkarmak gerektiğinde, belleğe ilk giren sayfayı doğrudan çıkarmak yerine, önce bu sayfanın erişim belirteci sınanır. Erişim belirteci bir bulunan sayfa, son sıfırlamadan bu yana kullanılmış bir sayfa demektir. Dolayısıyla bu sayfa, görev tarafından halen kullanılan bir sayfa olarak yorumlanıp bellekten çıkarılmaz. Listede bir sonraki sayfanın belirteci sınanır. Sınamalar sırasında belirteci sıfır bulunan ilk sayfa bellekten çıkarılır. İlk giren sayfayı çıkarma algoritmasının değişmiş bu biçimi, genelde yakın geçmişte kullanılmamış sayfayı çıkarma *(NRU Not Recently Used)* algoritması olarak bilinir.

İlk giren sayfayı çıkarma algoritmasının çalışma ilkesi, Çizim 5.24'te, bir görevin ardarda işletilen 10 sayfası için örneklenmiştir. Erişilen sayfa dizgisi, görevin arka arkaya erişilen mantıksal sayfa numaralarını içermektedir. Sözkonusu göreve ana bellekte üç fiziksel sayfa ayrıldığı varsayılmıştır. 00 ve 01 numaralı sayfalara, bellekten çıkarılmalarından hemen sonra yeniden gereksinim duyulması, algoritmanın yukarıda açıklanan sakıncasını örneklemektedir.

Bellekte bir göreve ayrılan boş sayfa sayısı ne kadar büyük olursa bu görevin üreteceği eksik sayfa uyarısı sayısının da o denli küçük olacağını düşünmek çok doğaldır. Ancak ilk giren sayfayı çıkarma algoritmasıyla, kimi zaman, göreve ayrılan boş sayfa sayısının artırılmasına karşın eksik sayfa uyarısının da arttığı görülür. Ters orantılı olan boş sayfa ve eksik sayfa uyarısı sayılarının düz orantılı olarak gelişmesi normal değildir. Bu durum *Belady* Anormalliği olarak adlandırılır. Bunun, görev sayfalarının ilk giren ilk çıkar türü bir veri yapısı içinde tutulmasından kaynaklandığı bilinir.

Erişilen sayfa dizgisi

00	01	00	02	00	03	00	01	04	01

Bellekte göreve ayrılan boş 3 sayfayı işgal eden program sayfaları

Sayfa-i	00	00	00	00	00	03	03	03	04	04
Sayfa-j		01	01	01	01	01	00	00	00	00
Sayfa-k				02	02	02	02	01	01	01

Belleğe taşınan program sayfaları

00	01		02	03	00	01	04	

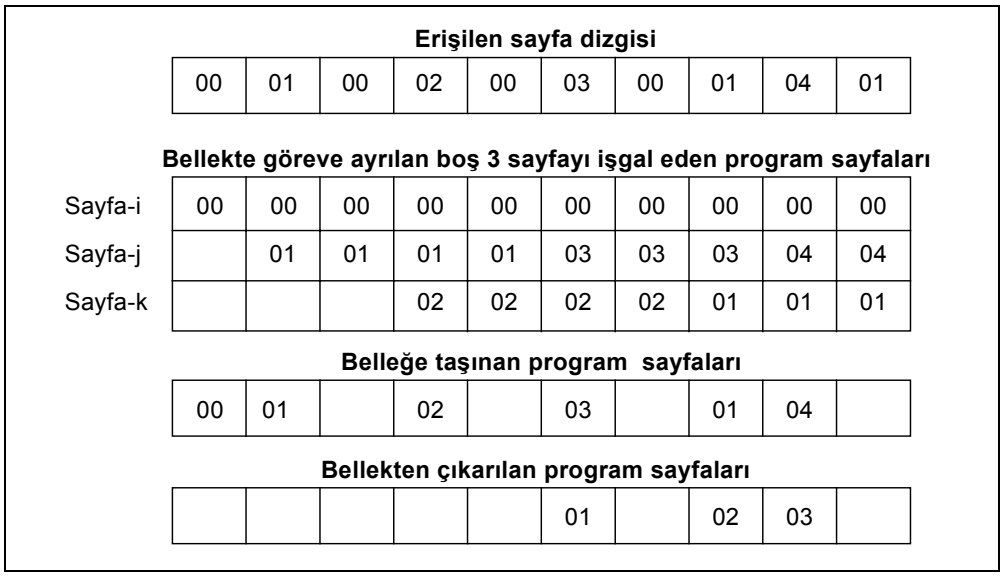
Bellekten çıkarılan program sayfaları

				00	01	02	03	

Çizim 5.24. İlk Giren Sayfayı Çıkarma Algoritmasına göre Sayfa Çıkarma

İlk giren sayfayı çıkarma algoritması, bellekten çıkarılacak sayfayı seçerken sayfaların belleğe yüklenme zamanını, en erken erişilmiş sayfayı çıkarma algoritması ise erişilme zamanını taban alır. Bu algoritma son zaman diliminde sıkça kullanılan sayfaların gelecek zaman diliminde de sıkça kullanılacağını ya da son zaman diliminde hiç kullanılmamış bir sayfanın gelecek zaman diliminde de kullanılma olasılığının düşük olacağını varsayar. Bu varsayıma göre bellekten bir sayfa çıkarılacağı zaman o ana kadar, en erken erişilmiş sayfanın seçimi yapılır. En erken erişilmiş sayfayı çıkarma algoritması, genelde, ilk giren sayfayı çıkarma algoritmasına göre daha iyi sonuç verir. Ancak en erken erişilmiş sayfayı belirlemek kolay bir işlem değildir. Bu belirlemeyi yapabilmek için, örneğin bir sayaçtan yararlanılabilir. Bu sayaç, her bellek erişiminde bir artırılır. İçeriği, sayfa tanım çizelgesinde sayfa başlangıç adresiyle birlikte saklanır. Bellekten sayfa çıkarmak gerektiğinde, sayaç değeri en küçük sayfa seçilir. Açıklanan bu sayışım ve günleme işlemlerini her bellek erişiminde yinelemek gereklidir. Bu,

sistem başarımını olumsuz yönde etkileyen bir durum ortaya çıkarır. Ancak en erken erişilmiş sayfayı belirlemenin kolay ve hızlı bir başka yolu da yoktur. Bu nedenle en erken erişilmiş sayfayı çıkarma algoritması, çoğu kez özel donanım desteği gerektirir. Çizim 5.25'te bu algoritmaya göre sayfa çıkarma örneklenmiştir. Çizim 5.24'teki aynı sayfa dizgisi taban alınmış, bu yolla örneklenen algoritmaların karşılaştırılabilmesi amaçlanmıştır.



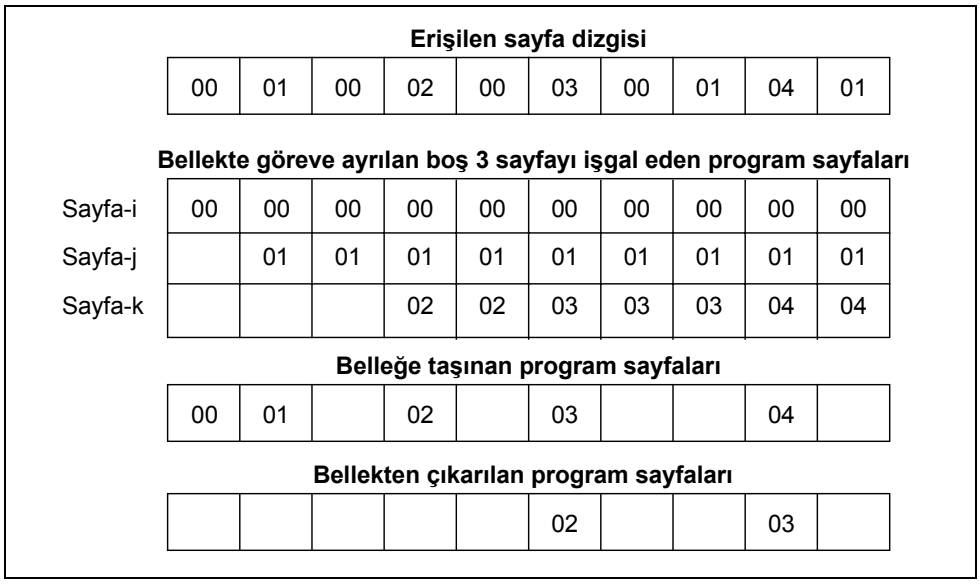
Çizim 5.25. En Erken Erişilmiş Sayfayı Çıkarma Algoritmasına göre Sayfa Çıkarma

Yukarıda, bellekten sayfa çıkarma algoritmaları arasında en son sırada sayılan en geç erişilecek sayfayı çıkarma algoritması, kuramsal olarak en iyi sonucu verecek algoritmadır. Zira bellekten bir sayfa çıkarmak gerektiğinde, bu algoritmaya göre, o an erişimi en geç gerçekleşecek sayfanın seçimi söz konusu edilecektir. Ancak herhangi bir anda hangi sayfanın en geç erişilecek sayfa olacağını bilmenin ya da kestirebilmenin kolay ve pratik bir yolu yoktur. Bu nedenle bu algoritma, daha çok kuramsal değerde, diğer algoritmalarla elde edilen sonuçların değerlendirilmesine yarayan ölçüt bir algoritma olarak algılanmalıdır. Bu kuramsal algoritmanın çalışması Çizim 5.26'da, daha önce iki kez kullanılan sayfa dizgisi için örneklenmiştir. Verilen son üç çizimden, ilk giren sayfayı çıkarma algoritmasının 7, en erken erişilen sayfayı çıkarma algoritmasının 6 ve bu kuramsal algoritmanın da 5 eksik sayfa uyarısına neden olduğu görülmektedir. Bu sonuç, incelenen üç algoritmanın başarımlarını da özetlemektedir.

Görevlere Sayfa Atama Politikaları

Sayfalı görüntü bellek yönetiminin uygulandığı sistemlerde, görevlere ayrılan bellek sayfası sayısı sistem başarımını etkileyen önemli bir parametre olarak ortaya çıkar. Bellekten sayfa çıkarma algoritmaları incelenirken belirtildiği gibi bir görevin, aynı anda sahip olabileceği toplam fiziksel sayfa sayısının, genelde bir üst sınırı bulunur. Çok iş düzeninin uygulandığı sistemlerde bellek, aynı anda birden çok görev tarafından paylaşılmak zorunda olduğundan herbir göreve, ana bellekteki sınırlı sayıdaki fiziksel

sayfadan ancak bir kesiminin ayrılabilmesi doğaldır. Bu bağlamda, ana bellekteki sayfa sayısının değişmez olduğu düşünüldüğünde, herbir göreve ayrılabilen toplam sayfa sayısı, sistemde aynı anda işletime alınan görev sayısına bağlıdır. Daha açık bir anlatımla, görevlere ayrılan sayfa sayısı, aynı anda işletime alınan görev sayısıyla ters orantılıdır. Buradan, görevlere atanabilecek sayfa sayısının bir üst sınırının bulunacağı anlaşılır. Bunun yanı sıra, görevlere ayrılan sayfa sayısının çok düşük tutulması durumunda, eksik sayfa uyarılarının artışına koşut olarak, ana bellek - disk arası sayfa gelgitlerinin ve işletim yükünün artmasıyla karşı karşıya kalınır. Buradan, görevlere ayrılan sayfa sayısının, bir de alt sınırı olacağı görülür. Bu nedenle görevlere aynı anda atanabilecek toplam sayfa sayısının, işletim sistemince dikkatlice belirlenmesi önem taşır. Bu belirleme, genelde eksik sayfa uyarısı sayısı taban alınarak yapılır.

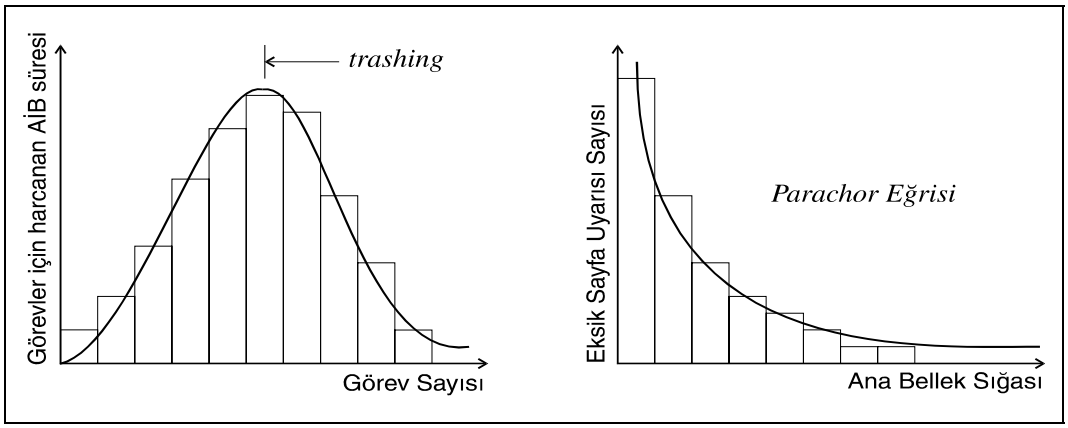


Çizim 5.26. En Geç Erişilecek Sayfa Algoritmasına göre Bellekten Sayfa Çıkarma

Sistemde kaynak kullanım düzeyini artırmak amacıyla aynı anda işletilen görev sayısını artırmak gerekir. Ancak aynı anda işletime alınan görev sayısı artıkça görevlerin herbirine ayrılabilen sayfa sayısı düşer. Bu düşüş belirli bir düzeyin altına indiğinde, birlikte çalışan görev sayısının (çok iş düzeyinin) artmasına karşın kaynak kullanım düzeyi, özellikle de ana işlem birimi kullanım düzeyi düşmeye başlar. Zira görevlere yeterli sayıda ana bellek sayfası ayıramama sonucunda bellekte bulunmayan sayfalara erişim istemleri, dolayısıyla eksik sayfa uyarıları sayıca artar. Bu durum bellek yöneticinin, sayfa gelgitlerini düzenlemek üzere, çok sık bir biçimde ana işlem birimine anahtarlanmasını gerektirir. Böylece görevlerin ana işlem birimi kullanım payı ve sistem başarımı aşırı düşer; işletim yükü artar.

İşletim sisteminin eksik sayfa uyarılarının gerektirdiği işlemlerle yoğun bir biçimde uğraşmaya başlamasına, İngilizce *trashing* olayı denir. Bu olay, ana işlem biriminin bir göreve anahtarlanmasından sonra, bu görevin komutlarını işletmek yerine, eksik sayfa uyarılarıyla devreye giren çıkarılacak sayfa seçimi, sayfa aktarımı gibi, görev yönünden

kayıp sayılan işlemlerle uğraşması olarak da tanımlanır. Ana belleğin fiziksel sığasının değişmez olduğu varsayıldığında bu olay, çok iş düzeyinin bir işlevi olarak görülür. Bunun gibi birlikte çalışan görev sayısının değişmez olduğu varsayıldığında ise ana belleğin fiziksel boyunun bir işlevi olarak ortaya çıkar. Belirli bir sistem için eksik sayfa uyarısı sayısının ana belleğin fiziksel sığasına göre değişimi, *Parachor* Eğrisi olarak adlandırılan bir eğri ile gösterilir (Çizim 5.27).



Çizim 5.27. Eksik Sayfa Uyarısı Sayısının Bellek Sığası ve Görev Sayısı ile İlişkisi

Çok iş düzeyini artırmak üzere, görevlere ayrılan sayfa sayısının düşük tutulması *trashing*'e neden olarak sistem başarımını, arananın tersine düşürür. Görevlere atanacak toplam sayfa sayısını, *trashing* yaratmayacak biçimde belirleyebilmek için *Working Set- WS* olarak adlandırılan bir kavram geliştirilmiştir. Görevlerin, belirli bir zaman aralığında ana bellek erişimleri incelendiğinde bu erişimlerin belirli adres bölgeleri içinde sınırlı kaldığı görülmektedir. Görev işletiminin, belirli bir adres bölgesinde ne kadar süreyle kaldığı, görevin yerellik düzeyi (*locality*) ile açıklanmaktadır. Yüksek yerellik gösteren bir görev, uzunca bir süre, dar bir adres bölgesi içindeki ana bellek sözcüklerine erişim yapan görev demektir.

WS, bir görevin, belirli bir zaman aralığı içinde, eksik sayfa uyarısı üretmeden çalışabileceği en küçük sayfa takımıdır. *WS*, görevin yerellik düzeyiyle uyumlu sayıda sayfadan oluşan takım olarak da tanımlanabilir. Sözkonusu zaman aralığı görev işletim süresinin tümü olarak düşünüldüğünde *WS*'in de görev boyuna yaklaşacağı görülür. Bir sistemde *trashing* olayının ortaya çıkmasını engellemek, birlikte çalışan görevlere *WS*'leriyle uyumlu sayıda sayfa ayırarak gerçekleşir. *WS,* her görev için, işletim sistemi tarafından kestirilmesi gereken bir parametredir. Ancak ortalama değerler de kullanılabilir. *WS* bir kez belirlendikten sonra, ilgili görevin, en az, *WS* ile tanımlanan sayıda sayfasının ana bellekte bulunmasına özen gösterilir.

Trashing olayını, *WS* kavramına dayalı olarak, dolaylı bir biçimde engellemeye çalışmak yerine, her görev için eksik sayfa uyarısı sıklığını doğrudan ölçerek önlem almak da olanaklıdır. Bu yöntemle, her görevin, son eksik sayfa uyarısı zamanı görev iskeletinde saklanır. Bir görev için, iki eksik sayfa uyarısı arasında geçen süre, bu yolla ölçülerek uyarı sıklığı hesaplanır. Bu sıklık, sistemce benimsenen bir sınır değerinin

altında kaldığı sürece ilgili sayfa gereksinimi, görevin sahip olduğu sayfalardan biri boşaltılarak karşılanır. Göreve ayrılan bellek sayfa sayısında bir değişiklik yapılmaz. Eğer ölçülen sıklık, sınır değerinin üstüne çıkmışsa göreve, kendi sayfalarının dışından, ek bir fiziksel sayfa sağlanmaya çalışılır. Bu yolla, bir görevin sahip olduğu toplam sayfa sayısı, işletim sırasında, çok iş düzeyinin elverdiği ölçüde artabilir.

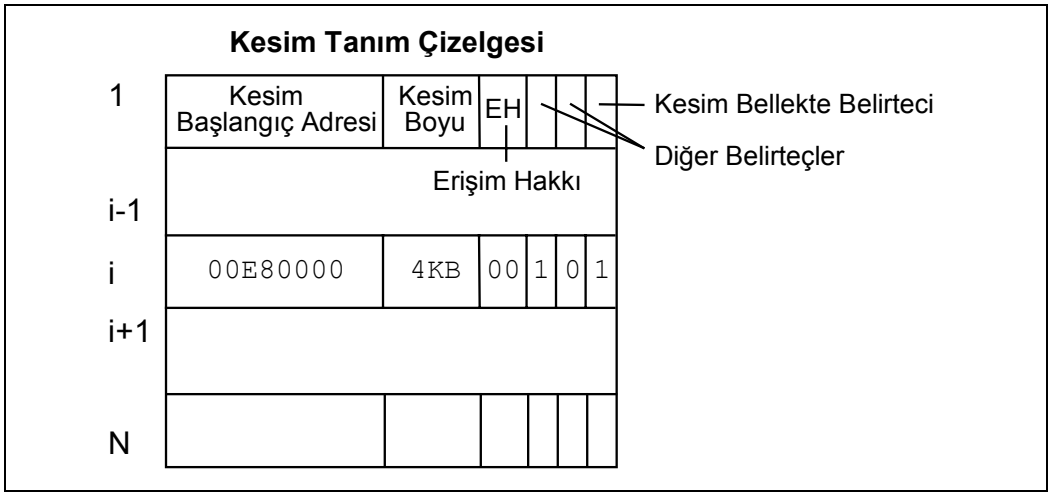
Sayfalı bellek yönetiminin uygulandığı sistemlerde sayfalar değişmez boydadır. Sayfa boyları da *trashing* olayını etkileyen bir unsurdur. Sayfa boyunun seçimi, bir yandan bellek yönetimiyle ilgili olarak tutulan çizelgelerin boyutları, diğer yandan diske erişim süresi gözetilerek belirlenir. Sayfa boyları ne kadar büyükse bellek ve sayfa tanım çizelgelerinin boyları da o kadar küçük olur. Bunun gibi, sayfa boyu ne kadar küçükse disk ana bellek arası aktarım süreleri o kadar kısa olur. Bunlara ek olarak sayfa boyunun büyüklüğü sayfa içi parçalanmayı artırır. Sayfa boylarının küçüklüğü ise eksik sayfa uyarısı sayısını artırıcı bir rol oynar. Bu parametreler göz önüne alınarak, günümüz bilgisayarlarında 2 - 4 KB'lık sayfalar kullanılır.

5.8. Kesimli Görüntü Bellek Yönetimi

Sayfalı bellek yönetiminde sayfalar, programların mantıksal adres evrenlerini mantıksal bir bütünlük gözetmeksizin rasgele parçalarlar. Oysa programları oluşturan yordamlar, blok yapıları mantıksal bütünlük içeren birimlerdir. Bu birimlerin eşit uzunlukta sayfalar biçiminde bölünmeleri görüntü bellek düzeninde, eksik sayfa uyarısı sayısını artıran bir etken olarak ortaya çıkar. Zira sayfalı bölümleme bir yordamı, bir döngüyü oluşturan komutları hatta bir komutu oluşturan sözcükleri ayrı sayfalara ayırabilir. Bu sayfaların herbirinin, aynı anda ana bellekte bulunamaması aynı yordam, aynı döngü ve hatta aynı komut için bir dizi eksik sayfa uyarısının üretilmesine neden olur. Bir komutu oluşturan sözcüklerin aynı sayfada yer alması, derleyiciler tarafından sağlanabilir. Ancak döngü ve yordam gibi mantıksal bütünlük içeren öbeklerin hep aynı sayfada yer almasını sağlamak olanaklı değildir. Zira bunu yapabilmek için program adres evrenlerini bölümlerken bu mantıksal bütünlüğü gösteren kıstaslara dayanmak gerekir. Sayfalamada program adres evrenleri, mekanik olarak, eşit uzunluk dışında herhangi başka bir kıstas gözetilmeksizin bölümlenir. Mantıksal bütünlüğü taban alan bölümleme kesimleme ile gerçekleşir. Görüntü bellek düzeni çerçevesinde, kesimli bellek yönetimi, programları yordam, yığıt gibi bütünselliği olan kesimler biçiminde ele alan, böylece yüksek oranda erişim yerelliği sağlayan bir yöntem olarak ortaya çıkar.

Bilindiği üzere kesimli bellek yönetiminde mantıksal adreslerden fiziksel adreslere geçiş, çoğu kez, her kesim türü için en az bir tane olmak üzere birden çok kesim yazmacı taban alınarak gerçekleştirilir. Mantıksal adres içindeki kesim numarası ya da kesim türü bilgisine göre seçilen kesim taban yazmaç içeriği kesim içi göreli adrese eklenerek fiziksel adres elde edilir. Kesim yazmaçlarının içerikleri, kesim tanım çizelgesinde saklanan kesim başlangıç adresleriyle günlenir. Kesimli bellek yönetimine görüntü bellek özelliği katabilmek için kesim tanım çizelgelerinde, kesim başlangıç, boy, erişim hakkı gibi bilgilerin yanı sıra kesimin belleğe yüklü olup olmadığını gösteren belirtece de yer verilmesi gerekir. Kesim taban yazmacı, kesim tanım çizelgesindeki değerle günlenirken bu bitin sıfır olması bir iç kesilmeye kaynaklık eder.

Bu kesilme, komut işletiminin ortasında, denetimin donanımdan bellek yöneticiye geçmesini sağlar. Bu yolla bellekte olmayan kesimin belleğe taşınma işlemleri başlatılır. İlgili kesim belleğe taşındıktan sonra işletim, yarıda kesilen komuttan başlamak üzere, kalınan yerden sürdürülür. Sayfalı görüntü bellek yönetiminde olduğu üzere, kesim tanım çizelgelerinde kesim bellekte belirtecinin yanı sıra görüntü bellek yönetiminin gerektirdiği erişim ve değişim belirteçlerine de yer verilir. Bellekten bir kesim çıkarılacağı zaman bu belirteçlerden yararlanılır (Çizim 5.28).

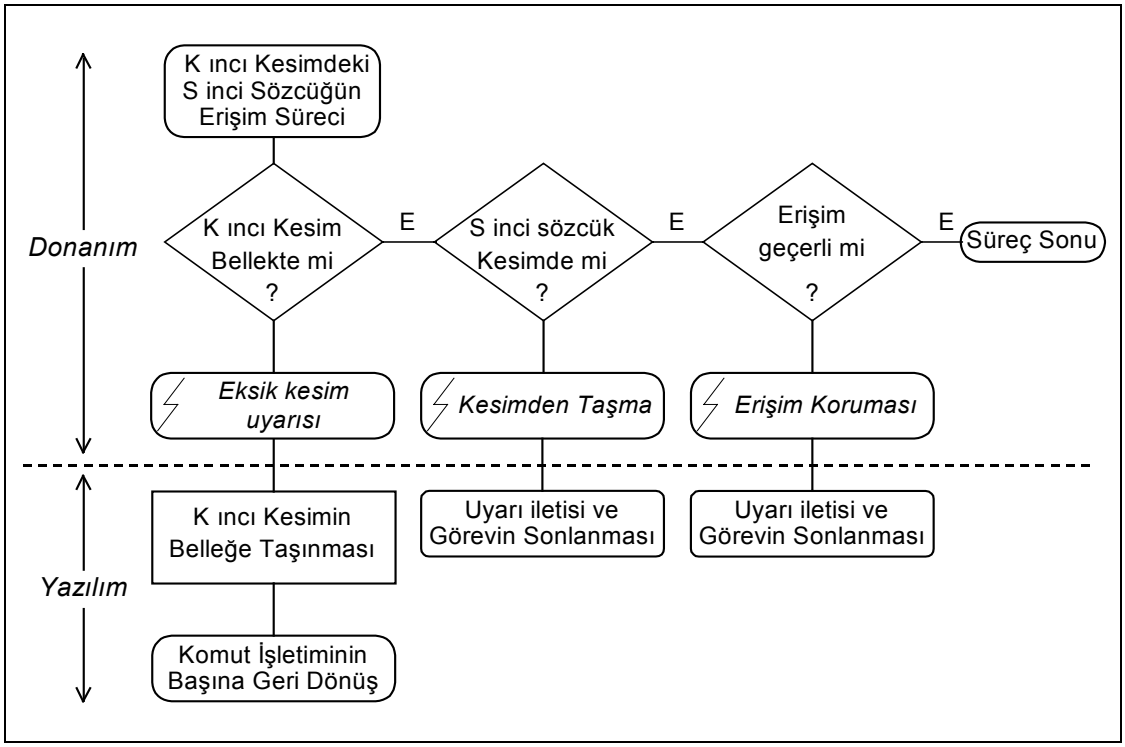


Çizim 5.28. Kesimli Görüntü Bellek Yönetimi ve Kesim Bellekte Belirteci

Kesimli görüntü bellek yönetimi sayfalı görüntü bellek yönetiminin sayfalarla yaptığını kesimlerle gerçekleştirir. Eksik kesim uyarısının yanı sıra, bu yöntemde, daha önce de belirtildiği gibi, kesimden taşmalarla, kesim paylaşımının gerektirdiği erişim haklarının da denetlenmesi gereklidir. Kesimli görüntü bellek yönetimine olanak veren bir işleyicinin belleğe erişim süreci Çizim 5.29'da özetlenmiştir. Bu süreç erişilen sözcüğün komut mu yoksa işlenen mi olduğuna bakılmaksızın K ıncı kesimdeki S inci sözcüğe erişim çerçevesinde, genel olarak verilmiştir. Ancak eksik kesim uyarısı alınması durumunda, erişilen sözcük, ister komut kodu ister işlenen olsun, eksik kesimin ana belleğe taşınmasından sonra hep komut algetir evresinin başına dönüleceği unutulmamalıdır.

Kesimli görüntü bellek yönetimi programların adres evrenlerini yüksek yerellik içeren kesimlere ayırarak sayfalı bellek yönetiminin yarattığı *trashing* adlı olayı ortadan kaldırır. Örneğin her kesim türü için bir kesim taban yazmacının tutulduğu sistemlerde kod, veri ve yığıt kesimleri ana bellekte bulunan bir görev çok seyrek olarak eksik kesim uyarısına neden olur. Sayfalı görüntü bellek yönetimi kapsamında sözü edilen ve belirlenmesi oldukça zor olan *WS (Working Set),* kesimli görüntü bellek yönetimi için kolayca belirlenir. Aynı anda yararlanılan bir takım kod, veri ve yığıt kesimi, ilgili görevin *WS*'ini oluşturur. Bunun yanı sıra kesimli görüntü bellek yönetimi, işletim aşamasında devingen biçimde büyüyebilen kesimlere ve yordam bağlama işlemlerinin işletim sırasında devingen olarak gerçekleşmesine olanak verir. Bu yolla işletime kimi

özel koşullarda katılacak yordamların gereksiz yere, işletimin başında ana belleğe yüklenme zorunluluğunu ortadan kaldırır ve yer tasarrufu sağlar.



Çizim 5.29. Kesimli Görüntü Bellek Yönetiminde Bellek Erişim Süreci

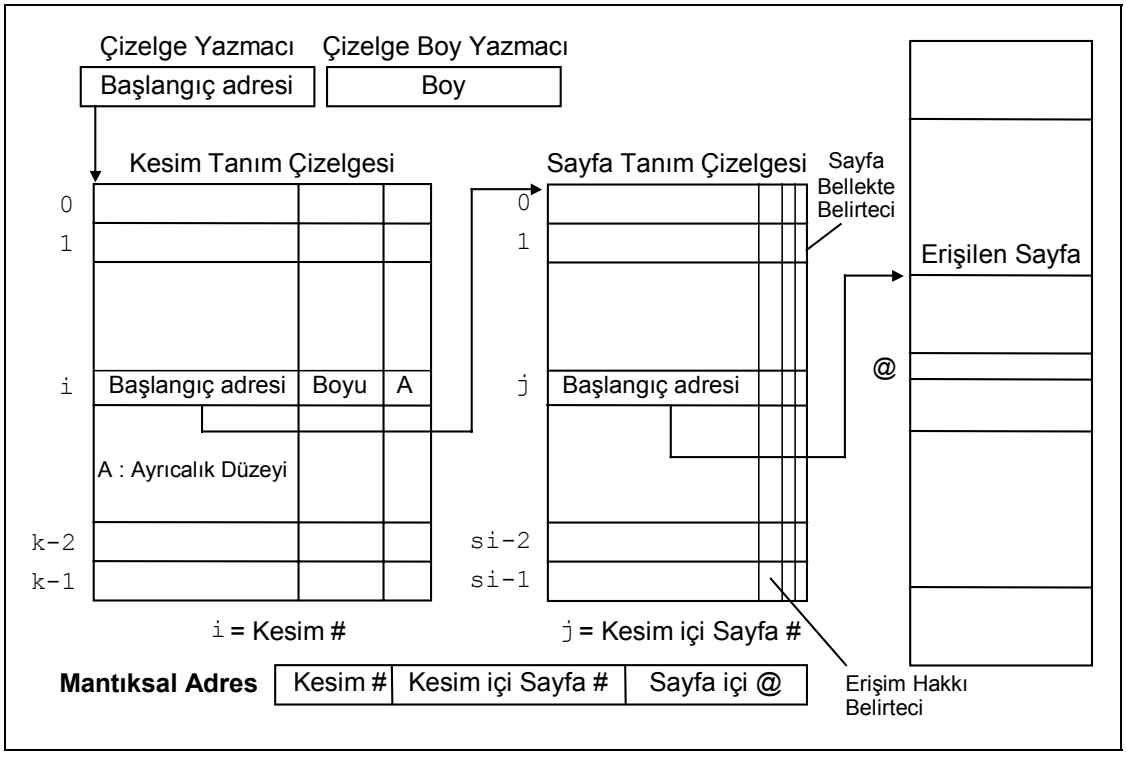
Bu olumlu yönlerine karşın kesimli görüntü bellek yönetimi değişken boydaki kesimlere, ana bellekte bitişken yer sağlama zorunluğu nedeniyle parçalanma sorununu yeniden gündeme getirir ve karmaşık bellek atama yöntemlerinin kullanımını gerektirir. Değişken boydaki kesimlerin diskte saklanması da işletim yükünü artıran bir etmen olarak ortaya çıkar. Kesimli görüntü bellek yönetimi ile sayfalı görüntü bellek yönetimi karşılaştırıldığında her iki yönetimin de olumlu ve sakıncalı yönleri bulunduğu görülür. Bunların olumlu yönlerini korumak ve sakıncalarını aşmak, her ikisini birlikte kullanmakla mümkündür. İzleyen kesimde açıklanan kesimli - sayfalı görüntü bellek yönetiminde bu gerçekleştirilir.

5.9. Kesimli - Sayfalı Görüntü Bellek Yönetimi

Kesimli ve sayfalı görüntü bellek yönetimlerinin iyi yönlerini birleştirmek için en uygun yol programların mantıksal adres evrenlerini kesimlere, kesimleri de sayfalara ayırmaktır. Bu yolla, kesimli görüntü bellek yönetiminin temel sakıncasını oluşturan değişken boydaki kesimleri sayfa tabanında ele almak mümkün olur. Kesimli-sayfalı görüntü bellek yönetiminde mantıksal adresler üç birleşenden oluşur. Bu birleşenler:

- Kesim numarası
- Kesime göreli sayfa numarası
- Sayfa içi adrestir.

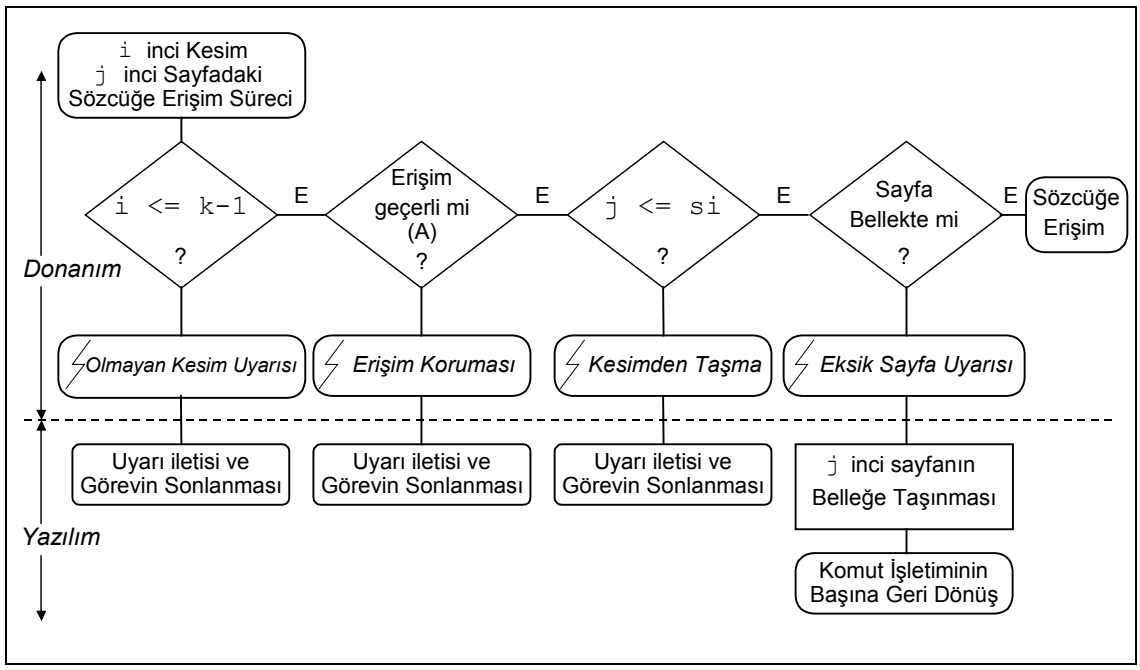
Mantıksal adreslerden fiziksel adreslere geçişte iki temel çizelge kullanılır. Bunlardan ilki kesim tanım çizelgesi, ikincisi ise sayfa tanım çizelgesidir. Kesimli - sayfalı görüntü bellek yönetiminde kesim tanım çizelgesi, programları oluşturan kesimlerin başlangıç adresleri yerine, bu kesimlerin herbiri için ayrı ayrı öngörülen sayfa tanım çizelgelerinin başlangıç adreslerini tutar. Kesim boyu bilgisi yerine, ilgili sayfa tanım çizelgesinin boy bilgisine yer verilir. Kesime erişim denetim kodu, genelde çizelgedeki yerini korur. Kesime erişim, genelde kesim tabanında denetlenir. Ancak kesim bellekte belirtecine çoğu kez çizelgede yer verilmez. Bunun yerine, sayfa tanım çizelgesindeki sayfa bellekte belirteci kullanılır. Zira kesimin ana belleğe yüklenmesi sayfa sayfa gerçekleşir. Bu durumda kesimin tümünün ana bellete bulunup bulunmadığı hususu anlamını yitirir. Sayfa tanım çizelgeleri, ilgili oldukları kesime ilişkin sayfaların bellek başlangıç adresleri ile sayfa bellekte, erişim, değişim gibi belirteçlerini tutar.



Çizim 5.30. Kesimli-Sayfalı Bellek Yönetiminde kullanılan Tanım Çizelgeleri

Çizim 5.30'da kesim tanım ve sayfa tanım çizelgeleri ile bunların içerdikleri bilgiler örneklenmiştir. Bu çizime göre işletilmekte olan görevin `k` kesimi bulunduğu; `i` inci kesimin `si` sayfa içerdiği varsayılmıştır. Kesimli - sayfalı görüntü bellek yönetiminde bir görev ana işlem birimine anahtarlandığında, ana işlem birimi kesim çizelge yazmacı ile çizelge boyu yazmacı, bu görevin kesim tanım çizelgesi başlangıç adresi ve boyu bilgileri ile günlenir. Adres dönüştürme kapsamında, mantıksal adres içindeki kesim numarası, ana işlem birimi kesim çizelge yazmacına eklenerek kesim tanım çizelgesinden, o kesimle ilgili sayfa tanım çizelgesi başlangıç adresi okunur. Mantıksal adres içindeki kesim numarası, çizelge boyu yazmacında saklanan en büyük kesim numarasından daha büyük ise olmayan kesim uyarısı ile görev işletimi sonlandırılır.

Kesim tanım çizelgesinde tutulan ayrıcalık bilgisi, kesime erişimin denetlenmesini sağlar. İşletilen görevin iskeletinde tutulan ayrıcalık düzeyi, çizelgede saklanan ayrıcalık bilgisiyle uyumsuzsa, ya da başka bir deyişle bu kesime erişim yapmak için yeterli değilse görev işletimi, sayfa tanım çizelge başlangıç adresinin ana işlem birimi sayfa tanım çizelge yazmacına taşınması sırasında, erişim koruma iç uyarısı aracılığıyla sonlandırılır. Mantıksal adres içindeki sayfa numarası, içeriği bir önceki adımda günlenen sayfa tanım çizelge yazmacına eklenerek erişilen sayfanın başlangıç adresi elde edilir. Mantıksal adres içindeki sayfa numarası, kesim tanım çizelgesinde tutulan kesim boyundan daha büyük olamaz. Tersi durumlarda olmayan sayfa uyarısı ile görev işletimi sonlandırılır.



Çizim 5.31. Kesimli-Sayfalı Bellek Yönetimi ve Adres Dönüştürme Süreci

Sayfa tanım çizelgesinde, daha önceden de görülen çeşitli belirteçler yer alır. Bunlar erişim hakkı, erişim, değişim, sayfa bellekte belirteçleridir. Sayfaya erişimler salt okuma, okuma-yazma, uygulama gibi değişik haklarla yapılmak üzere, erişim hakkı belirteçlerine dayalı olarak denetlenir. Erişim ve değişim belirteçleri, ilgili sayfaya, sırasıyla ilk erişimde ve yazmada günlenir. Sayfa bellekte belirteci erişilen sayfanın bellekte bulunup bulunmadığını gösterir. Bu belirtecin sıfır olması durumunda, adres dönüştürme süreci eksik sayfa uyarısı ile kesilir ve sayfanın belleğe taşınma süreci başlatılır. Bu süreç sonunda komut işletiminin başına dönülür (Çizim 5.31).

Kesimli - Sayfalı bellek yönetiminde, mantıksal adreslerden fiziksel adreslere geçiş kesim ve sayfa olmak üzere iki değişik tanım çizelgesinin kullanımını gerekli kılar. Bu çizelgeler, bilindiği üzere ana bellekte saklanır. Bu nedenle, adres dönüştürme işlemleri üç adımda gerçekleşir. İlk aşamada kesim tanım çizelgesine erişilip sayfa tanım çizelgesi başlangıç adresi elde edilir. İkinci bir adımda, sayfa tanım çizelgesinden sayfa

başlangıç adresi okunur. İlgili bellek sözcüğüne erişim, ancak üçüncü adımda gerçekleşir. Kesimli - Sayfalı bellek yönetimi bellek erişim süresini üçe katlar. Bu durumda, sayfalı bellek yönetimi kapsamında konu edilen ön bellek olmadan bu yönetimin kullanılması anlamlı olmaz. Ön bellek kullanımına karşın sayfalamanın yanı sıra bir de kesimlemenin kullanılması bellek erişim süresini, yine de artıran bir sonuç doğurur. Bu sakıncası bir yana bırakılırsa kesimli - sayfalı bellek yönetimi, şimdiye değin incelenen en yetkin ve dolayısıyla en pahalı bellek yönetim biçimidir.

Görüntü bellek yönetimlerinin tümüne ilişkin önemli bir sorun, gerçek adreslerle işlem yapılması gerektiğinde ortaya çıkar. Bir bilgisayar sisteminde ana belleğe erişim yapan tek birim ana işlem birimi değildir. Doğrudan bellek erişim denetleme birimleri, giriş/çıkış işleyicileri, kanallar gibi donanım birimleri de ana belleğe erişim yapabilen birimlerdir. Bilindiği üzere görüntü bellek düzeni, adres dönüştürme, komut bölünürlüğünün sağlanması gibi, ana işlem birimine yapılan özgün ekler sayesinde gerçekleşir. Bu ekler ana işlem biriminin pahasını artırır. Aynı ekleri, ana belleğe erişim yapan diğer birimlere de yapmak ekonomik nedenlerle anlamlı değildir.

Bu nedenle, ana işlem birimi dışındaki birimler adres dönüştürme işlemlerini yerine getiremezler. Bu nedenle, giriş/çıkış birimleri - ana bellek arası veri aktarımlarında, aktarılan verilerin ana bellekte saklandıkları konumlar fiziksel adresleriyle tanımlanmak zorundadırlar. Bu zorunluluk görüntü adreslerle fiziksel adreslerin içiçe kullanılması gibi sakıncalı bir durumu ortaya çıkarır. Örneğin bir görev diskten bir tutanak okumak istediğinde, kanala / giriş/çıkış işleyicisine / doğrudan bellek erişim denetleme birimine bu görevin adres evreni içinden bir alanı, aktarım alanı olarak bildirmek gerekir. Görevin o an elinde tuttuğu bir bellek sayfası içinden bir kesim, aktarım alanı olarak tanımlanabilir. Ancak görüntü bellek düzeni çerçevesinde, görevlere atanan fiziksel sayfalar işletim sırasında değişebilir. Giriş/çıkış istemi başında görevin fiziksel adres evreni içinde yer alan bir sayfa, giriş/çıkış işlemlerinin sonunda, artık o görevin sayfaları arasında yer almayabilir. Bu durum okuma işlemi başlatılan disk tutanağının başka bir görevin adres evreni içinde yer alan sayfaya yazılması ve bu sayfanın bozulması sonucunu doğurur.

Görüntü bellek düzenine olanak veren sistemlerde, bu sakıncayı aşmak üzere değişik yollar kullanılır. Bu yollardan biri sayfa belirteçleri arasına yeni bir belirteç ekleyerek sayfanın bellekten çıkarılıp çıkarılamayacağını kodlamaktır. Bellekten bir sayfa çıkarmak gerektiğinde bellek yönetici bu belirteci kurulu olmayan sayfalar arasından seçim yapar. Giriş/çıkış aktarım alanı olarak belirlenen sayfalar giriş/çıkış işlemi sonlanana değin belleğe bağlı sayfa olarak tanımlanırlar. Böylece bellekten çıkarılmaları engellenerek yukarıda açıklanan sakınca aşılmış olur.

Bellekte kimi sayfaların yerlerinden oynatılamamaları bellek yönetimini zorlaştırır. Giriş/çıkış işlemi başlatarak işletimi kesilen ve bir bekleme kuyruğuna bağlanan bir görevin, göreli yavaş giriş/çıkış işlemi sonlanana değin kimi sayfalarının, başka görevlerin kullanımına kapalı olarak bekletilmesi kaynak kullanım verimliliği yönünden her zaman benimsenemez. Bu nedenlerle giriş/çıkış işlemleri, doğrudan görev bellek alanını hedef almak yerine, işletim sistemi kapsamında bu amaçla, tüm görevlere ortak,

belleğe bağlı bir yastık alanına yönlendirilir. Bir görevin, örneğin okunmasını istediği tutanağın içeriği, önce sürücüden sistem yastık alanına aktarılır. İlgili görev, giriş/çıkış işleminin sonlanmasıyla ana işlem birimine yeniden anahtarlandığında bu yastık alan içeriği görev adres evrenine aktarılır.

Görüntü bellek düzeninin kurulabildiği sistemlerde, programlar içinde geçen adresler, bağlama - yükleme sonrasında da program başına göreli değerlerini korurlar. Mantıksal olarak nitelenegelen bu değerlerin fiziksel adreslere dönüşmesi, tanım çizelgelerinin yardımıyla, komut işletimi sırasında gerçekleşir. Ancak kimi durumlarda mantıksal adreslerin fiziksel karşılıklarının, erişimden bağımsız olarak da hesaplanabilmesi gerekir. Bu amaçla, bu tür sistemlerde `load effective address (lea reg,değişken)` gibi adlarla anılan komutlar bulunur. Bu komutlar ilgili tanım çizelgelerini kullanarak, adı verilen değişkenin mantıksal adresinden fiziksel adresini hesaplamaya ve kimliği verilen yazmaç içinde oluşturmaya yararlar. Örneğin bir program kapsamında, diskten aktarım yapılması istenen alanın başlangıç adresinin fiziksel karşılığı, işletim sistemi tarafından, fiziksel adreslerle çalışan kanala aktarılmak üzere, bu tür komutlar kullanılarak hesaplanır.

Programların adres evrenlerini önce kesimlere, sonra da sayfalara ayırma yönteminin genelleştirilmiş biçimi çok düzeyli adres dönüştürme yöntemi olarak bilinir. Bu bağlamda adres evrenleri, aşama aşama, küçülen boyutlarda kesimlere ayrılır. Örneğin 2 MB'lık bir adres evreni, ilk düzeyde 256 KB'lık 8 kesime ayrılır. 256 KB'lık kesimlerin herbiri de, kendi içlerinde 64 KB'lık ikinci düzey kesimlere ayrılırlar. 64 KB'lık kesimler, üçüncü düzeyde 2 KB'lık 16 sayfaya ayrılırlar. Çok düzeyli adres dönüştürme yönteminde, her düzey için bir tanım çizelgesi tutulur. Bu çizelgeler çok düzeyli adres dönüştürme çizelgeleri olarak anılırlar. (i) düzeyli bir çizelge, (i+1) düzeyli alt çizelgelerin ana bellek başlangıç adreslerini tutar. İlk düzey adres dönüştürme çizelgesi başlangıç adresi ilgili görev iskeletinde saklanır. Görev, ana işlem birimine anahtarlandığında, ana işlem birimi çizelge yazmacı bu başlangıç değeri ile günlenir. Bu değerden başlayarak, aşama aşama diğer çizelgelere ve erişilmek istenen sayfaya ulaşılır. Doğal olarak, bu yöntemle, ana bellek erişim süresi, düzey sayısıyla orantılı biçimde, katlanarak artar. Bu nedenle, çok düzeyli adres dönüştürme yöntemi ön bellek kullanımını zorunlu kılar. Çok düzeyli adres dönüştürme yöntemini kullanan bellek yönetimi, kesimli-sayfalı bellek yönetiminin genelleşmiş bir biçimi olmakla birlikte, burada yapılan kesimlemenin mantıksal bir bütünlük gözetmediği açıktır.

6. BÖLÜM

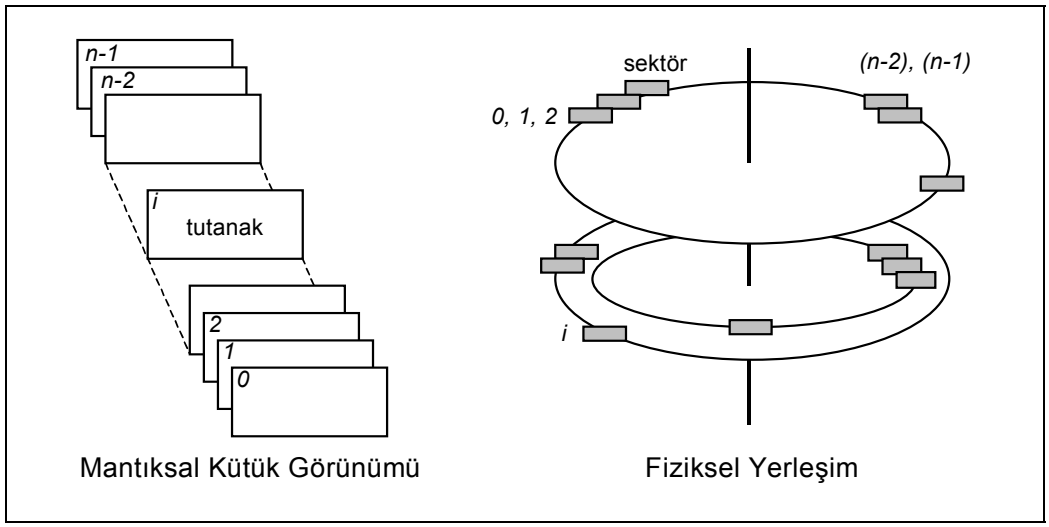
İŞLETİM SİSTEMLERİ

KÜTÜK YÖNETİMİ

Bilindiği gibi ana bellek dışında saklanan veri kümeleri, kütük olarak adlandırılır. Bu nedenle ikincil bellekler, daha geniş anlamıyla giriş/çıkış birimleri üzerinde tutulan verilerin yönetimi Kütük Yönetimi kapsamında ele alınır. Kütük yönetimi, işletim sistemlerinin önemli bir birleşenini oluşturur. Bu birleşenin en önemli işlevi kullanıcılara kütüklerini yalın bir mantıksal yapı çerçevesinde düşünme ve kullanma olanağı sunmaktır. Kullanıcılar, kaynak ve amaç programlar, veriler, yazılı metinler gibi veri kümelerini tutan kütüklerini, doğrusal bir tutanak ya da bayt dizisi olarak düşünürler. Ancak bunlar, ikincil bellekler üzerinde, kaynak kullanım verimliliği yönünden, fiziksel olarak, değişik silindir, sektör hatta sürücü ve sistemler üzerine serpiştirilmiş biçimde yer alabilirler. Bu durum, kullanıcıların programlarını, mantıksal olarak ardarda gelen sayfalardan oluşuyormuş gibi düşünmelerine ancak bu sayfaların, ana bellekte bitişken olmayan boş bellek sayfalara serpiştirilmesine benzer. Bunun gibi kullanıcılar kütüklerini oluşturan tutanakların konumlarını kütük başına göreli olarak düşünürlerken bunların yer aldığı disk öbeklerinin, kütüğün bulunduğu sürücü başına göreli, silindir-kafa-sektör üçlüsünden oluşan mutlak bir adresi bulunur. Kullanıcılara, kütükleri üzerinde düşüncelerinde gerçekliği olan mantıksal yapılar çerçevesinde işlem yapabilme olanağı, mantıksal yapılarla fiziksel yapılar arasında geçişi sağlayan kütük yönetim sistemince verilir (Çizim 6.1).

Kullanıcılara kütükleri üzerinde, yalın mantıksal modellere göre işlem yapabilme olanağının verilmesi bu işlemlerin de yalın mantıksal (kavramsal) tanımlara dayalı olmasını gerektirir. Mantıksal nitelikli işlemler, sistemin karmaşık fiziksel ayrıntılarını

kullanıcılardan gizleme olanağını da verir. Giriş/Çıkış Sistemi kesiminden anımsanacağı üzere, kütüklerin sürücülere fiziksel olarak yazılıp okunması; arabirimler düzeyindeki kimi giriş/çıkış kapı adresleri, silindir-kafa-sektör üçlüsü, kesilme vektörü gibi, sistemden sisteme değişen fiziksel ayrıntıları kullanmayı gerektirir. Kütüklerle ilgili işlemlerin bu ayrıntılardan soyutlanması, işletim sistemlerinin iki temel amacından biri olan kolay kullanım ilkesinin de bir gereğidir.



Çizim 6.1. Kavramsal Kütük Modeli ve Fiziksel Yerleşimi

Bilindiği gibi işletim sistemlerinin temel amaçlarından bir diğeri sistem kaynaklarının verimli kullanımının sağlanmasıdır. Bu bağlamda, kütük yönetim sistemi, kullanıcılara kütükleri üzerinde kolay işlem yapma olanağı sunarken bu kütüklerin saklandığı ikincil belleklerin de verimli bir biçimde kullanılmasını amaçlamak zorundadır. Bu amaçla, ikincil belleklerin düzenlenmesi, buralardan kütüklere yer sağlanması, boş alanların izlenmesi, kütüklerin ikincil bellekler üzerinde yerleşimlerinin, erişimin en hızlı olacağı biçimde düzenlenmesi gibi işlevler de kütük yönetimi kapsamında yerine getirilir.

Kolay ve verimli kullanım ilkelerinin yanı sıra, sistemde saklanan kütüklerin güvenliğinin gözetilmesi de zorunludur. Bu bağlamda, kütüklere erişim ve paylaşım haklarının belirlenmesi ve bunların denetlenmesi gereklidir. Güvenli kullanımla ilgili bu işlevler de kütük yönetim sisteminin yükümlülüğündedir. Güvenli kullanım ilkesi, kütüklere erişimleri denetim altında tutmanın yanı sıra bu kütüklerin bozulmalara karşı korunmasını da gerektirir. Sistemin hatalı çalışması sonucu kütüklerde istenmeyen bozulmalar oluşabilir. Hatta kimi durumlarda kütükler tümüyle yitirilebilir. Ancak kullanıcılar sistemde sakladıkları kütüklerinin korunacağına ve içerdikleri bilgilerin bütünlüğünün bozulmayacağına inanmak ve güvenmek isterler. İşletim sistemleri de güvenli kullanım ilkesi kapsamında kullanıcılara bu hizmeti sağlamayı amaçlarlar. Bu bağlamda, kütük yönetim sistemleri, yedekleme, kurtarma gibi, bozulmalara karşı önlemlerle ilgili temel altyapı araçlarını içerirler.

Yukarıda açıklandığı biçimiyle kütük yönetim sisteminin temel işlevleri, özetle üç madde altında toplanabilir. Bunlar:

- Mantıksal kütük yapılarından fiziksel yapılara geçişin sağlanması,
- İkincil belleklerin verimli kullanımlarının sağlanması,
- Kütüklerin paylaşılması, korunması ve kurtarılmasıyla ilgili araçların sağlanması

işlevleridir. İlerleyen kesimlerde, kütük yönetimi bu üç temel işlev çerçevesinde incelenecektir. Bu bağlamda önce, kütük yönetim sistemlerinin kullanıcılara sunduğu kavramsal modelle uyumlu kütük işlemleri incelenecektir. Bundan sonra, bu kavramsal modelin, gerçekleştirim yönünden ele alınışı açıklanacaktır. Bunun için önce ikincil bellek olarak diskin fiziksel yapısı açıklanacak sonra gerçekleştirimde bu yapıyı en verimli biçimde kullanmayı sağlayan düzenleme, atama ve izleme yöntemlerinden söz edilecektir.

Yukarıda verilen öncü tanıma göre, kütük ana bellek dışında saklanan veri kümelerine verilen addır. Bilindiği gibi, giriş/çıkış birimleri verilerin kullanıcı ile bilgisayar ortamı arasında aktarılmasına ya da bu ortamda saklanmasına yarayan araçlardır. Modern işletim sistemlerinde kütük kavramı, işlevi ne olursa olsun tüm giriş/çıkış birimlerini kapsayacak biçimde genişletilir. Bu durumda, kullanıcılar açısından ikincil belleklerde saklanan kütükler arası işlemlerle, kullanıcı - bilgisayar ortamı arası veri aktarım işlemleri arasında herhangi bir ayrım kalmaz. Veri aktarım amaçlı giriş/çıkış birimleri aynı kütükler gibi düşünülür ve adlandırılırlar. Bunlara erişimler, kütüklere ilişkin yöntem ve araçlarla gerçekleştirilir. Bu genellemeyi kullanan *UNIX* işletim sisteminde metin türü bir kütüğün yazıcıdan dökülmesi, bu kütüğün sanki yazıcıyı simgeleyen kütüğe kopyalanması işlemi gibi, `cp prog.c /dev/lp0` biçiminde, kütük kopyalama komutu kullanılarak gerçekleştirilir. Burada `prog.c`, dökümü alınacak kütüğü `/dev/lp0` ise yazıcıyı göstermektedir.

Bu genellemeye koşut diğer bir yaklaşım da, sürücüden bağımsız giriş/çıkış yaklaşımıdır. Sürücüden bağımsız giriş/çıkış yaklaşımında, gerek kütükler ve gerekse veri aktarım amaçlı giriş/çıkış birimleri, program geliştirme ve derleme sonrasında simgesel kimliklerini korurlar. Simgesel kimlikler, ancak işletim sırasında (başında) fiziksel kütük ve sürücü adlarıyla, örneğin iş denetim dili araçları kullanılarak ilişkilendirilirler. Bu yolla, bir yandan aynı programı, yeniden derleme yapmaksızın yeni verilerle işletme olanağı yaratılırken diğer yandan programların değişik sistemler arasında taşınabilmesi sağlanır.

Kütük kavramıyla ilgili bir diğer soyutlama, kütükleri programların görüntü adres evreni içinde düşünebilmektir. Bu yolla verileri, ana bellek, giriş/çıkış birimleri gibi saklandıkları ortamlardan bağımsız bir biçimde düşünebilme ve işleyebilme olanakları yaratılır. Böyle bir ortamın yaratılması durumunda, verilerin giriş/çıkış birimlerinden ana belleğe ve ana bellekten giriş/çıkış birimlerine taşınması, kullanıcıların sorumluluğunun dışına çıkar. Ana bellek, ikincil bellek, giriş/çıkış birimi ayrımı ortadan kalkar. İlk kez *UNIX*'e atalık eden *MULTICS* işletim sistemince ele alınan bu ilginç yaklaşım günümüzde ne yazık ki kullanılmamaktadır.

6.1. Kavramsal Kütük İşlemleri

İşletim sistemleri, kullanıcıların düşüncesinde gerçekliği olan yalın kütük modelleri üzerinde kavramsal işlemler yapmaya olanak verirler. Kavramsal kütük işlemleri kullanıcılara, işletim sistemince, değişik katmanlarda değişik soyutlama düzeyleriyle sunulurlar. Kabuk katman uç kullanıcıların dolaysız eriştiği, soyutlama düzeyinin en yüksek olduğu ilk katmandır. Bu katmandan kullanıcılar, üst düzey işletim sistemi komutları aracılığıyla hizmet alırlar. İşletim sistemi komutlarının sağladığı hizmetler, programlar içinden sistem çağrıları aracılığıyla alınır. Örneğin bir kütükten bir tutanak okuma işlemi, bununla ilgili sistem çağrılarına başvurularak gerçekleştirilir. İzleyen kesimde kavramsal kütük işlemleri, işletim sistemi komutları ve sistem çağrıları olmak üzere iki ayrı bağlamda ele alınacaktır.

Çizelge 6.1. Kütüklerle ilgili Sistem Komut Örnekleri

Komut Adı	Komut Parametreleri
Kütük İşlemleri	
create	kütük adı
delete	kütük adı
rename	eski kütük adı, yeni kütük adı
attributes	kütük adı, öznitelik sözcüğü
copy	kaynak kütük adı, hedef kütük adı
type	kütük adı
mount	sürücü / birim adı
dismount	sürücü / birim adı
compare	kütük adı1, kütük adı2
backup	kütük,altkılavuz,sürücü ad(lar)ı
restore	kütük,altkılavuz,sürücü ad(lar)ı
Kılavuz Kütük İşlemleri	
list / dir	kılavuz kütük adı
make directory	kılavuz kütük adı
remove directory	kılavuz kütük adı
change directory	kılavuz kütük adı

6.1.1. Sistem Komutlarıyla Gerçekleştirilen Kavramsal İşlemler

İşletim sistemi komutlarından önemli bir kesimi kütük işlemlerine dönüktür. Gerek adlandırma, gerekse sayı ve türleri yönünden sistemden sisteme değişiklik gösterseler de, bu katman düzeyinde sunulan kütük işlemlerine ilişkin klasikleşmiş ortak bir alt küme bulmak olanaklıdır. Bu alt küme, çoğu kez kütüklerin yaratılması, silinmesi, adlandırılması, kopyalanması, yedeklenmesi, içeriklerinin karşılaştırılması, listelenmesi, özniteliklerinin günlenmesi, gibi doğrudan kütüklere yönelik işlemler ile bunlara erişimleri düzenleyen ve kılavuz olarak adlandırılan özel kütüklere ilişkin işlemleri içerir. Sistem komutlarına ilişkin bir alt küme örneği, Çizelge 6.1'de İngilizce adlarla verilmiştir.

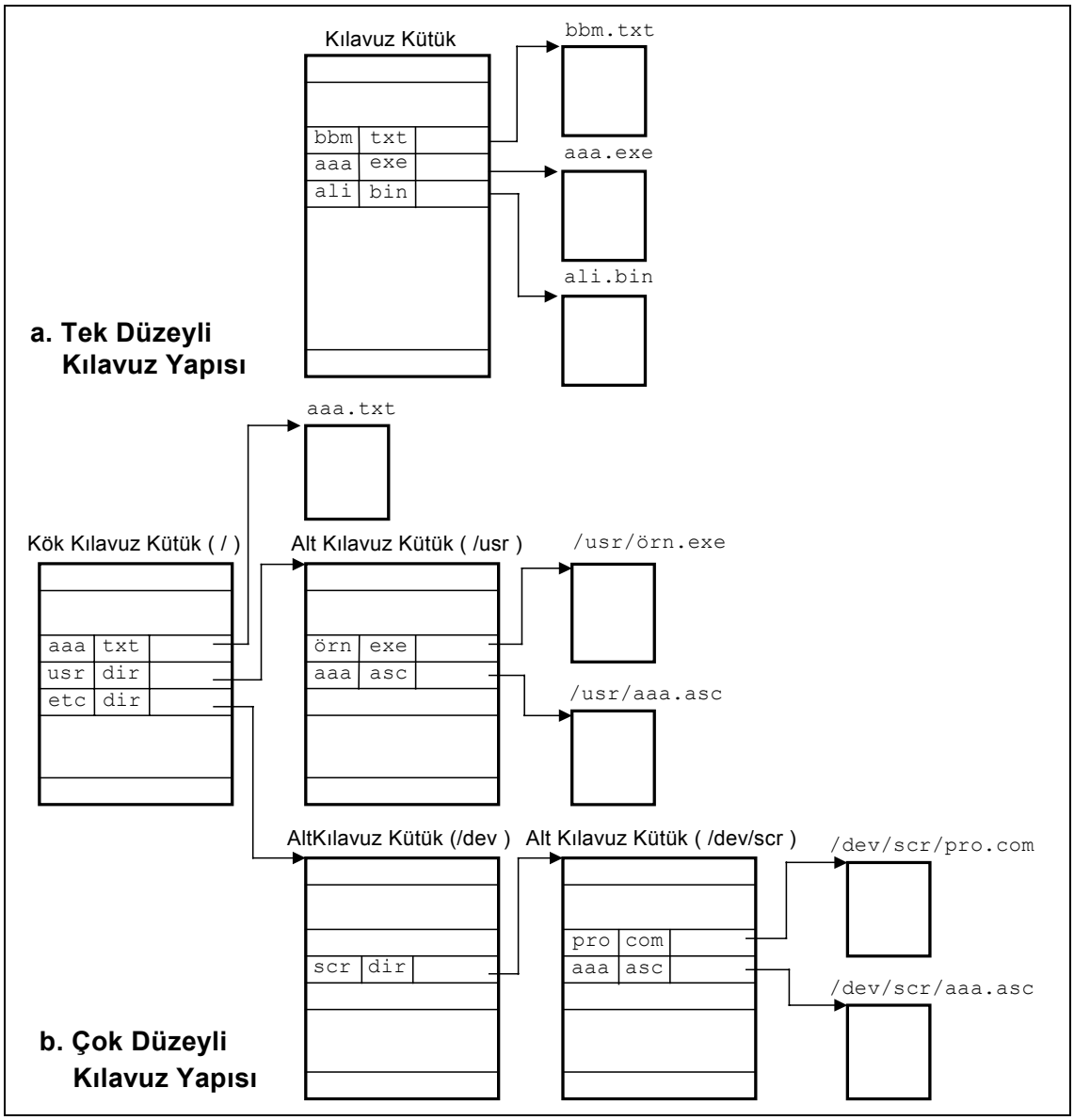
Çizelge 6.1'de yer alan komutlardan `create` ve `delete` komutları uç kullanıcıların kütüklerini yaratmak ve silmek amacıyla kullandığı komutlardır. Kimi sistemlerde kütük yaratma amacıyla öngörülen `create` komutu bulunmaz. Kullanıcılar kütüklerinde, yazılı metinlerini, kaynak ve amaç programlarını ve verilerini saklarlar. Bunları oluşturmada yararlanılan metin düzenleyici türü araçlara, oluşturulan bilgilerin saklanacağı kütükleri yaratma sorumluluğu da verilir. Kütük yaratma işlemi bu durumda, kullanıcı adına, metin / program düzenleme, derleyici gibi araçlarca gerçekleştirilir. Böylece uç kullanıcının kütük yaratma gibi bir gereksinimi kalmaz. Bu yüzden kabuk katman düzeyinde kütük yaratma komutuna yer verilmeyebilir.

Her kütük belirli bir ad ile anılır. İşletim sistemleri kullanıcılara, kütüklerine özgürce simgesel adlar takma olanağını verirler. Kütük adı, kütüğün bulunduğu sistem, sürücü ve erişim izi kimliği ile birleşerek kütük kimliğini oluşturur. Her kütüğün kimliği, varlığını sürdürdüğü evrende biriciktir. Kütüğe erişim bu kimlik kullanılarak gerçekleştirilir. Örneğin *DOS*'ta, `C:\bbm\program.pas` kütük kimliği içinde `C:` sürücü adını, `\bbm` erişim izini, `program.pas` ise kütük adını simgeler. *UNIX* gibi, kütüklerin yer aldığı sürücü kimliğinin, kullanıcı tarafından bilinmesi zorunlu olmayan işletim sistemlerinde kütük kimliği, erişim izi ve kütük adından oluşur.

`rename` komutundan kütük adlarının değiştirilmesinde yararlanılır. `attributes` komutu kütük özniteliklerinin günlenmesi amacıyla kullanılır. Salt okunur, okunur-yazılır, işletilir gibi erişim hakları; sistem ya da kullanıcı gibi iyelik bilgileri, kütük öznitelik bilgileri arasında yer alır. `copy` ve `type` komutları, sırasıyla kütükten kütüğe kopyalama ve kütük içeriklerini listeleme amacıyla kullanılır. İçerikleri, `ascii`, `ebcdic` gibi ekranda görüntülenebilen kodlardan oluşmayan kütüklerin (örneğin ikili kodlardan oluşan amaç program kütüklerinin) listelenmesi doğal olarak anlamlı değildir. `compare` komutu kütükler arası karşılaştırma amacıyla öngörülür. Kimlikleri verilen kütükler, bu komut aracılığıyla genelde, bayt bayt karşılaştırılır. Bu yolla kütüklerin aynı mı yoksa değişik kütükler mi oldukları sınanır.

`backup` komutu, kütüklerin yedeklenmesinde kullanılır. Birden çok kütük, bir alt kılavuz ya da bir sürücü üzerinde yer alan tüm kütükler, bu komut aracılığıyla, bir seferde yedeklenebilir. Yedekleme genelde, sıkıştırarak yazma yoluyla gerçekleştirilir. Saklanan arşiv kopyalarının birden çok disket, makara üzerine yayılabilmesine izin verilir. Yedeklemede özel sıkıştırma yöntemleri kullanıldığından `backup` komutuyla yedeklenen kütükler, özel bir sistem komutu kullanılmaksızın diske yeniden yüklenemezler. Bu özel sistem komutu `restore` komutudur.

Sisteme, sökülür / takılır nitelikli bir sürücü üzerinden (disket, makara gibi) yeni bir birimin takılması durumunda `mount` komutu kullanılır. Bu komut aracılığıyla, eklenen birime ilişkin kılavuz kütük, tüm sisteme ilişkin genel kılavuz kütüğe işlenir. `dismount` komutu, sistemden çıkarılan alt kılavuzların ana kılavuzdan da çıkarılmasını sağlar.



Çizim 6.2. Tek ve Çok Düzeyli (Sıradüzensel) Kılavuz Yapıları

Çizelge 6.1'in alt kesiminde yer alan komut örnekleri kılavuz kütüklerle ilgilidir. Adlarından da anlaşılacağı üzere, kılavuz kütükler sistemde yer alan kütüklere erişimi sağlamada yararlanılan ve bu kütüklere ilişkin ad, tür, yaratılma ve son günlenme zamanı gibi bilgilerle fiziksel erişim bilgilerinin tutulduğu özel kütüklerdir. Kılavuz kütükler çoğu kez çok düzeyli ya da sıradüzensel olarak nitelenen ağaç yapısında gerçekleştirilirler. Bu yapı çerçevesinde kılavuz kütükler, bir kök ve buna bağlı alt kılavuzlardan oluşurlar. Kök kılavuz olarak nitelenen ilk düzey kılavuz kütük, **:** \ / gibi damgalarla simgelenir ve sistem tarafından yaratılır. Alt kılavuzlar sıradan kütükler gibi kullanıcılar tarafından özgürce adlandırılıp yaratılırlar. Alt kılavuzlar ya doğrudan kök kılavuza ya da bir üst düzeyde yer alan diğer bir alt kılavuza bağlı olurlar. Sistemde yer alan tüm kütükler kök ve alt kılavuzlara bağlı olarak yer alırlar. Çok düzeyli kılavuz

kütük yapısının kullanıldığı sistemlerde kütük kimlikleri, (sistem ve sürücü kimlikleri dışarında) kütük adı ile kütüğün bulunduğu alt kılavuz adından oluşur. Bu nedenle değişik alt kılavuzlar altında yer alan kütükleri aynı adla adlandırmak olanaklıdır. Kütük kimlikleri içinde kök kılavuz adından başlayarak kütük adına varıncaya kadar yer alan kılavuz adları ilgili kütüğün izi olarak bilinir (Çizim 6.2). Bu bağlamda:

```
/etc/scr/aaa.asc
```

satırında `/etc/scr` kütük izini, `aaa.asc` ise kütük adı ve türünü temsil etmektedir. `scr` alt kılavuzuna `etc` alt kılavuzundan erişilmektedir. Ağaç yapısı gereği `scr` alt kılavuzunun `etc` alt kılavuzu altında yer aldığı söylenir. `etc` alt kılavuzu ise, `/` damgası ile simgelenen kök kılavuz altında yer almaktadır. Çok düzeyli kılavuz kütük yapısında kütük kimliği, kütük izi ve kütük adından oluştuğundan kütük adları, bulunulan alt kılavuz altında anlam taşır. Bu nedenle, Çizim 6.2'de `aaa.asc` adı, biri `/etc/scr`, diğeri de `/usr` olmak üzere iki ayrı kılavuz kütük altındaki iki ayrı kütüğü temsil etmektedir. Daha önce de belirtildiği üzere, bu yapı, kütüklerin adlandırılmasında kullanıcılara önemli bir esneklik sağlamaktadır.

Çok düzeyli kılavuz kütük yapısının atası tek düzeyli kılavuz kütük yapısıdır. Tek düzeyli kılavuz kütük yapısının kullanıldığı sistemlerde tüm kütüklere tek bir kılavuz kütük üzerinden erişilir. Böyle bir kılavuz kütük, sistemde yer alan tüm kütüklere ilişkin ad, tür, öznitelik, yaratılma ve son günlenme zamanı gibi bilgiler ile erişim bilgilerini tutar. Bir bilgisayar sisteminde yüzlerce hatta binlerce kütük olabileceği, kılavuz içinde, her kütük için, en az birkaç düzine baytlık bir yer gerektiği düşünüldüğünde kılavuz kütük boyunun aşırı büyük olacağı kolayca görülür. Kütüklere erişimde tek ve büyük bir kılavuz kütük kullanmanın sakıncaları çeşitlidir. Bunlardan önemli bir tanesi adlandırmaya getirilen kısıtlamadır. Aynı kılavuz kütük içinde aynı kimliği taşıyan birden çok kütük bulunamayacağından tüm kütüklerin aynı kılavuz içinde tutulması kullanıcılara, kütükleri için, sistemde yer alan diğer kütük adlarını uzun listeler biçiminde inceleyerek halen kullanılmayan bir ad bulma zorunluluğu getirir. Bu, adlandırma özgürlüğüne ve esnekliğine getirilen önemli bir kısıtlamadır.

Kılavuz kütüklerle ilgili işletim sistemi komutları, genelde bu özel kütüklerin yaratılması, silinmesi, içeriklerinin listelenmesi gibi amaçlar için öngörülür. `make directory` komutu, alt kılavuz yaratma amacıyla kullanılır. Alt kılavuz adları, sıradan kütüklerin adlandırılmasında uyulan kurallar çerçevesinde, alt kılavuzu yaratan kullanıcı tarafından belirlenir. Kök kılavuz, sistem diskinin iz ve sektörlerinin işaretlenerek kullanıma hazır biçime getirildiği formatlama işlemi sonunda, sistem tarafından yaratılır. `remove directory` komutundan, var olan bir alt kılavuzu silmek için yararlanılır. Bir alt kılavuzu silmek, buna bağlı, varsa diğer alt kılavuz ve kütükleri yok sayarak gerçekleşemez. Hatalı işlemden geri dönüşü kolaylaştırmak üzere, kimi işletim sistemlerinde alt kılavuzların silinmesi bunlara bağlı başka alt kılavuz ve kütüklerin bulunmaması koşuluna bağlanır. Başka bir deyişle bir alt kılavuzun silinebilmesi için, bunun aracılığıyla erişilen alt kılavuz ve kütüklerin daha önce silinmiş olması koşulu aranır. Kök kılavuz, formatlama işlemi dışında silinemez. `list, dir` gibi kısaltmalarla anılan komutlar kılavuz kütük içeriklerinin listelenmesine yarar. Kılavuz kütükler

kendilerine bağlı kütüklere ilişkin ad, tür, tarih gibi bilgileri içerdiklerinden bu komutlar çalıştırılarak ilgili kılavuza bağlı kütüklerin ad ve özellik listesi elde edilir.

Kılavuz kütüklerle ilgili önemli bir sistem komutu, `change directory` olarak adlandırılan erişim izi değiştirme komutudur. Burada sözü edilen değiştirme işlemi, geçerli olan izin değiştirilmesi anlamındadır. Bilindiği üzere kütük kimliği, (sistem ve sürücü kimlikleri dışarında) kütük izi ve kütük adından oluşur. Kütüklerle ilgili sistem komutları işletilirken, her seferinde, ilgili kütüğün kimliğinin tümünü girmek uzun ve sıkıcı bir işlem olabilir. Bu nedenle kütük izi, kütük adından ayrı olarak ele alınır. Bu amaçla, `change directory` gibi kütük izi belirleme komutları sistem komutları arasına eklenir. Bu komut aracılığıyla önce ilgili erişim izi belirlenir. Bu belirleme bir kez yapıldıktan sonra, ilgili işlemler için, salt kütük adının girilmesi yeterli olur. Girilen kütük adı, o an geçerli erişim izine bağlı kütükler arasında aranır. Geçerli erişim izinin görüntülenmesini sağlayan sistem komutları da bulunur. Örneğin *UNIX*'te `pwd` *(present working directory)* komutu bu amaçla kullanılır.

6.1.2. Sistem Çağrılarıyla Gerçekleştirilen Kavramsal İşlemler

Kabuk katmanın işlevi, kullanıcıların girdiği komutların çözümlemesini yaparak istenen işlemi belirleme ve bu işlemle ilgili sistem çağrılarını çalıştırmaktır. Sistem komutlarının işletilmesinde kullanılan düzenek sistem çağrı düzeneğidir. Uç kullanıcılar bu düzenekten, sistem komutları aracılığıyla, ancak bu düzeneği görmeden yararlanırlar. Kütüklerle ilgili işletim sistemi hizmetlerinden, programlar içinden yararlanabilmek için, sistem çağrı düzeneğini kullanmak gerekir. Sistem çağrılarının doğrudan kullanımı, ayrıcalıklı diye nitelenen ve sistem programcılarını da içeren özel bir kullanıcı sınıfına açıktır. Sıradan kullanıcıların, üst düzey programlama dilleriyle tanımladıkları kütük işlemleri ise, derleme aşamasında derleyiciler tarafından sistem çağrılarına dönüştürülür. Sistem çağrıları, sistem komutlarına göre, soyutlama yönünden bir alt düzeyde yer alırlar. Kimi sistem komutları, birebir sistem çağrılarına dönüşürken kimileri birkaç sistem çağrısının ardarda işletilmesini gerektirir. Sistem çağrıları arasında, kütük yaratma, silme, adlandırma gibi yukarıda açıklanan kütük işlemlerini gerçekleştiren işlevlerin tümü bulunur. İzleyen kesimde bunların dışında kullanılan çağrı örneklerine yer verilecektir.

Kütük işlemlerinde en çok kullanılan sistem çağrıları, örnek İngilizce adlandırmalarla aşağıda verilmiştir:

open (kütük açma)
close (kütük kapama)
sequential-read (sıradan okuma)
sequential-write (sıradan yazma)
random-read (rasgele okuma)
random-write (rasgele yazma)
seek (tutanak arama)

`open` sistem çağrısının birden çok işlevi vardır. Bunlardan en önemlisi, program içinde geçen kütük kimliği ile fiziksel kütüğün, işletim sırasında eşleştirilmesidir. İşletim aşamasında gerçekleşmesi itibarıyla bu eşleştirme, sürücü bağımsızlığı ve program

taşınırlığına katkı verebilmeyi amaçlar. Kütük kimliğinin fiziksel kütük ile eşleştirilmesi, fiziksel kütüğe erişim bilgilerine ana bellekte yer ayrılarak yapılır. Bilindiği üzere bir kütüğe erişim, kütük izinde belirtilen kılavuz kütükler kullanılarak gerçekleştirilir. Kılavuz kütükler, erişim bilgilerini tuttukları kütüklerle birlikte aynı sürücüler üzerinde saklanırlar. Bu durumda, bir kütüğe erişim, önce kılavuz kütüğe sonra da ilgili kütüğe olmak üzere iki sürücü (disk) erişimi gerektirir. Erişim sayısını birde tutabilmek amacıyla, bir kütükle ilgili okuma - yazma işlemlerine geçmeden önce, bu kütüğe ilişkin erişim bilgilerinin tutulduğu kılavuz kütük ögeleri ana belleğe taşınmalıdır. Bu işlem `open` sistem çağrısı aracılığıyla gerçekleştirilir.

`open` sistem çağrısının bir diğer işlevi kütüklere erişimin denetlenmesi ve paylaşımın sağlanmasıdır. Genelde `open` sistem çağrısı işletilirken erişilmek istenen kütüğe (salt okuma, okuma-yazma gibi) ne amaçla erişileceği de belirtilir. Bu yolla işletim sistemi, çağrı parametreleri arasında yer alan erişim istem kodunu, ilgili görev / kullanıcı hakları ile karşılaştırma olanağı bulur. Bu durumda, open çağrısının parametreleri arasında, `open(kütük kimliği,erişim türü)` gibi, simgesel kütük kimliğinin yanı sıra bir de erişim türü bilgisi yer alır.

`close` sistem çağrısı, `open` sistem çağrısıyla birlikte düşünülür. Yukarıda belirtildiği gibi, `open` sistem çağrısı işletildiğinde, sözkonusu kütükle ilgili erişim bilgilerinin tümü ya da bir kesimi, ana bellekte, bu amaçla öngörülen yastık alanlarına aktarılır. Fiziksel kütüğe bu bilgiler kullanılarak erişilir. Bu durumda, açılan her kütüğe bir yastık alanı atamak gereklidir. Çoğunlukla işletim sistemi kapsamında düşünülen bu yastık alanlarının toplam sığaları için bir üst sınır vardır. Bu, sistemde eş zamanlı olarak açık tutulabilen kütük sayısının da bir üst sınırı bulunduğu anlamına gelir. Bu nedenle görevlere, açtıkları kütükler üzerinde (okuma - yazma gibi) gerekli işlemleri tamamladıklarında işletim sistemini uyarma zorunluluğu getirilir. Bu uyarı kütük kapama anlamına gelen `close` sistem çağrısıyla gerçekleşir. Bu yolla, kapanan kütüğe ilişkin yastık alanı, ilgili görevden geri alınır ve serbest yastık listesine bağlanır. Bir sistemde aynı anda açık tutulabilen kütük sayısının bir üst sınırının bulunması, görevlerin herbiri için de bir üst sınır belirlemeyi gerekli kılar. Bu nedenle, işletim sistemleri, her görevin açık tutabileceği kütük sayısına kısıtlama getirir. Bu kısıtlama görevle ilgili bir parametre olarak saklanır. Bu parametre, örneğin *UNIX* işletim sisteminde `maxfiles` komutu ile belirlenebilir. *MS-DOS*'ta ise buna benzer `files` komutu vardır. `close` çağrısı, genelde `close(kütük kimliği)` biçiminde, tek parametreli bir çağrıdır.

`open` ve `close` sistem çağrıları, kütüklerin birden çok görev tarafından paylaşılması için gerekli zamanuyumlama düzeneğinin kurulabilmesine de altyapı oluşturabilirler. Aynı kütüğe erişen birden çok görevden `open` komutunu ilk çalıştıran, ilgili kütüğün erişimini diğer görevlere kapatabilir. İşletim sistemi, `open` komutunu çalıştıran diğer görevlerin, `close` komutu işletilene değin ilgili kütüğe ilişkin bekleme kuyruğuna konmasını sağlayabilir. Bu biçimiyle, `open` ve `close` sistem çağrıları, üst düzey zamanuyumlama işleçleri gibi kullanılabilirler.

Bilindiği gibi, işletim sistemleri, kullanıcılara kütükleri, ya tutanak ya da damga dizisi olarak düşünebilme olanağı verirler. Kütüklerle ilgili okuma ve yazma işlemleri de, bu nedenle, tutanak ya da damga okuma ve yazma işlemleri olarak düşünülür. Okuma ve yazma işlemleri, genelde iki biçimde gerçekleştirilir. Bunlar sıradan ve rasgele okuma-yazma işlemleridir. Sıradan okuma - yazma işlemlerinde bir önceki işlemde sözkonusu edilen tutanağı izleyen tutanağın okuma - yazma işlemi gerçekleştirilir. Her okuma yazma sonrasında tutanak göstergesi kendiliğinden bir artırılır. Bu gösterge, tutanağın kütük içindeki mantıksal sırasıdır. Şimdiye değin sözkonusu edilen sistem komut ve çağrılarının, mantıksal nitelikli işlemleri yerine getirdiklerini; bunlara konu olan tutanakların da, gerek boyları gerek kütük içindeki yerleri bakımından mantıksal nitelikli olduklarını anımsamakta yarar vardır. Sıradan okuma - yazma çağrılarının, kullandıkları parametrelerle örnek görünümleri aşağıda verilmiştir:

```
sequentialread (kütük kimliği,yastık)
sequentialwrite(kütük kimliği,yastık)
```

Burada `yastık`, ilgili tutanağın aktarılacağı ve ilgili görevin adres evreni içinde tanımlı bellek alanını temsil etmektedir. Sıradan okuma - yazma çağrılarında, çağrı parametreleri arasında mantıksal tutanak numarasına genelde yer verilmez. Zira ardarda işleme gireceği varsayılan tutanakların numaraları, sıradan okuma - yazma çağrısı ile kendiliğinden artırılmaktadır. Ancak bu durum, ilk okuma işleminde hangi tutanak numarasının kullanılacağı sorusunu akla getirebilir. Sıradan okuma - yazma çağrılarında, kütük açma işleminden sonra ilk erişilen tutanağın, genelde hep sıfırıncı tutanak olacağı varsayılır. İşlemler sırasında kütük başına geri dönülmesi gerektiğinde, erişilen tutanak numarasını sıfırlamaya yarayan, `rewind` gibi özel çağrılar öngörülür.

Rasgele okuma - yazmada, ilgili tutanağın kütük içindeki sıra numarası verilerek herhangi bir tutanak üzerinde işlem yapabilme olanağı sağlanır. Bu amaçla kullanılan sistem çağrıları aşağıdaki görünümde olabilir:

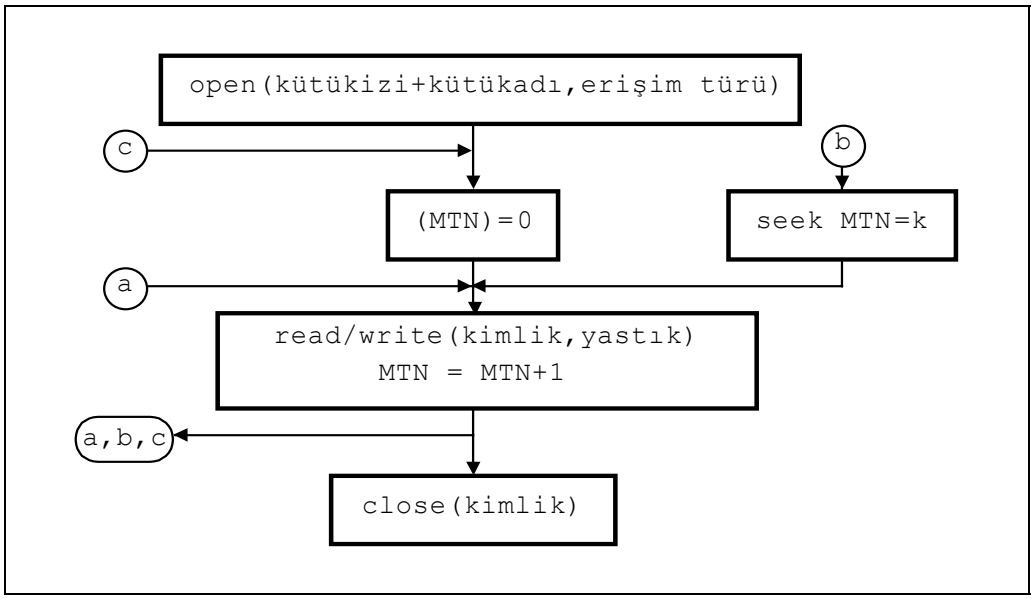
```
randomread (kütük kimliği,tutanak numarası,yastık)
randomwrite(kütük kimliği,tutanak numarası,yastık)
```

Kimi işletim sistemlerinde okuma - yazma işlemleri, yukarıda listelendiği biçimde, sıradan ve rasgele olarak ayrıştırılmaz. Bu işlemler, hep, sıradan okuma yazma işlemleri olarak sunulurlar. Ancak, bunların yanı sıra , `seek` adlı tutanak arama işlemi de sistem çağrıları arasına katılır. Rasgele okuma - yazma yapılacaksa, önce `seek` çağrısı ile sözkonusu tutanak belirlenir sonra okuma - yazma işlemi gerçekleştirilir. Bu durumda, örneğin rasgele okuma işlemi aşağıdaki biçimde gerçekleşir.

```
seek (kütük kimliği,mantıksal tutanak numarası)
read (kütük kimliği,yastık)
```

`read, write, seek, close` gibi sistem çağrıları içinde yer alan `kütük kimliği` parametresi, çoğu kez `open` işlemi sonrasında elde edilen bir kimlik numarasıdır. `open(kütük kimliği,erişim türü)` çağrısı çalıştırıldığında, bu çağrının gerektirdiği işlemler gerçekleştirilip işlem başarı ile sonuçlanmışsa çağrıyı yapan göreve İngilizce *handle, file descriptor* gibi adlarla anılan bir sayı geri döndürülür. Görev de

bu sayıyı, diğer kütük işlemlerinde kütük kimlik numarası ya da kütük göstergesi olarak kullanır. Doğal olarak `open` çağrısında yer alan `kütük kimliği`, kütük erişim izi ve kütük adından oluşmak zorundadır.



Çizim 6.3. Klasik Kütük İşlemleri

Sistem çağrıları aracılığıyla gerçekleştirilen kütük işlemlerinin özeti Çizim 6.3'te verilmiştir. Buna göre herhangi bir kütük üzerinde, herhangi bir işlem yapmadan önce kütük açma işleminin gerçekleştirilmesi, başka bir deyişle program içinde geçen simgesel kütük kimliğinin fiziksel kütük ile eşleştirilmesi gereklidir. Bu işlem bir kez gerçekleştikten sonra, ilgili kütüğün belirli sayıda tutanağı sıradan okunup yazılabilir (a). Mantıksal tutanak numarası istenen değerle günlenerek rasgele okuma - yazma işlemi gerçekleştirilebilir (b). Mantıksal tutanak numarası sıfırlanarak sıradan okuma - yazma işlemleri yinelenebilir (c). Kütük üzerinde öngörülen tüm işlemler tamamlandığında kütük kapama işlemi gerçekleştirilir. Kütüklerle ilgili sistem çağrıları, izleyen kesimde *MS-DOS*'un kimi sistem çağrılarıyla örneklenmiştir.

6.1.3. Kütüklerle ilgili kimi Örnek *MS-DOS* Sistem Çağrıları

Bilindiği üzere *MS-DOS* (kısaca *DOS*) *80X86* serisi işleyicilere dayalı, standart kişisel bilgisayar sistemlerinde kullanılan bir işletim sistemidir. *DOS*'ta sistem çağrıları *80X86* işleyicilerinin `int xxh` yazılım kesilmelerine dayalı biçimde gerçekleştirilmektedir. Bunlardan `int 21H` kimlikli yazılım kesilmesine dayalı olarak gerçekleştirilen çağrılar, *DOS* işlevleri olarak adlandırılmaktadır. *DOS* işlevleri, kütüklerle ilgili hemen hemen tüm sistem çağrılarını karşılamaktadır. `int 21H` yazılım kesilmesiyle çağrılan işlevin türü, *80X86* işleyicilerinin `ah` yazmacı içine yerleştirilen kod ile seçilebilmektedir. Belirlenen işlev ile bunu çağıran görev arasında parametre ve sonuç akışı, yine işleyici yazmaçları üzerinden gerçekleşmektedir. Bu yapıya göre, örneğin kütük açma işlemi:

```
asciiz     db     '\bbm\örnküt.doc',00H
handle     dw     ?
.
mov  ah,3DH                    ;işlev kodu
mov  al,02                     ;erişimin türü
mov  dx,offset asciiz          ;kütük adı
int  21h                       ;çağrı
jc   error                     ;hata
mov  handle,ax
.
```

komutlarıyla gerçekleştirilebilmektedir. Burada, seçilen işlev, `3D` kodlu *open handle* işlevidir. `int 21h` kesilmesi bu işlev koduyla çağrılmadan önce, işlevin tanımı gereği `DS:DX` yazmacı içinde kütük izi ve kütük adının yer aldığı `asciiz` olarak adlandırılan ve `00` değeriyle sonlandırılan damga dizgisinin başlangıç adresinin bulunması gerekmektedir. Kütük açma işleminin hangi amaçla yapılmak istendiği ise `al` yazmacı içine yazılan bir kodla işleve aktarılmaktadır. *DOS* tanımlarına göre `al` yazmacının en küçük ağırlıklı üç bitinin, örneğin `010` değerini taşıması, yapılmak istenen erişimin okuma - yazma işlemi olduğunu belirlemektedir. İşlev başarıyla sonuçlanırsa kesilmeden geri dönüşte *DOS*, *PSW carry* bitini sıfırlayarak *handle* diye adlandırılan ve kütükle ilgili bundan sonraki işlemlerde kütük kimliği yerine geçecek 16 bitlik işaretsiz tamsayıyı, `ax` yazmacı içinde, çağrıyı yapan göreve döndürmektedir. İşlevin, kütük izinin ya da kütüğün kendisinin bulunamaması, erişim koruması gibi nedenlerle başarısızlıkla sonuçlanması durumunda *carry* biti kurulmakta, `ax` yazmaç içeriği ise hatanın türünü belirleyen bir kod içermektedir.

Çizelge 6.2. Kütük Yönetimine ilişkin kimi *DOS* İşlevleri

Kod	Tanım
0E	*Select Disk Drive*
0F	*Open File*
10	*Close File*
13	*Delete File*
14	*Sequential Read*
15	*Sequential Write*
16	*Create File*
17	*Rename File*
19	*Get Current Disk*
1A	*Set Disk Transfert Area*
21	*Read Random Record*
22	*Write Random Record*
23	*Get File Size*
24	*Set FCB Random Record Field*
27	*Read Random Records*
28	*Write Random Records*
29	*Parse Filename*

Kod	Tanım
39	*Create Directory*
3A	*Remove Directory*
3B	*Change Current Directory*
3C	*Create File*
3D	*Open Handle*
3E	*Close Handle*
3F	*Read from File/Device*
40	*Write to File/Device*
41	*Delete File*
42	*Move File Pointer*
45	*Duplicate Handle*
46	*Force Duplicate Handle*
47	*Get Current Directory*
56	*Rename File*
5B	*Create New File*
5C	*Lock / Unlock File Region*
67	*Set Handle Count*

Kütük açma işleminin hatasız gerçekleşmesi durumunda elde edilen *handle* değeri, kütük üzerinde, okuma, yazma, kapama gibi diğer işlemler gerçekleştirilirken kullanılmaktadır. Bu bağlamda, `3F` kodlu, *read from file* işlevi ile belirli sayıda damga, *handle* değeri bilinen bir kütükten, ilgili görev adres evreni içinde tanımlı bir yastık alanına okunabilmektedir. *DOS*'un ilk versiyonunda, sistem işlevlerinin kullandığı kavramsal modelde kütükler, tutanak dizileri olarak düşünülmüştür. Ancak *UNIX*'in etkisiyle, ikinci versiyondan başlayarak kütükleri, tutanak dizileri yerine bayt dizileri olarak düşünebilme ve kullanabilme olanağını sunulmuştur. Bu nedenle, Çizelge 6.2'de, kütük yaratma, açma, okuma, yazma, kapama, silme gibi işlevlerden birer çift bulunmaktadır. Bunlardan birinci versiyonda tanımlananlar `00-2EH`, ikinci ve ileri versiyonlarda tanımlananlar ise `2F-68H` kodları arasında yer almaktadır. Burada, açıklaması yapılarak örneklenen işlevler, ikinci versiyondan başlayarak tanımlanan ve kütükleri bayt dizileri olarak gören işlevlerdir. *DOS*'un ilk versiyonunda tanımlanmış ilk grup işlevlere ve bunları ikinci gruptan ayıran özelliklerine ileride değinilecektir. *read from file* işlevi kullanılarak 80 baytlık bir dizinin, `handle` adlı değişkende kimliği bulunan bir kütükten, `yastık` adlı ilgili görev ortamına okunması aşağıda örneklenmiştir:

```
yastık     db     80 dup(?)
.
mov ah, 3FH             ;read from file işlev kodu
mov bx, handle          ;kütük kimliği
mov cx, 80              ;dizi(tutanak) boyu=80
mov dx, offset yastık   ;aktarım alanı
int 21H                 ;çağrı
jc  error               ;carry = 1 ise hata
.
```

Yukarıda verilen örnek dikkatlice incelendiğinde, *read from file* işlevinin parametreleri arasında, ilgili dizinin hangi konumdan başlayarak okunacağına ilişkin herhangi bir bilginin bulunmadığı görülür. *read from file* işlevi, sözkonusu dizinin, o an, *file pointer* olarak anılan gösterge değerinden başlayarak okunacağını varsayar. Okuma işlemi sonrasında da bu göstergeyi okunan damga sayısı kadar artırır. *read from file* işlevi rasgele bir konumdan başlayarak okuma yapmak amacıyla kullanılmak istenirse, öncelikle *move file pointer* işlevi çalıştırılarak mantıksal okuma-yazma başlangıç konumunun günlenmesi gerekecektir. *move file pointer* işlevi, daha önce sözü edilen *seek* çağrısına karşılık gelen bir işlevdir. *move file pointer* işlevi, `bx` yazmacı içinde kimliği (`handle` değeri) bulunan kütüğün mantıksal okuma-yazma başlangıç konumunu (kütük göstergesini), `al-cx-dx` yazmaç üçlüsü içindeki yeni konumla günlemeye olanak vermektedir. `cx-dx` yazmaç ikilisinin içerdiği 32 bitlik tamsayı, al yazmacının taşıdığı taban koduna görelidir. `al` yazmacı `00` değerini içerirse, `cx-dx` yazmaç ikilisinin içerdiği 32 bitlik tamsayı, kütük başına göreli konumu gösterir. Bunun gibi `al` yazmacının `02` değerini içermesi durumunda, `cx-dx` yazmaç ikilisinin içerdiği 32 bitlik tamsayı, kütük sonuna göreli konumu gösterir. Son olarak `al` yazmacının `01` değerini içerirmesi durumunda da, `cx-dx` yazmaç ikilisinin içerdiği 32 bitlik tamsayı, o anki göstergeye göreli konumu belirlemeye yarar. *move file pointer*

işlevi hatasız çalıştığında, çağıran göreve, `dx-ax` yazmaçları içinde, kütük başına göreli 32 bitlik yeni gösterge değerini döndürür. Tüm diğer işlevlerde olduğu gibi, hatalı çalışma durumu, *carry* bitinin kurulması yoluyla ilgili göreve bildirilir. Bu durumda `al` yazmacının içerdiği değer hata kodu olarak yorumlanır. *move file pointer* işlevinin kullanımı, `handle` kimlikli bir kütüğe, 2048 inci baytından başlayarak `yastık` adlı bellek alanında yer alan 256 baytın yazılmasını gerçekleştiren, aşağıdaki örnek kapsamında gösterilmiştir:

```
yastık   db     256 dup(?)
.
mov ah, 42h            ;move file pointer işlevi
mov al, 0              ;kütük başına göreli
xor cx, cx             ;cx ← 0
mov dx, 2048           ;2048. konum
int 21h                ;çağrı
jc  error1             ;hatalı işletim
;
mov ah, 40h            ;write to file işlevi
mov bx, handle         ;kütük kimliği
mov cx, 256            ;tutanak boyu
mov dx, offset yastık ;aktarım alan adresi
int 21h                ;çağrı
jc  error2             ;hatalı işletim
.
```

Çizelge 6.3. ***File Control Block - FCB*'nin Yapısı**

Başa göreli Adres	***Uzunluk***	***Alan Tanımı***
00H	1	*Drive ID*
01H	8	*Filename*
09H	3	*File Extention*
0CH	2	*Current Block Number*
0EH	2	*Record Size (Bytes)*
10H	4	*File Size (Bytes)*
14H	2	*Date*
16H	2	*Time*
18H	8	*Reserved for DOS*
20H	1	*Current Record Number*
21H	4	*Random Record Number*

write to file işlevi, *read from file* işlevinde olduğu gibi, yerine getirdiği yazma işlemi sonrasında kütük göstergesini yazılan bayt sayısı kadar artırır. Bu işlem sonrasında yeniden yazma yapılırsa, yeni yazılan dizi, bir önce yazılan dizinin sonuna eklenir. Bu biçimiyle, *write to file* ve *read from file* işlevlerinin, *move file pointer* işleviyle birlikte, hem sıradan, hem de rasgele okuma ve yazma işlemlerini gerçekleştirmede kullanılabildikleri görülür.

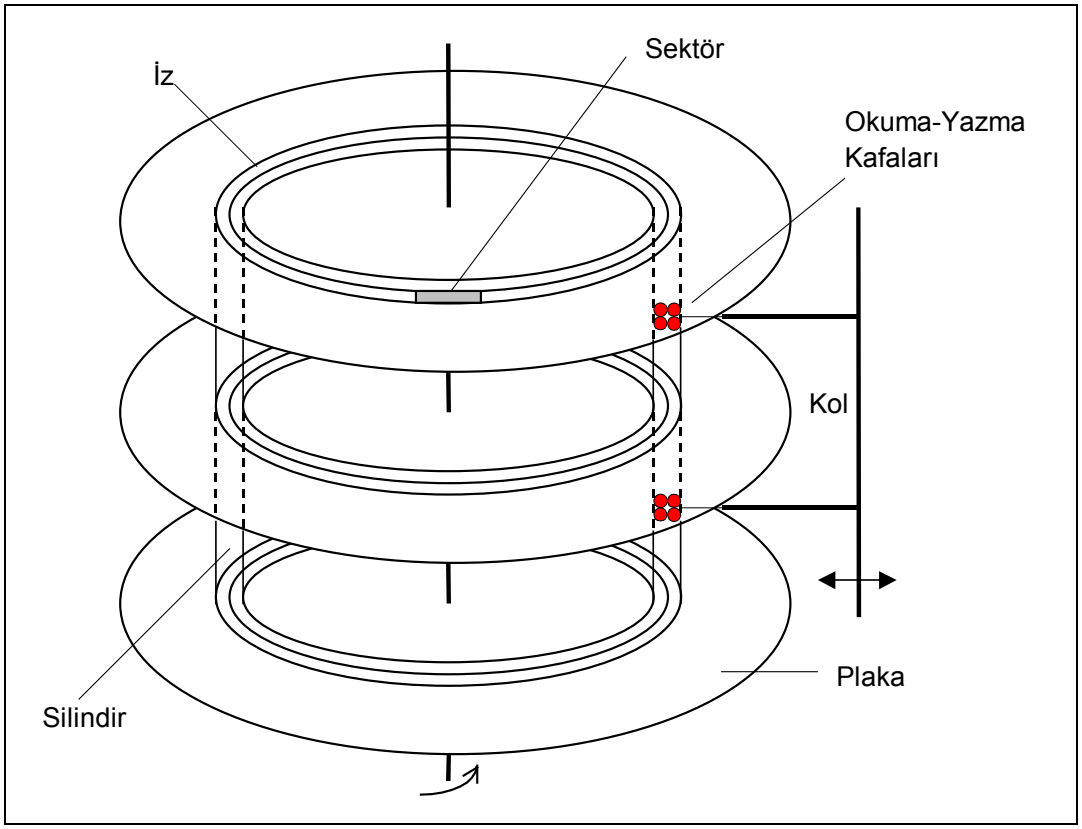
Daha önce de belirtildiği üzere, *DOS*'ta, aynı işe yarayan iki ayrı grup işlev bulunmaktadır. Şimdiye değin örneklenenler, ikinci versiyondan başlayarak tanımlanmış olanlardır. *DOS*'un birinci versiyonunda tanımlanan kütük yönetim işlevleri, *file control block (FCB)* olarak adlandırılan bir yapıyı kullanmaktadır. Bu yapının görünümü Çizelge 6.3'te verilmiştir. *FCB, DOS* ile ilgili görev arasında, üzerinde işlem yapılacak kütüğe ilişkin parametrelerin aktarıldığı bir veri yapısıdır. `0FH` kodlu *open* işlevi ile bir kütüğü açabilmek için, bir *FCB*'nin tanımlanması ve kütük kimliğinin buraya yazılarak işleve aktarılması gereklidir. *open* işlevi sonunda da *DOS*, *FCB*'nin ilgili alanlarını açılan kütük bilgileriyle günleyerek, işlevi çağıran göreve döndürür. Bunun gibi okuma ve yazma işlemlerine taban oluşturan, örneğin kütük göstergesi de *FCB* içinde tutulur. *FCB* yapısının kullanımı, Çizim 6.4'te verilen kütük kopyalama yordamı ile örneklenmiştir.

```
fcb1        db        'Cörneküt1txt',19 dup(?)
fcb2        db        'Cörneküt2txt',19 dup(?)
.
file_copy   proc      far
            open      fcb1
            open      fcb2
again:      read_seq  fcb1
            cmp       al,01h
            je        alldone
            write_seq fcb2
            jmp       again
alldone:    close     fcb1
            close     fcb2
            ret
file_copy   endp
read_seq    macro fcb
            mov   dx, offset fcb
            mov   ah, 14h
            int   21h
            endm
open        macro fcb
            mov   dx, offset fcb
            mov   ah, 0fh
            int   21h
            endm
close       macro fcb
            mov   dx, offset fcb
            mov   ah, 10h
            int   21h
            endm
```

Çizim 6.4. *FCB*'ye dayalı *DOS* İşlevlerinin Kullanım Örneği

FCB, işlevi çağıran görev adres evreni içinde tanımlanan bir veri yapısıdır. *DOS*'un ikinci versiyonundan başlayarak bu veri yapısını, işletim sistemi adres evreni içinde tutan ve görevlerin erişimine kapayan yeni bir yaklaşım benimsenmiştir. Yukarıda

handle olarak anılan kimlik, aslında bu yapının kimliğidir. Bu yeni yaklaşıma göre tanımlanan kütük yönetim işlevleri, *FCB*'ye dayalı eski işlevlerin tümünü karşılamaktadır. Bu nedenle ilk versiyonda tanımlanan kütük yönetim işlevlerinin ve *FCB* yapısının kullanımına gerek kalmamıştır.



Çizim 6.5. Diskte Verilerin Fiziksel Düzenlenişi

6.2. Diskin Fiziksel Yapısı

Kütük yönetim sisteminin temel işlevlerinden biri, kütüklerin, kullanıcıların düşüncesinde gerçekliği olan mantıksal yapılarının, ikincil bellekler üzerindeki fiziksel varlıklarıyla eşleştirilmesidir. Bu eşleştirmenin nasıl yapıldığını anlayabilmek için ikincil belleklerin fiziksel düzenleniş biçimlerinin bilinmesi gerekir. Bilgisayar sistemlerinde ikincil bellekler olarak çoğunlukla disk, mıknatıslı şerit birimleri kullanılmaktadır. Disk birimleri mıknatıslı ve optik olmak üzere iki sınıfa ayrılmaktadır. Optik diskler, 1990'lı yıllar itibarıyla, daha çok, ya tümüyle salt okunur ya da *WORM (Write-Once-Read-Many)* diye nitelenen ve salt okunur özelliğe daha yakın türde olmaktadır. Okunur-yazılır özellik içeren optik disklerin erişim süreleri mıknatıslı disklerinkine göre çok büyüktür. Bu nedenle, optik disk ve mıknatıslı şerit birimlerinden, daha çok veri yedekleme ve taşıma amacıyla yararlanılmaktadır. İşletim içi erişilen veri ve programların saklandığı birimler, daha çok mıknatıslı disk birimleridir. İzleyen kesimde ikincil bellekler olarak salt mıknatıslı disk birimleri ele alınacak ve kısaca disk olarak anılacaktır.

Bilindiği üzere, disk birimleri, tüm giriş/çıkış birimlerinde olduğu gibi, arabirim ve sürücülerden oluşur. Disk sürücüleri bir ya da birkaç dönen plaka, okuma-yazma kafaları ve bunları denetleyen elektronik ve elektromekanik aksamdan oluşmaktadır. Veriler diskte plakaların yüzeylerindeki mıknatıslı ortama, iz olarak adlandırılan içiçe çemberler biçiminde yazılıp okunmaktadır. İzler de kendi içlerinde sektör olarak anılan alt kesimlere ayrılmaktadır.

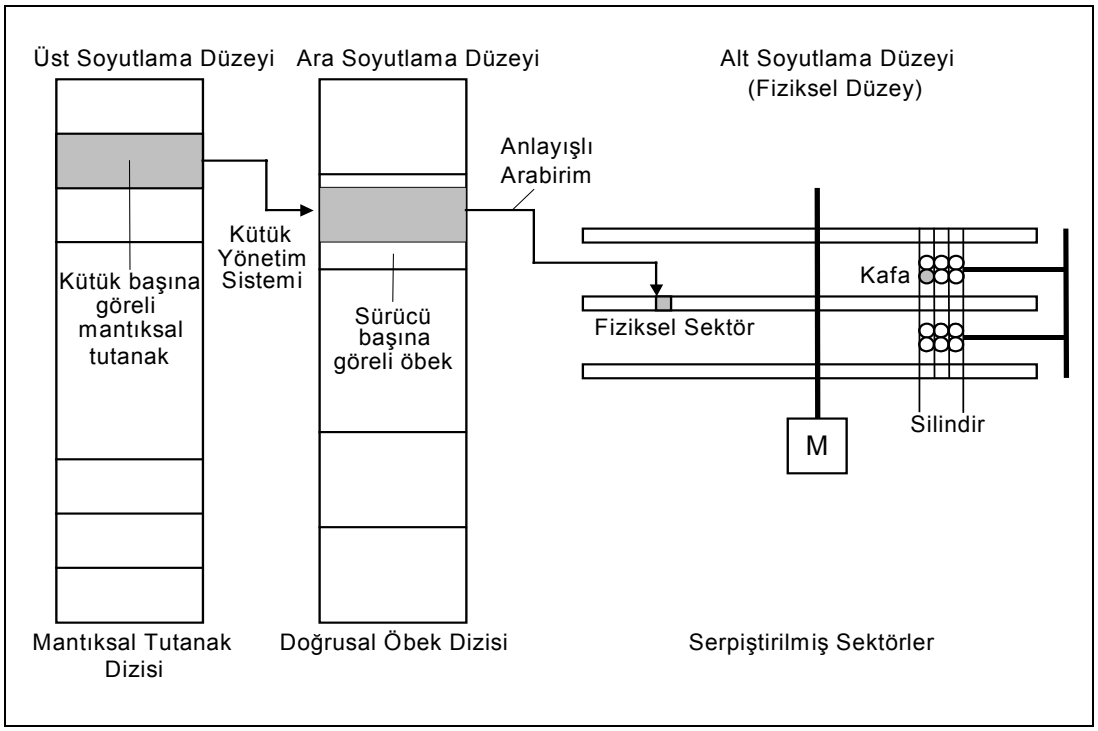
Plakalar üzerindeki verilerin okunup yazılması, okuma-yazma kafaları aracılığıyla gerçekleşmektedir. Disk birimlerinde, genellikle birden çok kafa bulunmaktadır. Okuma-yazma kafaları, hareketli ve hareketsiz olmak üzere iki biçimde olabilmektedir. Plakalar üzerinde her ize bir kafa öngörülmesi durumunda, kafaların hareket etmesine gerek duyulmaz. Bu tür disk sistemlerinde okuma-yazma kafaları hareketsizdir. Her yüzeye ilişkin kafa sayısının iz sayısından küçük olması durumunda ise kafaların izler arasında, yatay olarak hareket etmesi gerekir. Çoğu disk sisteminde, ekonomik nedenlerle kafa sayısı iz sayısının çok altındadır. Bu sistemler hareketli kafalara sahip sistemlerdir. Disket birimlerinde, örneğin kafa sayısı, her yüzeyde bir tane olmak üzere ikidir. Okuma-yazma kafaları altında kalan, değişik yüzeylere ilişkin izlerin oluşturduğu bütün silindir olarak adlandırılır. Disk birimi ile ana bellek arasında bir seferde aktarılabilen veri birimi öbek olarak adlandırılır ve çoğu kez, ikinin katları olarak birden çok sektör içerir. (Çizim 6.5).

Disk sürücüleri, ana işlem birimi ve ana bellek ikilisiyle, arabirimler ya da kanallar aracılığıyla bütünleşmektedir. Arabirim ve kanallar, genelde bu ikili ile aynı coğrafi ortamda bulunurken sürücüler sistemin uzağında (dışında) yer alabilmektedir. Ancak standart kişisel ve iş istasyonu türü bilgisayar sistemlerinin yaygınlaşmasına koşut olarak, hem sürücü ve hem de arabirimin, birlikte ve sistemin uzağında yer aldığı yapılar da ortaya çıkmıştır. Bu yapılarda sürücü-arabirim ikilisi sisteme *SCSI* (*Small-Computer-System-Interface*) gibi standart bağlantılarla bağlanmaktadır. Bu gibi sistemlerde, arabirim düzeyinde *cache* olarak adlandırılan, çoğu kez birkaç izlik veriyi saklayabilen bellek birimleri de bulunmaktadır. Disk sürücüden, örneğin okuma işlemlerinde, okunmak istenen veri öbeği, arabirim tarafından önce bu bellekte aranmakta bulunamazsa diske erişim yapılmaktadır. Ortalama erişim süreleri, bu bellek birimi sayesinde, büyük oranlarda düşürülebilmektedir.

Bilindiği gibi, diskte herhangi bir sektör içeriğine erişim süresi, ilgili okuma-yazma kafasının sektörün bulunduğu iz üzerine taşınma süresi, sektörün kafa altından geçmesinin beklendiği süre ve verinin sürücüden ana belleğe aktarım süresi toplanarak bulunmaktadır. Bu sürelerden ilki yatay erişim (ize erişim) süresi, ikincisi döngüsel bekleme (sektöre erişim) süresi ve sonuncu da aktarım süresi olarak adlandırılır. Yatay erişim ve döngüsel bekleme sürelerini tam olarak hesaplama olanağı bulunmadığından ortalama değerlerle yetinilir. Arabirim düzeyinde büyük sığalı yastık bellek alanları bulundurmak, doğal olarak bu ortalama değerlerin, önemli oranlarda düşürülebilmesine olanak sağlamaktadır.

Sektörlerin sürücü üzerindeki fiziksel adresleri silindir-kafa-sektör üçlüsü ile belirlenir. Değişik yüzeylere dağılmış kafaların altındaki izler bütünü silindiri oluşturduğundan

belirli bir silindire göreli kafa numarası, erişilen yüzey ve izi birlikte belirler. İze göreli sektör numarası da, doğal olarak erişilmek istenen sektör numarasıdır. Bilindiği gibi, bir bilgisayar sisteminde kütükler, değişik soyutlama düzeylerinde ele alınırlar. Bu soyutlama düzeylerinden en üstte olanı kullanıcılar tarafından kullanılanıdır. Yukarıda açıklanan kavramsal düzey kütük işlemlerinin taban aldığı soyutlama düzeyi, bu ilk düzeydir. Bu düzeyde kütükler, genellikle saklandıkları ortam ve sistemden bağımsız olarak düşünülürler. Kullanıcılar için kütüklerin herbiri doğrusal bir tutanak, hatta bayt dizisidir. Kütükleri oluşturan ögelerin (tutanak ya da baytların) adresleri mantıksal olup kütük başına görelidir. Mantıksal adreslerle tanımlı her kütük ögesinin, saklandığı sektör itibarıyla, bir de fiziksel adresi bulunur. Fiziksel adresler kütüğün saklandığı sistem ve sürücü kimliği ile sürücü başına göreli silindir-kafa-sektör üçlüsünden oluşur.



Çizim 6.6. Kütüklere ilişkin Değişik Soyutlama Düzeyleri

Sürücü üzerinde bir öbeğin okunması, önce ilgili okuma - yazma kafasının ilgili ize getirilmesini sonra da ilgili sektörlerin arabirim veri yastığına aktarılmasını gerektirir. Bu alt işlemler, giriş/çıkışların programlanmasıyla ilgili konu başlığı altında açıklandığı gibi, sürücü yazılımlar tarafından gerçekleştirilir. Bu yazılımlar, silindir-kafa-sektör üçlüsü ile başlangıç adresi verilen öbeğin ana belleğe aktarılmasını sağlarlar. Kütük ögelerinin silindir-kafa-sektör üçlüsü ile tanımlandığı düzey, alt soyutlama düzeyidir. Kütük yönetim sistemi, soyutlama düzeyleri arasındaki geçişi sağlamakla yükümlüdür. Soyutlama düzeyleri arasındaki geçiş, her zaman kütük başına göreli mantıksal tutanak adreslerinden, sürücüye göreli silindir - kafa - sektör üçlüsüne geçiş biçiminde olmayabilir. Kimi disk birimleri, anlayışlı arabirimler içermeleri sayesinde, bağlı oldukları bilgisayar sistemine, disk birimini, fiziksel yapısının ayrıntılarından

soyutlayarak, doğrusal bir öbek dizisi olarak sunabilmektedirler. Bir başka deyişle, sürücü başına göreli öbek adreslerinden silindir - kafa - sektör üçlüsünün oluşturduğu fiziksel adreslere geçiş, disk birimince bizzat gerçekleştirilmektedir. Bu durumda kütük yönetim sisteminin işlevi yalınlaşarak üst soyutlama düzeyinden ara soyutlama düzeyine geçişle sınırlı kalmaktadır. *SCSI* standartındaki disk birimleri, bu tür ara soyutlamaya olanak sağlamaktadır. Şimdiye değin sözkonusu edilen değişik soyutlama düzeyleri, Çizelge 6.5'de, her düzeye ilişkin nesneler ile bunlar üzerinde işlem yapan komut, çağrı ve işlevlerle birlikte özetlenmiştir.

Çizelge 6.5. Kütük Yönetim Sistemine ilişkin Soyutlama Düzeyleri

Soyutlama düzeyi	*Kullanıcı sınıfı*	*Nesneler-Kimlikleri*	*Komut, Çağrı, İşlevler*
1	Uç Kullanıcılar	Kütükler- Kütük Adı	`create, delete, rename, copy, list, type, cd`
2	Derleyiciler, Sistem Programcıları	Mantıksal Tutanak - Kütük başına göreli adres	`open, close, read, write, seek`
3	Kütük Yönetim Sistemi	Öbek - Sürücü başına göreli doğrusal adres	`seek, read, write`
4	Sürücü Yordamlar	Öbek - (Silindir, Kafa, Sektör) üçlüsü	`seek, read, write`

6.3. Kütük Yönetim Sisteminin Ele Alınışı

Kütük yönetim sisteminin, kavramsal kullanım modeli çerçevesinde yerine getirebildiği işlevler açıklandıktan sonra, şimdi, bunların işletim sistemi tarafından nasıl ele alındığı incelenecektir. Bu kapsamda, önce kılavuz kütükler incelenecek, bunlar aracılığıyla diskin değişik yüzey ve izlerine serpiştirilmiş kütük ögelerine nasıl erişildiği açıklanacaktır. Bu açıklamalar verilirken, çoğunlukla *MS-DOS*, *UNIX* gibi yaygın kullanılan işletim sistemlerinin kullandığı yaklaşımlardan yararlanılacaktır. Bunun sonrasında disk sektörlerinin kütüklere nasıl atandığı, boş sektörlerin nasıl izlendiği ve bu bağlamda disk sektörlerinin en verimli kullanımının nasıl sağlandığı konularına yer verilecektir.

6.3.1. Kılavuz Kütüklerin Ele Alınışı

Disk biriminden bir seferde okunup yazılabilen veri birimi öbek olarak adlandırılır ve çoğu kez birden çok sektör içerir. Kütükleri oluşturan mantıksal tutanaklar, disk üzerine serpiştirilmiş bu fiziksel öbekler içinde saklanır. Kılavuz kütükler, kullanıcılar

tarafından verilen simgesel kütük kimlikleri ile bu kütüklerin fiziksel öbeklerini ilişkilendirmeye yarayan yapılardır. Kılavuz kütükler, bu bağlamda kütük adları ve bunlarla ilgili fiziksel öbek adresleri listesi olarak düşünülebilir. Bu listede her kütüğe ilişkin bir satır yer alır. Her satırda, ilgili kütük adı ve fiziksel öbek adreslerinin yanı sıra kütük türü, öznitelik bilgileri, yaratılma ve günlenme tarihleri gibi bilgiler bulunur. Kullanıcılar, sistem komutlarını kullanarak, sistemde saklanan kütükleri görüntülemek istediklerinde bu kılavuz kütüğün dökümünü alırlar. Bu dökümde, doğal olarak, öbek adresleri gibi kullanıcıyı ilgilendirmeyen fiziksel erişim bilgileri yer almaz. Bu bilgiler, kütüklere fiziksel olarak erişmek gerektiğinde sistem tarafından kullanılır. Örnek bir kılavuz kütük satır içeriği aşağıda verilmiştir.

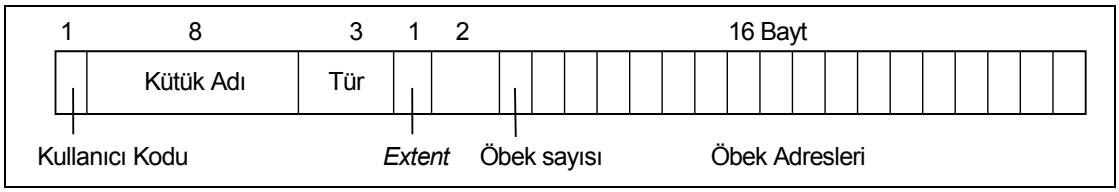
Kütük Adı	Kütük Türü	Öznitelik Bilgisi	Yaratılma Tarihi	Günlenme Tarihi	**Fiziksel Öbek Adresleri ya da İlk Öbek Göstergesi**

Bu bilgiler arasında kılavuz kütüğün temel işleviyle doğrudan ilişkili olanlar kütük adı ile fiziksel öbek adresleri ya da ilk öbek göstergesidir. Kütük adı, kullanıcının, aynı kılavuz kütük üzerinden erişilen diğer kütük adlarıyla çelişmemek koşuluyla, özgürce belirlediği, `örnek1, program` gibi simgesel bir addır. İşletim sistemleri kütüklere verilen adların içerdiği damga tür ve sayılarına, genelde, "kütük adı en çok 8 damgadan oluşmalıdır", "ilk damga, alfasayısal olmak zorundadır" gibi, kısıtlayıcı kurallar getirirler. Kütük türü kütük adının bir ekidir. Bu ek bilgiden kullanıcılar ve işletim sistemi, kütükleri sınıflandırmak, gruplandırmak için yararlanırlar. Kütüğün, örneğin ikili ya da *ASCII* türü bilgi içermesine göre, `bin, txt` gibi tür bilgileri, kütük adına eklenir. Kütük öznitelik bilgileri, çoğunlukla erişim haklarıyla ilgilidir. Kütüğe hangi grup kullanıcının hangi haklarla erişim yapabileceği, öznitelik kodu ile belirlenir. Bir kütüğe, örneğin sahibinin günleme, bunun yer aldığı kullanıcı grubunun salt okuma, diğer kullanıcıların ise salt işletim amacıyla erişim yapabileceği bilgisi öznitelik kodu içinde yer alır. Kılavuz kütük içinde kütükle ilgili kimi tarih bilgileri, özellikle karşılaştırma ve yedekleme işlemleri için bulundurulur.

Kütüğün, fiziksel öbek adresleri, kılavuz kütük içinde değişik biçimlerde yer alabilir. Bunlardan en yalını, bu adreslere, kılavuz satırı içinde ardarda yer vermektir. Örneğin, `personel` adlı bir kütük, sürücünün, `a, b, c, d, e` adresli 5 öbeğini kaplıyorsa, bu adreslerin tümüne, aşağıda gösterildiği biçimde kılavuz satırı içinde yer verilir:

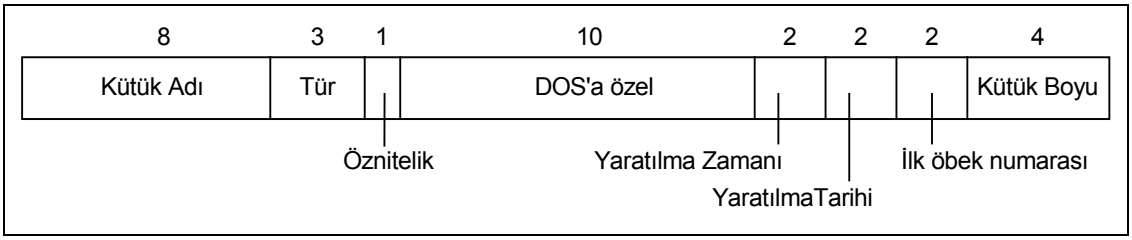
personel					a	b	c	d	e

Bir sistemde saklanan kütükler değişik boylarda olabileceklerinden, bu yaklaşıma göre, her kütükle ilgili kılavuz satırı da değişik uzunlukta olacaktır. Kılavuz kütüğü oluşturan satırların eşit ve benimsenebilir bir uzunlukta olması gerekir. Bu kısıtlama öbek adresleri alt alanına, dolayısıyla kütük boylarına önemli bir sınırlama getirir. Bu sınırlamayı aşmak üzere aynı kütük için birden çok kılavuz kütük satırı kullanılabilir. Ancak bu durumda, kılavuz kütük satırlarının, ilk satır ve ek satırlar olarak kodlanması gerekir. *MS-DOS* işletim sisteminin atası sayılan *CP/M* (*Control Program for Microcomputers*) işletim sisteminde buna benzer bir yaklaşım kullanılmıştır.



Çizim 6.7. *CP/M* Kılavuz Kütük Satırı Görünümü

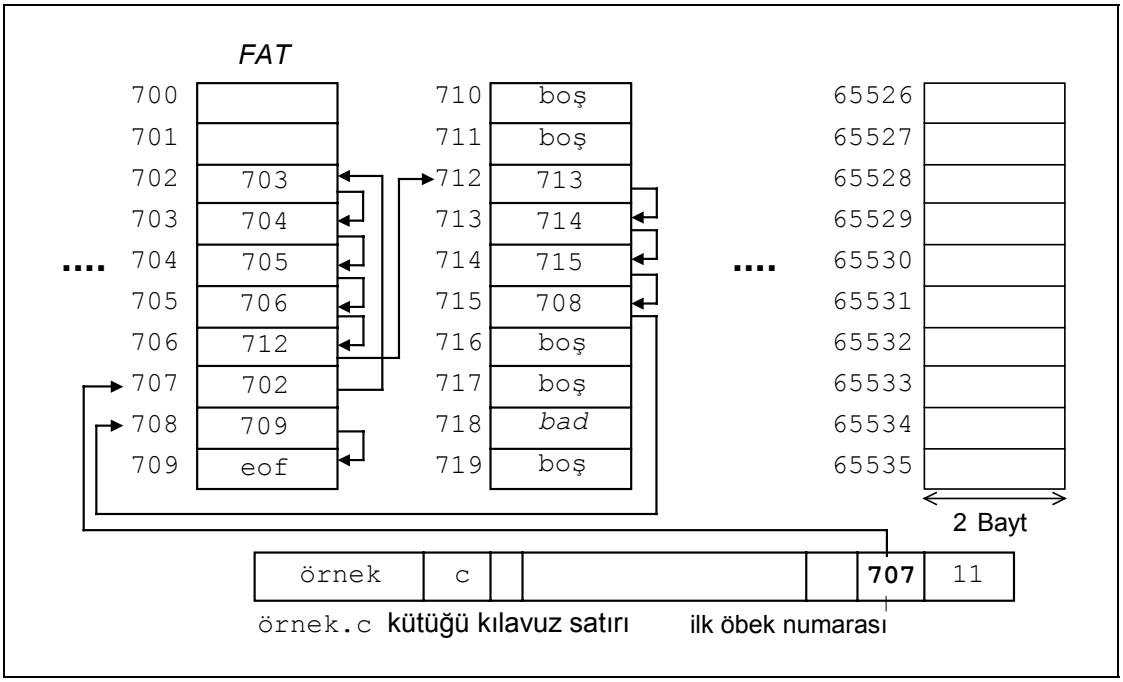
CP/M'in kullandığı kılavuz kütük satır uzunluğu 32 bayttır. Her kütük satırında 16 bayt disk öbek adreslerini tutmak için öngörülmüştür. İlk satır - ek satır kodlaması, *extent* adlı bir alt alan üzerinde yapılmaktadır. Bu alt alan sayesinde, 16 baytla kodlanabilen öbek sayısından daha çok öbeğe sahip bir kütüğün birden çok kılavuz kütük satırı bulunabilmekte ve *extent* alt alanında bunların sıra numarası tutulmaktadır. Mikrobilgisayarlar için öncü bir işletim sistemi olan ve tek düzeyli bir kılavuz kütük yapısına sahip *CP/M*'in kullandığı bu model çok kullanışlı ve esnek bir model değildir. Mikrobilgisayar işletim sistemi olarak *CP/M*'i izleyen *MS-DOS*'ta bu yapı ve yaklaşım kullanılmamıştır. *MS-DOS*'ta kullanılan yapıda, kılavuz kütük satırı içinde, kütüğe ilişkin tüm öbek adreslerini tutmak yerine bu öbek adreslerinin tutulduğu veri yapısını gösteren ve ilk öbek numarası olarak anılan bir gösterge tutulur (Çizim 6.8). Bu veri yapısı *FAT File Allocation Table,* kütük atama çizelgesi olarak adlandırılır.



Çizim 6.8. *MS-DOS* Kılavuz Kütük Satırı Görünümü

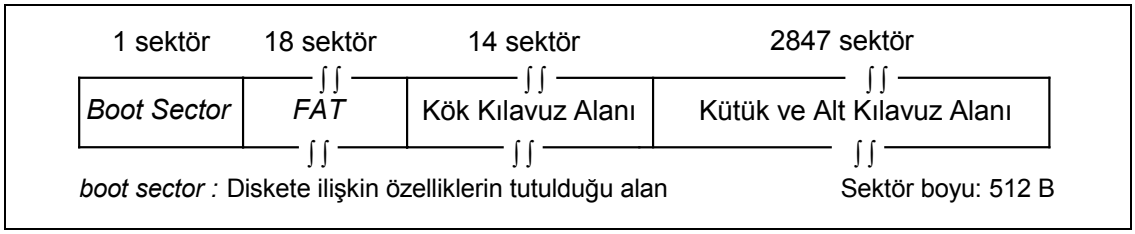
MS-DOS'ta kütükler sürücü tabanında ele alınırlar. Bir sürücüye ilişkin öbekler, kütüklere, *FAT* aracılığıyla atanırlar. *FAT,* ilgili olduğu sürücünün içerdiği öbek sayısı kadar ögeden oluşan bir dizidir. Bu dizide her öge, ilgili sürücünün bir öbeğini temsil eder. Her öge tüm öbekleri kodlayabilmeye olanak verecek sayıda bitten oluşur. Örneğin *MS-DOS*.3'te *FAT* öge uzunluğu 16 bittir. *MS-DOS* kılavuz kütük satırındaki ilk öbek numarası, *FAT*'ın bir ögesinin numarasıdır. *FAT*'ta bu numaraya karşı gelen öge ile başlayan bağlı liste, kütüğün diğer öbek numaralarını gösterir. Liste sonu, `eof`, kütük sonu damgasıyla belirlenir. Çizim 6.11'de `örnek1.c` adlı, ilk öbek numarası 707, son öbek numarası da 709 olan, 12 öbeklik bir kütüğün erişim zinciri gösterilmiştir. Çizim, öge boyunun iki bayt (16bit) olması dolayısıyla *FAT*'ın en çok 65536 öge içerebileceği düşünülerek çizilmiştir. Ancak disk birimlerinin sığalarına ve kullanılan öbek boyuna bağımlı olarak *FAT*'ın boyu, her sürücü için değişik olabilmektedir. Örneğin 1.44 MB sığa ve 512 Bayt (0.5KB) öbek boyuna sahip standart disketler için *FAT,* 1440 / 0.5 = 2880 ögeden oluşur. Bu durumda *FAT*'ın boyu 2880 x (12/8) = 4320; yaklaşık 4 KB olur. Burada *FAT* öge boyu 12 bit olarak alınmıştır.

Kılavuz kütükler, ilgili oldukları sürücüler üzerinde saklanırlar. Bu ilke, sürücülerin sistemden sisteme taşınmasına, geçici olarak işletim dışında tutulabilmelerine olanak verir. Bilindiği gibi kılavuz kütükler, diğer kütüklere erişimde yararlanılan, bu kütüklerin fiziksel öbek adreslerinin izini tutan kütüklerdir. Bu nedenle, herhangi bir kütüğe erişmeden önce kılavuz kütüğe erişmek gerekir. Kütüklerle ilgili işlemlerde, diske erişim sayısını artırmamak için, üzerinde işlem yapılan kütüğün kılavuz kütük satırı, daha önce açıklanan `open` işlemi aracılığıyla ana belleğe taşınır. `close` işlemi ile kütük üzerinde yapılan işlemlerin son bulduğu bilgisi sisteme aktarılana değin de ana bellekte tutulur. Bu gerekçeye dayalı olarak, *MS-DOS* kapsamında kılavuz kütüklerle birlikte kullanılan *FAT* 'ın da ana bellekte tutulması gerekir. *FAT* kılavuz kütükler gibi ilgili sürücü ortamında saklanan ancak işletimin başında ana belleğe yüklenen bir dizidir.



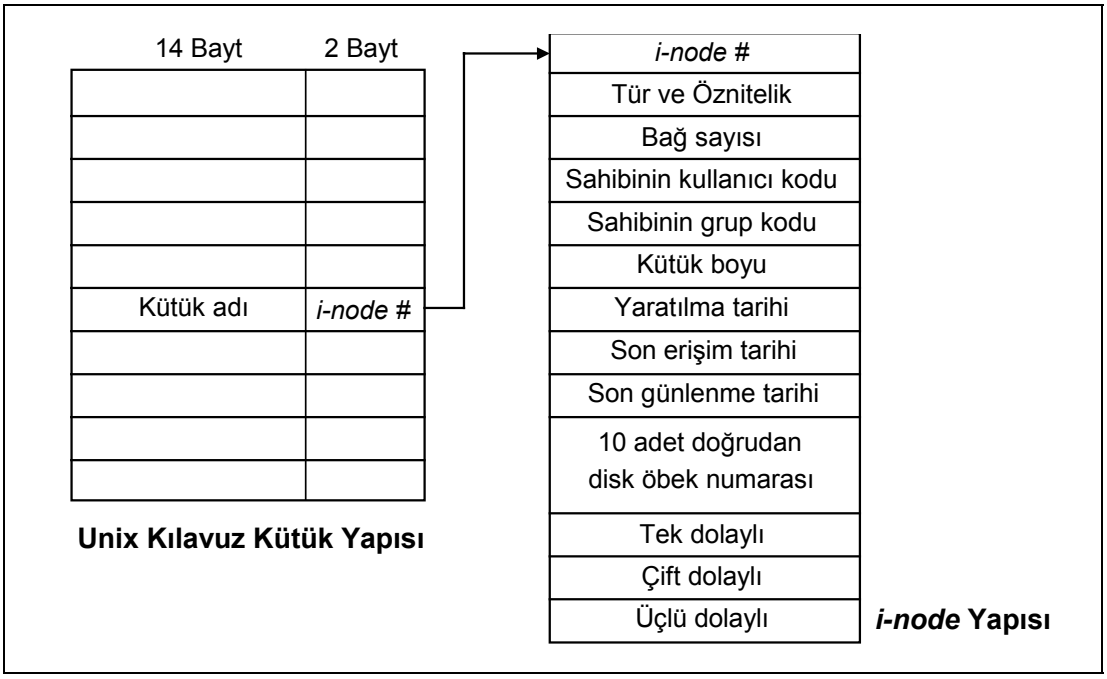
Çizim 6.9. *MS-DOS*'ta Kütük Atama Çizelgesi - *FAT*

Çok düzeyli kılavuz kütük yapısını kullanan kütük yönetim sistemlerinde kılavuz kütükler kök kılavuz ve alt kılavuzlar olmak üzere iki kategoriye ayrılır. Genellikle sıradan kütükler gibi ele alınan alt kılavuz kütüklere kök kılavuz aracılığıyla erişilir. Kök kılavuzun yer aldığı disk öbeklerine nasıl erişileceğini gösteren bir yapı bulunmadığından kök kılavuzların ilgili oldukları sürücü üzerindeki konumları değişmezdir. Bu bağlamda, *MS-DOS*, 1.44 MB'lık disketlerde, kök kılavuzu 19 uncu sektörden başlayarak, en çok 14 sektörlük yer kaplayacak biçimde yaratır. *MS-DOS* kapsamında disklerin genel düzenleniş biçimi, 1.44 MB'lık disket birimi ile Çizim 6.10'da örneklenmiştir. Kök kılavuzlar, sürücünün formatlanması (kullanıma ilk hazırlanışı) sırasında yaratılırlar.



Çizim 6.10. *MS-DOS*'ta 1.44 MB Disket Birimlerinin Düzenleniş Biçimi

Fiziksel öbeklere erişimde, *FAT* türü bir veri yapısının kullanılmasının önemli sakıncaları bulunur. Bunlardan en önemlisi bu veri yapısının boyutuyla ilgilidir. *FAT*, ana bellekte saklanması itibarıyla boyutları sınırlı kalması gereken bir veri yapısıdır. Öbek boyu 1 KB olan 32 MB sığasındaki bir disk sürücü için gerekli *FAT* uzunluğu, 16 bitlik *FAT* ögeleri için 64 KB'dır. Bu uzunluk, öbek boyu 4 KB olan 2 GB'lık bir disk sürücü için ise, 24 bitlik *FAT* ögeleri ile 1,5 MB olur. Sürücü sığasına bağımlı olarak daha da büyüyecek bu yapıların ana bellekte saklanabilmesi, belirli bir büyüklüğün üstünde düşünülemez. Bu nedenle, büyük sığada disk birimleri içeren sistemlerde, *FAT* ilkesine dayalı kılavuz kütük yapıları kullanılamaz.



Çizim 6.11. *UNIX*'te *i-node* ve Kılavuz Kütük Görünümü

UNIX işletim sisteminde, simgesel kütük adı ile bu kütüğün fiziksel disk öbeklerinin ilişkilendirilmesi *i-node* adlı yapılar aracılığıyla gerçekleştirilir. *i-node, FAT*'ın işlevini sistemde saklanan her kütük için, ayrı ayrı yerine getiren bir çizelgedir. *UNIX* işletim sisteminde kök kılavuz, sürücülerden bağımsız olarak düşünülür ve öyle ele alınır. Bu nedenle, kök kılavuz sistem için biriciktir. *UNIX* işletim sisteminde, giriş/çıkış bağımsızlığı ilkesi çerçevesinde, giriş/çıkış sürücüleri kütükler gibi düşünülür ve

herbirinin bir kütük kimliği bulunur. Herhangi bir sürücüye erişmek gerektiğinde, ilgili kütükten okuma-yazma yapılıyormuş gibi davranılır. Sisteme yeni bir sürücünün eklenmesi ya da çıkarılması durumunda da, bu sürücülere karşı gelen kütük adlarının, `mount, dismount` gibi sistem komutları ile kök kılavuza eklenmeleri ya da buradan çıkarılmaları gerçekleştirilir. Bu bağlamda, *UNIX*'te kılavuz kütükler kütük adı ve *i-node* numarasını içeren satırlardan oluşurlar (Çizim 6.11).

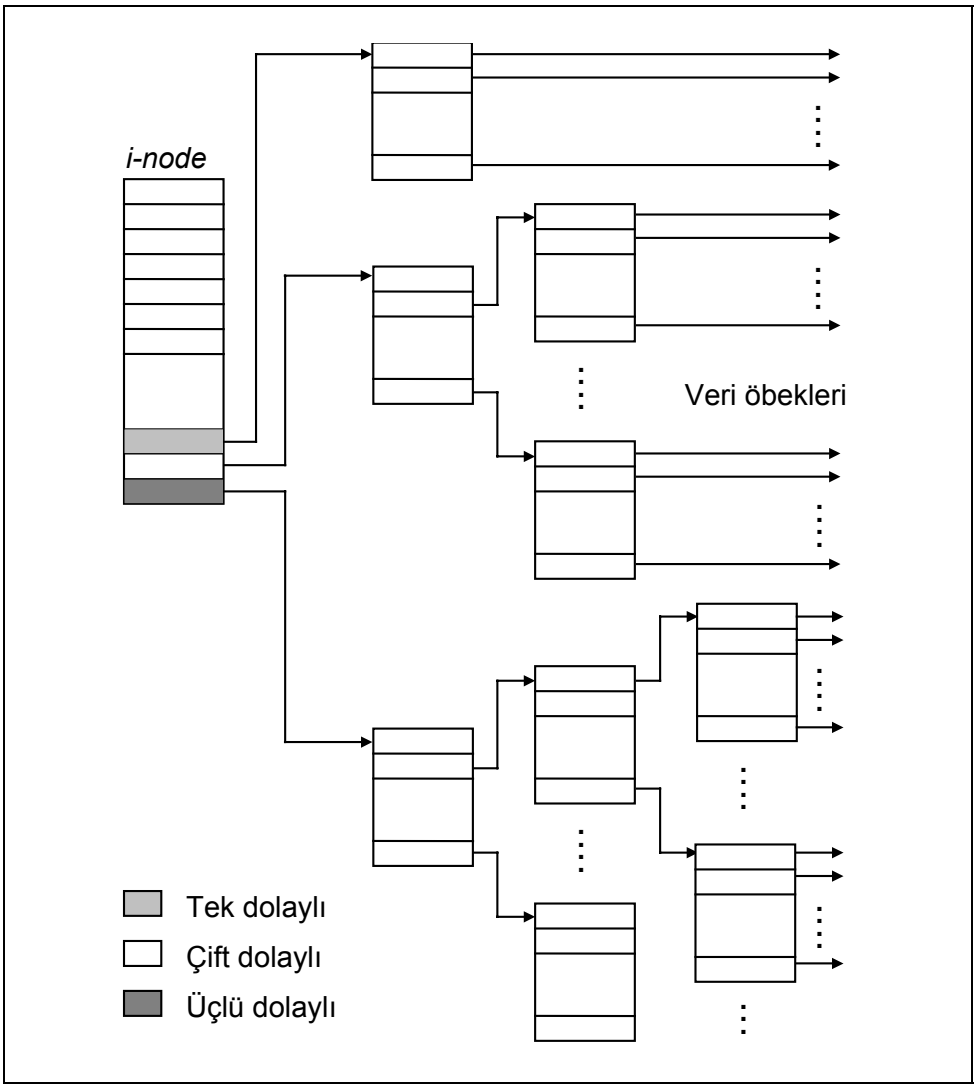
Kütük adının, kütüğün saklandığı disk öbek adresleri ile ilişkisi, *i-node* içinde bu amaçla öngörülen alt alanlar aracılığıyla kurulur. *i-node* içinde ilgili kütüğün öbek adreslerini tutmak üzere 4 değişik alt alan bulunur. Bu alt alanlardan ilki, ilk 10 öbek adresini tutar. Eğer bir kütük 10 ya da daha az öbek içeriyorsa bu öbek adresleri *i-node* içinde doğrudan yer alır. Diğer alt alanlar kullanılmaz. Kütük 10'dan daha çok öbeğe sahipse, ilk onun dışındaki öbek adresleri, ikinci alt alanda adresi bulunan öbek içinde saklanıyor demektir. Bu öbeklere erişim dolaylı olarak gerçekleşir. Bu bağlamda, eğer öbek adresleri 32 bit (4 bayt) ile kodlanıyorsa 2 KB'lık bir öbek içinde 512 dolaylı öbek adresi saklanabilir. Bu örnek çerçevesinde, ilk ve ikinci alt alanların kullanılmasıyla, toplam $10 + 512 = 522$ öbeği adreslemek olanaklıdır. İlk ve ikinci alt alanların kullanılması kütüğün tüm öbek adreslerini tutmakta yetersiz kalırsa, bu kez üçüncü alt alan kullanılır. Üçüncü alt alanda yer alan öbek, diğer kütük öbek adreslerini tutan öbeklerin adreslerini içerir. Yukarıda verilen varsayımlara göre, üçüncü alt alanın kullanılması durumunda, toplam $10 + 512 + 512^2$ öbek adreslenebilir. Bunun da yetersiz kalması durumunda dördüncü alt alan kullanılır. Bu durumda, yine yukarıdaki örneğe göre, en çok $10 + 512 + 512^2 + 512^3$ öbek adreslenebilir. *UNIX* kapsamında bir kütüğün içerebileceği en büyük öbek sayısı $[10 + \alpha + \alpha^2 + \alpha^3]$ olur[24]. Burada α öbek boyunun öbek adres boyuna oranıdır. Yukarıdaki örnekte bu oran 2048 / 4 = 512 olarak bulunmuştur (Çizim 6.12).

i-node içerisinde, ilgili kütüğün fiziksel öbek adreslerinin yanı sıra başka bilgilere de yer verilir. Bu bilgiler, kütük sahibinin kullanıcı ve grup kodu, kütük boyu, yaratılma, son erişim ve son günlenme tarihleri ile bağ sayacıdır. Bağ sayacı, kütüğe kaç alt kılavuzdan erişildiğini tutan bir sayaçtır. *UNIX*'de, kılavuz kütük için, döngüsüz yönlü çizge yapısı kullanılarak kütüklerin birden çok alt kılavuz altında yer almalarına olanak sağlanır. Bununla, kütüklerin kullanıcılar arasında ortaklaşa kullanılabilmesi amaçlanır. Birden çok alt kılavuz altında yer alan bir kütüğün, sistemde tek bir kopyası saklanır. Kütüğün *i-node* yapısı da biriciktir. Kütüğün yer aldığı değişik alt kılavuzların herbiri aynı *i-node* yapısını gösterir. Bu yapı doğal olarak kimi sorunları da beraberinde getirir. Bu sorunlardan en önemlisi kütüğün silinmesinin ortaya çıkaracağı sorundur. Kütüğün bir alt kılavuz altından silinmesi diğer alt kılavuzlar altındaki varlığını etkilememelidir. Bu amaçla, *i-node* içinde, bağ sayacı olarak adlandırılan bir alt alan öngörülür. Bağ sayaç içeriği, kütük yaratıldığında birdir. Kütüğün başka bir alt kılavuz altında

---

[24] En büyük kütük boyuyla ilgili bir diğer kısıtlama da öbek adreslerini kodlamak üzere öngörülen bit sayısından gelir. Bu örnekte öbek adreslerinin 32 bit ile kodlandığı varsayılmıştır. Bu, en büyük öbek sayısını 2^{32}, en büyük kütük boyunu da 2^{32} x 2 KB = 8 TB ile sınırlar. Ancak bu sınırlama $10 + 512 + 512^2 + 512^3$ değerinin çok ötesindedir.

gösterilme işlemi sonrasında bu içerik bir artırılır. Kütüğün herhangi bir alt kılavuz altından silinmesi durumunda da bu içerik bir eksiltilir. Kütüğün sistemdeki varlığı, ancak bağ sayısı birken silinmesi durumunda son bulur.

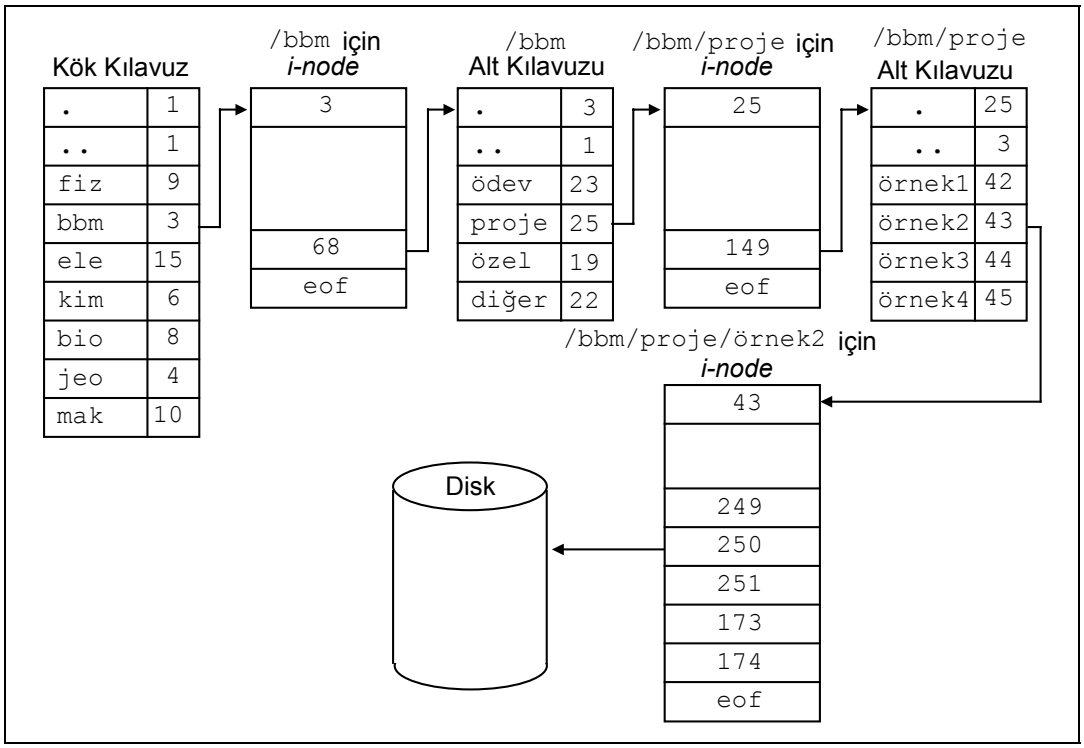


Çizim 6.12. *UNIX*'te Fiziksel Disk Öbeklerine Erişim Düzeneği

UNIX'in *i-node* içinde tuttuğu bağ sayacı ile gerçekleştirdiğini, başka işletim sistemleri simgesel bağ olarak anılan başka bir yöntemle gerçekleştirirler. Simgesel bağ yönteminde bir kütüğün bir başka alt kılavuz altında da görünmesi (bağ kurulması) istendiğinde, bu alt kılavuz altında, işletim sistemi tarafından, kütüğün erişim izini içeren *link* türü yeni bir kütük yaratılır. Kütüğe bu alt kılavuz altından erişmek gerektiğinde, kütük izini içeren özel kütük okunur. Elde edilen iz kullanılarak kütüğe erişim sağlanır. Doğal olarak kütük izini içeren *link* türü özel kütüğün yaratılması, kütük izini kullanarak erişimin sağlanması işlemleri, işletim sistemi tarafından kendiliğinden yerine getirilen, kullanıcıya saydam işlemlerdir. Simgesel bağ yönteminde kütük silme

işlemlerini denetleme olanağı bulunmaz. Birden çok alt kılavuz altında görünen bir kütüğün sahibi tarafından silinmesi sonrasında, diğer alt kılavuzlardan, kütük izi kullanılarak yapılacak erişimler, olmayan kütük uyarısı ile sonlanırlar.

i-node içerisindeki kullanıcı ve grup kodları ile öznitelik bilgilerinden kütüğe erişim haklarının denetlenmesi amacıyla yararlanılır. Yaratılma, son erişim ve son günlenme tarihleri, çoğunlukla kütüklerin sistemli yedeklenmesi, yer açma amacıyla kimi kütüklerin disklerden çıkarılması gibi işletim amaçlı işlemlerde yararlanılan bilgilerdir. Bu konulara Koruma ve Güvenlik başlığı altında ayrıntılı olarak yer verilecektir.



Çizim 6.13. *UNIX*'te Kütük Kimliğinden Fiziksel Disk Öbek Adreslerine Geçiş

Çizim 6.13'te, *i-node* ve kılavuz kütük yapıları kullanılarak, kimliği verilen (`/bbm/proje/örnek2` adlı) bir kütüğe nasıl erişildiği örneklenmiş ve bu yolla, *UNIX* kapsamında simgesel kütük kimliğinden fiziksel öbek adreslerine geçiş düzeneği özetlenmiştir. Bu örneğe göre, önce `/bbm` alt kılavuzuna erişim sağlanmaktadır. Bunun için, kök kılavuzdan `/bbm` alt kılavuzuna ilişkin *i-node* numarası elde edilmekte, bu *i-node* yapısına erişilerek /bbm kütüğünün 68 inci fiziksel disk öbeğinde bulunduğu belirlenmektedir. Bunun sonrasında 68 inci disk öbeğinden /bbm alt kılavuz kütüğü okunmaktadır. İkinci aşamada, aynı /bbm alt kılavuzu için olduğu gibi, `/proje` alt kılavuz okunmaktadır. Bu kütüğün *i-node* numarasının 25, bulunduğu disk öbeğinin ise 149 uncu öbek olduğu belirlenmektedir. 149 uncu öbekten, `örnek2` kütüğünün 43 numaralı *i-node* yapısını kullandığı anlaşılmaktadır. 43 numaralı *i-node* yapısından, `örnek2` kütüğünün `249, 250, 251, 173, 174` numaralı 5 disk öbeğini kapladığı görülmektedir.

6.3.2. Kütüklere Diskte Yer Atama Yöntemleri

Kütük yönetim sisteminin temel işlevlerinden bir tanesi de, sistemde disk alanlarının verimli kullanılmasının sağlanmasıdır. Kütüklere, gereksedikleri disk alanlarının atanması, bu temel işlev gözetilerek yapılır. Kütük yönetim sisteminin kütüklere yer atayabilmesi için boş disk alanlarını izlemesi de gerekir. Disk alanları genelde öbek dizisi olarak düşünülür. Öbek, ikinin katları kadar sektörden oluşan bir bütündür. Her öbeğin, disk üzerindeki konumunu belirleyen, genelde silindir, kafa ve başlangıç sektörü üçlüsünden oluşan fiziksel bir adresi bulunur. Kütüklere diskte yer atamalar öbek ya da öbek kümesi tabanında yapılır. Bu durum, sayfalı bellek yönetiminde, belleğin sayfalar dizisi biçiminde ele alınmasına ve görevlere de sayfa, sayfa atanmasına benzer.

Kütüklere diskte yer atanırken en çok gözetilen kıstas erişim hızıdır. Disk öbeklerinin kütüklere atanması aşamasında, erişim hızının en yüksek olacağı düzende atama yapılmaya çalışılır. Bir kütüğe ilişkin tüm öbeklerin aynı silindirde yer alması durumunda, bu öbeklere ardarda yapılan (sıradan) erişimlerde yatay kafa hareketi gerekmez. Kütüğe erişim hızı artar. Bunun tersine, öbekleri diskin birbirine uzak silindirlerine serpiştirilmiş bir kütükten yapılacak sıradan okuma işlem hızının da düşük olacağı açıktır. Bu gözlemlere dayalı olarak, disk öbeklerinin kütüklere elverdiğince bitişken konum ve silindirlerde atanmasına özen gösterilmesi gerekeceği söylenir[25].

Disk öbeklerinin kütüklere elverdiğince bitişken olarak atanması diskin verimli kullanımını olumsuz yönde etkiler. Disk alanlarının verimli kullanımı, kütük yönetim sisteminin gözetmesi gereken temel bir kıstastır. Boş disk alanları yeni yaratılan ya da işletim aşamasında büyüyen kütüklerce tüketilir. Silinerek sistemdeki varlığına son verilen kütüklerin serbest bıraktığı alanlar ise, yeniden boş alanlara dönüşürler. Kütüklere bitişken öbekler atama, bu süreç içerisinde, aynı ana bellekte olduğu üzere disk alanlarının parçalanmasına neden olur. Kullanılan alanların arasına sıkışmış küçük boş alanlardan yararlanılamaz. Bu, disk kullanım verimliliğini olumsuz yönde etkiler.

Kütük yönetim sistemi, bu durumda, erişim hızı ile kullanım verimliliği gibi birbiriyle çelişen iki kıstası uzlaştırmak zorunda kalır. Bu uzlaştırmada bir kıstasa, diğerine göre daha çok ağırlık veren değişik atama yöntemleri kullanılır. Bunlar:

- Bitişken ve
- Bitişken olmayan

atama yöntemleri olarak genelde ikiye ayrılır. Bitişken olmayan atama yöntemleri de, kendi içlerinde:

- Zincirli ve
- Dizinli

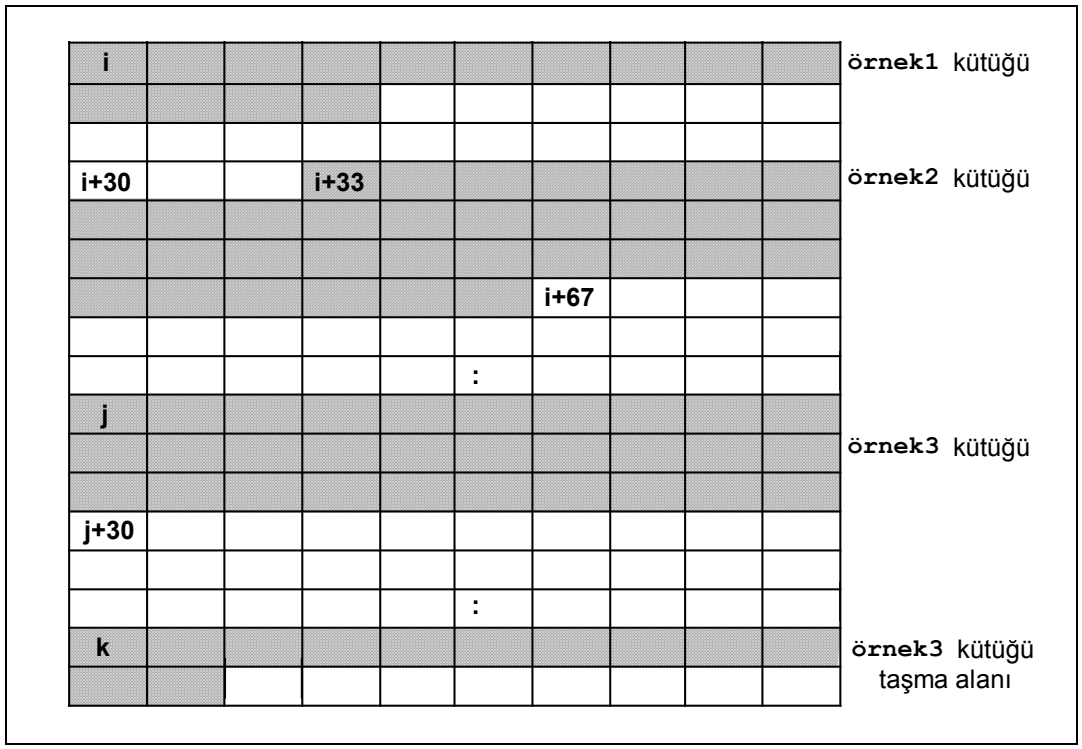
---

[25] Diskte herhangi bir öbeğe erişim hızı, okuma-yazma kafasının, o öbeğin yer aldığı silindire olan yakınlık ya da uzaklığına bağımlıdır. Ana bellekte ise, disktekinin tersine herhangi bir sözcüğe erişim hızı, sözcüğün bellek adres evreni içindeki konumundan (belleğin başında ya da sonlarında bulunmasından) bağımsızdır. Ana bellek bu nedenle *RAM* (*Random Access Memory* - rasgele erişimli) bellek olarak da adlandırılır.

olarak ikiye ayrılırlar. İzleyen kesimde bu yöntemler açıklanacak ve herbirinin olumlu ve olumsuz yönleri tartışılacaktır.

a. Bitişken Yer Atama Yöntemi

Kütüklere bitişken yer atama yönteminde bir kütüğün gereksediği tüm alan, fiziksel adresleri ardarda gelen bitişken disk öbekleriyle karşılanır. Kütük yaratılırken gereksenen toplam alan boyu belirlenerek atama yapılır. Ancak bir kütüğün gereksediği tüm disk alanının, kütüğün yaratıldığı anda saptanabilmesi her zaman olanaklı değildir. Örneğin bir metin düzenleyici aracılığıyla yazılan bir metnin ya da kaynak programın ne uzunlukta olacağını önceden kestirebilmek, ilgili kütüğün yaratılmasından da sorumlu olan metin düzenleyici yazılım için olanaksızdır. Bunun kullanıcıdan istenmesi de anlamsızdır. Toplam kütük boyunun önceden kestirilemediği durumlarda, kimi ortalama değerlere başvurulur. İşletim aşamasında büyüyerek bu ortalama değeri aşan kütükler için, ilk atanan alanla bitişken olma zorunluluğu bulunmayan taşma alanları öngörülür. Dönem dönem, diskte, işletim dışı bitiştirme işlemlerine başvurularak taşma alanları bulunan kütüklerin yeniden tümüyle bitişken biçime getirilmesi sağlanır.



Çizim 6.14. Kütüklere Diskte Bitişken Alan Atama Yöntemi

Çizim 6.14'de, sırasıyla 14, 33 ve 42 öbeğe sahip `örnek1`, `örnek2` ve `örnek3` adlı 3 kütüğün, bitişken yer atama yöntemine göre diskteki yerleşimleri örneklenmiştir. Burada, boyları önceden kestirilemeyen kütükler için 30 öbeklik bitişken bir alan atandığı varsayılmıştır. Bu bağlamda `örnek1` ve `örnek3` adlı kütüklere, başlangıçta 30 bitişken öbek atandığı, `örnek3` adlı kütüğün, işletim sırasında bu ortalama değeri

aşması nedeniyle taşma alanına sahip olduğu gösterilmiştir. `örnek2` adlı kütüğe, boyutları önceden bilindiği için, 33 öbeğin tümünün, yaratma aşamasında atandığı varsayılmıştır. Çizimi karmaşıklaştırmamak üzere fiziksel disk öbekleri, sürücü başına göreli doğrusal adresleri ile gösterilmiştir.

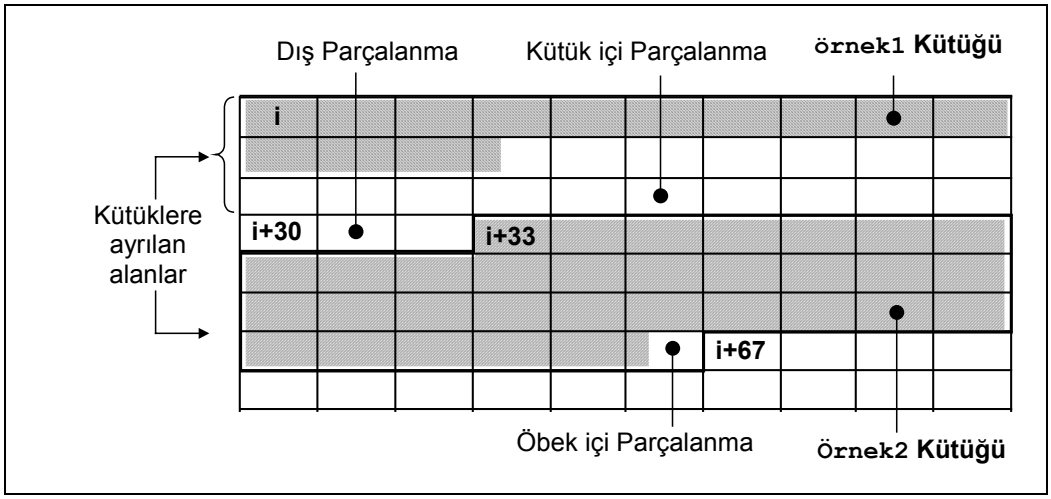
Bitişken yer atama yönteminde kütüklere yer atanırken, boş alan çizelgesi olarak adlandırılan bir çizelgeden yararlanılır. Bu çizelge diskte, kütüklere atanmış bitişken öbek dizileri arasında kalan boş alanların başlangıç öbek adreslerini ve öbek sayılarını tutar. Yeni bir kütük yaratılacağı zaman, boş alan çizelgesi taranarak yeterli büyüklükte bitişken bir alan bulunmaya çalışılır. Bu tarama, daha önce ana belleğin yönetimi kapsamında açıklanan ilk uyan (*first fit*) ya da en uygun (*best fit*) adlı algoritmalar kullanılarak yapılabilir. Bulunan alan başlangıç adresi, ilgili kütüğün kılavuz kütük satırına, öbek sayısıyla birlikte işlenir. Bitişken yer atama yönteminin kullanıldığı sistemlerde kılavuz kütük satırları kütüklerin, taşma alanları da dahil, disk alan başlangıç adreslerinin tümünü tutacak biçimde öngörülebilir. Bir kütüğün, örneğin en çok 3 taşma alanı bulunabileceği kısıtlaması getirilebilir. İlk alanla birlikte, en çok 4 bitişken alana sahip olabilecek bir kütüğün bu alan adresleri, doğrudan kılavuz satırında tutulabilir. Bu durumda, kılavuz kütükler, hangi alanın hangi kütüğe atandığı bilgisinin tutulduğu disk atama çizelgelerinin işlevini de yerine getirebilirler.

Bir kütüğün silinmesi durumunda, bu kütüğe ilişkin bitişken alan başlangıç adresi ve öbek sayısı, yeni bir öge olarak boş alan çizelgesine işlenir. Kılavuz kütüklerde silinen kütüklerle ilgili satırlar, çoğu kez yeniden kullanılabilir olduklarını gösterir biçimde damgalanırlar. Başka bir deyişle içerdikleri eski bilgileri korurlar. Bu durumda yeni yaratılacak bir kütüğe boş alan atama, kılavuz kütükler üzerindeki silinmiş kütük satırları taranarak da yapılabilir. Bu biçimde kılavuz kütükler, boş alan çizelgesinin işlevini de yüklenecek biçimde tasarlanabilirler. Başka bir anlatımla, bitişken alan ataması yapılan sistemlerde kütük yönetim sistemi, kılavuz kütük yapısıyla yetinerek kütük atama çizelgesi, boş alan çizelgesi gibi ek veri yapılarına ayrıca gereksinim duymayabilir.

Bitişken alan atama yöntemi kütüklere, gerek sıradan gerekse rasgele erişimlerde en hızlı erişimi sağlayan bir yöntemdir. Zira bu yöntemle bir kütüğe atanan fiziksel öbekler, bu kütüğün mantıksal tutanakları gibi ardarda saklanırlar. Sıradan okuma ve yazmalarda ardarda okunmak istenen mantıksal tutanaklara karşı gelen fiziksel öbekler de ardarda olduğundan disk kafa hareketleri en aza iner. Bilindiği gibi, disk arabirimleri, genelde birkaç izlik verinin arabirim düzeyinde saklanabilmesine olanak veren *cache* adlı yastık bellek birimleri içerirler. Mantıksal tutanakların ardarda fiziksel öbekler içinde saklanması, sıradan okunan bu tutanakların tümünün yastık bellekte bulunabilmesine olanak verir. Okunmak istenen tutanaklar yastık bellekten ana belleğe doğrudan aktarılıverir. Böylece disk erişim hızı olağanüstü artar. Bitişken alan atama yöntemi, öbek adresi hesaplama işlemlerinin yalınlığı sayesinde, rasgele erişimlerde de hız artışı sağlar.

Bitişken alan atama yönteminin en belirgin sakıncası diskte yarattığı parçalanmadır. Bu parçalanma, iç ve dış parçalanma olarak ikiye ayrılır. Yaratılma sırasında, boyu

hakkında kestirim yapılamayan kütüklere ortalama değerler üzerinden, belirli sayıda bitişken öbek ayrılır. Kimi kütükler, işletim aşamasında büyüyerek bu alandan taşabilirken, kimi kütükler de, kendilerine ayrılan alanın tamamını hiçbir zaman dolduramazlar. Kütük içi kullanılmayan bu kesimler, iç parçalanma kapsamında düşünülür. Dış parçalanma, kütük yaratma - silme süreci içerisinde kütüklere ayrılan alanlar arasında kalan, atanabilirlik yönünden kritik boyun altında, küçük kullanılamaz öbek dizilerinin yarattığı parçalanmadır. Bitişken alan atama yöntemi için, her iki türün yarattığı sakınca da önemlidir. Dış parçalanmanın yarattığı olumsuzluk, bitiştirme işlemi ile ortadan kaldırılmaya çalışılır. Dönem dönem, işletim dışında, kütüklere ayrılan disk alanları bitiştirilir. Bu yolla, bu alanlar arasına sıkışmış, kullanılamayan küçük parçaların bütünleşmesi ve büyük bitişken disk alanlarının elde edilmesi sağlanır. İç parçalanma sorununun genelde çözümü yoktur. Kütük içi kayıplar, ancak ilgili kütüğün silinmesi sonucu, öbek dizisinin boş alan biçimine dönüşmesi ile ortadan kalkar (Çizim 6.15).



Çizim 6.15. Disk Alanlarının Parçalanması ve Parçalanma Türleri

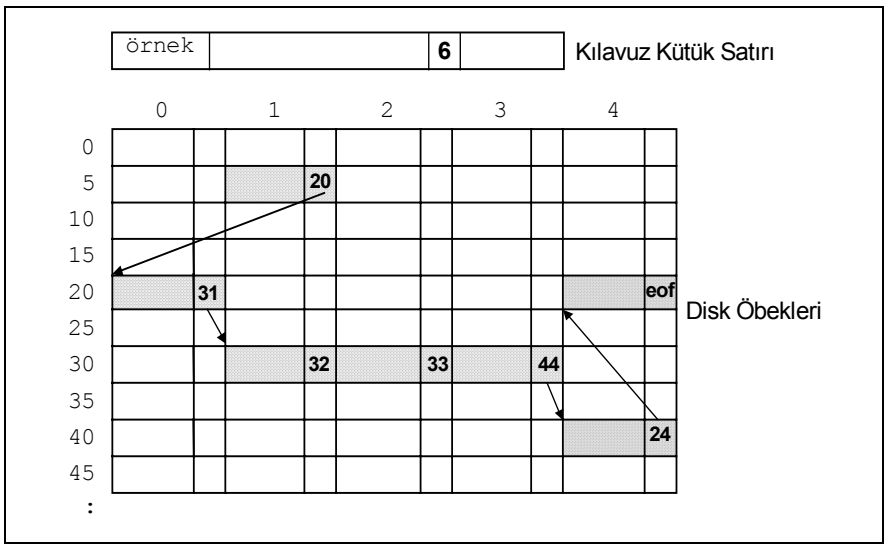
Diskte, yukarıda sözü edilen parçalanmaların dışında, yine iç parçalanma kapsamında düşünülebilecek bir diğer parçalanma daha vardır. Kütüklere atanan alanlar, genelde öbek kümelerinden oluşur. İleride incelenecek diğer yöntemler kapsamında da, kütüklere alan atamaları öbek kümesi tabanında, birim birim yapılır. Her kütüğün son öbek kümesi, doğal olarak tümüyle (sonuna kadar) doldurulamaz. Son kümedeki kullanılamayan disk alanları, küçük de olsa iç parçalanmaya kaynaklık ederler. Bu son parçalanma, türü ne olursa olsun tüm atama yöntemleri için sözkonusudur.

Diskte iç ve dış parçalanma, erişim hızını en üst düzeyde tutabilmek amacıyla kütüklere bitişken alan atama ilkesinden kaynaklanır. Bu ilkeden vazgeçmeden parçalanmaya etkili bir çözüm yolu bulma olanağı yoktur. Bitişken alan atama yönteminde de, boyu işletim aşamasında büyüyen kütükler için taşma alanlarının kullanılması yoluyla, bitişkenlik ilkesinden zaten taviz verilmektedir. Bu olumsuzluğu ortadan kaldırmak üzere düşünülen bitiştirme işlemleri ise, zamana mal olan, işletimin dönem dönem

kesilmesini gerektirerek genel başarımı düşüren işlemlerdir. Bu durumda bitişken alan atama ilkesi terkedilip erişim hızından biraz taviz verilerek parçalanmayı azaltma yoluna gidilir. Zincirli ve dizinli alan atama yöntemleri bu bağlamda düşünülür.

b. Zincirli Yer Atama Yöntemi

Zincirli yer atama yönteminde bir kütüğün sahip olduğu öbek ya da öbek kümeleri birbirlerine zincirleme olarak bağlıdır. Kütüklere atanan öbeklerin bitişken olma zorunluluğu yoktur. Her öbeğin sonunda küçük bir kesim bağ alanı olarak öngörülür. Her öbeğin bağ alanı bir sonraki öbeğin adresini tutar. Öbek zincirinin ilk öbek göstergesi kütüğün kılavuz kütük satırı içinde tutulur. Kütüğe erişimde bu göstergeden yararlanılır. Bu yöntemle, bir kütüğün *i* inci öbeğine erişebilmek için *(i-1)* inci öbeğini daha önce okumuş olmak gerekir. Bu bağlamda bir kütüğün, örneğin 6 ıncı öbeğine erişebilmek için, kılavuz satırında (dolayısıyla ana bellekte) adresi bulunan ilk öbekten başlayarak, sırayla ilk 5 öbeği okumak gerekir. Çizim 6.16'da 7 öbekten oluşan `örnek` adlı bir kütüğün, bu yönteme göre, diskteki yerleşimi verilmiştir.



Çizim 6.16. Zincirli Yönteme göre Bir Kütüğün Diskte Yerleşim Örneği

Zincirli yer atama yönteminde, kütüklere atanan öbekler gibi, diskteki boş öbekler de zincirli biçimde tutulurlar. Bir kütüğe yeni bir öbek atanacağı zaman boş öbek zincirinin ilk öbeği, kütük öbek zincirinin sonuna bağlanır. Bunun için, kütüğün son öbeğinin bağ alanı, yeni atanan öbek adresiyle, yeni atanan öbeğin bağ alanı ise son öge olarak günlenir. Bir kütüğün silinmesi durumunda, serbest kalan öbeklerin, boş öbek zincirine birer birer bağlanması gerekir. Bu, zincire bağlanan boş öbek sayısının iki katı kadar diske erişim demektir. Zira zincire bağlanan her boş öbek için, bu öbeğin okunması, bağ alanının o anki zincir göstergesi ile, zincir göstergesinin ise bu öbek adresi ile günlenmesi ve öbeğin yeniden diske yazılması gerekir. Bu sakıncalı durumu ortadan kaldırmak üzere, bir kütüğün silinmesi sonucu serbest kalan öbekleri, zinciri bozmadan, boş öbek zincirine tümden eklemenin yolları aranır. Bunun için, kütük ilk öbek

göstergesinin yanı sıra, bir de son öbek göstergesi tutulur. Bu göstergenin belirlediği son öbek bağ alanı, boş öbek zinciri ilk öbek göstergesi ile günlenir. Bu gösterge de, silinme sonucu serbest kalan kütük öbek zincirine ait ilk öbek göstergesinin değerini alır.

Zincirli yer atama yöntemi sıradan kütük işlemleri için çok uygun bir yöntemdir. Zira öbekler sırayla işlenirken, işlenen öbek bir sonra işlenecek öbeğin adresini içerir. Ancak rasgele kütük işlemleri için aynı şey söylenemez. Bir kütüğün *i* inci öbeğinin adresi *(i-1)* inci öbeğinde saklandığından i inci öbeğe erişebilmek için 0 'dan *(i-1)* 'e kadar tüm öbekleri okumak ve birer birer, bir sonraki öbeğin adresini elde etmek gerekir. Bu yol, doğal olarak, pratikte uygulanamaz. Bu sakınca, değişik disk öbeklerine dağılmış bağ alanlarını, ayrı ve özel bir kütükte toplayarak aşılabilir. Bir sürücü üzerindeki tüm kütük öbek adresleri, topluca bu kütükte saklanır. Bu durumda kılavuz kütük satırları, kütüğün ilk öbek adresi yerine bu özel kütük içinde bir konumu gösterir. Bu konumdan başlayarak kütük öbek adresleri, bağlı bir liste biçiminde saklanır. Bir öbeğe rasgele erişmek gerektiğinde bu liste taranarak adresi elde edilir. Tarama işlemlerini hızlandırmak üzere, bu özel kütüğün, işletimin başında ana belleğe taşınması ve işlemlerin ana bellekte yapılması da anlamlıdır. Anımsanacağı gibi, açıklanan bu yol *MS-DOS*'ta kullanılmıştır. Sürücü öbek adreslerinin saklandığı özel kütük, *MS-DOS* kapsamında *FAT* olarak adlandırılmıştır (Çizim 6.9).

Zincirli yer atama yönteminin bir diğer zayıf noktası bağ bozulmalarına karşı aşırı duyarlı oluşudur. Bağ alanları ister öbeklere dağılmış, ister özel bir kütük içinde toplanmış olsun kütük zinciri içinde herhangi bir bağ alanının bozulması, o bağdan başlayarak diğer öbeklerin yitirilmesine ve ilgili kütüğün bozulmasına neden olur. Bozulan bir bağ alanının belirlenerek düzeltilmesi ve kütüğün kurtarılması hemen hemen olanaksızdır. Bu sakıncayı aşmak üzere, bağ alanlarının özel bir kütükte saklandığı durumlarda, bu özel kütüğün birden çok kopyası saklanır. Böylece, bozulma durumu ortaya çıktığında, ilgili bağ bilgisi için kütüğün diğer kopyalarına baş vurulur. Örneğin *MS.DOS*'ta *FAT* iki kopya olarak saklanır.

c. Dizinli Yer Atama Yöntemi

Zincirli yer atama yöntemiyle, rasgele kütük işlemleri için ortaya çıkan önemli sakınca, dizinli yer atama yöntemi kullanılarak da aşılabilir. Dizinli yer atama yönteminde kütüklere ilişkin öbek adresleri, dizin öbeği olarak anılan bir öbek içinde toplanır. Kütüğün kılavuz kütük satırı, dizin öbeğinin adresini tutar. Kütüklere erişimde önce dizin öbeği okunur. İlgili öbeğin adresi buradan elde edilerek, ikinci aşamada öbeğin kendisine erişilir. Dizin öbeği ana bellekte saklanarak diske, sistemli olarak çift erişim engellenmiş olur. Çizim 6.17'de, bir önceki çizimde söz konusu edilen `örnek` adlı kütüğün dizinli atama yöntemine göre diskteki yerleşimi ve dizin öbeği örneklenmiştir.

Dizin öbeğinin tutabileceği öbek adres sayısı sınırlıdır. Öbek boyunun 2 KB, adres boyunun ise 32 bit olduğu varsayılırsa dizin öbeğinin tutabileceği en büyük adres sayısı 512 olur. Bu, sistemde varlığına izin verilen en büyük kütük boyunun (512 x 2 KB) 1 MB ile sınırlı olması demektir. Bu önemli sınırlamayı aşmak üzere çok düzeyli

dizinleme kullanılır. İki düzeyli dizinlemede her kütük için birinci ve ikinci düzey dizin öbekleri bulunur. Birinci düzey dizin öbeği, yukarıdaki varsayımlara göre 512 değişik ikinci düzey dizin öbeğini gösterir. İkinci düzey dizin öbeklerinin herbiri de 512 kütük öbeğinin adresini tutar. Bu durumda iki düzeyli dizinleme ile 512 x 512 x 2 KB; 512 MB'lık kütüklerin adreslemesi gerçekleştirilir. Bunun da yetersiz olması durumunda üç düzeyli dizinlemeye geçilir. Üç düzeyli dizinleme ile 512 x 512 x 512 x 2 KB yaklaşık 256 GB'lık kütükleri adresleme olanağı bulunur.

Dizinli yer atama yönteminde boş öbeklerin ele alınışı genelde bit çizelgesi olarak anılan özel bir çizelge aracılığıyla olur. Bu çizelgede her bit disk sürücü üzerindeki bir öbeği temsil eder. Bir öbek bir kütüğe atandığı zaman çizelgedeki ilgili bit kurulur. Serbest öbeklere karşı gelen bitler ise sıfır değerini içerir. Kütük yönetim sistemi, yeni bir öbek istemi geldiğinde bit çizelgesini tarayarak atama yapar. Öbek boyu 4 KB olan 10 GB sığalı bir disk için tutulan bit çizelgesi 256 Mbit içerir. Bu da 32 MB demektir. Bit çizelgeleri ilgili oldukları sürücüler üzerinde saklanır, işletimin başında, kısmi olarak ana belleğe taşınırlar.

Çizim 6.17.Dizinli Yönteme göre Bir Kütüğün Diskte Yerleşim Örneği

Dizinli yer atama yöntemi gerek sıradan gerekse rasgele kütük işlemlerinde başarı ile kullanılır. Üzerinde işlem yapılan kütüğün dizin öbeği ana bellekte tutularak diske ek erişimler en düşük düzeyde tutulur. Bu yolla, kütüklere erişim hızı dizin yapısından çok etkilenmez. Her kütük için ayrı bir dizin öbeğinin tutuluyor olması disk kullanım verimliliğini belirli oranlarda düşürür. Büyük boyutlu kütükler için bu oran çok büyük değildir. Ancak birkaç öbeklik kütükler için sözkonusu oran oldukça yüksek olabilir. Örneğin, yukarıda yapılan varsayımlara göre, 1024KB'a kadar olan kütükler için bir dizin öbeği yeterli olmaktadır. 1024KB'lık bir kütük için yitirilen 2KB'lık bir dizin öbeğinin yarattığı kayıp oranı 2/1024'dür. Bu %0.2'lik küçük bir kayıptır. 10KB'lık küçük bir kütük için ise sözkonusu kayıp oranı %20'lere varır. Diskte saklanan

kütüklerin çoğunluğu küçük boyutlu kütüklerden oluşursa, genel yer kullanım verimliliği de oldukça düşer. Dizinli yer atama yönteminin önemli sayılabilecek tek sakıncası, diskte neden olduğu bu yer kaybıdır. *UNIX* ve *Windows NT* işletim sistemleri dizinli yer atama yöntemini kullanır. Bu yöntemin küçük boyutlu kütüklerde neden olduğu önemli orandaki yer kaybını en aza indirmek üzere, *UNIX*'te, 10 öbeğe kadar olan kütüklerde doğrudan öbek adresleme yöntemi kullanılır. Bu boydan daha büyük kütükler için dizinlemeye geçilir (Çizim 6.11, 12).

6.4. Kütük Yönetim Sisteminin Başarımı ve Güvenilirliği

Kütük yönetim sisteminin başarımı, mantıksal kullanım modeline dayalı işlemlerin yerine getirilişinde ortaya konan hızla ilgilidir. Kütük yönetim sistemleri, genelde, ana belleğin yönetiminde yararlanılan, ön bellek kullanımı gibi kimi başarım artırıcı yöntemleri kütüklerin yönetimine de uyarlarlar. Bu yolla diske yapılan erişimleri azaltarak ortalama erişim hızlarını yükseltmeyi amaçlarlar.

Kullanıcılar, nitelikli bir bilgisayar hizmeti kapsamında, sistemde sakladıkları kütüklerinin korunacağına ve bunların bilgi bütünlüğünün, kendi hataları dışında bozulmayacağına inanmak ve güvenmek isterler. Bu güven ortamının yaratılabilmesi için, kütüklerin, kullanıcıların istemi dışında, elektrik kesilmeleri, sistem hataları gibi nedenlerle bozulmalarına karşı kimi önlemler alınır. Bu önlemler kütük yönetim sisteminin güvenilirliği kapsamında düşünülür. İzleyen kesimde, kütük yönetim sisteminin başarım ve güvenilirlik konularına değinilecektir.

6.4.1. Disk Ön Bellek Alanlarının Kullanımı

Kütük yönetim sisteminin başarımını yükseltmenin yolu diske yapılan erişimleri hızlandırmaktan geçer. Bilindiği gibi disk arabirimleri düzeyinde, birkaç izlik verinin saklanabildiği yastık bellek birimleri bulunur. Disk biriminden bir öbeğin okunması gerektiğinde, ilgili öbek bu yastık alanında ise sürücüye erişime gerek kalmadan istem yanıtlanır. Bu düzenek sürücüye (mıknatıslı plakalara) erişim sayısını azaltarak diske ortalama erişim hızını yükseltmeye katkı verir. Arabirim düzeyinde yer alabilen bu yastık belleklerin yanı sıra, bir de ana bellek düzeyinde disk öbeklerini yastıklamaya yarayan ve bu yolla disk birimine erişimi azaltmayı hedefleyen düzenekler bulunur. Disk öbeklerinin yastıklandığı ana bellek kesimi, genel olarak disk ön bellek *(cache)* alanı olarak bilinir. *UNIX* işletim sisteminde bu, *buffer cache* olarak adlandırılır. Disk ön bellek alanları, herhangi bir anda en yoğun kullanılan ya da kullanılma olasılığı yüksek öbeklerin ana bellekte saklanmalarını ve bunlara gereksinim duyuldukça, disk birimine erişilmeden buradan okunmalarını sağlarlar. Disk ön bellek alanı belirli sayıda disk öbeğini taşıyabilecek sığada ana bellek kesimidir.

Disk ön bellek alanının yönetiminden, genelde kütük yönetim sistemi sorumlu olur. *UNIX*'te bu işlev çekirdek katmanda yeralan işlevler tarafından karşılanır. Diske ilişkin okuma ve yazma istemleri bu yönetim katmanına yönlendirilir. Eğer erişilmesi istenen öbek ön bellek alanında ise istem, bellekten belleğe aktarımla hızla karşılanır. Öbeğin ön bellekte bulunmadığı durumlarda diske erişilir. Okunan öbek, aynı zamanda ön

belleğe de yazılır. Kısıtlı sayıda öbeğin sığabildiği bir ana bellek alanının yönetiminin yanıtlaması gereken en önemli soru, ön bellek alanına yeni bir öbek taşımak gerektiğinde buradan hangi öbeğin çıkarılacağı sorusudur. Bu amaçla, sayfalı bellek yönetimi kapsamında da konu edilen kimi algoritmalar kullanılabilir. Anımsanacağı üzere bunlar *FIFO, LRU* gibi adlarla anılan algoritmalardır. Bunlardan *LRU*-ön belleğe en erken giren öbeğin çıkarılması algoritması disk ön belleğinin yönetimi için oldukça uygun bir algoritmadır. Bu algoritma aracılığıyla, en sık kullanılan öbeklerin ön bellekte kalması sağlanır. Bu bağlamda, bir görev, tutanak okuma isteminde bulunduğunda, bu tutanağın yer aldığı öbek, disk ön belleğine taşınır. Görevlere üzerinde işlem yaptıkları tutanakları, dolayısıyla öbekleri kilitleme olanağı verilir. Bir öbek kilitli olduğu sürece ön bellekte kalır. Kiliti kaldırılan öbeğin kimliği, *LRU* sırasında tutulan bir kuyruğun sonuna eklenir. Bu arada öbek, ön bellekteki varlığını sürdürür. Bir öbeğe erişim isteminde bulunulduğunda, önce bu kuyrukta aranır. Bulunursa kuyruktan çıkarılır ve yeniden kilitlenir. Ön bellekte bulunmayan bir öbeğe erişim yapılmak istendiğinde *LRU* kuyruğunun başındaki öbek ön bellekten çıkarılır. Üzerinde yazma işlemi yapılmışsa diske geri yazılır.

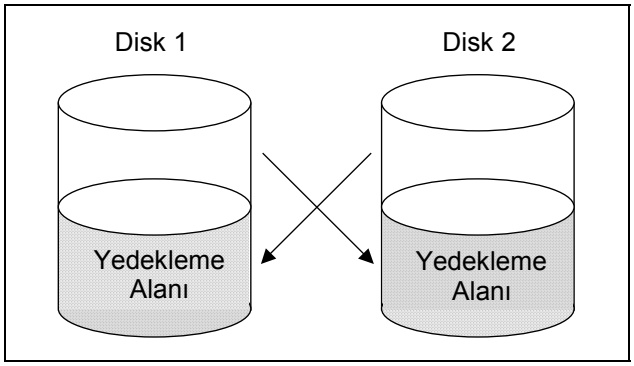
Diske erişimde ön bellek tekniğinin kullanılmasının en önemli sakıncası veri bütünlüğünün bozulma riskidir. Kimi öbeklerin, üzerlerinde yazma işlemi yapılmış olsa bile, ön bellek alanında belirli süre tutuluyor olmaları, istem dışı elektrik kesilmelerinde yitirilme risklerini artırır. Diske erişimde ön bellek tekniğinin kullanılması kullanıcılara saydam bir husustur. Bu nedenle, üzerinde günleme yapılan kimi kütük öbeklerinin, ilgili kütük kapatılmış, günlemeleri gerçekleştiren görev sonlanmış olsa bile, bir süre, ön bellek alanında kalmaları, başka bir anlatımla diske yazılmalarının gecikmesi olasıdır. Bu aradaki elektrik kesilmeleri, sözkonusu öbeklerle birlikte, kullanıcılara açıklaması zor, veri ve işlem kayıplarına neden olur. Hele kaybedilen öbeklerin, örneğin dizin öbekleri olması durumunda bu kayıplar çok ciddi boyutlara ulaşır.

Bu gerekçeye dayalı olarak, ön bellek tekniğinin kullanıldığı sistemlerde kimi önlemler alınır. Bu bağlamda, *UNIX* işletim sisteminde, `sync` olarak adlandırılan bir sistem çağrısı öngörülmüştür. `sync` çağrısının işlevi disk ön belleğinde bulunan günlenmiş öbekleri diske yazmaktır. Sistem işletime açıldığında, `update` ya da benzeri bir adla anılan özel bir sistem görevi de işletime alınır. Bu görev, 30-40 saniye gibi bir sıklıkla ana işlem birimine anahtarlanarak `sync` çağrısını çalıştırır. Böylece günlenmiş öbeklerin ön bellekte yitirilme riski azaltılmış olur. Aynı işlev `shutdown` olarak adlandırılan sistem kapama komutu ile de yerine getirilir. Bu komut çalıştırılmadan sistemin kapatılması durumunda da, yine, kimi günlenmiş öbekler yitirilebilir.

MS-DOS'ta kullanılan önlem ise başkadır. Burada herhangi bir öbeğe yazma yapılması durumunda bu öbek hemen diske aktarılır. Bu tür önlemlere, İngilizce *write-through-cache* adı verilir. Günlenen öbeklerin hemen diske yazılması, doğal olarak, *UNIX*'in kullandığı gecikmeli yazma yaklaşımına göre diske daha çok erişim gerektirir. Bu yaklaşımla, salt okunan öbeklerin ön bellekte tutulduğu da söylenebilir. Diske yapılan erişimlerin büyük çoğunluğu okuma amacıyla gerçekleştiğinden, bu yaklaşım diske erişim konusunda önemli başarım düşüşüne de pek neden olmaz.

6.4.2. Kütük Yönetim Sisteminin Güvenilirliği

Bir bilgisayar sisteminde saklanan kütüklerin, kullanıcıların istemleri dışında ortaya çıkabilecek bozulmalara karşı korunması kütük yönetim sisteminin güvenilirliği açısından çok önemlidir. Bu güvenirlik, en yaygın biçimde yedekleme işlemleri ile sağlanır. Yedekleme işleminden, disk birimleri üzerinde saklanan verilerin, dönem dönem başka mıknatıslı saklama ortamlarına kopyalanmaları anlaşılır. Bu yolla, sistemde saklanan verilerin birden çok kopyası (yedeği) oluşturulur. Bir kütük üzerinde ortaya çıkabilecek bozulmaların diğer kopyalardaki bilgilerle düzeltilebilmesi amaçlanır. Diskler üzerinde saklanan kütüklerin yedeklemesi, çoğu kez, mıknatıslı şerit, kartuş, disket ve optik disk birimleri gibi taşınır birimler üzerine yapılır.

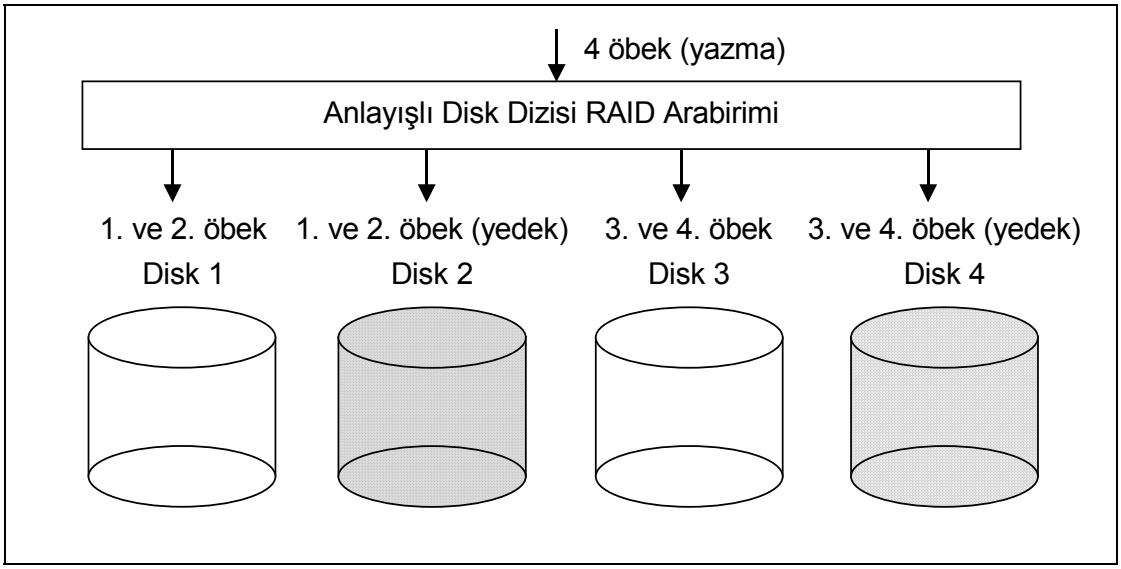


Çizim 6.18. Diskte Yedekleme

Yedekleme işlemleri, bilgisayar sisteminin işletiminden sorumlu kişiler (sistem işletmenleri) tarafından, belirli sıklıkta yerine getirilir. Bu sıklık, saklanan verilerin önem ve boyutuyla ilgili olarak her gün, her hafta ya da her ay gibi belirlenir. Sistemdeki tüm verilerin yedeklenmesi sözkonusu edildiğinde yedekleme işlemleri saatlerle ifade edilen sürelerde gerçekleşir. Aktarım sığaları 1 MBps olan 2 mıknatıslı şerit biriminin bulunduğu bir sistemde toplam disk sığasının 10 GB olduğu varsayılırsa tüm verilerin yedeklenme süresi, en az 5000 saniye olur. Bu da yaklaşık 2 saat demektir. Böyle bir yedekleme işlemini, örneğin her gün yerine getirmek, her sistem için gerekli olmayabilir. Bu durumda, yedeklemede zaman tasarrufu sağlayan önlemlere başvurulur. Daha önceden de belirtildiği üzere, kütüklerin son günlenme tarihleri kılavuz kütüklerde saklanır. Yedekleme işlemleri, son yedeklemenin gerçekleştiği tarihten bu yana günlenmiş kütüklere uygulanarak işlem süreleri kısaltılır. Yedekleme işlemleri birden çok kopya olarak da yapılabilir. Çok kritik verilerin saklandığı sistemlerde değişik kopyalar değişik coğrafi ortamlarda saklanarak, verilerin sel, yangın gibi felaketlere karşı da korunması *(disaster recovery)* sağlanır.

Yedekleme işlemlerinin ortaya çıkardığı zaman darboğazını aşmak üzere bir disk birimi, başka bir disk birimi üzerine de yedeklenebilir. Bu değişik biçimlerde gerçekleşir. Birinci yol disk birimlerinin eşlenip ikiye bölünerek kullanılmasıdır. Bu yolla bir birimin kopyası, eşi olan diğer birim üzerinde saklanır. Bir birimin bozulması durumunda eşi üzerinde saklanan verilere başvurulur. Yedekleme diskten diske aktarım yoluyla gerçekleşir (Çizim 6.18). Bu yolun bir türevi aynalama *(mirroring)* yöntemi

olarak bilinir. Sistemde her diskin, aynı sığada bir eşi bulunur. Diske tüm yazmalar her iki birime de koşut olarak yapılır. Aynalama yöntemi uygulanan sistemlerde her kütüğün, doğal olarak iki kopyası bulunur. Burada yedekleme için işletim dışı kopyalama işlemlerine gerek duyulmaz.



Çizim 6.19. *RAID10* Modeli Disk Dizilerinin Kullanımı

Aynalama kavramı genişletilirse *RAID (Redundant Arrays of Inexpensive /Independant Disks)* Disk Dizisi tekniğinden söz edilir. Bu teknikte belirli sayıda disk sürücü, disk dizisi denetleme birimi olarak anılan anlayışlı bir arabirim aracılığıyla sisteme bağlanır. Disk dizisine öbek yazma işlemleri, sanki tek bir sürücüye yapılıyormuş gibi yürütülür. Ancak anlayışlı disk dizisi arabirimi sözkonusu öbekleri (kimi durumlarda, bu öbeklere ilişkin *CRC, ECC* gibi adlarla anılan hata denetim sözcüklerini) dizi içinde yer alan sürücülere belirli bir model çerçevesinde dağıtarak, koşut bir biçimde yazar. Okuma işlemi de bu modele uyumlu olarak, koşut bir biçimde yapılır. Bu yolla, bir yandan disk erişim hızı artırılırken diğer yandan da yedeklemeye dayalı hataya duyarsız *(fault tolerant)* işleme olanak sağlanır.

Bu bağlamda, kullanılan modelin, *RAID0, RAID1, RAID2, RAID3, RAID4, RAID5, RAID10 (1+0)* gibi standart bir adı bulunur. *RAID0* modelinde kütükler, *strip* olarak anılan öbek kümelerine ayrılıp dizi içinde yer alan disk sürücülere koşut olarak yazılır ve okunur. *Strip* olarak anılan öbek küme boyu, (işletimin başında) değiştirilebilen bir parametredir. Bu modelde yedekleme yapılmaz, dolayısıyla hataya duyarsız işlem sözkonusu değildir. *RAID1* modeli, yukarıda aynalama olarak tanımlanan yöntemi gerçekleştirmede kullanılır. Öbek kümeleri, sistemli olarak iki ayrı sürücüye yazılır. Öbek kümeleri, iki ayrı sürücüden koşut olarak da okunabilir. *RAID0* ve *RAID1* modellerinin birlikte kullanılması durumunda *RAID10* modelinden söz edilir. Örneğin 4 disk sürücü içeren bir disk dizisinde, *RAID10* modeline göre, sürücüler ikişerli eşlenir. Eşlerden biri diğerinin yedeğini tutar. Disk çiftlerine koşut yazma yapılarak erişim hızı artışı sağlanır (Çizim 6.19).

Bilgisayar sistemlerinde herhangi bir anda, disklerde saklanan verilerin bütünlüğü, tutarlılık sınaması yapılarak denetlenir. Tutarlılık sınamaları özel sistem programları aracılığıyla yerine getirilir. Bu bağlamda, *MS-DOS*'ta `chkdsk` (*check disk*) adlı komut kullanılır. *UNIX*'te ise bu amaçla kullanılan işlevin adı `fsc`k (*file system check*) 'dir.

Bu tür programlar, genelde tüm disk öbeklerini kütük atama çizelgeleri ile boş öbek çizelgeleri içinde izlerler. İşletim sırasında, çeşitli nedenlerle, kimi disk öbekleri birden çok kütüğe atanmış görünebilir. Kimi öbekler de, ne boş öbek ne de atama çizelgelerinde yer almayabilirler. Bu tür öbekler kütük sisteminin bütünlüğünü ve tutarlılığını bozan öbeklerdir. Bu nedenle bu öbeklerle ilgili gerekli düzeltmeler yapılır. Çoğu kez, birden çok kütüğe atanmış görünen öbekler, içerikleri aynı kalacak biçimde ayrı fiziksel öbekler olarak çoğaltılarak atama çizelgelerine yeni kimlikleri ile eklenirler. Hiçbir çizelgede görünmeyen öbekler ise boş öbek çizelgesine işlenirler. Bu işlemler, doğal olarak belirli bir anda, kütük sistemine tutarlı görünümünü yeniden kazandıran ancak o ana kadar oluşmuş kütük bozulmalarını ortadan kaldırmayan işlemlerdir.

7. BÖLÜM

İ Ş L E T İ M S İ S T E M L E R İ

GÜVENLİK ve KORUMA

Güvenlik ve koruma dendiğinde, bilgisayar sistemlerinde saklanan verilerin güvenliği ve korunması amacıyla, işletim sistemlerince sağlanan işlev ve araçlar akla gelir. Bilgisayar sistem donanımlarının, yangın, sel gibi doğal afetler ile fiziksel bozulmalara karşı korunmaları, kurtarma *(disaster recovery)* kapsamında düşünülür ve bu bağlamda ele alınmaz. Bunun gibi, bilgisayar sistemi dışında saklanan, örneğin yedeklenmiş verilerin güvenliği de yine bu kapsamda ele alınmaz. Özellikle çok kullanıcılı bilgisayar sistemleri ile yerel ağlar içinde bütünleşmiş bilgisayar sistemlerinde güvenlik ve koruma konuları önemli bir yer tutar. Zira bu tür sistemlerde, günümüzde, parasal, kurumsal, askeri, tıbbi ve kişiye özel çok önemli bilgiler ile parasal değeri yüksek yazılım araçları saklanmaktadır. Bunların, bir yandan silinerek yok olmalarına, diğer yandan da üçüncü kişilerin eline geçmelerine karşı koruma önlemleri alınması zorunludur. Bu koruma önlemlerinin tümü, işletim sistemlerinde güvenlik ve koruma başlığı altında düşünülür. İşletim sistemlerinde:

- Bilgisayar sisteminden izinsiz yararlanmaya,
- Bilgisayar sisteminde saklanan bilgilere izinsiz erişimlere ve bunların izinsiz olarak günlenmesi ve silinmesine,
- Bilgisayar sisteminin sağlıklı çalışmasının engellenmesine karşı

çeşitli önlemler alınır.

Her türlü özel mülkiyette olduğu gibi, bilgisayar sistemlerinden yararlanabilmek için ya, kişisel bilgisayar sistemlerinde olduğu üzere bu sistemlere sahip olmak ya da, çok kullanıcılı bilgisayar sistemlerinde olduğu üzere, bu sistemlerin işletiminden sorumlu mercilerden izin almak gereklidir. Bilgisayar sistemlerinin kullanımının belirli kurallara bağlanması ve çoğu kez bunlardan, belirli bir bedel ödeyerek yararlanılması doğaldır. Bir bilgisayar sisteminden yararlanmada uyulması gereken kurallar, işletim sistemi tarafından denetlenir. Bu bağlamda, örneğin kullanıcı kodu ve parola gibi araçlardan yararlanılır. Bilgisayar sisteminden izinsiz yararlanmaları engelleme kapsamında düşünülen bu araçlara ve ilgili denetimlere, izleyen kesimde Bilgisayar Sistemine Girişlerin Denetlenmesi başlığı altında değinilecektir.

Bilgisayar sisteminde saklanan bilgilere her tür erişimin denetlenmesi bunların, izinsiz kişilerce; gerek okunarak gizlilik ilkesinin ihlal edilmesi, gerekse silme ve günleme gibi nedenlerle bütünlük ve tutarlılıklarının bozulmasına engel olunması çok önemlidir. Bilgisayar sistemleri güvenli birer işletim ortamı olmanın yanı sıra güvenli birer saklama ortamı da olmak zorundadırlar. Zira, çoğu kez veriler işlendikleri bilgisayar ortamında sürekli saklanmak ve korunmak durumundadır. Bir bankanın bilgisayar sisteminde işlenen müşteri hesap bilgileri, çevrim içi işlenen bilgiler olmaları itibarıyla sürekli bilgisayar sisteminde tutulması gereken bilgilerdir. Bunun gibi bir havayolu taşımacılık firmasının yer ayırtma sisteminin işlediği yolcu ve uçak bilgileri de sürekli bilgisayar sisteminde tutulması, korunması ve işlenmesi gereken tür bilgilerdir. İşletim sistemleri, bilgisayar ortamında tutulan verilerin:

- bütünlüğünü
- gizliliğini ve
- kullanılabilirliğini

korumakla yükümlüdür. Bilgisayar sistemine emanet edilen, çoğu gizli ve kıymetli bilgilerin korunmasının işletim sistemi tarafından ele alınışı, Erişim Denetimi başlığı altında incelenecektir.

Bilgisayar sistemlerinde saklanan verilerin güvenliğini artırmanın bir yolu da, bu verileri şifrelemekten geçer. Veriler hem disk ortamında saklanırken hem de bir sistemden diğer bir sisteme iletilirken şifrelenebilir. Şifreleme üçüncü kişilerin eline izinsiz geçmiş verilerden yararlanmayı engelleyerek korumayı güçlendirir. Şifreleme konusuna Şifreleme başlığı altında, kısaca değinilecektir.

İşletim sistemleri bilgisayar sistemlerinin sağlıklı çalışmasına engel olan kötü niyetli kullanıcılar ile bunların geliştirdiği aynı amaçlı uygulamalara karşı da önlemler almak zorundadır. Bilgisayar sistemine girişlerin denetlenmesi ve verilerin izinsiz erişimlere karşı korunması kapsamında ele alınan önlemler, bilgisayar virüsleri gibi, kötü niyetli uygulamalara karşı, özellikle kişisel bilgisayar sistemleri için yetersiz kalabilmektedir. Güvenlik ve Koruma konusunun sonunda, kısa bir kesim bu konuya ayrılmış ve bunlarla başa çıkmada kullanılan kimi yollardan söz edilmiştir.

7.1. Bilgisayar Sistemine Girişlerin Denetlenmesi

Bilgisayar ortamında tutulan verilerin bütünlüğünün, gizliliğinin ve kullanılabilirliğinin korunabilmesi için, öncelikle bilgisayar sistemine girişlerin denetlenmesi, başka bir anlatımla izinsiz kişilerin, sistemde çalışmasının engellenmesi gereklidir. Bir kullanıcının, kullanım amacıyla bilgisayar sistemi üzerinde geçirdiği zaman dilimi, oturum olarak adlandırılmaktadır. Bilgisayar sisteminden yararlanmak için bir oturumu başlatabilmek gerekir. Bilgisayar sistemine girişler, bu nedenle oturum başlarında denetlenir. Çok kullanıcılı bir bilgisayar sisteminden yararlanan kullanıcıların herbirinin, öncelikle sistemde tanımlı oldukları birer kullanıcı kodu bulunur. Hangi kullanıcının, hangi kaynağı, ne kadar süre ile kullandığının sayışımı bu kod aracılığıyla yapılır. Kullanıcıyla ilgili sayışım bilgilerinin yanı sıra, işletim hakları, denetim bilgileri ile istatistiksel bilgilerin tutulmasında da bu koddan yararlanılır.

Her oturum, kullanıcı kodunun girilmesiyle başlatılır. Kullanıcılarla ilgili denetimler, kullanıcı kodunun sınanmasıyla son bulmaz. Kullanıcı kodları genelde sistem tarafından, yalın bir mantığa dayalı olarak üretilen ve genelde gizli olmayan bilgilerdir. Örneğin, bir üniversite bilgisayar sisteminde, öğrenciler için kullanıcı kodları, ilgili oldukları bölüm kodunun arkasına, soyadı alfabe sırasına göre iki ya da üç basamaklık bir sayı eklenerek üretilebilir. Bu durumda, herhangi bir öğrenci, arkadaşının kodunu kolayca tahmin edebilir. Bu nedenle kullanıcı kodunun yanı sıra, bir de kullanıcıya özel ikinci bir bilgiyi kullanma zorunluluğu vardır. Kullanıcı kodunu girerek oturum başlatmak isteyen bir kişinin bu kodun gerçek sahibi olup olmadığını denetlemenin, genelde değişik yolları bulunur. Bunlar:

- parolaya dayalı
- kimlik kartına dayalı
- fiziksel özelliklere dayalı

olmak üzere, en yalınından en karmaşığına kadar üç sınıfta toplanır.

7.1.1. Parolaya Dayalı Denetim

Oturum başlatmak isteyen kullanıcıların kimlik denetiminde en yaygın biçimde kullanılan araç paroladır. Kullanıcı kodunu girerek oturum başlatma girişiminde bulunan kullanıcıdan, parola olarak adlandırılan bir sözcüğü girmesi istenir. Girilen parola sistemdeki ile karşılaştırılır. Uyuşma durumunda oturum başlatılır. Parola olarak adlandırılan gizli sözcük, kullanıcı sisteme tanımlanırken, sistem tarafından belirlenir ve kullanıcıya bildirilir. Ancak kullanıcı kendisiyle ilgili parolayı, genelde istediği zaman özgürce değiştirebilir. Parolalar sistemde şifrelenmiş biçimde saklanırlar. Bu yolla parola kütüğünün okunarak parolaların çalınması önlenmeye çalışılır.

Kullanıcılar parolalarını, çoğu kez hatırlanması kolay özel isimler arasından seçmeyi tercih ederler. Bu durum, kişilerle ilgili bu gizli denetim sözcüklerini tahmin etmeyi kolaylaştırır. Başkasına ilişkin bir kullanıcı kodu ile oturum başlatmak isteyen bir kişi, terminal başında, bir dizi değişik parolayı deneyerek sisteme girmeyi başarabilir. Bu tür parola tahmin girişimlerini engellemek üzere işletim sistemlerinde çeşitli önlemler

alınır. Örneğin kullanıcıların parolayı yanlış girme sayısı sınırlandırılır. Belirlenen sayı kadar deneme sonunda, hala doğru parolanın girilememesi durumunda ilgili terminal belirli süre kilitlenir. Parolaların tahminini zorlaştırmak üzere birden çok sayıda parola da kullanılabilir. Bu parolalar ya oturumun başında ardarda girilir ya da işletim sırasında terminali başında çalışan kullanıcıdan rasgele zamanlarda talep edilir. Bu yolla kaçak kullanıcıları yakalamaya yarayan elek sıkılaştırılır. Doğal olarak bu denetimlerin aşırıya kaçması yasal kullanıcılara rahatsızlık verebilir. Kaçak kullanıcıların yakalanması hedeflenirken yasal kullanıcılara rahatsızlık vermek bilgisayar sisteminin verdiği hizmetin niteliğini olumsuz yönde etkiler. Çok sıkı denetimlerin gerekli olduğu sistemlerde, parola sistemini karmaşıklaştırmak yerine özel kimlik kartlarına ya da fiziksel özelliklere dayalı denetimlere geçilir.

7.1.2. Kimlik Kartına Dayalı Denetim

Bilgisayar sistemlerine girmede, parolaya dayalı denetimden daha güvenilir bir diğer denetim, kimlik kartlarına dayalı denetimdir. Bu kapsamda her kullanıcının sistem tarafından okunabilen bir kimlik kartı bulunur. Bu kartlar üzerinde, genellikle kullanıcı kimlik bilgilerinin kayıtlı olduğu mıknatıslı bir bant yer alır. Bunun gibi, *smart* olarak adlandırılan, kimlik bilgilerinin, doğrudan karta yapıştırılan tümleşik çevrim içinde saklandığı tür kartlar da kullanılır. Bilgisayar sisteminden yararlanmak isteyen kullanıcılar, çoğunlukla terminal birimiyle bütünleşmiş bir kart okuyucu ile kartlarını okutarak oturumları başlatırlar. Çalınma ya da kaybolma yoluyla üçüncü kişilerin eline geçme olasılığı bulunduğundan kimlik kartları parola ile birlikte kullanılır. Parola, kartı kullanan kişinin kartın gerçek sahibi olup olmadığını denetlemeye yarar. Kartını okutan kişiden parolasını girmesi istenir. Girilen parola, çoğu kez kart üzerinde saklanan asıl parola ile karşılaştırılır. Uyuşma durumunda oturum başlatılır. Elektronik banka işlem makinalarının, özel terminal birimleri olarak bağlı olduğu bankacılık sistemleri, kimlik kartı ile denetim uygulayan sistemlere verilebilecek örneklerdir.

7.1.3. Fiziksel Özelliklere Dayalı Denetim

Bilgisayar sistemine girişte kimlik denetiminin çok güvenilir bir biçimde yapılması gerektiğinde taklit edilmesi olanaksız bilgilere dayanma zorunluluğu vardır. Bu durumda, kullanıcılara özel kimlik kartları ve parolalar yerine, kişiden kişiye mutlaka farklılıklar gösteren parmak izi, resim, ses, göz retinası gibi kişisel özellik bilgilerine başvurulur. Doğal olarak, bu tür bilgilere dayalı denetim işlemleri pahalı özel giriş birimlerinin kullanımını gerektirir. Denetim amacıyla, örneğin kullanıcıların yüz resimlerine başvurulan sistemlerde özel bir kamera aracılığıyla sisteme girmek isteyen kişinin o anki yüz resminin elde edilmesi, sistemdeki resimle karşılaştırılarak uyuşma olup olmadığına hızla karar verilmesi zorunludur. Ses ve resme dayalı tanıma sistemleri, hızlı işlem hızına sahip de olması gereken özel, dolayısıyla pahalı sistemlerdir. Fiziksel özelliklere dayalı denetim düzenekleri, bu nedenle çok yaygın kullanılan düzenekler değildir.

7.2. Erişim Denetimi

Bilgisayar sistemlerinde erişim denetim zorunluluğu, ilk olarak, işletim sistemi ile kullanıcı programlarının, aynı anda ana belleği paylaşmalarının yarattığı sorunlardan kaynaklanmıştır. Kullanıcı programlarının, genelde hatalı çalışma sonucu işletim sistemine ayrılan ana bellek alanını bozmalarını engellemek ve bu yolla işletim bütünlüğünü korumak üzere, ana bellekte kimi alanlara erişimleri kısıtlamak gerekmiştir. Tek iş düzeninden çok iş düzenine geçişle, işletim sisteminin yanı sıra kullanıcı programlarını da, ana bellekte birbirlerine karşı koruma gereği doğmuştur.

Ana belleğin yönetimi kapsamında açıklandığı gibi, kullanılan bellek yönetim modeline bağlı olarak, ana işlem birimine yapılan kimi özgün ekler sayesinde, ana belleğe erişim denetimini sağlayan düzenekler elde edilir. Örneğin, bölümlü bellek yönetimi kapsamında, ana işlem birimine eklenen sınır yazmaçları aracılığıyla, hem işletim sistemini işlerden, hem de işleri birbirlerinden koruma düzenekleri yaratılır. Bir iş, kendisiyle ilgili sınır yazmaç içeriklerinin dışına düşen bir adres ürettiğinde kesilerek sistemin işletim bütünlüğü korunur. Sayfalı ya da kesimli bellek yönetimlerinde, görevlere ilişkin sayfa ya da kesimlerin herbirinin erişim hakkı, sayfa ya da kesim tanım çizelgelerinde tutulur. Yapılan erişim türü, ilgili sayfa ya da kesimin erişim hakkı ile uyuşmadığı durumlarda görev işletimi sonlandırılarak koruma sağlanır. Görüntü bellek yönetimlerinde görevlerin mantıksal adres evrenleri ile fiziksel ana bellek evreni birbirlerinden tümüyle soyutlanarak daha esnek koruma düzenekleri kurulur.

Koruma düzeneklerinin kullandığı, sınır yazmacı, yer değiştirme yazmacı, sayfa / kesim tanım çizelge yazmaçları gibi özgün ana işlem birimi yazmaçları sistemin işletim bütünlüğünü doğrudan etkileyen ögeler olarak ortaya çıkarlar. Bu yazmaçlara erişimlerin de denetim altında tutulması; başka bir deyişle, bunlara erişim hakkının kimi özel görevler dışında diğer görevlere kapalı tutulması gerekir. Bu, görevlerin:

- sıradan ve
- ayrıcalıklı

olmak üzere iki değişik kategoriye ayrılmasını zorunlu kılar. Sıradan görevler, koruma altındaki ana işlem birimi yazmaçlarına erişmeye (ya da bu amaçla ilgili makina komutlarını işletmeye) kalktıklarında, engellenemez bir iç kesilme ile işletimleri kesilir. İşletilmekte olan görevin hangi kategoriye ilişkin bir görev olduğu, görev işletime alındığında günlenen, ana işlem birimi program durum yazmacı içindeki, örneğin bir bit ile belirlenir. Bu bitin kurulu olması, yazmaçlara erişim yapıldığında işletimin kesilmesi sonucunu doğurur. Sıradan görevlerin program durum yazmaçlarındaki bu bit hep kuruludur. Bir bilgisayar sisteminde çalışan görevler ilgili oldukları kullanıcılara göre sınıflandırılırlar. Bu nedenle kullanıcılar da,

- sıradan kullanıcılar ve
- ayrıcalıklı kullanıcılar

olarak iki sınıfa ayrılırlar. Kullanıcıların hangi sınıfa dahil oldukları sisteme ilk tanımlandıklarında belirlenir. Ayrıcalıklı bir görev ana işlem birimine anahtarlandığında program durum yazmacı içindeki sıradan / ayrıcalıklı durum biti sıfırlanır. Bu durum,

ana işlem biriminin işletim sistemi moduna geçmesi olarak tanımlanır. Bu bağlamda ana işlem biriminin

- işletim sistemi modu'nda ya da
- kullanıcı modu'nda

çalışmasından söz edilir. Bu adlandırma, ayrıcalıklı görevlerin çoğunlukla işletim sistemi görevleri, sıradan görevlerin ise sıradan kullanıcı görevleri olmasından kaynaklanır[26].

Ana bellekte saklanan veriler için yaratılan koruma düzenekleri, daha çok işletim ve veri bütünlüğü kapsamında düşünülür. Ana bellek dışında saklanan ve kütük olarak adlandırılan verilerin korunması, veri bütünlüğünün yanı sıra veri gizliliğini de gözetmek durumundadır. Kütükler için erişim denetimi, genelde her kütük için ayrı ayrı ve kullanıcının (sahibinin) bizzat belirlediği, salt okunur, okunur-yazılır, işletilir gibi haklara dayalı olarak ele alınır. Erişim denetiminde, ana bellek ve ikincil bellek ayrımı yapmak yerine, bu denetimi, korunması amaçlanan nesnelerden soyutlayarak gerçekleştirmek de olanaklıdır. Bu soyutlama, erişim matrisi modeli olarak adlandırılan bir model çerçevesinde ele alınabilir.

Nesneler

Koruma Alanları	kütük1	kütük2	...	kütük i	...	yazıcı	disket
KA0	okuyaz	oku-yaz		oku-yaz		yaz	oku-yaz
KA1		salt-oku					
KA2				işletim			salt-oku
KA3				salt-oku			

Çizim 7.1. Erişim Matrisi Örneği

Bu model kapsamında, sistemdeki amaç programlar, veri kütükleri, giriş/çıkış birimleri, bellek sayfa ya da kesimleri gibi donanım ve yazılım nitelikli ögeler nesneler olarak genellenir. Görevler ise, nesneler üzerinde işlem yapan ve bunları dönüştüren ögeler olarak düşünülür. Görevlerin nesneler üzerinde, okuma-yazma, salt okuma, işletim gibi, gerçekleştirebilecekleri işlem türleri nesnelerin erişim hakkı ile tanımlanır. Bir nesne ve buna ilişkin erişim hakkının oluşturduğu ikiliye nesne-erişim hakkı ikilisi adı verilir. Bir sistemde yer alan ikililer, koruma alanları adı verilen değişik alanlar içinde kümelenmiş biçimde düşünülebilirler. Görevlerin nesnelere erişimleri bu koruma alanları içinde ele alınabilir. Görevlerin, hangi koruma alanları içinde işlem yapabilecekleri, ayrıcalık düzeyleriyle belirlenir.

[26] İngilizcede, bu bağlamda, *supervisor mode / user mode, superuser mode / user mode* gibi ifadeler kullanılmaktadır.

Koruma alanlarının satırları, nesnelerin de sütunları oluşturduğu matris, erişim matrisi olarak adlandırılır. Bu matrisin ögeleri sözkonusu koruma alanı için ilgili nesnenin erişim hakkını temsil eder. Aynı nesne, değişik erişim haklarıyla değişik koruma alanlarında bulunabilir. Çizim 7.1'de, bir erişim matrisi örneği verilmiştir. Bu örneğe göre, `kütük2` adlı nesne, oku-yaz hakkı ile sıfırıncı koruma alanında, salt okuma hakkı ile de birinci koruma alanında bulunmaktadır.

7.2.1. Erişimde Koruma Halkaları

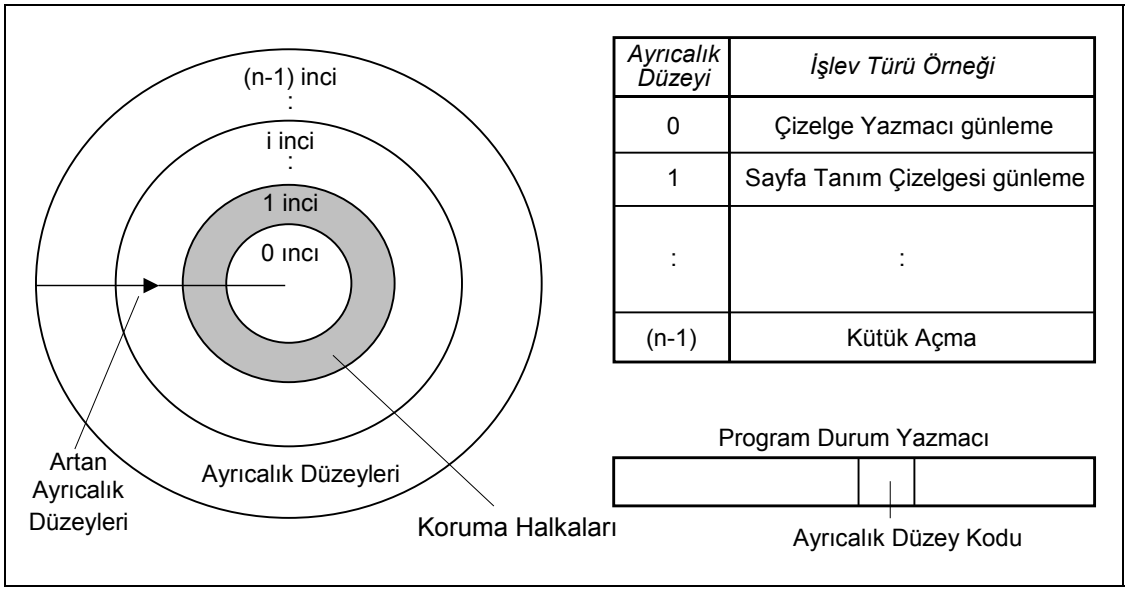
Nesnelerin, değişik koruma alanları içinde kümelenmesi, bu nesneler üzerinde işlem yapan görevlerin de, değişik koruma alanları içinde işlem yapan görevler olarak düşünülmelerini gerektirir. Bir görevin herhangi bir koruma alanı içinde işlem yaparken diğer bir koruma alanına geçmesi, bu yeni koruma alanına anahtarlanması olarak adlandırılır. Görevlerin değişik koruma alanları arasında anahtarlanması bir nesne kümesi üzerinde işlem yaparken yeni bir nesne kümesine erişmesi demektir. Bu nedenle görevlerin nesnelere erişimini denetlemek, bunların değişik koruma alanları arasında anahtarlanmalarını denetim altında tutarak gerçekleştirilebilir. Kimi işletim sistemlerinde koruma alanları iç içe halkalar olarak düzenlenir. Görevlerin hangi halkalar içinde işlem yapabileceği, donanımca da desteklenen bir düzenek ile denetim altında tutulur. İç içe halkaların içten dışa azalan önemde bir sıradüzenleri bulunur. İç halkayla simgelenen çekirdek koruma alanına çok az sayıda görev erişebilirken çekirdek halkadan dışa doğru gidildikçe daha çok sayıda göreve erişim ya da anahtarlanma hakkı verilir. *n* adet koruma halkasının tanımlandığı bir sistemde *i* inci halkanın simgelediği koruma alanına anahtarlanabilen bir görev, *(i+1), (i+2), .., (n-1)* inci halkalarla simgelenen koruma alanlarına da anahtarlanma hakkına sahip olur.

Görevler işlem yaptıkları bir koruma alanından daha altta bir koruma alanına erişmek istediklerinde bunu işletim sisteminden isterler. Değişik koruma alanlarında yer alan nesnelere erişim, işletim sistemi işlevleri tarafından yerine getirilir. Görevlerin bu işlevleri çağırma hakları, ayrıcalık düzeyleriyle belirlenir. Bilindiği gibi, ana işlem birimi program durum yazmacında, görev ayrıcalık düzeyini belirleyen durum bitleri bulunur. Bir görev ana işlem birimine anahtarlandığında bu bitler, görevin ilgili olduğu ayrıcalık düzey koduyla günlenir. Program durum yazmacındaki ayrıcalık düzey kodu uzunluğu, koruma halkası sayısına bağlıdır. 4 adet koruma halkasının bulunduğu sistemlerde, düzey kodunun içerdiği bit sayısı en az 2, 8 koruma halkasının bulunduğu sistemlerde ise, en az 3 olmak zorundadır. Görevler, bulundukları koruma alanının altındaki bir alana anahtarlanmak için ilgili işletim sistemi işlevini çağırdıklarında, işlevin gerektirdiği ayrıcalık düzeyi ile program durum yazmacındaki düzey kodu karşılaştırılır. Uyumsuzluk durumunda görevin işletimi sonlandırılarak nesnelere erişim genel anlamda denetim altında tutulur[27] (Çizim7.2).

---

[27] *Pentium* işleyicileri, *EFLAG* yazmacı içinde iki bitlik düzey kodu ile 4 adet koruma halkasının oluşturulmasına olanak verir.

İşletimin, örneğin işletim sistemi / kullanıcı olmak üzere iki değişik modda sürdürülebildiği sistemlerde, iç içe iki koruma halkası bulunur. Görevler, iç koruma halkasına, `svc xx (supervisor call), int xx` gibi komutlarla çağrılan işlevler aracılığıyla anahtarlanırlar. Bellek yönetiminde kullanılan tanım çizelgeleri, özel ana işlem birimi yazmaçları gibi, örneğin, sistemin işletim bütünlüğünü doğrudan etkileyen veri yapıları üzerinde yerine getirilmesi gereken kimi kritik işlemler, işletim sistemi işlevleri olarak ele alınırlar. Bu kritik işlevlerden yararlanma hakkı, işletim sistemi modunda çalışma hakkına sahip, başka bir anlatımla iç koruma halkasında yer alan nesnelere erişim hakkı bulunan görevlere açık tutulur. Bu yolla, sıradan kullanıcı görevlerinin, işletim bütünlüğünü etkileyen nesnelere erişimleri engellenir.



Çizim 7.2. Ayrıcalık Düzeyleri ve Koruma Halkaları

7.2.2. Erişim Listeleri

Erişim matrisi modeli çerçevesinde, değişik koruma alanları içinde kümelenen nesnelere erişimin denetlenmesinde kullanılan bir yöntem de erişim listeleri yöntemidir. Erişim listeleri yönteminde her nesne için bir liste tutulur. Bu liste içinde bu nesneye erişim yapabilecek tüm görev ya da kullanıcıların kimlikleri bulunur. Nesneye erişim sözkonusu olduğunda bu liste taranarak ilgili görev ya da kullanıcının bunu yapmaya hakkı olup olmadığı sınanır. Olumsuzluk durumunda erişim engellenerek koruma sağlanır. Bir nesneyle ilgili erişim listesi, bu nesneyi yaratan kullanıcı tarafından oluşturulur. Kullanıcılar, kendilerine ait nesnelerin erişim listelerini istedikleri biçimde günleyerek diğer görev ya da kullanıcıların bu nesnelere erişimlerini, her an açıp kapama hakkına sahip olurlar.

Erişim listeleri, daha çok kütük türü nesnelerle kullanılan bir yöntemdir. Bir kütüğün erişim listesi ya doğrudan ya da göstergesi ile ilgili kütüğün kılavuz kütük satırında yer alır. Kütüğe erişim sözkonusu olduğunda erişim listesinin taranması ve erişim isteminin

geçerli olup olmadığının sınanması gereklidir. Bu yöntemin en önemli sakıncası, kütüklere her erişimde uzun listelerin, denetim amacıyla taranma zorunluluğu ve bu nedenle yitirilen zamandır. Bu sakıncayı aşmak üzere kimi sistemlerde bu denetimin salt kütük açma aşamasında yapılmasıyla yetinilir. *UNIX* gibi sistemlerde ise bu sakınca, erişim listelerinin boyları kısıtlı tutularak aşılır.

Bu amaçla, *UNIX* işletim sisteminde kullanıcılar üç sınıfa ayrılır. Bunlar; kütüğü yaratan kullanıcı *(owner)*, bu kullanıcının yer aldığı kullanıcı grubu *(group)* ve diğer kullanıcılar *(others)* olarak bilinir. Bu durumda bir kütüğün erişim listesi üç grup kullanıcı ile sınırlanır. *UNIX* 'te erişim listeleri *ACL's (access control lists)* olarak adlandırılır. Kütüklere erişim hakları, her sınıf kullanıcı için oku-yaz (`rw`), salt-oku (`r`), işletim (`x`) hakkı biçimindedir. Erişim listeleri:

- `ll kütükadı`
- `lsacl kütükadı`
- `chmod kod kütükadı`
- `chacl 'kullanıcıkimliği = erişimhakkı' kütükadı`

komutlarıyla ekrandan listelenip günlenebilir. `chacl` komutu ile, kimliği belirtilen bir kullanıcının, adı verilen bir kütüğe ilişkin erişim hakları günlenebilmektedir.

```
chacl 'saatcı.bbm = r-x' /usr/lab/uygula
```

komutu ile `bbm` grubundaki `saatci` adlı kullanıcıya, `/usr/lab/uygula` kütüğünün `(rwx)` haklarından, okuma ve işletme (`r-x`) hakları verilebilmektedir. Bunun gibi:

```
lsacl /usr/lab/uygula
```

komutu ile de, `/usr/lab/uygula` kütüğüne erişim yapabilen kullanıcılar, erişim hakları ile, aşağıdaki örnekte olduğu üzere listelenebilmektedir:

```
$ lsacl /usr/lab/uygula
(asistan.%,rwx) (saatci.bbm,r-x) (%.bbm,r--) (%.%,---)
```

Buradan:

- `/usr/lab/uygula` adlı kütüğün sahibi `asistan`'ın kütüğe okuma, yazma ve işletim (`asistan.%,rwx`),
- `asistan`'ın içinde bulunduğu `bbm` grubundaki kullanıcıların salt okuma (`%.bbm,r--`),
- bu gruptan `saatci`'nin hem okuma hem de işletim (`saatci.bbm,r-x`) hakkı ile bu kütüğe erişebilecekleri,
- diğer kullanıcıların ise (`%.%,---`) `/usr/lab/uygula` kütüğüne hiçbir erişim hakları bulunmadığı

anlaşılmaktadır. Dökümde yer alan `%` damgası, “herhangi bir” anlamını taşımaktadır. Örneğin `%.bbm` ifadesi `bbm` grubundaki herhangi bir kullanıcıyı, `%.%` ifadesi ise diğer kullanıcıları göstermektedir.

7.2.3. Görevlerin Yetkilerine Dayalı Erişim Denetimi

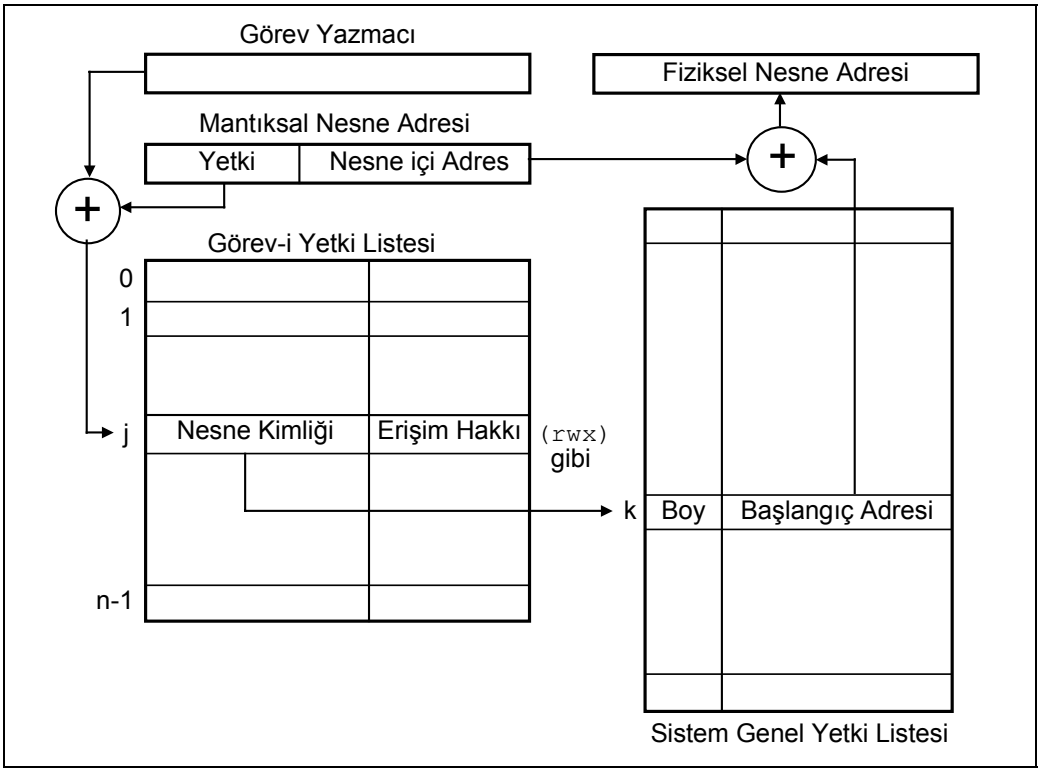
Erişim listeleri yönteminde yapılanın tersine, nesneler için erişim denetim listeleri tutmak yerine, görev ya da kullanıcı tabanında listeler tutmak da düşünülebilir. Bu durumda, örneğin her görevin, hangi nesnelere hangi haklarla erişebileceğinin listesinin tutulması gerekir. Bir görevin belirli bir nesneye erişim hakkına yetki adı verilir. Bu tanıma göre görevlerin, hangi nesnelere hangi haklarla erişebilecekleri bilgisinin tutulduğu listeler yetki listeleri olarak adlandırılır (Çizim 7.3).

Çizim 7.3. Görev Yetki Listesi

Yetki tabanlı sistemlerde, adresleme ve koruma düzeneklerini bütünleşik olarak ele alma ilkesi uygulanır. Bu bağlamda, nesne adreslerinin, yetki ve nesne içi göreli adresten oluşması öngörülür. Adresleme süreci içinde, nesne adresinin yetki birleşeni, işletilmekte olan ilgili görevin yetki listesindeki bir satırı gösterir. Eğer gerçekleştirilmekte olan işlem bu satırda belirtilen erişim hakkıyla uyumlu bir işlem ise, yetki listesinden ilgili nesnenin kimliği alınıp nesne içi göreli adresle birleştirilerek ilgili nesneye erişim sağlanır. Gerçekleştirilen işlem sözkonusu erişim hakkı ile uyumsuzsa, örneğin, görevin ilgili nesneyi günleme hakkı bulunmadığı halde, örneğin bir yazma işlemi yerine getirilmeye çalışılıyorsa adresleme süreci kesilerek görev işletimi sonlandırılır. Açıklanan bu düzenek, kesimli bellek yönetimi kapsamında, hem adres dönüştürme hem de kesime erişimin birlikte denetlendiği düzeneğe benzer. Burada kesimler yerine nesneler sözkonusu olup nesnelere erişimi sağlayan adresleme süreci ile erişim denetimi aynı anda, tek bir düzenekle gerçekleştirilir (Çizim 7.4).

Yetki tabanlı sistemlerde nesneler, bir ana bellek kesimi ya da bir kütük gibi, hem ana bellekte hem de ikincil belleklerde saklanan ögeleri temsil edebilirler. Böylece bu sistemlerde ana belleğe ve ikincil belleklere erişim ve bu erişimin denetimi bütünleşik olarak gerçekleştirilir. Bunun yanı sıra, bu sistemlerde yetki, nesne adresinin içinde yer aldığından aynı görevin değişik yordamlarına farklı yetkiler verilebilir. Bu da aynı görevin değişik ayrıcalık düzeylerinde çalışabilmesine olanak verir. Diğer koruma düzeneklerinde bir görevin değişik kesimleri için değişik ayrıcalık düzeyleri tanımlama olanağı bulunmaz.

Görevlere verilen yetkilerin sonradan geri alınmasında karşılaşılan zorluklar, yetki tabanlı sistemlerin temel sakıncasını oluşturur. Yetki tabanlı sistemlerde bir görev, yeni bir nesne yarattığında bu nesne için, işletim sistemi tarafından bir yetki belirlenir. Yeni nesne görevin yetki listesine, belirlenen erişim haklarıyla birlikte eklenir. Bunun gibi bir nesnenin, değişik erişim haklarıyla başka görevlerin yetki listelerine eklenmesi de söz konusu olabilir. Bu yolla değişik yetki listelerine dağılmış bir nesnenin, sonradan kimi listelerden silinmesi ya da erişim haklarının değiştirilmesi kolay değildir. Zira silinecek ya da erişim hakkı günlenecek nesnenin hangi listelerde yer aldığı belirleyebilmek üzere sistemdeki tüm yetki listelerinin taranması gibi, gerçekleştirilmesi hemen hemen olanaksız bir zorunlulukla karşı karşıya kalınacaktır. Bu sakıncayı aşmanın benimsenebilir kolay bir yolu da yoktur. Yetki tabanlı koruma ve adresleme düzeneğini kullanan, tanınmış ticari bir işletim sistemi bulunmamaktadır.

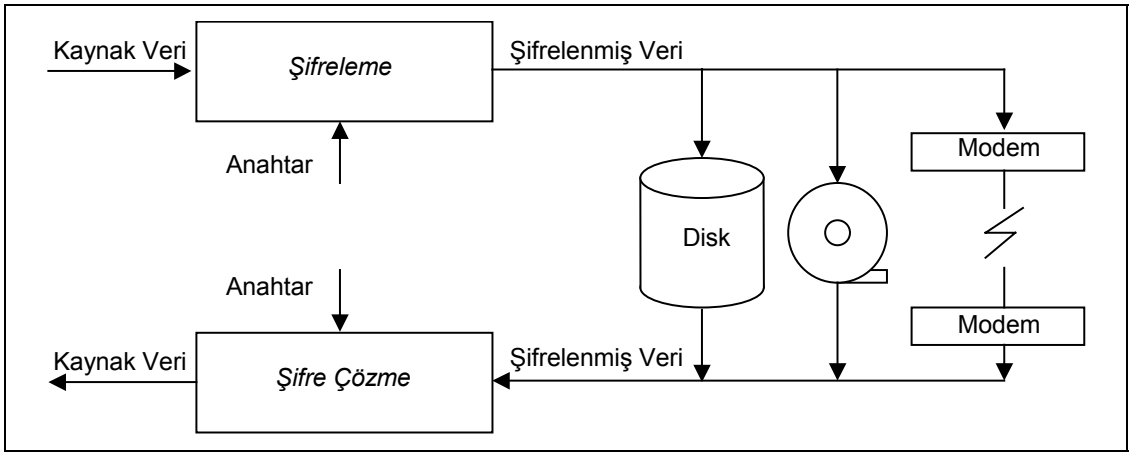


Çizim 7.4. Yetki Tabanlı Sistemlerde Adresleme Süreci

7.3. Şifreleme

Şifreleme, verileri izinsiz olarak ele geçirenlerin bunlardan yararlanmalarına engel olarak güvenliği artırmayı sağlayan bir yöntemdir. Şifreleme, verilerin, belirli bir işlev ve bu işlevin kullandığı bir anahtara göre dönüştürülmesi demektir. Şifrelemenin gerçekleştirilmesinde yararlanılan model, Çizim 7.5'te verilmiştir. Bu modele göre kaynak veriler anahtar olarak adlandırılan bir parametre ile işleme sokularak şifrelenmiş veri elde edilmektedir. Şifrelemenin uygulandığı sistemlerde veriler, disk, mıknatıslı şerit gibi ortamlarda şifrelenmiş biçimleriyle saklanırlar. Bir sistemden

diğerine şifrelenmiş biçimleriyle aktarılırlar. Şifrelenmiş veriden kaynak veriye geri dönüş şifre çözme olarak adlandırılır.



Çizim 7.5. Verilerin Şifrelenmesi

Şifrelemede sözkonusu edilen işlev ve anahtar kavramlarını açıklamak üzere aşağıda bir şifreleme örneği verilmiştir. Bu örneğe göre kaynak veri, alfabetik sıraya göre *BCD* kodlanmış bir metindir. Şifreleme işlevi olarak, damgalar arası dışlayan ya da işlevi kullanılmıştır. Anahtar ise, kaynak metin ile dışlayan ya da işlemine sokulan `ECE` sözcüğüdür.

Kaynak Veri : `İŞLETİM-SİSTEMLERİ`
Anahtar : `ECEECEECEECEECEECE`
Şifrelenmiş Veri : `KPI-VKHCTKRS-LI-SK`

Kodlanmış olarak:

Kaynak Veri : `12 23 15 06 24 12 16 00 22 12 22 24 06 16 15 06 21 12`
Anahtar : `06 03 06 06 03 06 06 03 06 06 03 06 06 03 06 06 03 06`
Şifrelenmiş Veri : `14 20 11 00 27 14 10 03 24 14 21 22 00 15 11 00 22 14`

Örnekte, kaynak veri olarak seçilen `İŞLETİM-SİSTEMLERİ` metnini oluşturan damga kodları, anahtar sözcüğün (`ECE`) damga kodlarıyla, birebir "dışlayan ya da" işlemine sokulmaktadır. Bu bağlamda:

- `İ` damgasının kodu olan `12(00010010B)` değeri ile `E` damgasının kodu olan `06 (00000110B)` değeri arasında,
- `Ş` damgasının kodu olan `23(00100011B)` değeri ile `C` damgasının kodu olan `03 (00000011B)` değeri arasında,
- `L` damgasının kodu olan `15(00010101B)` değeri ile `E` damgasının kodu olan `06 (00000110B)` değeri arasında,
- **...**

"dışlayan ya da" işlemi uygulanmaktadır. Bunun sonucu olarak:

`14 20 11 00 27 14 10 03 24 14 21 22 00 15 11 00 22 14`

değerlerini içeren kod dizisi elde edilmektedir. Bu kod dizisine karşı gelen damga dizisi, başka bir deyişle şifrelenmiş veri, K P I - V K H C T K R S - L I - S K damgalarından oluşmaktadır.

Yukarıda verilen örnekte olduğu gibi, şifreleme ve şifre çözmede kullanılan anahtar aynı ise "simetrik" şifrelemeden söz edilir. Simetrik şifrelemenin veri iletişim amacıyla kullanıldığı durumlarda şifreleme anahtarının vericiden alıcıya güvenli bir biçimde nasıl aktarılacağı sorunu ortaya çıkar. Bu sorunun aşılabilmesi amacıyla "simetrik olmayan şifreleme yöntemi kullanılır. Bu yöntemde, gönderici ve alıcıların herbirinin şifreleme ve şifre çözme amacıyla kullandıkları, özel *(private)* ve ortak *(public)* olarak adlandırılan iki değişik anahtar bulunur. Bunlardan şifreleme amacıyla kullanılan ortak anahtar, herkese açık tutulur. Gönderici alıcıya herhangi bir veri aktarmadan önce alıcının "ortak" şifreleme anahtarını elde eder. Bu anahtara göre şifrelemeyi yaparak veri aktarımını gerçekleştirir. Alıcı, kendisinden başka kimsenin bilmediği "özel" şifre çözme anahtarını kullanarak şifrelenmiş veriyi çözer. Ortak şifre anahtarından şifre çözme anahtarını tahmin etmek olanaksızdır. Şifreleme amacıyla kullanılan anahtarın alıcıdan talep edildiği bu yöntem, şifreleme anahtarının herkesin bilgisine açık olması dolayısıyla, İngilizcede *Public Key* (ortak anahtar) şifreleme yöntemi olarak adlandırılır.

7.4. Bilgisayar Virüsleri

Başka programların içine yerleştirilip bunların çalıştırılması sonucu gizlice çalışmaya başlayarak bilgisayar sistemine zarar vermeyi amaçlayan özel programlar bilgisayar virüsleri olarak adlandırılmaktadır. Bu adlandırmanın temel gerekçeleri şunlardır:

- Bu programlarlar, çalışabilmek için mutlaka başka bir programın varlığına gereksinim duyan ve ancak, başka bir program içine yerleşerek çalışabilen programlardır.
- İçine yerleştikleri program aracılığıyla, bir kez, bir bilgisayar sistemine girdikten sonra, kendi kendilerini sistemdeki diğer programlar içine de kopyalayarak çoğalabilmekte ve bunlar aracılığıyla diğer bilgisayar sistemlerine de taşınabilmektedirler.
- Çoğu kez, çalıştıkları bilgisayar sistemine; kütükleri silme, işletim sistemini bozma yoluyla zarar vermektedirler.

Bilindiği gibi, bakterilerden daha küçük, asalak canlılar olan biolojik virüsler de:

- Yalnızca canlı bir hücreye girip yerleştikten sonra çoğalabilmekte,
- Bitki, hayvan ve insan gibi canlılara, birbirlerinden bulaşmakta ve
- İçine yerleştikleri hücreleri bozmaktadırlar.

Bu temel benzerlikler nedeniyle kötü niyetli bilgisayar programları bilgisayar virüsleri olarak adlandırılmaktadır. Adlandırmadaki bu koşutluğun yanı sıra, bilgisayar virüsleriyle ilgili diğer terimlerde de biyolojik virüs terimlerinden yararlanılmaktadır. Bu bağlamda bilgisayar virüslerinin bulaşmasından, virüs salgınlarından, bilgisayar

sistemlerinin bunlara karşı bağışıklık kazanmasından, bağışıklık sağlayan koruyucu programlardan, aşılardan söz edilebilmektedir.

Bilgisayar virüsleri, bilgisayar sistemlerine:

- disket, mıknatıslı şerit gibi birimlerle ya da
- bilgisayarın bağlı bulunduğu ağ üzerinden

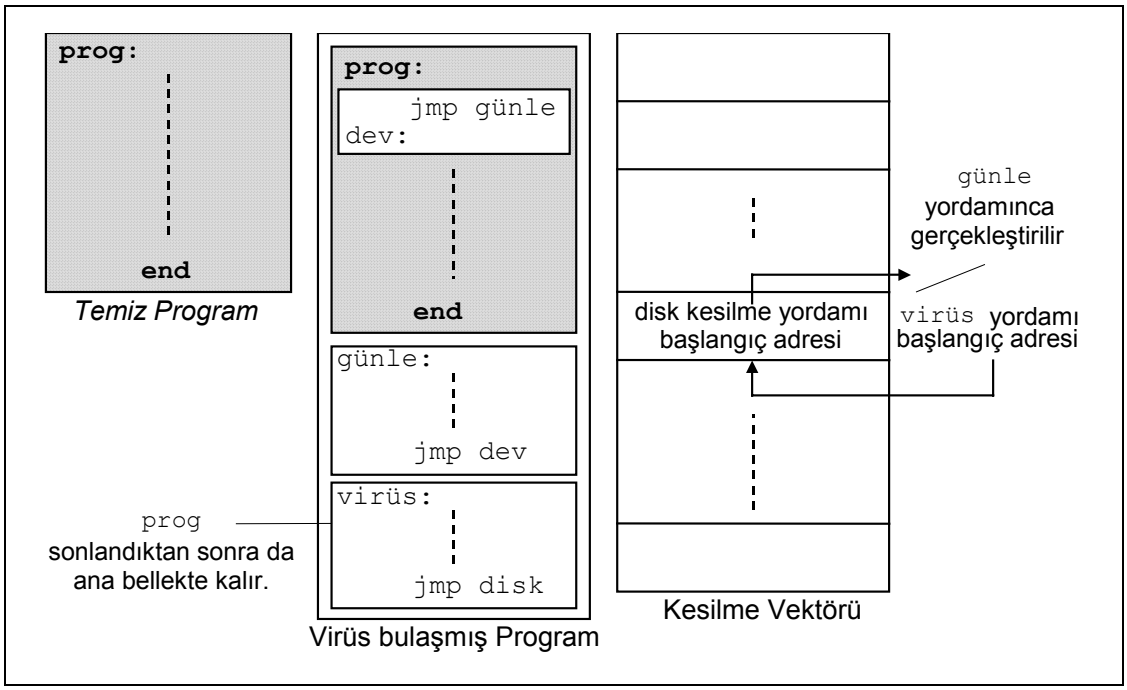
kopyalanan programlar aracılığıyla taşınmaktadır. Virüsü taşıyan bir program bir bilgisayar sistemine girdikten sonra, bu programın çalışmasıyla birlikte çalışmaya başlamaktadır. Bu bağlamda, virüs programı, öncelikle bir kopyasını, sistemdeki temiz programlar içine yerleştirmektedir. Bunun sonrasında, içerdiği mantığa göre, bulunduğu sisteme zarar vermek üzere, kimi kütüklerin kimi öbeklerini silme, kütük atama çizelgelerindeki kimi bağları koparma, işletim sistemine ilişkin kimi veri yapıları üzerinde değişiklik yapma gibi zararlı işlemleri yerine getirmektedir. Virüs programları, sistemdeki varlıklarını kamufle edebilmek amacıyla, etkilerini, çalışmaya başlar başlamaz göstermemekte ve verecekleri zararı genelde, bir seferde olmak yerine azar azar vermektedirler. Bu amaçla, ilk çalıştıklarında, örneğin disk kesilme vektörünü, içerdikleri bir yordam adresine kurmakla yetinmekte, bu yordamın, ancak belirli bir sayıda çağrılması sonrasında etkilerini azar azar göstermeye başlamaktadırlar. Bu yolla ortaya çıkarılmalarını geciktirerek sistemde daha uzun süre kalma ve böylece daha çok çoğalma ve yayılma fırsatı bulmaktadırlar.

Çizim 7.6'da verilen örnekte, `prog` adlı bir programın virüs bulaşmış görünümü verilmiştir. Burada virüs programı, `günle` ve `virüs` adlı iki kesimden oluşmaktadır. Virüs programı kendini `prog` adlı programın sonuna kopyaladıktan sonra bu programın başına `jmp günle` komutu ile `dev:` etiketini eklemektedir. Bu yolla `prog` çalıştırıldığında, ilk iş olarak `günle` kesimine sapılmakta ve disk kesilme vektörü `virüs` adlı yordamın başlangıç adresi ile günlenmektedir. Böylece disk arabiriminden her kesilme gelişte, denetim `virüs` yordamına geçmektedir. `virüs` yordamı işletildikten sonra `disk` yordamına sapılarak sistemin görünürdeki işleyişi bozulmamaktadır. Bu yolla `virüs` yordamının sisteme verdiği zarar kullanıcıdan gizlenebilmektedir.

Bilgisayar virüslerine karşı tam koruma sağlayacak ne genel bir yöntem ne de bir araç, ne yazık ki bulunmamaktadır. Bunlarla başa çıkmada geçerli olabilen hemen hemen tek yol, koruyucu önlemlere özen göstermekten geçmektedir. Sisteme yeni bir program yüklemek gerektiğinde bu programın kaynağını iyi bilmek gerekmektedir. Programlarının kopyalanarak bedava kullanılmasından bıkan yazılım üreticilerinin, izinsiz kopyalama durumunda etkili olabilen virüs programlarını kullandıkları bilinmektedir. Bu yolla bedel ödemeden program kullanan kişileri, bir biçimde cezalandırmayı amaçlamaktadırlar.

Önemli kütüklerin düzenli olarak yedeklenmesi de virüslerin yol açacağı bozulmaların yaratacağı zararları en aza indirmek için geçerli bir önlemdir. Bu tür önlemlerin yanı sıra virüs tarama, virüs izleme ve bütünlük denetleme programlarının kullanılması da yararlı olmaktadır. Virüs tarama programları bilinen kimi virüslerin, örneğin ilk 50

bayttan oluşan örüntülerini tüm program kütükleri içinde tarayarak varlıklarını belirlemeye yarayan programlardır. Virüs izleme programları sistemin çalışmaya başlamasıyla birlikte işletime giren ve virüslerin gerçekleştirebileceği türden şüpheli işlemlerin yapılıp yapılmadığını izleyen programlardır. Bütünlük denetleme programları ise sistemdeki kimi program kütüklerinin denetim toplamlarını (*checksum* değerlerini) dönem dönem hesaplayarak ilk değerleriyle karşılaştıran ve bu yolla bu kütüklerin içeriklerinin değişip değişmediğini denetleyen programlardır. Günümüzde, kişisel bilgisayar sistemleri için binlerce virüs programı bulunmaktadır. Bunlara hergün yenilerinin eklendiği ve yeni geliştirilen virüs programlarının, alınan önlemler gözetilerek tasarlandığı düşünülecek olursa tarama, izleme ve denetleme programlarının etkilerinin kısıtlı kalacağı kolayca anlaşılır.



Çizim 7.6. Bilgisayar Virüslerinin Çalışma Biçimini Açıklayan Bir Örnek

Bilgisayar virüslerinin yanı sıra, bir de bilgisayar kurtları bulunmaktadır. Bilgisayar kurtları, virüslerin tersine veri bütünlüğüne zarar veren programlar değildir. Bilgisayar kurtları ilk kez, bilgisayar ağlarının aşırı yüklendiği durumları öykünebilmek için, araştırma amacıyla ortaya çıkmıştır. Araştırma amacıyla belirlenen çalışma ilkeleri sonradan kötü niyetli uygulamalara taban olmuştur. Bu programların amaçları, içine girdikleri bilgisayar sistemlerinde hızla çoğalarak, ana işlem birimi, ana bellek, giriş/çıkış kanalı, ağ bağlantıları gibi sistem kaynaklarını aşırı bir biçimde tüketerek sistemi aşırı yüklemektir. Bilgisayar kurtları veri bütünlüğüne zarar vermemekle birlikte, sistem kullanılırlığını olumsuz yönde etkileyerek güvenliğe karşı bir tehdit oluştururlar.

8. BÖLÜM

İ Ş L E T İ M S İ S T E M L E R İ

AYGIT SÜRÜCÜLER

Giriş/Çıkış Sistemi adlı İkinci Bölümde, Giriş/Çıkış Sistemiyle ilgili olarak, giriş/çıkışların programlanmasında kullanılan programlama teknikleri ile bunların gerektirdiği donanım altyapısı açıklanmıştır. Sözkonusu tekniklerin kullanımı, anlatım kolaylığı sağlayan yalın ve adanmış örnek sistemler taban alınarak ve işletim sisteminin bütünü ve katmanlı mimarisi gözetilmeden örneklenmiştir. Açıklanan programlama yöntemleri kullanılarak gerçekleştirilen giriş/çıkış hizmet ve kesilme yordamlarının, günümüz işletim sistemlerinin bütünü içinde, çekirdek katman düzeyinde nasıl ele alındıkları, *UNIX/LINUX* işletim sistemi taban alınarak, bu bölümde açıklanıp örneklenecektir.

İzleyen kesimde de görüleceği üzere, Aygıt Sürücüler konusunun Kütük Yönetimi, Görev Yönetimi, Birlikte Çalışan Görevler başlıkları altında açıklanan önemli kavramları da sözkonusu etmesinden dolayı, bu konuya, doğal uzantısı olduğu Giriş/Çıkış Sistemi adlı Bölümün sonunda yer vermek yerine, anılan kavramlar açıklandıktan sonra, ayrı bir başlık altında bu bölümde yer verilmiştir.

Aygıt sürücü kavramı *UNIX* işletim sistemi ile ortaya çıkmış bir kavramdır. Bilindiği gibi, uygulama programları işletim sisteminden gereksedikleri hizmetleri, sistem çağrılarını kullanarak alırlar. Sistem çağrıları, daha genel anlamda, işletim sistemini oluşturan değişik katmanlar arası hizmet alma düzeneğini oluşturur. Bu bağlamda, bir uygulama programı, örneğin simgesel kimliği ile andığı bir kütük üzerinde, kütük başına göreli belirli sayıda baytın, örneğin okunmasını, kütük yönetim sistemi olarak

adlandırılan işletim sistemi katmanının `read()` sistem çağrısını kullanarak bu katmandan ister. Kütük yönetim sistemi, sözkonusu belirli sayıda baytın yer aldığı öbeğin, örneğin sürücü (aygıt), silindir, kafa, sektör cinsinden fiziksel konumunu (adresini) belirledikten sonra, öbeğin, fiziksel olarak ilgili aygıt üzerinden okunmasını, bu kez çekirdek katmana özgü bir sistem çağrısıyla bu katmandan ister. Fiziksel okuma işlemi, ilgili giriş/çıkış birimine ilişkin, çekirdek katman içindeki sürücü yordamlarca gerçekleştirilir. Okunan öbeğin içindeki ilgili baytlar ya doğrudan ya da kütük yönetim sistemi aracılığıyla uygulama programına aktarılarak işlem tamamlanır.

Bir bilgisayar sisteminin çok çeşitli giriş/çıkış birimi bulunur. Bunlar üzerinde, okuma, yazma gibi fiziksel işlemler, yukarıda da belirtildiği gibi, ilgili giriş/çıkış birimine özgü, çekirdek katman içinde yer alan sürücü yordamlar aracılığıyla yerine getirilir. Sözkonusu sürücü yordamların herbirinin, ilgili giriş/çıkış birimine özgü bir yapıda olması, çekirdek katmandan hizmet alan üst katmanların her giriş/çıkış birimine özgü ayrı bir sistem çağrı kümesini kullanmasını gerektirir. Bunun yerine, giriş/çıkış biriminin niteliği ne olursa olsun, tüm giriş/çıkış birimleri için aynı sistem çağrı kümesinin kullanılması, üst katmanları, donanımın görünümünden ve çekirdek katmandan bağımsızlaştırır. Sisteme yeni giriş/çıkış birimlerinin eklenmesini kolaylaştırır. Bu, giriş/çıkış birimlerinin standart bir görünümle ele alınmasıyla gerçekleşir.

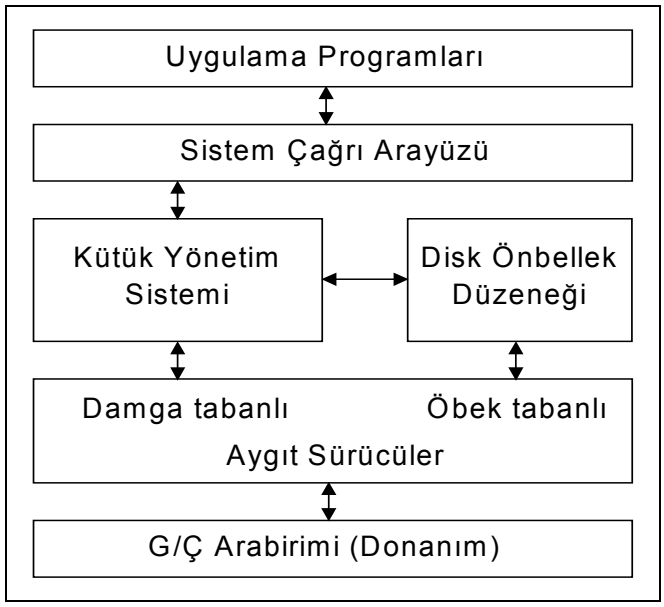
UNIX işletim sisteminde sözkonusu standart görünüm, tüm giriş/çıkış birimlerinin kütükler gibi ele alınmasıyla sağlanır. Başka bir deyişle, tüm giriş/çıkış birimlerine erişim, kütüklere erişimde kullanılan, klasik `open()`, `close()`, `read()`, `write()`, `ioctrl()` gibi tek bir sistem çağrı kümesiyle gerçekleşir. Bunun sonucunda, örneğin klavyeden girilen damga kodlarının okunmasıyla, bir disk üzerinde saklanan baytların okunması arasında, kullanılan sistem çağrıları yönünden bir ayrım kalmaz. Bu amaçla, sistemdeki her giriş/çıkış birimi için, `open()`, `close()`, `read()`, `write()`, `ioctrl()` gibi, çekirdek katman düzeyi ortak sistem çağrıları, sözkonusu giriş/çıkış birimine özel ayrı bir kütük belirteci (*file descriptor*) ile işletildiğinde gerekseme duyulacak sürücü yordamların çekirdek katmana katılması gereklidir. Bir giriş/çıkış birimine özgü sürücü yordam kümesi, *UNIX* bağlamında *Device Driver* ya da aygıt sürücü olarak adlandırılır. İzleyen kesimde böyle bir yordam kümesinin, *LINUX*'te hangi yordamlardan oluştuğu ve nasıl ele alındığı açıklanacaktır.

8.1. Aygıt Türleri

LINUX (*UNIX*) işletim sistemi, giriş/çıkış birimlerini, damga ve öbek tabanlı olmak üzere iki kategoride ele alır. Klavye, yazıcı gibi aygıtlar damga tabanlı, disk, disket, *CD-ROM* gibi aygıtlar ise öbek tabanlı aygıtlardır. Öbek 512, 1024, 2048 gibi sayıda baytın birlikte oluşturduğu bütüne verilen addır. Adlarından da anlaşılacağı üzere damga tabanlı aygıtlar arabirimleri ile iletişimlerinde damgayı, öbek tabanlı aygıtlar ise öbeği taban alırlar. Başka bir deyişle, damga tabanlılarda bir seferde aygıt-arabirim arasında aktarılan veri birimi bir baytan, öbek tabanlılarda ise bir öbekten oluşur. Bu durumda, iki aygıt türü için, damga tabanlı aygıt sürücüler *(character device drivers)* ve

öbek tabanlı aygıt sürücüler *(block device drivers)* olmak üzere iki tür aygıt sürücü bulunur.

Damga tabanlı aygıt sürücülerde, uygulama programlarının kullandığı sistem çağrıları, aygıt sürücü içinde tanımlı, aygıta fiziksel erişim yapan yordamlarla doğrudan ilişkilidir. Öbek tabanlı aygıt sürücülerde ise, okuma ve yazma işlemleri, aygıta doğrudan erişmek yerine, Kütük Yönetimi başlıklı kesimde konu edilen ve öbek tabanlı giriş/çıkış birimlerine erişimde, erişim süresini kısaltmak, dolayısıyla sistem başarımını yükseltmek üzere, son erişilen öbekleri ana bellekte tutan disk ön bellek *(buffer cache)* düzeneğine erişim yapar (Çizim 8.1).



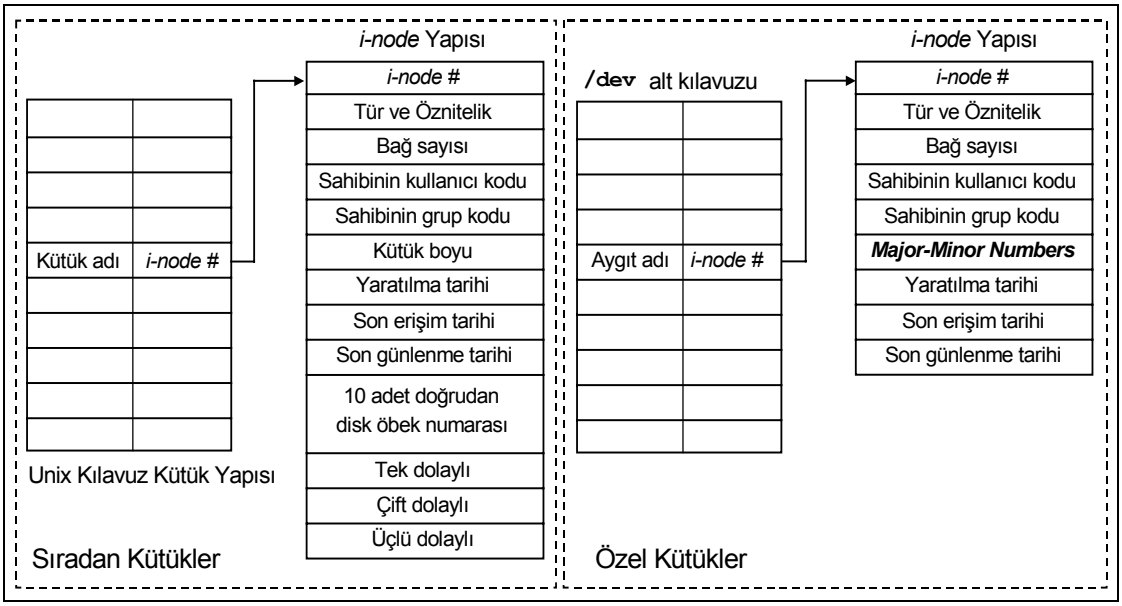
Çizim 8.1. Damga ve Öbek Tabanlı Aygıt Sürücülerin Konumu

LINUX (*UNIX*)'te aygıt sürücülerin temel görevleri şunlardır:

- İşletime başlama aşamasında (sistem açıldığında) ilgili donanımın ön belirlemesinin yapılması
- İlgili aygıtı hizmete açma-kapama *(open-close)* işlemlerinin yapılması
- Aygıttan çekirdek katman ortamındaki yastık alanına okuma
- Yastık alanından aygıta yazma
- Donanım hatalarının saptanması ve ele alınması

Yukarıda da belirtildiği gibi, *LINUX* (*UNIX*) altında çalışan bir bilgisayar sisteminde her giriş/çıkış birimi (başka bir deyişle aygıt) bir kütük gibi ele alınır. Bu nedenle, her aygıtın, bir kütük gibi, bir simgesel kimliği (başka bir deyişle adı) ve kılavuz kütük sıradüzeni içinde bir konumu (başka bir deyişle erişim izi) bulunmak durumundadır. Aygıtlarla ilişkilendirilmiş bu sanal kütükler, özel kütükler *(special files)* olarak anılırlar. Türlerine göre damga tabanlı ve öbek tabanlı özel kütüklerden *(character/ block special files)* söz edilir. Bu sanal kütükler, işletim sisteminin `/dev` alt kılavuzu

altında yer alırlar. Bir kullanıcı programı, herhangi bir aygıta erişmek istediğinde, kütüklerde olduğu üzere, önce `open()` sistem çağrısı ile, aygıtın simgesel kimliğinden *file descriptor* adlı bir kütük belirteci elde eder. Aygıta ilişkin okuma-yazma işlemlerinde bu belirteci kullanır. Sözkonusu bu belirteç, bir kütük olarak ele alınan aygıtın, *LINUX* (*UNIX*) bağlamında `i-node` olarak bilinen ve diskte saklanan veri yapısının, `open()` sistem çağrısı ile ana belleğe taşınan kopyasının göstergesidir. Sıradan kütüklerde *(ordinary files)* kütüğün fiziksel disk öbek adreslerini tutan `i-node` veri yapısı, özel kütüklerde aygıta erişimi sağlayan yordam kümesinin göstergesini *(major-minor device number)* saklar (Çizim 8.2).



Çizim 8.2. *LINUX* (*UNIX*)'te `i-node` Yapıları

`/dev` alt kılavuzu listelendiğinde aşağıdakine benzer bir liste elde edilir:

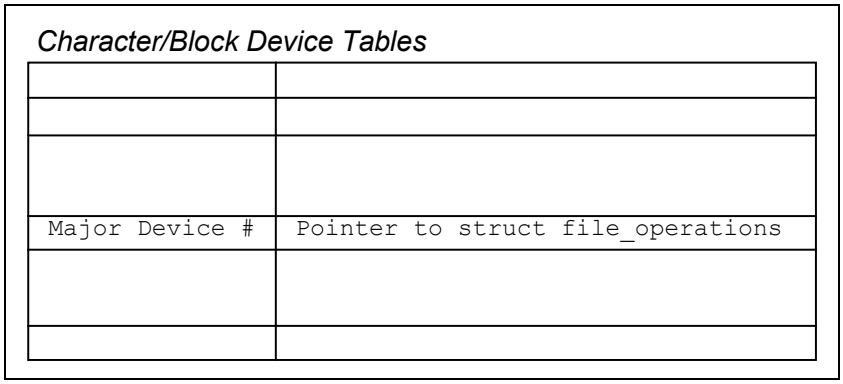
```
.
.
brw-------  1 root  floppy   2, 0  May  5 1998  /dev/fd0
brw-rw----  1 root  disk     3, 1  May  5 1998  /dev/hda1
crw-rw-rw-  1 root  sys      14,1  Apr 17 1999  /dev/sequencer
.
.
```

Burada her satırın başındaki ilk harf ilgili aygıtın damga tabanlı mı, ya da öbek tabanlı mı olduğunu gösterir. `b` öbek tabanlıyı, `c` ise damga tabanlıyı gösterir. Sıradan kütükler için kütük boyunun gösterildiği yerde (5 ve 6. konumlarda `2,0` gibi) iki sayı bulunmaktadır. Bunlardan ilki *major device number,* ikincisi ise *minor device number* olarak anılır. Bilgisayar sistemini oluşturan değişik nitelikteki herbir aygıt için ayrı bir aygıt sürücü öngörülür. Sistemde aynı nitelikte (örneğin 1.44 MB Disket Birimi gibi) birden çok aygıt var ise, bu aygıtların tümü için aynı aygıt sürücü kullanılır. Bu

durumda *major device number* kullanılan aygıt sürücüyü (başka bir deyişle aynı nitelikteki aygıt kümesini), *minor device number* ise aynı aygıt sürücüyü kullanan özel bir aygıtı gösterir.

8.2. Aygıt Sürücü Yordamları

Çekirdek katman içinde iki çizelge tutulur. Bu çizelgelerden birisi damga tabanlı, diğeri de öbek tabanlı aygıt sürücülere ilişkin `struct file_operations` olarak bilinen veri yapılarının göstergelerini *(pointers)* tutar. Herhangi bir aygıtın *major device number* adlı parametresi, aslında, bu çizelgelerde bir satırı gösteren dizin değeridir (Çizim 8.3).



Çizim 8.3. Damga ve Öbek tabanlı Aygıt Çizelgelerinin Yapısı

`struct file_operations` olarak bilinen veri yapısı, ilgili aygıtın sürülmesinde kullanılan ve çekirdek katmanda yer alan yordamların göstergelerini *(functions' pointers)* tutan bir veri yapısıdır. Bu veri yapısının *LINUX* 2.2 numaralı çekirdek katman sürümü için geçerli görünümü Çizim 8.4'te verilmiştir. Burada yer alan işlevlerin tamamı, tüm aygıtlar için gerekli olmayabilir. Eğer bir aygıt için, örneğin `llseek()` işlevi gerekli değilse, ilgili satıra, `NULL;` yazmak yeterli olmaktadır.

`struct file_operations` adlı yapı içinde tanımlı işlevler, uygulama programları içinde aynı ya da benzer adlarla yer alan sistem çağrısı nitelikli işlevler çalıştırıldığında işletilen çekirdek katman düzeyi işlevlerdir. Bir uygulama programı, kütük gibi ele alabildiği `aaa` adlı bir aygıta yönelik, örneğin `open(aaa,O_RDWR)` işlevini çalıştırdığında, çekirdek katmanda bununla ilgili çağrılan işlev, `struct file_operations` adlı yapı içinde tanımlı `open()` işlevi olmaktadır.

8.3. Aygıt Sürücü Yordamlarının İşletimi

Bir kullanıcı programı, işletimi sırasında, bir aygıta erişmek üzere, bir sistem çağrısı yaptığında, yukarıda açıklanan düzenek çerçevesinde, kütük yönetim sistemi aracılığıyla, `struct file_operations` içinde tanımlı ilgili bir işlevi (çekirdek katman düzeyi bir sistem çağrısını) çalıştırır. Bu aşamada, sözkonusu kullanıcı programına ilişkin görevle ilgili iki değişik yol izlenebilir. Bu yollardan birincisi,

işletilen görev bağlamında, çağrılan işleve doğrudan sapmaktır. Bu durumda, çağrılan işlevin adres evreni, çağrıyı yapan kullanıcı görevinki ile aynı olur. Varsa her türlü aktarım, aygıt ile program adres evrenleri arasında doğrudan gerçekleşir.

```
struct file_operations {
loff_t (*llseek)(struct file*, loff_t, int);
ssize_t (*read)(struct file*, char*, size_t, loff_t*);
ssize_t (*write)(struct file*, const char*, size_t, loff_t*);
int (*readdir)(struct file*, void*, filldir_t);
unsigned int (*poll)(struct file*, struct poll_table_struct*);
int (*ioctl)(struct file*, unsigned int, unsigned long);
int (*mmap)(struct file*, struct vm_area_struct*);
int (*open)(struct inode*, struct file*);
int (*flush)(struct file*);
int (*release)(struct inode*, struct file*);
int (*fsync)(struct file*, struct dentry*);
int (*fasync)(int, struct file*, int);
int (*check_media_change)(kdev_t dev);
int (*revalidate)(kdev_t dev);
int (*lock)(struct file*, int, struct file_lock*);
};
```

Çizim 8.4. `struct file_operations` (*LINUX* Çekirdeği V2.2)

Ancak aygıtla kullanıcı programı arasında veri (bayt) aktarımını konu eden sistem çağrılarının işletimi, çoğu kez, giriş/çıkış birimlerinin ana işlem birimine göre aşırı yavaş olmaları nedeniyle göreli çok uzun zaman alır. Bu sürede, çağrıyı yapan görevin ana işlem birimine anahtarlı kalması sistem başarımını çok olumsuz yönde etkiler. Bu nedenle böyle bir çağrı yapıldığında, görev yönetici[28] olarak adlandırılan ve ana işlem biriminin yönetiminden sorumlu sistem görevi, çağrıyı yapan kullanıcı görevini, yaptığı çağrı (çağırdığı işlev) sonlanıncaya değin, geçici olarak bekler durumuna alır. Başka bir görevi ana işlem birimine anahtarlar. Beklemeye alınan görevin yeniden ana işlem birimine anahtarlanarak çalışabilmesi için, yaptığı çağrının sonlandığını gösteren bir kesilmenin üretilmesi gerekir. Giriş/Çıkış Sistemi başlıklı kesimde örneklendiği gibi, arabirim ya da doğrudan bellek erişim denetleme birimi gibi bir birimden, işlemlerin sonlandığına ilişkin kesilme geldiğinde, görev yönetici, bekler duruma aldığı sözkonusu görevi, ana işlem birimine anahtarlanmaya hazır görev durumuna sokar. Sırası geldiğinde ana işlem birimine anahtarlanan sözkonusu görev, işletimini, beklemesine neden olan sistem çağrısını izleyen komuttan başlayarak sürdürür. Bu durumda, aygıt-program arası veri aktarımı doğrudan yapılamaz. Aktarım, örneğin okuma için, önce

[28] Bu konu Görev Yönetimi adlı bölümde ayrıntılı olarak açıklanmaktadır.

aygıttan işletim sistemi bağlamında bir yastığa, oradan da, program adres evreninde bu amaçla öngörülmüş yastık alana aktarılır.

8.4. Kesilme Yordam(lar)ı

Yukarıda açıklandığı üzere, giriş/çıkış arabirimleri, sürücülere dönük aktarım işlemlerinin sonlanması durumunda kesilme imi üreterek ana işlem birimini uyarırlar. Bir aygıta yönelik, varsa bu tür uyarıları ele alan kesilme yordamları da, aygıt sürücü kapsamında bulunmak zorundadır. Başka bir deyişle, `struct file_operations` yapısı içinde tanımlı yordamların dışında, bu tür yordamların da, aygıt sürücü kapsamında öngörülmesi gereklidir.

Aygıt sürücü kapsamında kesilme yordamları kullanılacak ise, öncelikle bu yordamların, örneğin *PC* donanımı[29] içinde yer alan kesilme önceliği denetleme biriminin hangi kesilme girişine *(IRQ)* ilişkin olduğunun belirlenmesi gereklidir. Bu ilişkilendirme, *LINUX* Çekirdek V2.2 için:

```
int request_irq(unsigned int irq, void (*handler)(int),
                unsigned long type, char* name);
```

sistem çağrısıyla yapılmaktadır.

Burada `irq` *IRQ (Interrupt ReQuest)* numarasını,
`handler` kesilme yordamının başlangıç adresini,
`type` kesilmenin *Intel 80X86* işleyicileri için türünü,
`name` aygıt sürücünün adını
göstermektedir.

Bir kesilme yordamıyla ilişkilendirilmiş bir kesilme girişinin serbest bırakılması:

`void free_irq(unsigned int irq);` sistem çağrısı ile mümkün olabilmektedir.

8.5. Aygıt Sürücü Özel Kütüğünün Yaratılması

Bir aygıt sürücüye ilişkin özel kütüğün sistemde varlık bulması (yaratılması) `mknod` komutu ile gerçekleşir.

```
> mknod /dev/aygıtsürücüm c 30,0
```

komutu, damga tabanlı `aygıtsürücüm` adlı, *major device* numarası 30, *minor device* numarası da 0 olan yeni bir aygıt sürücüye ilişkin özel kütüğün `/dev` alt kılavuzu altında yaratılmasını gerçekleştirir.

Bu biçimde yaratılan aygıt sürücü özel kütüğünün *major device* numarasının, Çizim 8.3'te örneklenen ilgili çizelgeye eklenerek bu numaranın ilgili `struct file_operations` adlı veri yapısıyla ilişkilendirilmesi, niteliği (damga ya da öbek tabanlı) ne olursa olsun tüm aygıt sürücülerde bulunması gereken `init()` işlevi bağlamında gerçekleştirilir. Bu ilişkilendirme işlemine kayıt *(registration)* işlemi denir.

---

[29] Burada verilen açıklamalar, standart bir kişisel bilgisayar *(PC)* üzerinde çalışan *LINUX* işletim sistemi bağlamında geçerlidir.

`init()` işlevi ve bu bağlamda yapılan kayıt işlemi, izleyen kesimde, damga tabanlı aygıt sürücüler kapsamında açıklanmıştır.

8.6. Damga Tabanlı Aygıt Sürücüler

Bu kesimde damga tabanlı aygıt sürücülerin yapıları ve nasıl gerçekleştirildikleri açıklanacaktır. Daha önce de belirtildiği üzere, bir aygıt sürücü:

- `init()` işlevi,
- `struct file_operations` adlı veri yapısı içinde tanımlı işlevler ve
- gerekli ise, ilgili arabirime ilişkin kesilme yordam(lar)ından

oluşan bir bütündür. İzleyen kesimde bu işlev ve yordamların damga tabanlı aygıt sürücüler için tanım ve özellikleri açıklanacaktır.

8.6.1. Adlandırma

Yukarıda da tanımlandığı gibi, aygıt sürücü, bir yordamlar kümesidir. Tüm aygıt sürücüler aynı tür ve sayıda yordama sahiptirler ve bu yordamların kaynak kütüklerinin tümü, *LINUX* bağlamında `/usr/src/`*`LINUX`*`/drivers/char` adlı aynı alt kılavuz altında yer alırlar. Bu nedenle aygıt sürücülere ilişkin yordam adlarının biricik olma zorunluluğu vardır. Bu biriciklik, yordam adının başına aygıt adı eklenerek sağlanabilir. Bu durumda, örneğin `aygıtsürücüm` adlı aygıtın `open()` işlevinin adı, `aygıtsürücüm_open()` olur.

8.6.2. `init()` İşlevi ve Aygıtın Kaydedilmesi

Bilgisayar sistemi açıldığında her aygıt sürücü, kendisi ve ilişkili olduğu giriş/çıkış birimine (aygıta) ilişkin kimi önbelirlemeleri yapmak durumundadır. Örneğin aygıt sürücüyü oluşturan yordamlarda kullanılan kimi değişkenlere ilk değer atama işlemleri ile arabirim güdüm yazmacı günleme gibi işlemler bu önbelirlemelere bir örnektir. Sözkonusu bu önbelirleme işlemleri `init()` işlevi ile yerine getirilir.

`aygıtsürücüm` adı verilen bir aygıt sürücü için, `aygıtsürücüm_init()` olarak adlandırılan önbelirleme işlevi, diğer işlevlerde olduğu gibi tasarlanıp yazıldıktan sonra elde edilen kaynak kod `/usr/src/`*`LINUX`*`/drivers/char` alt kılavuzu altına saklanır. `aygıtsürücüm_init()` satırı, `/usr/src/`*`LINUX`*`/drivers/char/mem.c` adlı kütükte yer alan `chr_dev_init()` işlevinin kaynak kodunun sonuna, `return 0` satırından önce eklenerek, sistem açıldığında işletilmesi sağlanır.

`init()` işlevinin, arabirimle ilgili önbelirlemeler dışında yerine getirmesi gereken bir diğer işlem de aygıt sürücünün kayıt işlemidir. Bu işlem:

```
register_chrdev(major, name, file_op)
```

işlevi ile gerçekleştirilir.

Burada:

`major` aygıtın *major device* numarasını,
`name` aygıtın adını (Örneğin `aygıtsürücüm`),
`file_op` aygıt sürücünün `struct file_operations` yapısının göstergesini

tanımlar. İşlevin sıfır değerini döndürmesi, kayıt işleminin başarılı olduğunu; eksi bir değer döndürmesi ise başarısızlığı gösterir. İşlev, `major` için sıfır değeri ile çağrıldığında ve kayıt işlemi başarılı olduğunda çekirdek tarafından atanan *major device* numarasını döndürür.

`init()` işlevinin içerebileceği bir diğer işlev de, aygıt sürücünün içerebileceği kesilme yordamının hangi donanım kesilme girişiyle ilişkili olduğunu belirleyen işlevdir. Yukarıda açıklandığı üzere, bu belirleme:

```
int request_irq(unsigned int irq, void (*handler)(int),
                unsigned long type, char* name);
```

işleviyle yerine getirilebilmektedir.

8.6.3. Çalışma Alanı

Aygıt sürücü bağlamındaki yordamların gereksedikleri yastık alanların tanımlanması, üç değişik yolla gerçekleştirilebilir:

- Durgun *(static)* yastık tanımı
- `kmalloc()` işlevi ile devingen *(dynamic)* yastık tanımı
- `vmalloc()` işlevi ile devingen yastık tanımı

a. Durgun *(static)* yastık tanımı

Bu şıkta, gereksenen yastıklar, `static char buffer[1024]` gibi bir bildirimle yordamlar içinde, durgun bir biçimde tanımlanırlar. Bu yolla ayrılan bellek alanı, sözkonusu yordamın dışında kullanılamaz. Yeniden derleme yapılmadan artırılıp eksiltilemez. İşletim sırasında aygıt kullanılmasa bile bellekte varlığını sürdürüp yer kaybına neden olur. Bu nedenlerle, aşağıda açıklanan devingen *(dynamic)* bellek kullanımı tercih edilir.

b. `kmalloc()` işlevi ile devingen yastık tanımı

`kmalloc` ve `kfree` işlevleri, uygulama programı düzeyi `malloc` ve `free` kütüphane işlevlerinin çekirdek katmandaki eşdeğerleridir. `kmalloc` ile ayrılan bellek `kfree` ile serbest bırakılabilir. `kmalloc` işlevi aşağıdaki gibidir:

```
void *kmalloc(size_t size, int priority);
```

`size` ayrılacak bellek öbeğinin boyunu,
`priority` önceliği

belirtir. `priority` parametresi, `GFP_KERNEL` ve `GFP_ATOMIC` değerlerini alabilmektedir. `kmalloc` işlevi çağrıldığında, çağrıyı yapan görevin, ana bellekte yer

olmaması nedeniyle bekler duruma geçmesinde sakınca yoksa `GFP_KERNEL`, aksi halde `GFP_ATOMIC` değerinin kullanılması gerekmektedir. Ayrılan bellek üzerinde doğrudan bellek erişim *(DMA)* işlemi gerçekleştirilecek ise `priority` parametresinin, `GFP_DMA` değeri ile "mantıksal ya da" işlemine sokulması gerekmektedir. `kmalloc` işlevi ile 128KB'a kadar büyüklükte bellek alanı ayırmak mümkün olabilmektedir.

`kmalloc` işlevi ile ayrılan bellek `kfree` işlevi ile serbest bırakılabilmektedir:

```
void kfree(void* ptr, int size);
```

`ptr` ayrılmış bellek öbeğinin göstergesini
`size` sözkonusu öbeğin boyunu

göstermektedir.

c. `vmalloc()` işlevi ile devingen yastık tanımı

Aygıt sürücü bağlamındaki yordamların gereksedikleri bellek yastık alanlarının elde edilmesinde kullanılabilen üçüncü yöntem ise `vmalloc` işlevi ile bellek ayırma yöntemidir.

```
void *vmalloc(unsigned long size);
```

`vmalloc()` işlevi, adından da anlaşılacağı üzere görüntü *(virtual)* bellek yönetimiyle uyumlu bir işlevdir. Bu nedenle `vmalloc()` işlevi ile ayrılan bellek alanları, `kmalloc()` ile ayrılan bellek alanlarının tersine, bir kez elde edildikten sonra varlığı garanti edilebilen alanlar olamamaktadır. Bu nedenle `vmalloc()` işlevi ile ayrılan bellek alanları üzerinde *DMA* işlemlerine izin verilemediği gibi, bu işlevin kesilme yordamlarında kullanılması da son derece sakıncalıdır.

`vmalloc()` işlevi ile ayrılan bellek alanları `void vfree(void *ptr)` ile serbest bırakılabilmektedir.

8.6.4. `struct file_operations` yapısında tanımlı İşlevler

Bu kesimde, aygıt sürücünün ana giriş noktaları olan ve `struct file_operations` yapısında adresleri bulunan işlevlerden önemli olanları ve bunların gerçekleştirdikleri işlemler açıklanacaktır.

a. `open` işlevi

Aygıt sürücünün bu işlevi, uygulama programları `open()` sistem çağrısını, bu aygıt sürücüyü temsil eden özel kütük adı ile çalıştırdığı durumda çağrılır.

```
int open(struct inode* inode, struct file* file);
```

`inode` aygıt sürücüyü temsil eden özel kütüğün *i-node* yapısının göstergesidir.
`file` aygıt sürücüyü temsil eden özel kütüğün *file* yapısının göstergesidir.

Bu işlevin gerçekleştirdiği ana işlemler şunlardır:

- İlgili donanım biriminin hazır durumda olup olmadığının denetimi

- Verilen *minor device* numarasının geçerliliğinin denetimi
- Aynı anda yalnız tek bir uygulama programının aygıtı kullanabilmesinin gerektiği durumlarda aygıtın meşgul olup olmadığını gösteren bir değişkenin denetimi, meşgul değilse kurulması, aygıt kullanımdaysa (denetim değişkeni kurulu ise) işlevi çalıştıran göreve `EBUSY` değerinin döndürülmesi
- Herhangi bir hata durumunda, işlevi çalıştıran göreve, bu durumu gösteren eksi bir değerin döndürülmesi
- Başarı durumunda, işlevi çalıştıran göreve 0 değerinin döndürülmesi

b. `release` işlevi

`release` işlevi, aygıtı kullanım için açan kullanıcı görevinin aynı aygıt için `close()` sistem çağrısını çalıştırması durumunda işletilir.

```
int release(struct inode* inode, struct file* file);
```

`inode` aygıt sürücüyü temsil eden özel kütüğün *i-node* yapısının göstergesidir.
`file` aygıt sürücüyü temsil eden özel kütüğün *file* yapısının göstergesidir.

Bu işlevin gerçekleştirdiği ana işlemler şunlardır:

- Sonuçlanmamış G/Ç işlemleri varsa bunların gerektirdiği temizleme *(reset)* işlemlerinin yapılması
- Donanım kaynaklarının serbest bırakılması
- `open()` işlevi tarafından kurulmuş ve birden fazla programın aygıtı kullanmasını engelleyen değişkenlerin sıfırlanması

c. `read` işlevi

Aygıt sürücünün bu işlevi, uygulama programlarının `read()` sistem çağrısını yapması durumunda işletilir.

```
ssize_t read(struct file* file, char* buf, size_t length,
             loff_t* ppos);
```

`file` aygıt sürücüyü temsil eden özel kütüğün *file* yapısının göstergesidir.
`buf` kullanıcı adres evreninde tanımlı yastık alanın göstergesidir.
`length` okunacak bayt sayısıdır.
`ppos` okumaya başlanacak konumu gösterir.

Bu işlev, belirtilen sayıdaki baytı giriş/çıkış biriminden okuyup kullanıcı programı adres evreni içinde tanımlı `buf` yastık alanına aktarmakla yükümlüdür. Bu aktarmayı yapacak sözkonusu `read()` işlevi çekirdek katman adres evreninde yer aldığından, giriş/çıkış biriminden okuduğu baytları, kullanıcı programı adres evreninde tanımlı bir alana, doğrudan aktaramaz. Bu işlem için aşağıdaki makro kullanılabilir:

```
copy_to_user(to, from, n)
```

Bu makro kullanılmadan önce, kullanıcı programının, belirttiği yastık alanına yazma hakkı bulunup bulunmadığının denetlenmesi de gerekebilir. Bu işlem için aşağıdaki işlev kullanılabilir:

```
int verify_area(int access, void* u_addr, unsigned long size);
```

`access`	`VERIFY_WRITE` olmalıdır.
`u_addr`	kullanıcı adres evreninde, aktarım yapılacak yastık alanının adresidir.
`size`	yastık alanın boyudur.

Bu işlev belirtilen hak varsa sıfır yoksa `EFAULT` değerini döndürür.

d. `write()` işlevi

Aygıt sürücünün bu işlevi, uygulama programları `write()` sistem çağrısı yapmaları durumunda işletilir.

```
ssize_t write(struct file* file, char* buf, size_t length,
              loff_t* ppos);
```

Bu işlev, belirtilen sayıdaki baytı, kullanıcı programı adres evreni içinde tanımlı `buf` yastık alanından giriş/çıkış birimine aktarmakla yükümlüdür. Bu aktarımı yapacak sözkonusu `write()` işlevi çekirdek katman adres evreninde yer aldığından, kullanıcı programı adres evreninde tanımlı bir alandan giriş/çıkış birimine doğrudan aktarım yapamaz. Bu aktarım için aşağıdaki makro kullanılabilir:

```
copy_from_user(to, from, n).
```

Bu makro kullanılmadan önce, kullanıcı programının, belirttiği yastık alanını okuma hakkı bulunup bulunmadığının denetlenmesi de gerekebilir. Bu denetleme işlemi için aşağıdaki işlev kullanılabilir:

```
int verify_area(int access, void* u_addr, unsigned long size);
```

`access`	`VERIFY_READ` olmalıdır.
`u_addr`	kullanıcı adres evreninde, aktarım yapılacak yastık alanının adresidir.
`size`	yastık alanın boyudur.

Bu işlev belirtilen hak varsa sıfır yoksa `EFAULT` değerini döndürür.

e. `ioctl` işlevi

Giriş/çıkış birimleriyle ilgili yukarıda tanımlanan açma, kapama, yazma, okuma gibi temel işlevlerin yanı sıra, önbelirleme *(init)* aşamasının dışında da, giriş/çıkış arabirimine güdüm değeri yazma, arabirim durum yazmacını okuma gibi işlemlere gerekseme duyulabilir. Bu işlemler için aygıt sürücü `ioctl()` işlevini içermek durumundadır.

```
int ioctl(struct file* file, unsigned int cmd,
          unsigned long arg);
```

`file`	aygıt sürücüyü temsil eden özel kütüğün *file* yapısının göstergesidir.
`cmd`	arabirimle ilgili denetleme işlem kodunu gösterir.
`arg`	4 baytlık, denetleme işlem koduyla ilişkili bir değeri gösterir.

8.6.5. Aygıt Sürücünün Çekirdek Katmana Eklenmesi

Sisteme yeni bir aygıt sürücü eklemek gerektiğinde aşağıdaki dört adım yerine getirilir:

a. Birinci Adım

Aygıt sürücüyü oluşturan işlevler tasarlanıp yazıldıktan sonra, kaynak kodları ve gerekli başlık *(header)* bilgilerini taşıyan bir kütük elde edilir. `aygıtsürücüm.c` adının verildiği varsayılan bu kütüğün, ilk adımda `/usr/src/LINUX/drivers/char` alt kılavuzu altına yüklenmesi gereklidir. `aygıtsürücüm.c` adlı kütüğün içerdiği başlık bilgileri şunlar olabilmektedir:

```
#include <LINUX/kernel.h>          #include <LINUX/sched.h>
#include <LINUX/tty.h>             #include <LINUX/signal.h>
#include <LINUX/errno.h>           #include <LINUX/malloc.h>
#include <asm/io.h>                #include <asm/segment.h>
#include <asm/system.h>            #include <asm/irq.h>
#include <asm/uaccess.h>           #include "aygıtsürücüm.h"
```

Bunlardan `aygıtsürücüm.h` gerekli diğer başlık bilgilerinin yanı sıra, örneğin:

```
struct file_operations aygıtsürücüm_fops
{
  NULL,
  aygıtsürücüm_read,
  aygıtsürücüm_write,
  NULL,
  NULL,
  aygıtsürücüm_ioctl,
  NULL,
  aygıtsürücüm_open,
  NULL,
  aygıtsürücüm_release,
  NULL,
  NULL,
  NULL,
  NULL,
  NULL,
};
```

gibi bir tanımı da içermek durumundadır.

b. İkinci Adım

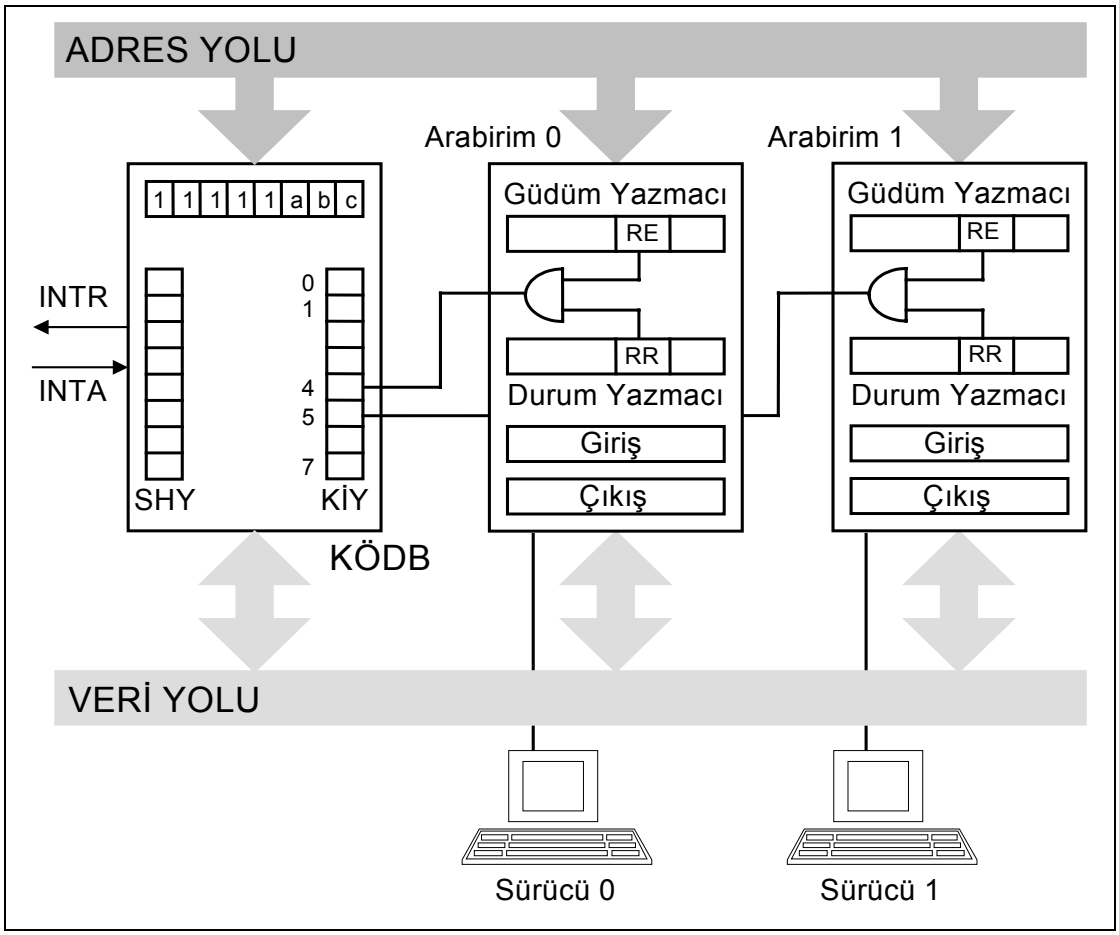
İkinci adımda, `aygıtsürücüm_init()` satırı, `/usr/src/LINUX/drivers/char/mem.c` adlı kütükte yer alan `chr_dev_init()` işlevinin kaynak kodunun sonuna, `return 0` satırından önce eklenir.

c. Üçüncü Adım

Üçüncü adımda, `/usr/src/LINUX/drivers/char/Makefile` kütüğünde `OBJS` tanımlarının sonuna `aygıtsürücüm.o`, `SRCS` tanımlarının sonuna da `aygıtsürücüm.c` satırı eklenir.

d. Son Adım

Son adımda Çekirdek yeniden derlenerek yeni aygıt sürücü sisteme katılmış olur.



Çizim 8.5. Örnek Aygıt Sürücünün Taban Aldığı Sistem Görünümü

8.6.6. Damga Tabanlı Aygıt Sürücü Örneği

İzleyen kesimde yalın bir damga tabanlı aygıt sürücü örneklenmektedir. Sözkonusu aygıt sürücü, daha önce, giriş/çıkışların programlanmasında kullanılan yöntemleri açıklarken örneklediğimiz ardıl giriş/çıkış birimine ilişkin olacaktır (Çizim 8.5). Aygıt sürücü, `ardıl_open()`, `ardıl_release()`, `ardıl_read()`, `ardıl_write()` ve `ardıl_ioctl()` olarak adlandırılan işlevleri içerecektir. `ardıl_read()` ve `ardıl_write()` işlevleri, sırasıyla ardıl giriş/çıkış biriminden belirli sayıda (en çok `MAKSIMUM`) baytı okuma ve ardıl giriş/çıkış birimine bayt yazma işlemlerini yerine

getirecektir. Arabirim Güdüm yazmacı günlenerek iletişim hızı, damga boyu, eşlik denetimi gibi işletim parametrelerinin değiştirilmesi `ardıl_ioctl()` işlevi ile gerçekleştirilecektir.

Aygıt sürücü, sistemde, sözkonusu ardıl giriş/çıkış arabiriminden 2 tane bulunacağını ve bunların RR durum bitine dayalı olarak, sistem kesilme önceliği denetleme biriminin 4 ve 5 numaralı girişleri üzerinden kesilme üreteceklerini varsayacaktır. Bu durumda, aygıt sürücü, temel işlevlerin yanı sıra, arabirimlere ilişkin 2 ayrı kesilme yordamını da içerecektir. Kesilme yordamlarının, ilgili oldukları Giriş yastığı içeriğini, `ardıl_read()` işlevinin de eriştiği, `kuyruk_0` ya da `kuyruk_1` adlı yapının `alan[]` kesimine ekleyecekleri düşünülecektir.

`ardılsürücü` olarak adlandırılan aygıt sürücünün *major device* numarasının 30 olduğu ve 0, Arabirim0'a, 1 ise Arabirim1'e adanan iki *minor device* numarasının bulunduğu varsayılacaktır.

Bu varsayımlara göre aygıt sürücünün görünümü Çizim 8.6'da verilmiştir.

```
/*Damga Tabanlı Aygıt Sürücü Örneği*/
#define KERNEL /*Derlemenin Kernel bağlamında yapılması için*/

/*Kernel headers*/
#include <LINUX/kernel.h>          #include <LINUX/sched.h>
#include <LINUX/tty.h>             #include <LINUX/signal.h>
#include <LINUX/errno.h>           #include <LINUX/malloc.h>
#include <asm/io.h>                #include <asm/segment.h>
#include <asm/system.h>            #include <asm/irq.h>
#include <asm/uaccess.h>
. . .

/*Minor Device Numaraları*/
#define ARABIRIM_0 0
#define ARABIRIM_1 1

/*Arabirim Değişmezleri*/
#define MESGUL  0
#define SERBEST 1
#define MAKSIMUM 16
#define SATIR 80
#define BIP 0x08
#define GUDUM_ADR 0xA0;   /*ARABIRIM_0 Güdüm Yazmaç Adresi*/
#define DURUM_ADR 0xA1;   /*ARABIRIM_0 Durum Yazmaç Adresi*/
#define GIRIS_ADR 0xA2;   /*ARABIRIM_0 Giriş Yazmaç Adresi*/
#define CIKIS_ADR 0xA3;   /*ARABIRIM_0 Çıkış Yazmaç Adresi*/
#define SRN_HZMT_ADR 0xB0;/*KÖDB Süren Hizmet Yazmaç Adresi*/
#define GONDERMEYE_HAZIR 0x01; /*Arabirim Göndermeye hazır*/
#define ALMAYA_HAZIR 0x02;     /*Arabirim Almaya hazır*/
#define ESLIK_HATASI 0x03;     /*Arabirim Eşlik Hatası*/
#define TASMA_HATASI 0x04;     /*Arabirim Taşma Hatası*/
#define KODB_MASKE_0 0xEF; /*KÖDB Süren Hizmet Yazmacı 4.Bit*/
#define KODB_MASKE_1 0xBF; /*KÖDB Süren Hizmet Yazmacı 5.Bit*/

/*Aygıt Sürücünün kullandığı Değişkenler*/
static int mesgul_0, mesgul_1;
static char ilk_güdüm_değeri_0 = 0x84;
static char ilk_güdüm_değeri_1 = 0x84;
static char tranceiver_enable  = 0x03;
static unsigned int irq_0 = 4;
static unsigned int irq_1 = 5;
static unsigned type = 1;
```

```
struct kuyruk {
       char alan[MAKSIMUM];
       int bas, son;
       boolean dolu;
};
static struct kuyruk kuyruk_0;
static struct kuyruk kuyruk_1;


/*İşlevler*/
static int ardıl_open(struct inode*,struct file*);
static int ardıl_release(struct inode*,struct file*);
static ssize_t ardıl_read(struct file*,char*,size_t,loff_t*);
static ssize_t ardıl_write(struct file*,char*,size_t, loff_t*);
static int ardıl_ioctl(struct inode*,struct file*,unsigned int,
                       unsigned long);


/*file_operations struct*/
struct file_operations ardıl_fops =
{
    NULL,           /* llseek            */
    ardıl_read,     /* read              */
    ardıl_write,    /* write             */
    NULL,           /* readdir           */
    NULL,           /* poll              */
    ardıl_ioctl,    /* ioctl             */
    NULL,           /* mmap              */
    ardıl_open,     /* open              */
    NULL,           /* flush             */
    ardıl_release,  /* release           */
    NULL,           /* fsync             */
    NULL,           /* fasync            */
    NULL,           /* check_media_change */
    NULL,           /* revalidate        */
    NULL            /* lock              */
};
```

```
/*Önbelirleme İşlemleri*/
void ardıl_init(void)
{
    /*30 numaranın kayıt edilmesi*/
    if((register_chrdev(30, "ardıl", &ardıl_fops)) < 0)
       printk("Ardıl-30 Kayıt işlemi başarısız\n");
    else {
    printk("Ardıl-30 Kayıt işlemi başarıyla tamamlandı\n");
    *Arabirim_0 ve 1'ın ilk güdüm değerlerinin yazılması*/
    _asm {
    mov al,ilk_güdüm_değeri_0
    out GUDUM_ADR,al
    mov al,ilk_güdüm_değeri_1
    out GUDUM_ADR+4,al
    }
    mesgul_0 = SERBEST;
    mesgul_1 = SERBEST;
    request_irq(unsigned int irq_0, void (*kesilme_yordamı_0)(int),
                unsigned long type, char* ardıl);
    request_irq(unsigned int irq_1, void (*kesilme_yordamı_1)(int),
                unsigned long type, char* ardıl);
    }
}
```

```
/*Okuma İşlemi*/
static ssize_t ardıl_read(struct file* file, char* buf,
                          size_t length, loff_t* ppos)
{
    int i, boy, kboyu;
    char yastik[MAKSIMUM];
    /*Hangi Arabirim ?*/
    switch(MINOR(file->f_dentry->d_inode->i_rdev))
    {
      case ARABIRIM_0:
      _asm cli
      if(!bos(&kuyruk_0)){
            kboyu = boybul(&kuyruk_0);
            boy =(kboyu < length) ? kboyu : length;
            for(i=0; i<boy; i++)
                  yastik[i] = damgaal(&kuyruk_0);
            _asm sti
           /*Kullanıcı alanına yazabilme denetimi*/
            if(!verify_area(VERIFY_WRITE, buf, boy))
                  return -EFAULT;
            /*damga'yı kullanıcı alanına yazma*/
            copy_to_user(buf, yastik, boy);
            return boy;
            }
      else
            _asm sti
            return -ENODATA;
      case ARABIRIM_1:
      _asm cli
      if(!bos(&kuyruk_1)){
            kboyu = boybul(&kuyruk_1);
            boy =(kboyu < length) ? kboyu : length;
            for(i=0; i<boy; i++)
                  yastik[i] = damgaal(&kuyruk_1);
            _asm sti
           /*Kullanıcı alanına yazabilme denetimi*/
            if(!verify_area(VERIFY_WRITE, buf, boy))
                  return -EFAULT;
            /*damga'yı kullanıcı alanına yazma*/
            copy_to_user(buf, yastik, boy);
            return boy;
            }
      else
            _asm sti
            return -ENODATA;
      /*Geçersiz minor device numarası*/
      default:
            return -EINVAL;
      }
}
```

```
/*Yazma İşlemi*/
static ssize_t ardıl_write(struct file* file, char* buf,
                           size_t length, loff_t* ppos)
{
    char dizgi[SATIR];
    /*Hangi Arabirim ?*/
    switch(MINOR(file->f_dentry->d_inode->i_rdev)) {
        case ARABIRIM_0:
        /*Kullanıcı alanından okuyabilme denetimi*/
        if(!verify_area(VERIFY_READ, buf, length))
             return -EFAULT;
        /*Kullanıcı alanından damga okuma*/
        copy_from_user(&dizgi, buf, length);
        _asm {
             mov cx, length
             lea bx, dizgi
             xor si, si
        dön0:in al, DURUM_ADR
             test al, GONDERMEYE_HAZIR
             jz dön0
             mov al, [bx][si]
             out CIKIS_ADR, al
             inc si
             loop dön0
        }
        return boy;
        case ARABIRIM_1:
        /*Kullanıcı alanından okuyabilme denetimi*/
        if(!verify_area(VERIFY_READ, buf, boy))
             return -EFAULT;
        /*Kullanıcı alanından damga okuma*/
        copy_from_user(&dizgi, buf, boy);
        _asm {
             mov cx, length
             lea bx, dizgi
             xor si, si
        dön1:in al, DURUM_ADR+4
             test al, GONDERMEYE_HAZIR
             jz dön1
             mov al, [bx][si]
             out CIKIS_ADR+4, al
             inc si
             loop dön1
        }
        return boy;
        /*Geçersiz minor device numarası*/
        default:
        return -EINVAL;
    }
}
```

```
/*İşletim Parametreleri Günleme*/
static int ardıl_ioctl(struct inode* inode, struct file* file,
                       unsigned int cmd, unsigned long arg)
{
    switch(MINOR(file->f_dentry->d_inode->i_rdev))
    {
        case ARABIRIM_0:
          gunle(GUDUM_ADR, cmd, arg);
          return(0);

        case ARABIRIM_1:
          gunle(GUDUM_ADR+4, cmd, arg);
          return(0);

        /*Geçersiz minor device numarası*/
        default:
          return -EINVAL;
    }
}


void gunle(char adres,unsigned int komut,unsigned long arguman)
{
/*****************************************************/
/*                                                   */
/*  "Adres"i verilen Güdüm Yazmacı içeriğini,        */
/*                                                   */
/*  "komut"la belirlenen:                            */
/*                                                   */
/*        - iletişim hızı (01),                      */
/*        - damga boyu (02),                         */
/*        - eşlik türü (03)                          */
/*                                                   */
/*  değiştirme gibi bir işleme ilişkin,              */
/*                                                   */
/*  "arguman" içinde tanımlı değerle günleyen işlev. */
/*                                                   */
/*****************************************************/
}
```

```
/*Açma İşlemi*/
static int ardıl_open(struct inode* inode, struct file* file)
{
    switch(MINOR(file->f_dentry->d_inode->i_rdev))
    {

    case ARABIRIM_0:
        /*Daha önce açılmışsa*/
        if(mesgul_0 == MESGUL)
           return -EBUSY;
        else
           /*ARABIRIM_0 Meşgul göstergesini kur*/
           mesgul_0 = MESGUL;
           /*ARABIRIM_0 Kuyruk Göstergelerini Kur*/
           kuyruk_0.bas = kuyruk_0.son = MAKSIMUM -1;
           kuyruk_0.dolu = FALSE;
         /*ARABIRIM_0'ı Alma ve Göndermeye Aç (Enable)*/
         _asm {
               mov al, ilk_güdüm_0
               or  al, tranceiver_enable
               out GÜDÜM_ADR, al
         }
           return 0;

    case ARABIRIM_1:
        /*Daha önce açılmışsa*/
        if(mesgul_1 == MESGUL)
           return -EBUSY;
        else
           /*ARABIRIM_1 Meşgul göstergesini kur*/
           mesgul_1 = MESGUL;
           /*ARABIRIM_1 Kuyruk Göstergelerini Kur*/
           kuyruk_1.bas = kuyruk_1.son = MAKSIMUM -1;
           kuyruk_1.dolu = FALSE;
         /*ARABIRIM_1'i Alma ve Göndermeye Aç (Enable)*/
         _asm {
               mov al, ilk_güdüm_1
               or  al, tranceiver_enable
               out GÜDÜM_ADR+4, al
         }
           return 0;

    /*Geçersiz minor device numarası*/
    default:
        return -ENXIO;
    }
}
```

```
/*Kapama İşlemi*/
static int ardıl_release(struct inode* inode,struct file* file)
{
    switch(MINOR(file->f_dentry->d_inode->i_rdev))
    {

    case ARABIRIM_0:
        /*ARABIRIM_0 Meşgul göstergesini sıfırla*/
        mesgul_0 = SERBEST;
        /*ARABIRIM_0'ı Alma ve Göndermeye Kapa (Disable)*/
        _asm {
              mov al, ilk_güdüm_0
              out GÜDÜM_ADR, al
        }
        return 0;

    case ARABIRIM_1:
        /*ARABIRIM_1 Meşgul göstergesini sıfırla*/
        mesgul_1 = SERBEST;
        /*ARABIRIM_1'i Alma ve Göndermeye Kapa (Disable)*/
        _asm {
              mov al, ilk_güdüm_1
              out GÜDÜM_ADR+4, al
        }
        return 0;

    /*Geçersiz minor device numarası*/
    default:
        return -ENXIO;
    }
}
```

```
/*Arabirim0 Kesilme Yordamı*/
void kesilme_yordamı_0(void)
{
char damga, durum;
    _asm {
    pusha
    sti
    in al, DURUM_ADR
    mov durum, al
    }
    if(hata(durum)) {
          _asm {
          in al, GIRIS_ADR /* RRDY sıfırlama için boş okuma */
          cli
          in al, SRN_HZMT_ADR
          and al, KODB_MASKE_0
          out SRN_HZMT_ADR, al
          popa
          iret
          }
          }
    _asm {
    in al, GIRIS_ADR
    mov damga, al
    }
    if(!damgaekle(&kuyruk_0, damga) {
           bip_0();
           _asm {
           cli
           in al, SRN_HZMT_ADR
           and al, KODB_MASKE_0
           out SRN_HZMT_ADR, al
           popa
           iret
           }
    _asm {
    bekle: in al, DURUM_ADR /*Girilen damganın "echo" işlemi*/
           test al, GONDERMEYE_HAZIR
           jz bekle
           mov al, damga
           out CIKIS_ADR, al
           cli
           in al, SRN_HZMT_ADR
           and al, KODB_MASKE_0
           out SRN_HZMT_ADR, al
           popa
           iret
    }
}
```

```
/*Arabirim1 Kesilme Yordamı*/
void kesilme_yordamı_1(void)
{
char damga, durum;
    _asm {
    pusha
    sti
    in al, DURUM_ADR+4
    mov durum, al
    }
    if(hata(durum)) {
          _asm {
          in al, GIRIS_ADR+4 /RRDY sıfırlama için boş okuma/
          cli
          in al, SRN_HZMT_ADR
          and al, KODB_MASKE_1
          out SRN_HZMT_ADR, al
          popa
          iret
          }
          }
    _asm {
    in al, GIRIS_ADR+4
    mov damga, al
    }
    if(!damgaekle(&kuyruk_1, damga) {
           bip_1();
           _asm {
           cli
           in al, SRN_HZMT_ADR
           and al, KODB_MASKE_1
           out SRN_HZMT_ADR, al
           popa
           iret
           }
    _asm {
    bekle: in al, DURUM_ADR+4  /Girilen damganın "echo" işlemi/
           test al, GONDERMEYE_HAZIR
           jz bekle
           mov al, damga
           out CIKIS_ADR+4, al
           cli
           in al, SRN_HZMT_ADR
           and al, KODB_MASKE_1
           out SRN_HZMT_ADR, al
           popa
           iret
    }
}
```

```
/*Kuyruk boş mu? işlevi*/
boolean bos(struct kuyruk* kg)
{
   return((kg->bas == kg->son) ? TRUE : FALSE);
}


/*Kuyruktaki damga sayısını bul işlevi*/
int boybul(struct kuyruk* kg)
{
   int deger;

   if((deger = (kg->son)-(kg->bas)) < 0)
       return(MAKSIMUM + deger);
   else
       return(deger);
}


/*Kuyruktan bir damga al işlevi*/
char damgaal(struct kuyruk* kg)
{
   if(kg->bas == MAKSIMUM - 1)
       kg->bas = 0;
   else
       (kg->bas)++;
   kg->dolu = FALSE;
   return(kg->alan[kg->bas]);
}


/*Kuyruğa bir damga ekle*/
boolean damgaekle(struct kuyruk* kg, char damga)
{
   if(kg->dolu)
       return(FALSE);
   if(kg->son == MAKSIMUM - 1)
       kg->son = 0;
   else
       (kg->son)++;
   if(kg->son == kg->bas) {
       kg->dolu = TRUE;
       return(FALSE);
       }
   kg->alan[kg->son] = damga;
   return(TRUE);
}
```

```
/*Arabirimden okumada hata var mı?*/
boolean hata(char durum)
{
   if((durum & ALMAYA_HAZIR)==0 || (durum & ESLIK_HATASI)!=0 ||
      (durum & TASMA_HATASI)!=0)
       return(FALSE);
   return(TRUE);
}

/*Arabirim0 Kuyruk Dolu Uyarısı*/
void bip_0(void)
{
    _asm {
   bip: in al, DURUM_ADR
        test al, GONDERMEYE_HAZIR
        jz bip
        mov al, BIP
        out CIKIS_ADR, al
    }
}

/*Arabirim1 Kuyruk Dolu Uyarısı*/
void bip_1(void)
{
    _asm {
   bip: in al, DURUM_ADR+4
        test al, GONDERMEYE_HAZIR
        jz bip
        mov al, BIP
        out CIKIS_ADR+4, al
    }
}
```

Çizim 8.6. Damga Tabanlı Aygıt Sürücü Örneği

8.6.7. Damga Tabanlı Aygıt Sürücü Örneğiyle ilgili Ek Açıklamalar

İzleyen kesimde, okuyucunun, Çizim 8.6'da verilen örnek aygıt sürücü yordamlarını ayrı ayrı inceledikten sonra, sürücünün işleyişini bir bütün olarak algılanmasını kolaylaştıran, özellikle yordamlar arası ilişki ve etkileşime ilişkin kimi ek açıklamalara yer verilmiştir:

Örnek aygıt sürücünün taban aldığı sistem görünümü içinde yer alan Arabirim0 ve Arabirim1'in Güdüm ve Durum yazmaçları ile Giriş ve Çıkış yastıklarının adresleri, Giriş/Çıkış Sistemi adlı İkinci Bölümde verilenlerle uyumlu olarak, sırasıyla: `A0H, A1H, A2H, A3H; A4H, A5H, A6H, A7H`; kesilme önceliği denetleme biriminin Süren Hizmet Yazmaç adresi ise `B0H` olarak alınmıştır.

Örnek aygıt sürücü, bir aygıt sürücünün içermesi gereken işlevlerden, salt `open()`, `close()`, `read()`, `write()` ve `iocntl()` işlevlerini içermektedir. Diğer işlevler, kullanılmadıklarından dolayı, `file_operations struct` adlı yapı içinde `NULL` olarak tanımlanmıştır. Tanımlı işlevler `ardıl_` öneki ile (örneğin `ardıl_open()` gibi) anılarak adlandırmada biriciklik sağlanmaktadır.

Arabirim0 ve Arabirim1 için, `kuyruk_0` ve `kuyruk_1` olarak adlandırılan,

```
struct kuyruk {
       char alan[MAKSIMUM];
       int bas, son;
       boolean dolu;
};
```

biçiminde tanımlanan `static` nitelikli, çekirdek katman düzeyi iki kuyruk öngörülmüştür. Terminal klavyesinden girilen damgalar, bu kuyruklara, ilgili kesilme yordamınca eklenmekte ve `read()` işlevi ile de, `length` parametresinin belirlediği sayıda damga, buradan, uygulama programı adres evrenine aktarılmaktadır. `MAKSIMUM` adlı parametre 16 ile sınırlı olduğundan (`kmalloc()`, `kfree()` işlevlerini kullanarak) devingen bellek atama yerine durgun bellek atama tercih edilmiştir.

`kuyruk_0` ve `kuyruk_1` adlı kuyruklara, kesilme yordamlarının yazma, `read()` işlevinin de okuma amacıyla koşut erişimi söz konusudur. Bu nedenle, kesilme yordamları ile `read()` işlevinin zamanuyumlanma zorunluluğu ortaya çıkmaktadır. Bu zamanuyumlama, `read()` işlevi içinde, kuyruğa erişim yapan komutlar,

```
_asm cli
if(!bos(&kuyruk_0)){
     kboyu = boybul(&kuyruk_0);
     boy =(kboyu < length) ? kboyu : length;
     yastik[i] = damgaal(&kuyruk_0);
_asm sti
.
.
```

örneğinde görüldüğü gibi, `cli` ve `sti` komutları ile parantez arasına alınmakta ve bu yolla, `read()` işlevi kuyruğa erişim yaparken kesilme yordamlarının bu işlevi keserek kuyruğa erişim yapmaları, dolayısıyla karşılıklı dışlamanın ihlali engellenmektedir.

Bilindiği gibi, Birlikte Çalışan Görevler başlıklı Dördüncü Bölümde, kesilme düzeneğinin görevler arası zamanuyumlamada kullanımı konu edilmiş ve bu yöntemin önemli sakıncaları açıklanmıştı. Bu bağlamda, bir yandan, sistemin kesilmelere, göreli uzunca bir süre kapalı tutulması durumunda gerçek zaman sayışımının bozulması, diğer yandan da uygulama düzeyi programlara (sıradan kullanıcılara) kesilme gibi, işletim bütünlüğünü, dolayısıyla sistem güvenliğini çok yakından ilgilendiren bir düzeneğin açılması sakıncaları konu edilmişti. Çekirdek katman düzeyi bir yazılım olan aygıt sürücüler bağlamında, bu sakıncalardan ikincisinin, kendiliğinden ortadan kalktığı açıktır. Yukarıda verilen örnekten de görüleceği gibi, zamanuyumlama amacıyla kesilmelere kapatılan komut (satır) sayısı, gerçek zaman sayışımını bozmayacak kadar

kısıtlı olduğundan, bu özel durumda, karşılıklı dışlama gereksiniminin, `cli` ve `sti` komutları kullanılarak, yalın bir biçimde yerine getirilmesi mümkün olmuştur.

Örnek aygıt sürücünün kullanıma girmesinden önce:

```
> mknod /dev/ardıl0 c 30,0 ve
> mknod /dev/ardıl1 c 30,1
```

işletim sistemi komutlarıyla, sistemde, 30 *major device*, 0 *minor device* numarası ve 30 *major device*, 1 *minor device* numarasına karşı gelen `ardıl0` ve `ardıl1` adlı iki özel kütüğün, başka bir deyişle, iki ayrı *i-node* yapısının sistem içinde varlık bulması gerekmektedir.

`init()` işlevi içinde, `register_chrdev(30,"ardıl",&ardıl_fops)` çağrısıyla, 30 *major device* numarası, (*character device table* içinde) her iki özel kütüğe ortak `ardıl_fops` yapısı ile ilişkilendirilmektedir. Bu yolla, uygulama programı düzeyinde, örneğin, `open(/dev/aygıt0,O_RDONLY)` çağrısı işletildiğinde, `aygıt0` adlı özel kütüğün *i_node* yapısından, işletilecek çekirdek katman düzeyi `open()` çağrısının, 30 *major device* numarasına karşı gelen, `ardıl_fops` içinde tanımlı `ardıl_open(struct inode* inode, struct file* file)` çağrısı olacağı sistemce belirlenebilmekte ve bu çağrı içinde `file` argümanının yerini `/dev/aygıt0` değeri almaktadır. `ardıl_open()` işlevi içinde *minor device* numarası sınama, bu argümana dayalı olarak yapılmaktadır. Bu nedenle, örnek aygıt sürücünün içerdiği `open()`, `close()`, `read()`, `write()` ve `iocntl()` işlevlerinin tümünde, öncelikle *minor device* numarası sınanmakta ve sözkonusu işlevin yerine getirmesi gereken fiziksel arabirim işlemleri, bu sınama sonucunda belirlenen arabirim (`ARABIRIM_0` ya da `ARABIRIM_1`) üzerinde yürütülmektedir. Bu durum, aynı nitelikli arabirimlerin tümü için tek bir *major device* numarası tanımlayıp tüm arabirimler için tek bir sistem çağrı kümesini kullanma; her bir arabirime de, ayrı bir *minor device* numarası vererek her arabirim üzerinde, aynı fiziksel arabirim işlemlerini, bu tek sistem çağrı kümesiyle yerine getirebilme olanağını yaratmanın, başka bir deyişle, *major-minor device* numarası kavramının açık bir örneğini oluşturmaktadır.

Kayıt işlemi başarıyla sonuçlandığı takdirde, `ardıl_init()` işlevinin yerine getirdiği diğer işlemler, arabirimlere ilk güdüm değerlerinin (`87H`: 9600 bps, 7 bit-çift eşlik, 1 stop bit) yazılması ve `ardıl_open()` işlevinin sınadığı `mesgul_0` ve `mesgul_1` adlı değişkenlerin `SERBEST` değerine kurulmasıdır. `ardıl_init()` işlevi ile arabirimlere yazılan ilk güdüm değerleri içinde, TE *(Transmit Enable)* ve RE *(Receive Enable)* bitleri sıfır olarak alınmıştır. Başka bir deyişle, önbelirleme aşamasında, arabirimler, alma ve gönderme işlemlerine kapalı tutulmakta ve kuyruk başı ve sonu göstergeleri kurulmadan kesilme üretmeleri engellenmektedir. Bu bitlerin bire kurularak arabirimlerin alma ve gönderme işlemlerine açılması, sözkonusu göstergelerin de kurulduğu `ardıl_open()` işlevi içinde, bu kurma işleminden sonra, örneğin:

```
_asm {
       mov al,ilk_güdüm_0
       or  al,tranceiver_enable
       out GÜDÜM_ADR,al
}
```

komutları ile gerçekleşmektedir. `ardıl_open()` işlevi ile alma ve gönderme işlemlerine açılan arabirim, ilgili `ardıl_release()` işlevi ile, örneğin:

```
_asm {
     mov al,ilk_güdüm_0
     out GÜDÜM_ADR,al
}
```

komutları ile bu işlemlere kapatılmaktadır. Bu nedenle, açma işlemi yapılmamış bir arabirim üzerinde, okuma ya da yazma işlevlerini yürütmek mümkün olamamaktadır.

`ardıl_read()` işlevi ile kesilme yordamları arası veri aktarımı *FIFO* yapılı kuyruklar (`kuyruk_0` ve `kuyruk_1`) üzerinden gerçekleşmektedir. Bu amaçla, örnek programın son kesiminde yer alan `bos()`, `boybul()`, `damgaal()`, `damgaekle()` işlevleri kullanılmaktadır.

Uygulama programının çalışma mantığına dayalı olarak, `ardıl_read()` işlevinin işletilme hızının, ilgili kuyruğun ilgili kesilme yordamınca doldurulma hızından düşük olabileceği düşünülerek kuyruk dolduktan sonra ek damgaların girilmesinin engellenmesi amacıyla, kesilme yordamları içinde `bip()` işlevi kullanılmaktadır. Bu işlev aracılığıyla damga giren kullanıcı düdük sesiyle uyarılmakta ve girdiği damgaya *echo* işlemi uygulanmamaktadır.

(`ardıl_write()` gibi) çekirdek katman düzeyi işlevler işletilirken, bu işlevler, kendiliklerinden ana işlem birimini bırakmadıkları sürece başka bir görevin ana işlem birimine anahtarlanması söz konusu olamaz. Çizim 8.6 ile uyumlu olarak (arabirim TRDY- Transmit Ready durumunda kesilme üretmediğinden), `ardıl_write()` işlevi, damga yazma işlemlerini seçmeli giriş/çıkış programlama yöntemine dayalı olarak gerçekleştirmektedir. Bu durum, `ardıl_write()` işlevi işletilirken ana işlem biriminin verimsiz kullanımına neden olur. Zira, n adet damganın durum biti sınamalarıyla ekrandan dökümü sırasında, `ardıl_write()` işlevi buna izin vermediğinden başka herhangi bir görevin ana işlem birimini kullanması mümkün değildir. Arabirim-Terminal arası iletişim hızının 9600 bps olduğu varsayıldığında n adet damganın, seçmeli giriş/çıkış programlama yöntemiyle dökümü n/960 saniye gibi çok uzun bir süre alır.

Çok iş düzeninde çalışan çok kullanıcılı bir bilgisayar sisteminde, bu kadar uzun bir süre ana işlem biriminin diğer görevlere kapalı kalması düşünülemez. Bu sakınca, damga yazma işlemlerinde de kesilme düzeneğinin kullanılmasıyla aşılabilir. Ancak bu tek başına yeterli değildir. Yazılacak damganın arabirim çıkış yastığına yazıldıktan sonra TRDY bitinin kurulmasına kadar geçen sürede, `ardıl_write()` işlevinin ana

işlem birimini bırakmasına olanak veren bir düzeneğe de gereksinim vardır. Bu düzenek *UNIX/LINUX* çekirdek katmanında bulunan `sleep_on()` ve `wake_up()` hazır işlevleriyle kurulabilir. `sleep_on()` işlevi, kendisini çağıran görevin `wait_queue` adlı bekleme kuyruğuna bağlanarak ana işlem birimini bırakmasını, `wake_up()` işlevi de, `wait_queue` adlı bekleme kuyruğundaki tüm görevlerin hazır görev durumuna getirilmesini sağlamaktadır.

Bu durumda, giriş/çıkış işlemleri nedeniyle beklemesi gereken bir görev `sleep_on()` işleviyle, bekleme süresince ana işlem birimini başka görevlere bırakabilmekte, giriş/çıkış işlemlerinin sonlandığını belirten kesilmenin ana işlem birimine anahtarladığı kesilme yordamı da, `wake_up()` işleviyle bu görevin tekrar hazır görev durumuna gelmesini sağlayabilmektedir.

`ardıl_write()` işlevi	TRDY Kesilme Yordamı

.
.
.
`case ARABIRIM_0:`
`/*Kullanıcı alanından okuyabilme denetimi*/`
`if(!verify_area(VERIFY_READ, buf, length))`
`  return -EFAULT;`
`/*Kullanıcı alanından damga okuma*/`
`copy_from_user(&dizgi, buf, length);`
`_asm {`
`  mov cx, length`
`  lea bx, dizgi`
`  xor si, si`
`dön0:out CIKIS_ADR, al`
`  inc si`
`  sleep_on  /makro olarak düşününüz/`
`  loop dön0`
`}`
`return boy;`
.
.
.
	.
	.
	.
	`wake_up()`
	.
	.
	.
	`iret`

Çizim 8.7. `sleep_on` ve `wake_up` işlevlerinin Kullanım Örneği

Örnek aygıt sürücünün taban aldığı arabirimlerin (Çizim 8.5) TRDY durumunda da ayrı birer kesilme ürettikleri varsayılısa, `ardıl_write()` işlevi içerisinde, durum biti sınama döngüsü yerine, görüntülenecek damgayı arabirim çıkış yastığına yazdıktan sonra `sleep_on()` işlevini çağırarak, `ardıl_write()` işlevini çağırmış olan kullanıcı görevini bekleme kuyruğuna bağlamak; damganın arabirimden terminale gönderilmesinin sonlanmasıyla oluşacak TRDY kesilmesinin ana işlem birimine

anahtarlayacağı kesilme yordamı içinde de, `wake_up()` işleviyle, `bu` görevin tekrar hazır görev durumuna gelmesini sağlamak, yukarıda anılan sakıncayı ortadan kaldırmak için bir çözüm yoludur (Çizim 8.7). Çok iş düzeninin ve etkileşimli işlemin kurulduğu çok kullanıcılı bilgisayar sistemlerinde kullanılan yöntem de, genel hatlarıyla budur. Bu yöntem, bir kullanıcı görevinin, giriş/çıkış istemesi durumunda, hangi aşamada ve hangi düzenekle görev yönetici ile etkileşime girdiğini ve Görev Yönetimi başlıklı üçüncü konuda anlatılanları da somut biçimde örneklemektedir.

8.7. Öbek Tabanlı Aygıt Sürücüler

Damga tabanlı aygıt sürücülerle ilgili açıklanan hususların büyük bir çoğunluğu, öbek tabanlı aygıt sürücüler için de geçerlidir. Bu bağlamda, *major/minor device* numarası, *major device* numarasıyla erişilen `struct file_operations` yapısı, aygıt sürücünün `init()`, kesilme ve hizmet yordamlarının oluşturduğu bir bütün olma özelliği gibi hususlar, öbek tabanlı aygıt sürücüler için de geçerlidir.

Öbek tabanlı aygıt sürücüleri damga tabanlı aygıt sürücülerden ayıran en önemli husus, uygulama programlarının giriş/çıkış amacıyla kullandığı standart sistem çağrı kümesi ile aygıt sürücünün içerdiği sistem çağrıları (hizmet yordamları) arasında, kütük yönetim sistemine ek olarak disk ön bellek *(buffer cache)* düzeneğinin yer almasıdır (Çizim 8.1). Kütük yönetim sistemi uygulama programlarının talep ettiği öbekleri, sistemli olarak disk önbellek düzeneğinden sağlar. Talep edilen öbeğin, disk önbelleğinde bulunmadığı durumlarda, disk önbellek düzeneği tarafından disk sürücüye erişim yapılır. Bu amaçla, öbek tabanlı aygıt sürücü hizmet yordamları arasında `request()` adlı bir işlev öngörülür. Disk önbellek düzeneği, disk sürücüye erişim gerekli olduğunda bu erişimle ilgili istemini, ilgili sürücüye (aygıta) ilişkin bir G/Ç kuyruğuna ekler. Sözkonusu bu kuyruğun öge yapısı:

```
struct_request {
      int dev;
      int cmd;int errors;
      unsigned long sector;
      unsigned long nr_sector;
      unsigned long current_nr_sector;
      char *buffer;
      struct semafore *sem;
      struct buffer_head *bhead;
      struct buffer_head *btail;
      struct request *next;
};
```

biçimindedir. `request()` adlı işlev, bu kuyrukta tutulan istemleri karşılamada kullanılır. Öbek tabanlı aygıt sürücü hizmet yordamları `open()`, `release()`, `ioctl()` ve `request()` işlevlerinden oluşur. Damga tabanlı aygıt sürücülerden farklı olarak `read()` ve `write()` işlevleri, `request()` işlevinin içinde gömülü olarak yer alır. `request()` işlevinin algoritmik görünümü aşağıda verildiği gibidir:

```
static void öbek_tabanlı_aygıtım_request(void)
{
baş: INIT_REQUEST;
  if(CURRENT ->cmd == READ)
     {end_request(öbek_tabanlı_aygıtım_read());goto baş;}
  if(CURRENT ->cmd == WRITE)
     {end_request(öbek_tabanlı_aygıtım_write());goto baş;}
  end_request(0);
  goto baş;
}
```

`INIT_REQUEST`, `blk.h` adlı sistem kütüğü içinde tanımlı bir makrodur. İşlevi, ilgili kuyruğun boş olup olmadığını sınamak, kuyruk boşsa `request()` işlevinden geri dönüşü sağlamaktadır.

`CURRENT` parametresi `blk.h` adlı sistem kütüğü içinde tanımlı olup ilgili kuyruğun başını göstermektedir.

`end_request()`, 0 ya da 1 değeri ile çağrılan bir işlevdir. `request()` işlevinin, 1 değeri ile çağrılması durumunda, gereği yerine getirilmiş istemi kuyruktan silmek, `CURRENT` parametresini (varsa) bir sonraki istemi gösterecek biçimde günlemek ve bu istemle ilgili bekleme kuyruğuna bağlanmış görevin hazır görev durumuna getirlmesini sağlamak gibi işlemleri yerine getirmesi sözkonusudur. 0 değeri ile çağrılması durumunda ise, istemi kuyruktan silmek, `CURRENT` parametresini (varsa) bir sonraki istemi gösterecek biçimde günlemek ve hata uyarısı üreterek istemi yapan görevin sonlanmasını sağlamak gibi işlemler sözkonusudur.

Yukarıda, öbek tabanlı aygıt sürücülerle ilgili verilen özet açıklamadan da anlaşılacağı üzere, damga ve öbek tabanlı aygıt sürücülere yaklaşım, genel hatlarıyla benzerdir. Ancak öbek tabanlı giriş/çıkış arabirimlerinin daha karmaşık yapıda olması, arabirimin yapısı dışında, sistem yapısının (doğrudan bellek erişim düzeneği gibi) kimi diğer ayrıntılarının da bilinmesi zorunluluğu, disk önbellek düzeneğiyle etkileşim (dolayısıyla zamanuyumu) gibi nedenlerle öbek tabanlı aygıt sürücülerin gerçekleştirimi, damga tabanlı aygıt sürücülere göre daha karmaşık olabilmektedir.

9. BÖLÜM

İŞLETİM SİSTEMLERİ

DAĞITILMIŞ İŞLEM

Giriş Bölümünde, işletim sistemi, bir bilgisayar sistemini oluşturan donanım ve yazılım nitelikli kaynakları kullanıcılar arasında kolay, hızlı ve güvenli bir işletim hizmetine olanak verecek biçimde paylaştırırken bu kaynakların kullanım verimliliğini en üst düzeyde tutmayı amaçlayan bir yazılım sistemi olarak tanımlanmıştı. Bilgisayarların ufalıp ucuzlayarak yaygınlaşmasının temelinde yer alan mikro-elektronik teknolojisi, bilgisayarlar arası iletişimin de kolay, ucuz ve hızlı bir biçimde yapılabilmesine olanak sağlamıştır. Bu sayede, 1980'li yıllardan başlayarak, irili ufaklı bilgisayar sistemleri, ilk adımda yerel ağlar içinde, yerel ağların da geniş alan ağlarını oluşturması yoluyla, *Internet* adıyla anılan, Dünya ölçeğindeki bir ağ içerisinde bütünleşmiştir. Bu durum, herhangi bir bilgisayar kullanıcısının, kendi bilgisayarının yanı sıra, bu ağ içinde yer alan diğer bilgisayar sistem kaynaklarını da kullanabilmesi olanağını yaratmış ve işletim sistemleri açısından kaynak kavramının genişletilmesi ihtiyacını doğurmuştur. Başka bir deyişle, işletim sistemi, tek bir sistem yerine, ağ içinde bütünleşmiş donanım ve yazılım nitelikli tüm kaynakları kolay, hızlı ve güvenli bir biçimde ağ kullanıcıları arasında paylaştırmak işlevini üstlenmek durumunda kalmıştır. Bu işlev, genelde dağıtılmış işlem olarak anılan bir bağlamda yerine getirilmektedir.

Bir bilgisayarın donanım nitelikli kaynakları, ana işlem birimi, ana bellek ve giriş/çıkış birimleridir. Bu nedenle, işletim sistemleri, yukarıda belirtilen yeni işlev yönünden,

ağda çoğu kez kişisel bir bilgisayar aracılığıyla[30] yer alan bir kullanıcıya, kendi ana işlem birimi, ana bellek ve giriş/çıkış birimlerinin yanı sıra, ağdaki diğer sistemlerin ana işlem birimi, ana bellek ve giriş/çıkış birimlerinden de, bir bütün olarak yararlanabilme olanağı sağlamak durumundadır.

Bilindiği üzere, günümüz işletim sistemlerinde, giriş/çıkış birimleri kütükler olarak ele alınmaktadır[31]. Bu nedenle, dağıtılmış işlem bağlamında, giriş/çıkış birimlerinin paylaşımının ele alınması, işletim sistemlerinin klasik kütük yönetim sistemlerinin dağıtılmış kütük yönetim sistemlerine dönüştürülmesi yoluyla gerçekleşmektedir.

Ana işlem birimi ana bellek ikilisinin, ağ düzeyi paylaşımı yönünden değişik yaklaşımlar kullanılmaktadır. Bu yaklaşımlardan biri "Ağ Düzeyi Birlikte (Koşut) İşlem" ya da "Görev Göçü" yaklaşımı olarak adlandırılır. Bu yaklaşımda, bir bilgisayar sistemine sunulmuş bir işe ilişkin görev ya da işletim dizileri, kullanıcıya saydam bir biçimde, ağda o anda yükü göreli az başka bilgisayar sistemlerine taşınarak işletilir. Bu yolla, ağın toplam işlem kapasitesi değerlendirilerek işlem hızı artışı sağlamak amaçlanır. Bu yaklaşımı kullanan işletim sistemlerine en iyi örnek *MOSIX* işletim sistemidir. *MOSIX* işletim sistemi, *LINUX* işletim sisteminin, çekirdek katmanı görev göçüne olanak verilecek biçimde değiştirilmiş biçimidir.

Ağ bağlamında, bilgisayar sistemlerinin birbirlerinin işlem kapasitelerinden, görev göçü gibi göreli karmaşık işlemlere başvurmadan yararlanmalarının çok daha yalın yöntemi istemci-sunucu yaklaşımına (paradigmasına) dayalıdır. Bu yaklaşım çerçevesinde, sunucu olarak nitelenen bir bilgisayar sistemi üzerinde çalışan ve yine sunucu olarak nitelenen bir program, istemci olarak nitelenen ve sunucu bilgisayarla aynı ağ içinde yer alan diğer bilgisayar sistemleri üzerinde çalışan istemci nitelikli programlara hizmet üretir. Bu yolla, istemci nitelikli bilgisayarların, sunucu nitelikli bilgisayar sistemlerinin işlem kapasitelerinden, belirli işbirliği kuralları (protokolları) çerçevesinde, dolaylı olarak yararlanmaları sağlanır. Bu yaklaşım bir bilgisayar sisteminden yararlanabilmek için mutlaka o bilgisayar sistemine giriş yapıp (bağlanıp) kullanıcıya özel bir program çalıştırma zorunluluğuna da ortadan kaldırmıştır. Bunun sonucunda, çok kullanıcılı bilgisayar sistemleri, yerlerini, istemci-sunucu yaklaşımına dayalı olarak işbirliği yapan "kişisel" bilgisayar sistem kümelerine bırakmıştır. *Windows NT/2000/XP* işletim sistemi, bu gözle, çok kullanıcılı *UNIX* işletim sistemine eşdeğer ancak kişisel bilgisayar işletim sistemi olarak tasarlanmıştır.

İzleyen kesimde, öncelikle istemci-sunucu yaklaşımının uygulamaya sokulmasında işletim sistemine getirilen eklere, bu bağlamda *TCP/IP* ve *Socket* konusuna; klasik kütük yönetim sistemlerinin dağıtılmış kütük yönetim sistemlerine dönüştürülmesi hususuna değinilecektir. İstemci-sunucu yaklaşımıyla yapılabilenin ötesinde, daha çok, bir bilgisayar sistemi üzerinde göreli uzun işletim süresi gerektiren bir işe ilişkin birden çok görev ya da işletim dizisinin, ağ içinde eşanlı (koşut) işletimini ve bu yolla, iş

---

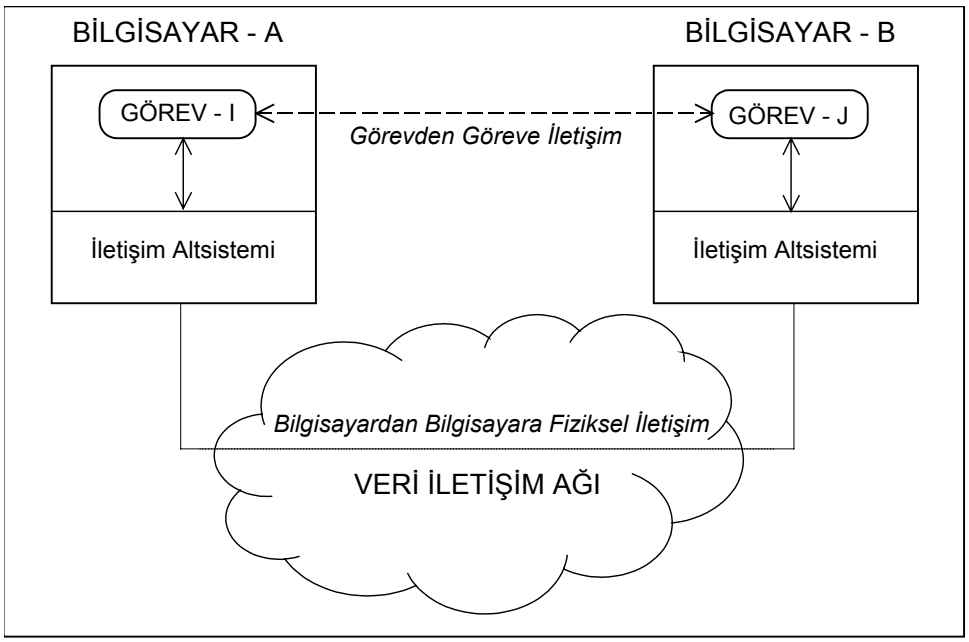
[30] Bkz. Bölüm 1, Giriş, 1.10. Çok Kullanıcılı Bilgisayar Sistemi Kavramının Evrimi.

[31] Bkz. Bölüm 6, Kütük Yönetim Sistemi.

düzeyi işletim hızı artışını amaçlayan "Ağ Düzeyi Birlikte İşlem" ya da "Görev Göçü" yaklaşımı, tek başına bir kitaba konu olabilecek boyut ve karmaşıklıktadır. İzleyen kesimde bu konuya ayrıca değinilmeyecektir.

9.1. İşletim Sistemleri İletişim Alt Kesimi

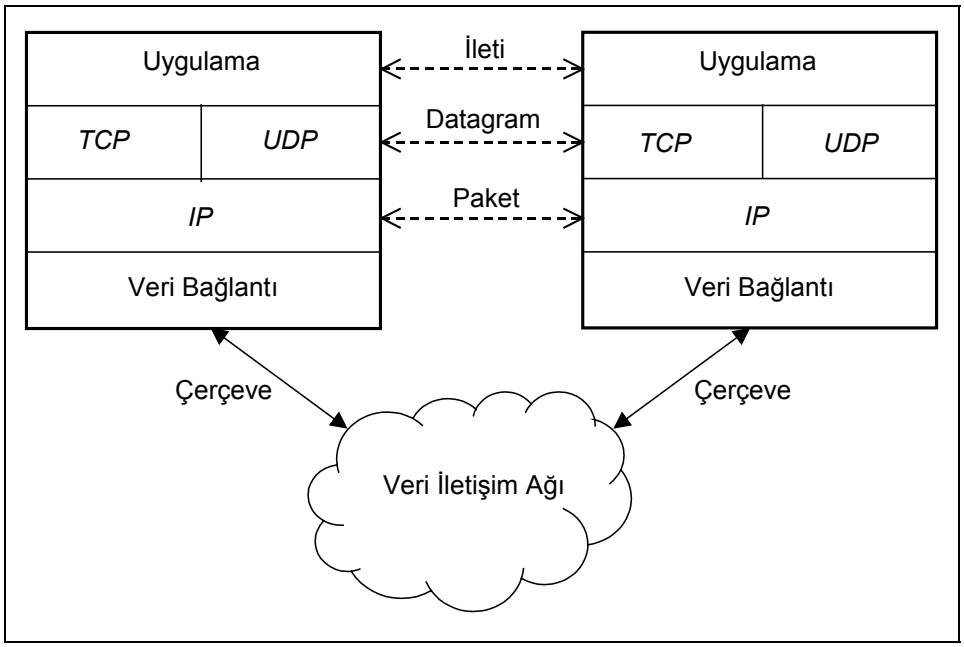
Bilgisayar sistemleri arası veri iletişimi, bu sistemlerden hizmet alan kullanıcılar ya da bunların işlettiği uygulama programları arası (görevler arası) veri aktarımı amacıyla yapılır (Çizim 9.1). İşletim sistemleri, kullanıcılara kolay, hızlı ve güvenli bir işletim hizmeti sunan temel sistem yazılımları olarak, uygulama programlarına, kolay, hızlı ve güvenli bir veri iletişim hizmeti de sunmak durumundadır. Bu amaçla geçmişte, bilgisayar ağlarının gelişmesine koşut olarak, işletim sistemlerine veri iletişim alt kesimi eklemek üzere, yoğun araştırma çalışmaları yürütülmüştür. Veri iletişiminin, doğası gereği değişik marka ve türde bilgisayar arasında yürütülme zorunluluğu nedeniyle, ilgili çalışmalar, öncelikle, bilgisayarlar arası iletişim protokollarına yönelik olmuştur.



Çizim 9.1. Bilgisayarlar arası Veri İletişimi

Bu amaçla, Uluslararası Standartlar Örgütü *(ISO)* düzeyinde çalışma grupları oluşturulmuş, tüm işletim sistemi üreticilerinin uyacağı ve bu yolla türü ve markası ne olursa olsun tüm bilgisayar sistemlerinin sorunsuz veri iletişimi yapabilmesine olanak sağlayacak ortak kurallar ve referans yapılar tanımlanmaya çalışılmıştır. Bu çalışmalar sonunda, *ISO* Referans Modeli diye bilinen bir çerçeve model ortaya çıkmıştır. *ISO* Referans Modelinde, işletim sistemi veri iletişim alt kesiminin; konumu, işlevi, kullandığı veri birimleri ve alt standartları belirlenmiş yedi değişik katmanda ele alınması önerilmektedir. Bu katmanlar, kullanıcı ile veri iletişim arabirimi arasında yer alan çizgide, fiziksel den mantıksala uzanan sıradüzende: Fiziksel, Veri Bağlantı, Ağ, Ulaşım, Oturum, Sunuş ve Uygulama olarak adlandırılmaktadır. Bu yapıda her katman,

her iki yönde (alma ve göndermede) komşu katmanın kendisine aktardığı veri birimine *(PDU-Protocol Data Unit)*, yerine getirmekle yükümlü olduğu işlevi uygulamakta ve işlenmiş veri birimini diğer (alt ya da üst) komşu katmana aktarmaktadır. Bu bağlamda, örneğin uygulama katmanı "ileti" aktarım işlevini yerine getirirken, ulaşım katmanı, iletileri oluşturan "kesim"lerin verilen sırada ve hatasız aktarılmasını, ağ katmanı bunların, "paket"ler olarak ağ üzerinde yönlendirilmesini, veri bağlantı katmanı da "çerçeve"lere dönüştürdüğü paketlerin, fiziksel kanal üzerinden gönderilip alınmasını gerçekleştirmekle yükümlü olmaktadır.



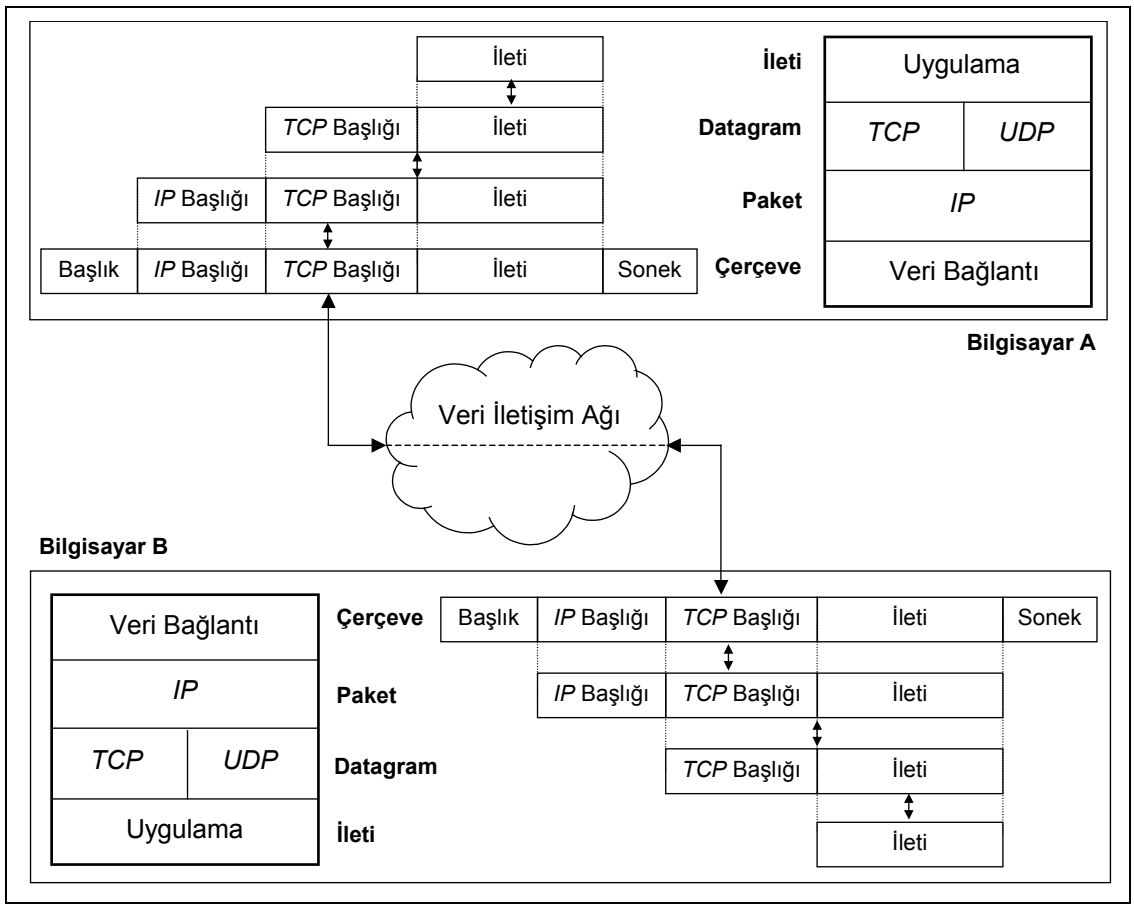
Çizim 9.2. *TCP/IP* Protokol Katmanları

ISO Referans Modeli, uzun ve kapsamlı bir çalışma sonucu ortaya çıkmış ciddi bir çalışma olmakla birlikte, ticari herhangi bir gerçekleştirime konu olamamıştır. Bunun temel nedeni, sözkonusu çalışmalar sürerken, *UNIX* işletim sisteminin, önerilen modele çok benzer bir yapıya sahip, *TCP/IP* diye bilinen iletişim alt sisteminin *Internet* standartı olarak benimsenmesidir. *TCP/IP* yazılımının *Internet* standartı olması, diğer işletim sistemi üreticilerini de, ister istemez bu yazılımın yapısını ve protokollarını benimsemeye yöneltmiştir. Günümüzde *TCP/IP,* yaygın kullanıma sahip işletim sistemlerinin *(UNIX, LINUX, Windows)*, *de facto* nitelikli iletişim alt sistem standartı olarak kullanılmaktadır.

9.1.1. *TCP/IP* Yazılımı

TCP/IP yazılımı beş sıradüzensel katmandan oluşmaktadır. Bunlar: Uygulama, Ulaşım, Ağ, Veri Bağlantı katmanları ile Fiziksel katmandır. Uygulama katmanında ele alınan veri birimi, iletidir. Bir uygulamadan diğer bir uygulamaya gönderilmek istenen bir ileti, *TCP (Transport Control Protocol)* diye adlandırılan katmana aktarılır. Bu katman

iletiyi, gerektiğinde *(segments)* kesimlere bölüp bunları, hatasız ve sıraları bozulmadan, karşı eşdeğer katmana (*TCP* katmanına) gönderme işlevini üstlenir.



Çizim 9.3. *TCP/IP* Protokol Katmanlarına ilişkin Veri Birimleri

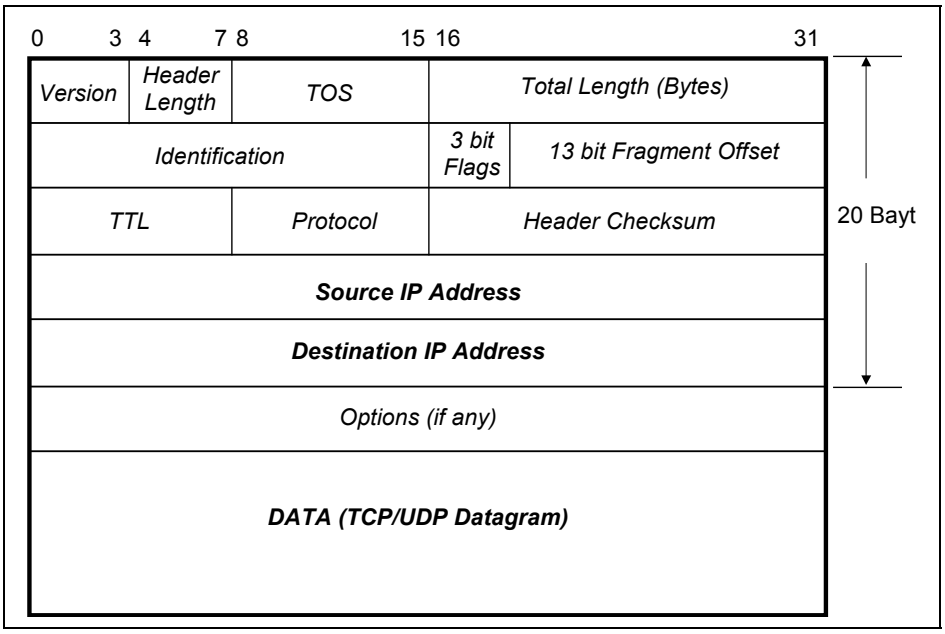
Bu katman düzeyinde, karşı taraftan alınan her kesim, hata ve sıra denetimine sokulur[32]. Hatasız alınan kesimlere ilişkin "alındı" *(ack)* gönderilir. Hatalı alınan kesimler, karşı taraftan yeniden istenir. Bunun yapılabilmesi için, *TCP* katmanı, uygulama katmanından kendisine aktarılan iletileri ya da bunları oluşturan kesimleri datagram olarak adlandırılan yapılara dönüştürür. Datagram, uygulama katmanından gelen iletiye bir başlık eklenerek oluşturulur. Sözkonusu başlık, *TCP* başlığıdır. *TCP* başlığı, sıra denetimi yapabilmek amacıyla bir "sıra numarası" da içerir. Oluşturulan datagram, *TCP* tarafından, *IP (Internet Protocol)* katmanına aktarılır.

IP katmanı da, yönlendirme işlevlerinin yerine getirilebilmesi için, sözkonusu datagrama kendi başlığını (*IP* başlığını) ekleyerek datagramı pakete dönüştürür (Çizim 9.4). *IP* başlığı, öncelikle "kaynak" ve "hedef" bilgisayarların *IP* adreslerini taşır. Hedef adrese dayalı olarak yol belirleme ya da yönlendirme işlemleri yürütülür. Oluşturulan

---

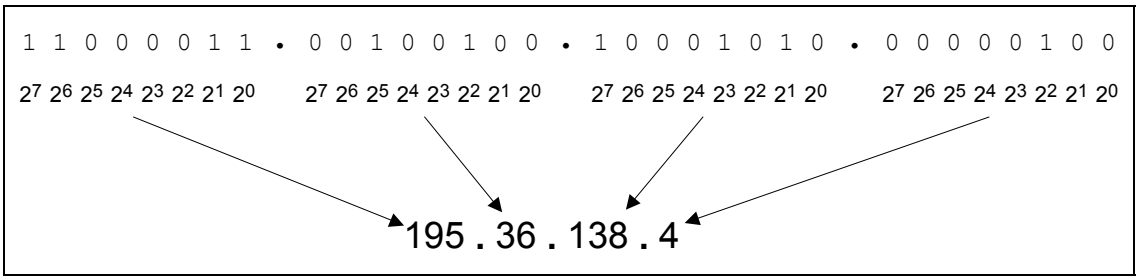
[32] Hata denetimi, Hata Denetim Sözcüğüne *(TCP Checksum)* dayalı olarak yürütülür. Sıra denetiminde, *Idle RQ, Continuous RQ Selective Repeat, Continuous RQ Go Back N* gibi adlarla bilinen algoritmalar kullanılır.

paket, veri bağlantı katmanınca, bağlı bulunulan (yerel ya da geniş) ağ standartına uygun çerçevelere dönüştürülerek, veri iletişim arabirimi (donanımı) aracılığıyla, veri iletişim ağı üzerinden aktarılır.



Çizim 9.4. *IP* Paket Görünümü

IP adresleri Internet ağı içinde yer alan bilgisayarları biricik olarak tanımlama amacıyla kullanılan 32 bit uzunluğunda adreslerdir[33]. Bir bilgisayarın *IP* adresi, *IP* numarası olarak da bilinir. Bu numaralar, telefon ağı içinde bir aboneyi tanımlayan telefon numaralarına benzetilebilir. 32 bitlik ikili kodlu numaralar kolay kullanım (okuma, yazma ve anımsama) amacıyla 8 bitlik gruplara ayrılır ve her grup onlu tabandaki değeriyle ifade edilir. Bu gösterim biçimi "noktalı onlu" gösterim olarak adlandırılır (Çizim 9.5).

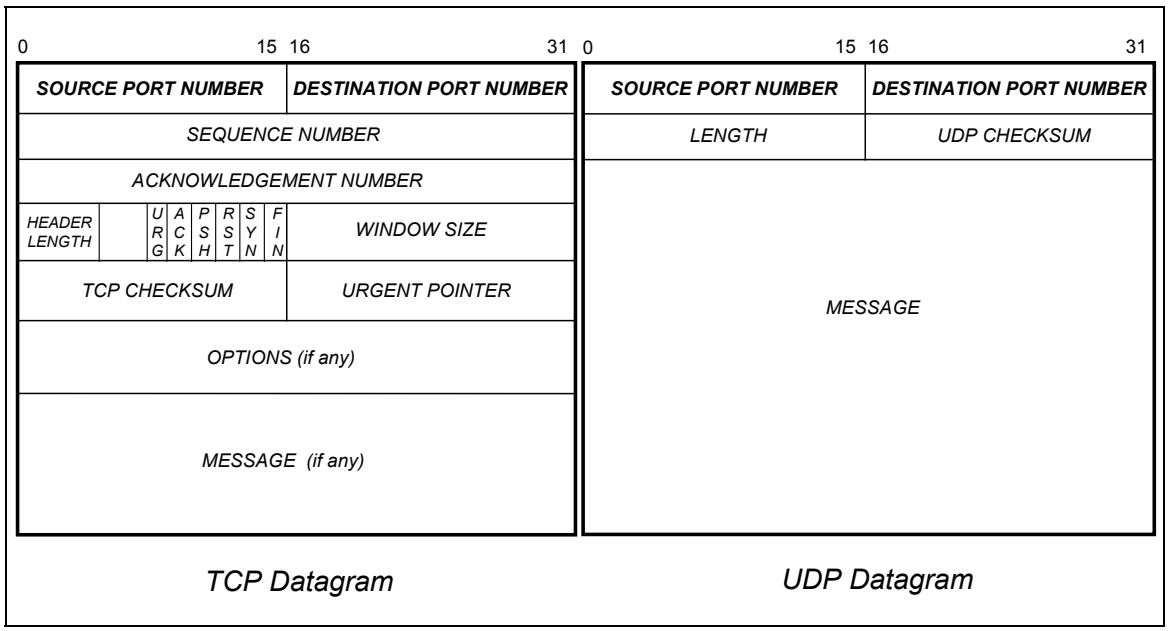


Çizim 9.5. *IP* Adreslerinin Noktalı Onlu Gösterimi

[33] 32 bitlik adresler, *Ipv4* olarak bilinen *TCP/IP* yazılımının bugünkü sürümü için geçerlidir. Adres boyları *IPv6* adlı yeni sürümde 128 bit olacaktır.

9.1.2. *TCP* ya da *UDP*

TCP/IP yazılımı ulaşım katmanında *TCP* kesimi yerine *UDP (User Datagram Protocol)* olarak bilinen kesim de kullanılır. Yukarıda belirtildiği gibi, *TCP* kesimi, uygulama düzeyinden gelen iletileri, hatasız ve sıraları bozulmadan, karşı eşdeğer katmana gönderme işlevini üstlenir. Eğer bir uygulama için, gönderilen ve alınan iletilerin sırasının bozulması bir sakınca oluşturmuyorsa ya da bu işlevi uygulama kendisi zaten üstleniyorsa, ulaşım katmanı düzeyinde *TCP* kesimi yerine, sıra denetimi yapmayan *UDP* kesimi kullanılabilir. *UDP* sıra denetim işlevi dışında *TCP* kesimine eşdeğer, bu kesime göre daha yalın dolayısıyla hızlı çalışan bir kesimdir.



Çizim 9.6. *TCP* ve *UDP* Datagram görünümleri

TCP kesiminin uygulama programlarına sunduğu, sıra denetim işlevini de içeren hizmet, oturum tabanlı, İngilizce, *connection oriented* hizmet olarak nitelenir. İletişime giren bilgisayarların *TCP* katmanları, sıra denetim işlevini yerine getirebilmek için, datagram aktarım öncesi bir oturum başlatmak ve sıra numaralarını tutacak sıra değişkenlerini, birlikte sıfırlamak durumundadırlar. Bu nedenle, uygulama katmanına ilişkin bir dizi ileti aktarımı sözkonusu olduğunda, *TCP* katmanının, karşı eşdeğer katmanı (*TCP* katmanını) arayarak bir oturum başlatması, sıra değişkenlerine dayalı, sıralı datagram aktarımını gerçekleştirmesi, aktarım sonlanınca da, sözkonusu oturumu kapatması gereklidir. Bu iletişim mantığı, abonelerin telefonda yaptığına benzediğinden, bu bağlamda verilen hizmet, İngilizcede, *(connection oriented)* "bağlantı kurmaya dayalı" hizmet olarak nitelenmektedir. Buradaki bağlantı kurma, doğal olarak telefondaki gibi fiziksel bir bağlantı kurma anlamında değildir. Bu nedenle, yukarıda verilen açıklamada "bağlantı kurma" yerine "oturum başlatma" ifadesi tercih edilmiştir.

Yukarıda da belirtildiği gibi, *UDP* kesiminin uygulama programlarına sunduğu hizmet sıra denetim işlevini içermez. Bu nedenle, sözkonusu hizmet, "oturum başlatmayı

gerektirmeyen" (oturumsuz) ya da İngilizce, *connection oriented* ifadesiyle tezat oluşturmak üzere *connectionless* hizmet olarak nitelenir. *UDP* kesimi, uygulama katmanından kendisine aktarılan iletileri, sıra denetimi yapmaksızın, *UDP* başlığı ekleyerek datagramlara dönüştürüp paket katmanına aktarmakla yetinir. Bu sayede, *UDP* kesiminin gerçekleştirimi, *TCP* kesimine göre çok daha yalın, işletimi de bir o kadar hızlıdır. Uygulamalar, (hız, hizmet) gereksinimlerine göre, ulaşım katmanında *TCP* ya da *UDP* kesimlerinden herhangi birini kullanabilmektedir. Bu tercihin nasıl yapıldığı, uygulamaların, *TCP/IP* yazılımından iletişim hizmeti alma düzeneğinin incelendiği, *Socket* Sistem Çağrı Düzeneği adlı kesimde açıklanacaktır.

Gerek *TCP* gerekse *UDP*, datagrama dönüştürmek üzere iletiye ekledikleri başlıkta, "kapı" *(port)* numarası olarak adlandırılan, 16 bit uzunluğunda numaralara yer verirler (Çizim 9.5). Kapı numaraları, *TCP/UDP* katmanından hizmet alan programları tanımlamaya yarar. Başlıkta kaynak ve hedef olmak üzere iki ayrı kapı numarasına yer verilir. Kaynak kapı numarası, datagramın taşıdığı iletiyi gönderen programı, hedef kapı numarası da, iletiyi alacak programı gösterir. *TCP/UDP* katmanı, bu numaralardan yararlanarak, kendisine ulaşan datagramları, üst katmanında yer alan uygulamalardan hangisine yönlendireceğini belirler. Kapı numaraları uygulama programlarınca belirlenip *TCP/UDP* katmanına aktarılır. Uygulama programlarının belirleyeceği kapı numaraları 1024'den 65535'e kadar herhangi bir değeri alabilir. 1024'den daha küçük numaralar, *Telnet (Terminal Emulation), FTP (File Transfert Program), SNMP (Simple Network Management Program), SMTP (Simple Mail Transfert Program)* gibi *TCP/IP* yazılımıyla birlikte sağlanan klasik hazır uygulamalara ayrılmıştır.

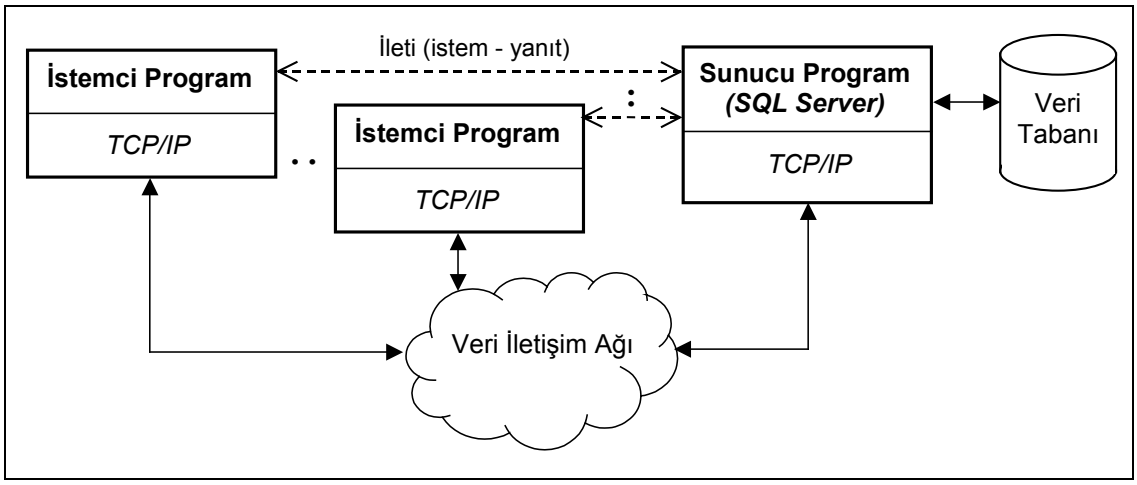
Kapı numaralarının yanı sıra, iletinin gönderileceği bilgisayarın *IP* numarası da uygulama programınca ulaşım katmanına sağlanır. Ulaşım katmanı, bu numarayı, kapı numaralarını da kullanarak oluşturduğu datagramla birlikte paket katmanına aktarır. Bu durumda, birbirleriyle iletişim içindeki uygulama programlarını, {*IP* Numarası; Kapı Numarası} ikilisi ile (`193.140.216.33; 1490` gibi), ağ içinde biricik olarak tanımlama olanağı bulunur.

9.2. *Socket* Sistem Çağrı Düzeneği

Bilindiği gibi, uygulama programları, işletim sisteminden sistem çağrı düzeneğine dayalı olarak hizmet alırlar. Nasıl uygulama programları, kütüklerle ilgili işlemlerde kütük yönetim sistemince ele alınan `open(), close(), seek(), read(), write()` gibi kütük sistem çağrılarını kullanırlarsa diğer bilgisayar sistemleri üzerinde çalışan programlarla iletişim amacıyla da, işletim sistemi iletişim alt kesiminin sistem çağrılarını kullanırlar. Bu bağlamda, *TCP/IP* yazılımından, uygulamalar, *socket* olarak adlandırılan düzeneğe ya da sistem çağrı kümesine dayalı olarak hizmet alırlar. *TCP/IP* sistem yazılımını iletişim alt sistemi olarak kullanan işletim sistemlerinden *UNIX*'te bu düzenek *Socket*, *MS-Windows* işletim sisteminde ise *WinSock* olarak anılmaktadır. Kullanılacak *socket* çağrı kümesi, sözkonusu uygulamanın "sunucu" ya da "istemci" olmasına göre farklılık gösterir. İzleyen kesimde, *socket* düzeneği, *UNIX* için açıklanıp örneklenecektir.

9.2.1. İstemci-Sunucu Yaklaşımı

Daha önce de açıklandığı üzere, Internet ağı içinde bütünleşen irili ufaklı bilgisayar sistemleri, birbirlerinin işlem gücünü paylaşabilmek amacıyla, istemci-sunucu yaklaşımını kullanırlar. Bu yaklaşımda, sunucu olarak nitelenen kimi programlar, istemci olarak adlandırılan programlardan gelen iletiler içinde tanımlı işlemleri gerçekleştirip sonuçlarını geri göndererek, istemci programlara, kendi bilgisayarları üzerinde hizmet üretmiş olurlar. Bir veri tabanı üzerinde, örneğin, sorgulama, günleme gibi bir işlemi, istemci-sunucu yaklaşımıyla uzaktan yapabilmek için; uzak bilgisayarlar (veri tabanının yer aldığı bilgisayar sistemiyle aynı ağ içinde yer alan diğer bilgisayarlar) üzerinde, bu amaca dönük, istemci nitelikli bir program öngörülür. Bu program, işlem istem iletilerini, veri tabanıyla aynı bilgisayar üzerinde yer alan sunucu bir programa (örneğin *SQL Server* sunucu programına) gönderir. Sunucu program sözkonusu sorgulama, günleme işlemlerini veri tabanı üzerinde yürütüp sonuçları istemci programa geri döndürür (Çizim 9.7).



Çizim 9.7. İstemci-Sunucu Yaklaşımı ile Uzak Bilgisayardan Hizmet Alma

Bunun, istemci-sunucu yaklaşımı dışındaki alternatifi, uzak bilgisayara *(login* ile) giriş yapmak ve sorgulama, günleme işlemlerini, bizzat bu bilgisayar üstünde çalışan bir uygulama programı ile ve bu bilgisayarın kullanıcısı olarak yürütmektir. İstemci-sunucu yaklaşımının, uzak bilgisayarlardan klasik hizmet alma yaklaşımından farklılığı da buradadır. Bu nedenle, dağıtılmış işlemin sunuş kesiminde, İstemci-sunucu yaklaşımı ile bir bilgisayar sisteminden yararlanabilmek için mutlaka o bilgisayar sistemine giriş yapıp (bağlanıp) kullanıcıya özel bir program çalıştırma zorunluluğunun ortadan kalktığından, bunun yerine bir uygulamaya (sunucu programa) bağlanıp hizmet almaktan söz edilmiştir.

9.2.2. Sunucu Program için *Socket* Sistem Çağrıları

Yukarıda da belirtildiği gibi, kullanılacak *socket* çağrı kümesi, sözkonusu uygulama programının "sunucu" ya da "istemci" olmasına göre farklılık gösterir. Zira, bu programların işlevleri ve çalışma mantıkları farklıdır. Sunucu nitelikli programlar,

sistem açıldığında çalışmaya başlayan, istemcilerden gelebilecek bağlantı ve hizmet istemlerini bekleyen, istem geldiğinde hizmet üreten ve genelde sonlanmayan programlardır. İstemci nitelikli programlar ise, hizmet gereksinimi ortaya çıktığında sunucuya bağlanarak çalışan ve sonlanan programlardır. *UNIX* bağlamında, sunucu programların kullandığı *socket* sistem çağrıları, işletim sıraları ile aşağıda verilmiştir:

```
socket();
bind();
listen();
accept();
read();/write();
close();
```

`socket()` sistem çağrısı, sunucu programa bağlantı noktası olarak da adlandırılabilen *socket* nesnesinin, işletim düzeyinde yaratılması amacıyla kullanılır. Görünümü ve argümanları aşağıda verilmiştir:

```
int sock = socket(arg1, arg2, arg3);
```

`arg1` *socket* nesnesinin kullanacağı adres sınıfını belirlemeye yarar. Adres sınıfı, genelde 2 değişik türden biri olabilmektedir. Bunlar:

```
AF_UNIX
AF_INET
```

adres sınıflarıdır. `AF_INET` adres sınıfı *Internet* adres sınıfını gösterir.

`arg2` sözkonusu *socket* nesnesini içeren sunucu programın, ulaşım katmanı düzeyinde *TCP* ya da *UDP* kesimlerinden hangisini kullanacağını belirlemeye yarar. Bu argümanın alabileceği değerler üç tanedir:

```
SOCK_STREAM
SOCK_DGRAM
SOCK_RAW
```

`SOCK_STREAM` *TCP*, `SOCK_DGRAM` *UDP* kesiminin kullanılacağını gösterir. `SOCK_RAW` değeri ise, kimi özel uygulamaların, ulaşım katmanını kısa devre yaparak doğrudan *IP* katmanını kullanabilmesi için öngörülmüş özel bir değerdir.

`arg3` *socket* nesnesinin, işletim sisteminin, varsa birden çok iletişim alt kesiminden hangisinin kullanılcağını belirlemek için öngörülmüştür. Bu değer genelde sıfır olarak verilir. Sıfır değeri, işletim sisteminin "*default*" olarak belirlediği iletişim alt sisteminin (örneğin *TCP/IP* alt sisteminin) tercih edildiğini gösterir. Bu durumda `socket()` sistem çağrısının örnek kullanımı aşağıdaki gibi olabilir:

```
int sock = socket(AF_INET, SOCK_STREAM, 0);
```

`socket()` sistem çağrısının, tamsayı türü geri döndürdüğü değer, izleyen kesimde açıklanan, `bind()`, `listen()` ve `accept()` sistem çağrıları tarafından kullanılır.

bind() sistem çağrısı, `socket()` sistem çağrısı ile yaratılan socket nesnesini, {IP Numarası; Kapı Numarası} ikilisiyle belirlenen bir adresle ilişkilendirmek için kullanılır. Görünümü aşağıdaki gibidir:

```
bind(arg1, arg2, arg3);
```

`arg1` `socket()` sistem çağrısının, geri döndürdüğü `sock` değeridir.

`arg2` socket nesnesini yaratan sunucu programın kullandığı (örneğin `struct sockaddr_in` adlı) veri yapısı ile ilişkili, *generic* nitelikli, (örneğin `struct sockaddr` adlı) bir veri yapısına göstergedir *(pointer)*.

```
struct sockaddr {
  u_short  sa_family;
  char     sa_data[14];
};
```

```
struct sockaddr_in {
  short    sin_family;         /* AF_NET */
  u_short  sin_port;           /* port number */
  struct   in_addr sin_addr;   /* IP Address */
  char     sin_zero[8],        /* padding */
};

struct in_addr {
  u_long   s_addr;
}
```

`arg3` `sockaddr` adlı yapının boyudur.

Bu durumda, `bind()` sistem çağrısının örnek kullanımı aşağıdaki gibi olabilir:

```
struct sockaddr_in sunucu;
bind(sock, (struct sockaddr *)&sunucu, sizeof sunucu);
```

`sunucu` adlı yapı içerisinde `sin_port` (sunucu programın belirlediği/kullandığı kapı numarası) ve `sin_addr` (sunucu programın üzerinde çalıştığı bilgisayarın *IP* adresi) içerikleri, `bind()` sistem çağrısı işletilmeden önce günlenmek zorundadır.

listen() sistem çağrısı, sunucu programın, aynı anda kaç istemci programın bağlantı istemini kabul edeceğini belirleme amacıyla öngörülmüş bir çağrıdır. Görünümü aşağıdaki gibidir:

```
listen(arg1, arg2);
```

`arg1` `socket()` sistem çağrısının, geri döndürdüğü `sock` değeridir.

`arg2` sunucu programa bağlantı isteminde bulunacak istemci programların kuyruklanacağı kuyruk boyu değeridir. Kuyruk boyu, örneğin (n) olarak belirlendiği bir durumda, (n) adet istemci programa ilişkin hizmet istemleri halen sürerken (n+1)inci bağlantı istemi *TCP/IP* sistem yazılımınca reddedilecektir. `listen()` sistem çağrısının örnek kullanımı aşağıdaki gibi olabilir:

```
listen(sock, 5);
```

accept() sistem çağrısı, sunucu programın, istemci programların bağlantı istemlerini üzerinde beklediği sistem çağrısıdır. Görünümü aşağıdaki gibi örneklenebilir:

```
struct sockaddr_in istemci;
int fd;
fd = accept(sock, &istemci, sizeof istemci);
```

`accept()` sistem çağrısını çalıştıran sunucu program, herhangi bir istemci program bağlantı isteyinceye (ileride açıklanacağı üzere, `connect()` sistem çağrısını işletene) değin bekler duruma geçer. `&istemci` olarak nitelenen argüman (adres göstergesi), sunucu programa bağlanmak isteyen istemcinin, `sockaddr_in` yapısında, {IP Numarası; Kapı Numarası} ikilisiyle belirlenen bilgilerinin sunucuya geri döndürülmesinde yararlanılan bir argümandır. Sunucu, bu bilgileri kullanarak, isterse, bağlantı talebinde bulunan kimi istemcilere hizmet vermeyi reddedebilsin (bağlantı talebinde bulunan istemcileri denetleyebilsin) diye öngörülmüştür.

`accept()` sistem çağrısı `fd` adlı bir değer geri döndürmektedir. Bu değer kullanılarak, bağlantı istemi kabul edilen istemci ile sunucu arasındaki bayt aktarım işlemleri, aynı, kütükten okuma/kütüğe yazma (**read**`(fd, buf, nbyte);` **write**`(fd, buf, nbyte);`) işlemleri gibi, sözkonusu istemciye özgü `fd` göstergesine dayalı olarak yürütülmektedir. Bu, *UNIX* işletim sisteminin tüm giriş/çıkış işlemlerini kütük işlemleri olarak ele alma felsefesine koşut bir yaklaşımdır. Okuma/yazma işlemleri son bulunca, yine kütük işlemlerinde olduğu gibi, **close**`(fd)` sistem çağrısı aracılığıyla, bir istemci ile kurulan bağlantının sonlandırılması gerçekleştirilebilmektedir.

UNIX Socket düzeneğinde, sunucu nitelikli programlar için, `socket()` ve `accept()` sistem çağrılarının geri döndürdüğü (`sock` ve `fd`) iki değişik gösterge kullanılmaktadır. Bunlardan `socket()` sistem çağrısının geri döndürdüğü (`sock`) *rendez vous descriptor*, `accept()` sistem çağrısının geri döndürdüğü (`fd`) de, *connection descriptor* olarak adlandırılmaktadır. Bu sonuncu, `read()/write()` sistem çağrılarının kullanmak zorunda olduğu göstergedir. İzleyen kesimde görüleceği üzere, istemci nitelikli programlarda, (`socket()` sistem çağrısı dışında) tüm sistem çağrıları, tek bir gösterge (`socket()` sistem çağrısının geri döndürdüğü göstergeyi (`sock`)) kullanmaktadır.

9.2.3. İstemci Program için *Socket* Sistem Çağrıları

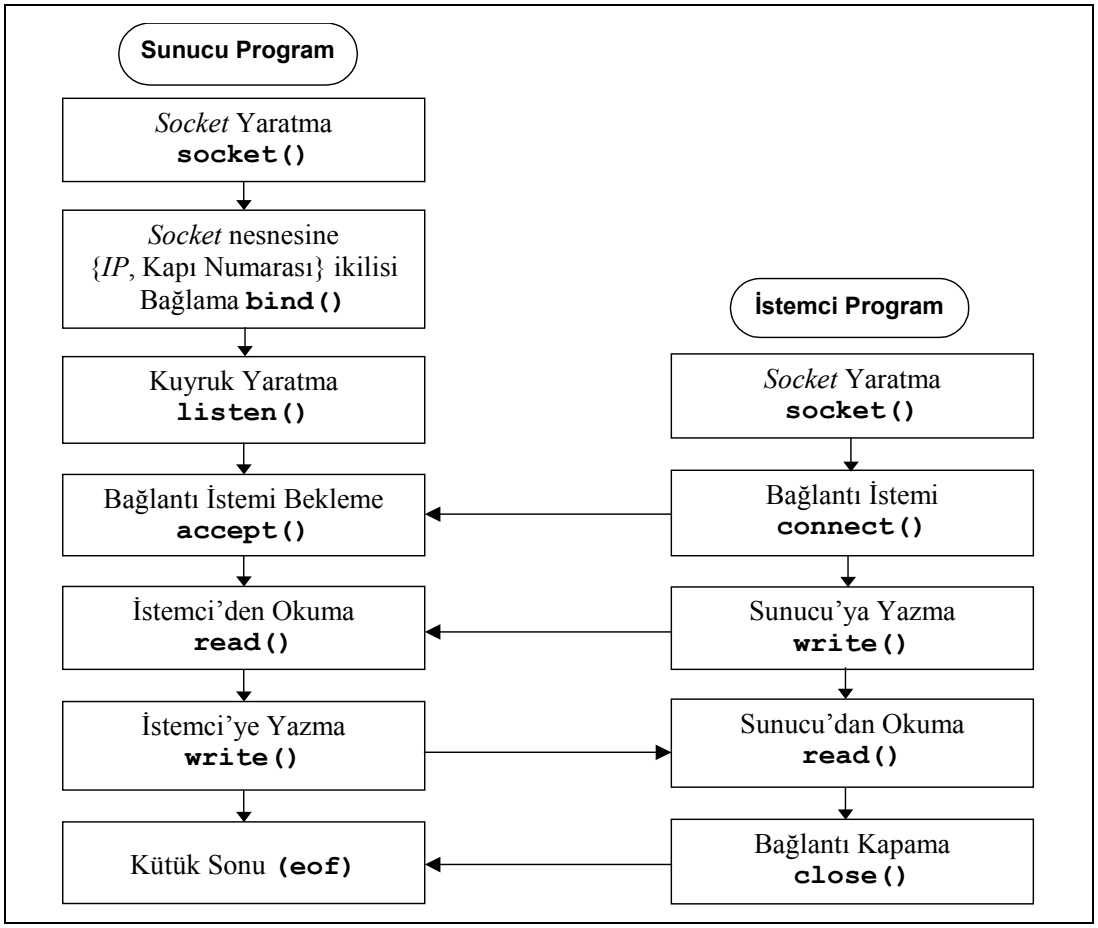
İstemci programların kullandığı *socket* sistem çağrıları, işletim sıraları ile aşağıda verilmiştir:

```
socket();
connect();
read();/write();
close();
```

Bunlardan, `socket()`, `read()`, `write()` ve `close()` sistem çağrıları, sunucu nitelikli programların kullandıklarıyla aynı işlev ve görünümdedir. `connect()` sistem çağrısının görünümü aşağıda verilmiştir:

```
struct sockaddr_in sunucu;
connect(sock, (struct sockaddr *)&sunucu, sizeof sunucu);
```

`connect()` sistem çağrısı, sunucu program düzeyi `accept()` sistem çağrısı ile iletişim kuran bir çağrıdır. Bu bilgiler ışığında istemci program-sunucu program iletişimi Çizim 9.8'de gösterildiği biçimde yürütülmektedir.



Çizim 9.8. *Socket* Sistem Çağrıları ve İstemci Program-Sunucu Program İletişimi

9.2.4. Sunucu ve İstemci Program Örnekleri

Socket sistem çağrılarının, istemci-sunucu mantığı içerisinde nasıl kullanıldıkları, sırasıyla, Çizim 9.9 ve 9.10'da verilen yalın, sunucu ve istemci nitelikli iki programla örneklenmektedir. Bu programlardan istemci nitelikli olanı (`istemci`), klavyeden "*enter*" tuşuna basılana değin girilen damgaları, sunucu nitelikli programa (`sunucu` adlı programa) göndermekte, bu program da, aldığı damgaları, olduğu gibi, gönderene geri yollamakta, başka bir deyişle "satır yankılama" *(line echo)* işlemi yapmaktadır.

```
/* Örnek sunucu program */

#include <sys/types.h>
#include <sys/socket.h>
#include <netinet/in.h>
#include <stdio.h>
#include <netdb.h>

#define MAX 80
#define SUNUCU_KAPISI 1490

main()

{

char yastik[MAX];
int fd, nb, sock;
unsigned long istemci_boyu;
struct sockaddr_in sunucu, istemci;

sunucu.sin_family = AF_INET;
sunucu.sin_addr.s_addr = INADDR_ANY;
sunucu.sin_port = htons(SUNUCU_KAPISI);

sock = socket(AF_INET, SOCK_STREAM, 0);
bind(sock, (struct sockaddr *)&sunucu, sizeof sunucu);
listen(sock, 5);
istemci_boyu = sizeof istemci;

while(1) {
  fd=accept(sock,(struct sockaddr *)&istemci, &istemci_boyu);
  while((nb = read(fd, yastik, MAX)) > 0)
       write(fd, yastik, nb);
  }

}
```

Çizim 9.9. Satır Yankılama Yapan Sunucu Program

```
/* Örnek istemci program */

#include <sys/types.h>
#include <sys/socket.h>
#include <netinet/in.h>
#include <stdio.h>
#include <signal.h>
#include <netdb.h>

#define MAX 80
#define SUNUCU_KAPISI 1490

main(int argc, char *argv[])
{
int nb, sock;
char yastik[MAX];
char sunucu_adi[MAX];
struct sockaddr_in sunucu;
struct hostent *s_bilgi;

sock = socket(AF_INET, SOCK_STREAM, 0);
strcpy(sunucu_adi, argv[1]);
s_bilgi = gethostbyname(sunucu_adi);
sunucu.sin_family = s_bilgi->h_addrtype;
memcpy(&sunucu.sin_addr, s_bilgi->h_addr, s_bilgi->h_length);
sunucu.sin_port = htons(SUNUCU_KAPISI);

if((connect(sock,(struct sockaddr *)&sunucu, sizeof sunucu))< 0){
  printf("Bağlanti yapilamiyor\n");
  exit(1);
  }
printf("%s adli sunucuya baglanti yapildi.\n", argv[1]);
while((nb = read(0, yastik, MAX)) > 0){ /* Klavyeden oku */
     write(sock, yastik, nb);           /* Sunucu'ya yolla */
     nb = read(sock, yastik, MAX);      /* Sunucu'dan geri al */
     write(1, yastik, nb);              /* Ekrana yaz */
     }
close(sock);
}
```

Çizim 9.10. Satır Yankılama İsteyen İstemci Program

Bu programlar, derlenip `sunucu` ve `istemci` adlı işletilir program kütükleri elde edildikten sonra, eğer her iki program da aynı bilgisayar üzerinde çalıştırılacaksa, örneğin:

```
defne> sunucu&
PID 24679
defne> istemci localhost
localhost adli sunucuya baglanti yapildi.
```

biçiminde işletime alınırlar. `sunucu` adlı program, `istemci` adlı programla koşut olarak arka planda çalıştırılmak üzere, adının sonuna & eklenerek sisteme sunulur.

Eğer, `istemci` adlı program, `sunucu` adlı programla farklı bir bilgisayar üzerinde çalıştırılacaksa, argüman olarak, `sunucu` adlı programın üzerinde çalıştığı bilgisayarın `(defne)` adını taşımak durumundadır. Bu şıkta, `istemci` adlı program, örneğin:

```
yunus> istemci defne
defne adli sunucuya baglanti yapildi.
```

biçiminde işletime alınır. `istemci` adlı program işletime alınmadan önce, `sunucu` adlı program, `defne` adlı bilgisayar üzerinde:

```
defne> sunucu&
```

komutuyla işletime alınmış olmalıdır.

`sunucu` adlı programdan, terminal öykünümü yapmak üzere, satır gönderme ve alma işlevlerini yerine getirebilen `telnet` adlı hazır (istemci) program aracılığıyla hizmet almak da mümkündür. `telnet` programı, `sunucu` adlı programın çalıştığı bilgisayardan `(defne)` farklı bir bilgisayar (`yunus`) üzerinde çalıştırılacaksa, `sunucu` adlı programın çalıştığı bilgisayarın adının yanı sıra kapı numarasını da, aşağıda gösterildiği biçimde, argüman olarak almak durumundadır:

```
yunus> telnet defne 1490
Trying...
Connected to defne@bil.hacettepe.edu.tr
Escape character is '^]'.
```

9.2.5. Alan Adı Sistemi *(DNS-Domain Name System)*

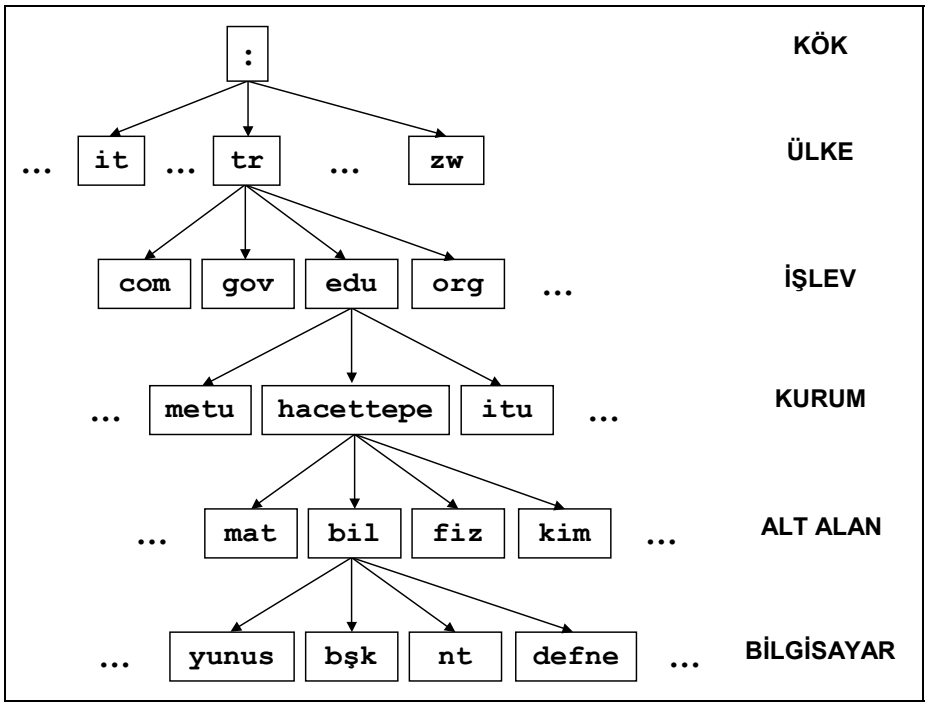
TCP/IP bağlamında istemci-sunucu yaklaşımı açıklanırken, istemci ve sunucu nitelikli uygulama programlarının, (`193.140.216.33; 1490` gibi), {*IP* Numarası; Kapı Numarası} ikilisi ile biricik olarak tanımlandığından söz edilmişti. Ancak, yukarıda verilen örneklerde, bilgisayarların, *IP* numaraları yerine, `lidya`, `eti` gibi simgesel adlarından söz edilmiştir. Internet içindeki bilgisayarları, anımsanmaları göreli zor ve uzun numaraları yerine yalın simgesel adları ile tanımlayabilme olanağı, *TCP/IP* uyumlu, Alan Adı Sistemi *(DNS, Domain Name System)* olarak adlandırılan dağıtılmış bir veri tabanı sistemi ile sunulur.

Bu sistem gereği, Internet içinde yer alan, bir kuruma özgü yerel ağ için, ilgili kurumca, Internet Merkezi Otoritesinden (ya da bu otoritenin yetkili kıldığı ülke düzeyi yerel bir otoriteden) alan adı satın alma zorunluluğu bulunur[34]. Bu bağlamda, Internet alan adları, kütüklerle ilgili sıradüzensel kılavuz yapısına benzer bir yapı içinde ele alınır. Kurumlar, ilk aşamada (`tr, uk, fr` gibi) ülkelere, ülke içinde (`com`, `net`, `gov`, `edu` gibi) işlev sınıflarına ayrılırlar. Bir kurumun, ilgili olduğu ülke ve işlev sınıfı içinde tanımlı (`hacettepe.edu.tr` gibi) biricik bir alan adı bulunur. Internet otoritesinden

[34] Merkezi otorite, ABD'de *INTERNIC (Internet Information Center)* adlı merkezdir. Bu Merkezin Türkiye'de yetkili kıldığı yerel otorite, ODTÜ, Orta Doğu Teknik Üniversitesi'dir. Sözkonusu bu otoriteler, alan adlarının yanı sıra bu alanlar altında yer alan bilgisayarların *IP* Numaralarını da sağlamaktadır.

satın alınan ad bu addır. Kurum isterse bu ad altında, (`bil.hacettepe.edu.tr` gibi) alt alan adları tanımlayabilir. Bunların altında, (`defne@bil.hacettepe.edu.tr` gibi) kurumun, o alt alanla ilgili bilgisayar adları yer alır (Çizim 9.11).

Simgesel bir ada sahip her bilgisayarın bir de *IP* numarası bulunur. Simgesel ad, doğal olarak kullanıcılar için anlamlıdır. Yukarıda örneklendiği üzere, *TCP/IP* yazılımı ve bununla uyumlu uygulama programları, bilgisayarları tanımlarken *IP* numaralarını kullanırlar. Bu durumda, bir bilgisayarın simgesel adından, gerektiğinde, *IP* numarasına geçişi sağlayacak bir düzeneğe gereksinim duyulur.

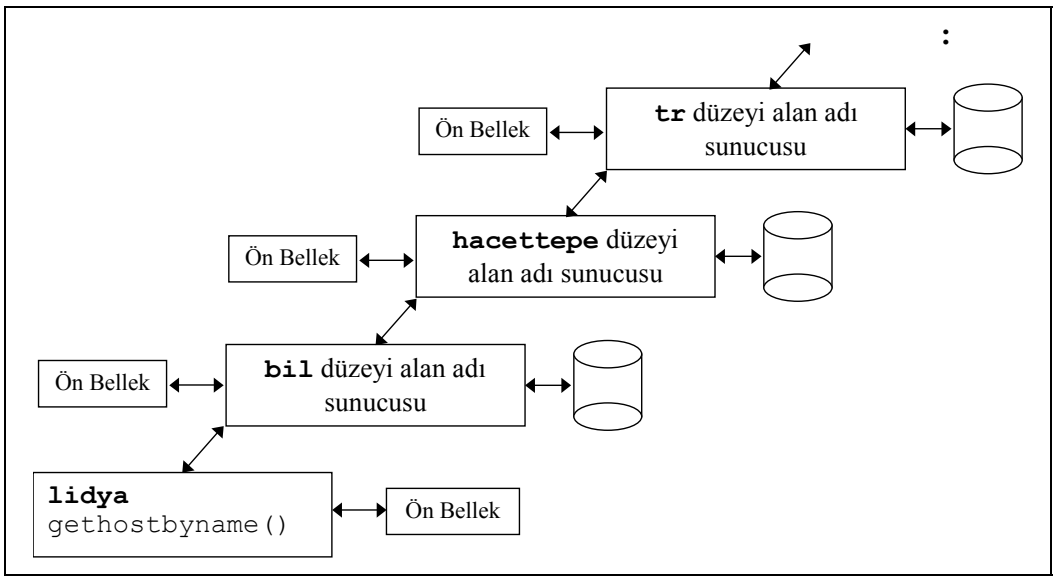


Çizim 9.11. Alan Adı Sistemi Adlandırma Sıradüzeni

Bu düzenek, istemci-sunucu mimarisinde, Alan Adı Sistemi *(DNS, Domain Name System)* olarak adlandırılan, dağıtılmış bir veri tabanı sistemidir. Uygulama programları, kendilerine sunulan simgesel bilgisayar adlarından *IP* numaralarına geçiş için, üzerinde çalıştıkları bilgisayar için tanımlı bir Alan Adı Sunucusuna (*Domain Name Server* ya da *Resolver* adlı bir sunucuya) başvururlar. Bunu yapabilmek için, *UNIX*'te `gethostbyname(simgeselad)` adlı bir kitaplık işlevinden yararlanılır[35]. Bu işlev, alan adı sunucusuna istemci olarak bağlanıp "`simgeselad`"a karşılık gelen *IP* numarasını elde eder. İstemin yöneltildiği alan adı sunucusu, sözkonusu ada karşılık gelen *IP* numarası kendi veri tabanında yer almıyorsa, kendisi için tanımlı, alan adı sıradüzeninde bir basamak üstte başka bir alan adı sunucusuna başvurabilir. Bu yolla,

[35] Bu bağlamda, `gethostbyname()` işlevinin yanı sıra, `gethostbyaddress()` diye diğer bir işlev daha vardır. Adından da anlaşılacağı üzere, bu işlev, *IP* adresi verilen bilgisayarın simgesel adını elde etmeye yarar.

Internet içinde tanımlı herhangi bir bilgisayarın *IP* numarasını elde etmek mümkün olur. Bu düzeneğin başarımını artırmak üzere, alan adı sunucuları, yapılan başvuruları sonuçlarıyla birlikte bir ön bellek alanında da tutarlar. Bu yolla, sık sık sorgulanan aynı bilgisayara ilişkin adres, doğrudan ön bellekten sağlanır (Çizim 9.12).



Çizim 9.12. Alan Adı Sunucuları Arası İşbirliği Düzeneği

Sözkonusu `gethostbyname()` işlevi, örnek istemci program içinde, bu programa argüman olarak sunulan ve bağlanılmak istenen sunucu programın üzerinde çalıştığı bilgisayarın (örneğin, `defne` ya da `localhost`) *IP* numarasını elde etmek amacıyla kullanılmıştır.

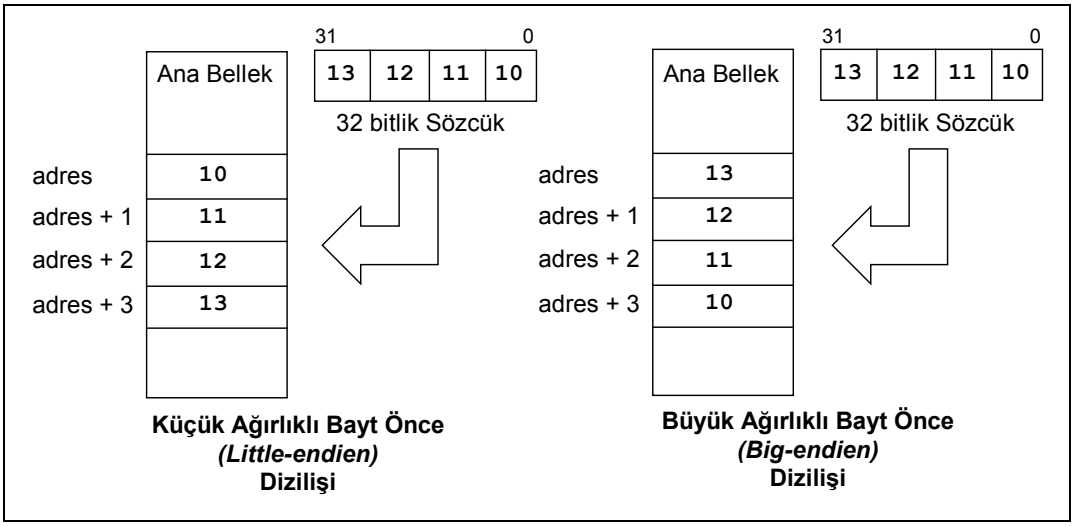
9.2.6. Örnek Sunucu ve İstemci Programlarla İlgili Açıklamalar

Alan adı sistemi açıklanırken, bilgisayar adlarının, bilgisayar_adı@alan_adı yapısında (`defne@bil.hacettepe.edu.tr` gibi) olması gerektiği söylenmiştir. Halbuki, yukarıda verilen işletim örneklerinde, istemci nitelikli programa (örneğin `istemci` ya da `telnet` adlı programa) argüman olarak sunulan bilgisayar adı, kısaca `defne` ya da `localhost` olarak verilmiştir. Bunun gerekçesi, *TCP/IP* uyumlu bilgisayar sistemlerinin, (@ işaretiyle başlayan) alan_adı kesimi içermeyen kısa bilgisayar adlarını, varsayılan *(default)* bir alan_adı ile tamamlamasıdır. Bu nedenle, eğer sözkonusu bilgisayar sisteminin (`yunus`) varsayılan alan_adı, `bil.hacettepe.edu.tr` ise, bilgisayar adını `defne` ya da `defne@bil.hacettepe.edu.tr` olarak, iki biçimde de vermek mümkün olmaktadır.

Socket düzeneğinin kullanımını örneklemek üzere verilen programlarda, çoğu sistem çağrısından geri dönüşte, program görünümünü karmaşıklaştırmamak amacıyla, bilinçli olarak, hata denetimine yer verilmemiştir. Ancak, gerçek uygulamalarda, her sistem çağrısından sonra, örneğin:

```
.
.
sock = socket(AF_INET, SOCK_STREAM, 0);
if(sock < 0) {
     printf("soket yaratma hatası\n");
     exit(1);
     }
if((bind(sock, (struct sockaddr *)&sunucu, sizeof sunucu)) < 0){
     printf("bağlama hatası\n");
     exit(2);
     }
.
.
```

biçiminde, hata denetimi yapmak gerektiği unutulmamalıdır.



Çizim 9.13. Bir Sözcüğe ilişkin Baytların Ana Bellekte Saklanış Biçimleri

`sunucu.sin_port = htons(SUNUCU_KAPISI)` komutu ile sunucu program kapı numarası günlenirken kullanılan `htons()` işlevi, bilgisayar ve ağ düzeyinde geçerli olan bayt dizilişlerini uyumlama amacını güdmektedir. Bilindiği üzere, birden çok bayt uzunluğundaki sözcükler (örneğin 16 bitlik kapı numarasını tutan sözcük), ana bellekte, iki değişik biçimde saklanabilmektedir. Bunlardan ilki, küçük ağırlıklı bayt önce *(little-endien)*; ikincisi, büyük ağırlıklı bayt önce *(big-endien)* yerleşim biçimi olarak bilinmektedir (Çizim 9.13).

Küçük ağırlıklı bayt önce *(little-endien)* yerleşiminde, örneğin 16 bitlik bir sözcüğe ilişkin iki bayttan, küçük ağırlıklı olanı, ana bellekte küçük adresli ilk baytta, büyük ağırlıklı ikinci bayt ise, bunu izleyen büyük adresli ikinci baytta yer almaktadır. Bilgisayar sistemleri bu yerleşimlerden birini tercih edebilmektedir. Bu bağlamda, örneğin *Intel* işleyicileri, küçük ağırlıklı bayt önce *(little-endien)* yerleşim biçimini kullanmaktadır. *TCP/IP* yazılımı ise, kullandığı veri yapılarında, büyük ağırlıklı bayt

önce *(big-endien)* yerleşim mantığını taban almaktadır. Ağ uygulama programları, üzerinde çalışacakları bilgisayar sistemine ilişkin varsayımlar yapılarak geliştirilmediğinden, işletim aşamasında ortaya çıkabilecek olası bayt diziliş sırası uyuşmazlığını ortadan kaldırmak üzere, `htons()` *(host-to-network short)* komutundan yararlanılmaktadır. Bu komut, *TCP/IP* yazılımı ile farklı bayt diziliş ilkesini kullanan bilgisayar sistemlerinde diziliş dönüşümünü gerçekleştirmekte, *TCP/IP* yazılımı ile aynı bayt diziliş ilkesini kullanan bilgisayar sistemlerinde ise, doğal olarak, boş bir komut olarak işletilmektedir.

Örnek programlarda, `sock()` sistem çağrısının ikinci argümanı, *TCP/UDP* katmanlarından *TCP* katmanının kullanılacağını gösterir biçimde `SOCK_STREAM` olarak seçilmiş ve programların yapıları *connection oriented* moduna göre düzenlenmiştir. Bu programların, *connectionless* moduna göre düzenlenmiş biçimleri 9.2.8 numaralı kesimde açıklanacaktır.

Sunucu program içinde, bu programın üzerinde çalışacağı bilgisayar sisteminin *IP* Numarası günlenirken, `INADDR_ANY` parametresi kullanılmıştır. Bu parametre ile, dönem dönem bilgisayarların *IP* numaralarının değişebileceği düşünülerek, belirgin bir numarayı programa gömmek yerine, sözkonusu bilgisayarın, sistemde o an tanımlı Internet adresinin kullanılması sağlanmaktadır.

Örnek sunucu programdan birden çok istemci programın hizmet alması mümkündür. Örnek sunucu programın yukarıda verilen yapısı gereği, bir istemci program, sunucudan hizmet almaya başladıktan sonra, sunucu programa `connect()` komutu ile bağlanmaya çalışan diğer istemci programlar `listen()` komutunun tanımladığı kuyruğa bağlanarak sunucu programın `accept()` komutuna dönmesini (ya da sunucuya daha önce bağlanan istemci programın sonlanmasını) beklemek durumunda kalırlar. Başka bir deyişle, sözkonusu bu yapı, istemci programların koşut çalışmalarına ya da sunucudan eşanlı hizmet almalarına olanak vermez. Bu durum kimi uygulama türleri için önemli bir sakınca oluşturur. Bu gerçekleştirim biçimi, "satır yankılama" örneği için de sakıncalıdır. Zira bir istemci program, sunucu programa bağlanıp satır yankılama işlemleri yürüttüğü sürece, diğer istemci programlar, çalıştırdıkları `connect()` komutu üzerinde, uzunluğu belirsiz bir süre bekler durumda kalırlar. Bu önemli sakıncanın nasıl aşılacağı izleyen kesimde açıklanmıştır:

9.2.7. İstemci Programlar için Koşut Hizmet Yapısı

Yukarıda açıklanan önemli sakıncayı aşmanın yolu, sunucu nitelikli programı, her istemci program için ayrı bir görev olarak işletmekten geçer. Bunun nasıl yapıldığı, satır yankılama sunucu program üzerinde yapılan değişiklikle, Çizim 9.14'te örneklenmiştir. Çizim 9.14'te görüldüğü üzere, sunucu program düzeyinde, `accept()` komutundan hemen sonra, `fork()` komutu ile yeni bir sunucu görev yaratılmaktadır. `accept()` komutu ile bağlantı kurulan istemci programın satır yankılama işlemlerini yürütecek görev, oğul nitelikli bu görevdir. `fork()` komutunu işleten ata görev, miras *(inheritance)* yoluyla göstergesini `(fd)` oğul göreve aktardığı bağlantıyı (`close(fd)`

komutu ile) kapatarak hemen `accept()` komutuna geri dönmekte ve bu yolla, varsa diğer istemcilere de, yeni bir oğul görev aracılığıyla, koşut işlem bağlamında hizmet verebilmektedir.

```
/* Örnek koşut sunucu program */

#include <sys/types.h>
#include <sys/socket.h>
#include <netinet/in.h>
#include <stdio.h>
#include <netdb.h>
#include <signal.h>

#define MAX 80
#define SUNUCU_KAPISI 1490

main()

{
char yastik[MAX];
int fd, nb, sock;
unsigned long istemci_boyu;
struct sockaddr_in sunucu, istemci;

sunucu.sin_family = AF_INET;
sunucu.sin_addr.s_addr = INADDR_ANY;
sunucu.sin_port = htons(SUNUCU_KAPISI);

signal(SIGCHLD, SIG_IGN);

sock = socket(AF_INET, SOCK_STREAM, 0);
bind(sock, (struct sockaddr *)&sunucu, sizeof sunucu);
listen(sock, 1);
istemci_boyu = sizeof istemci;

while(1) {
  fd=accept(sock,(struct sockaddr *)&istemci, &istemci_boyu);
  if(fork() == 0) {
     /* Bu kesimi "oğul sunucu" görev çalıştırır */
     while((nb = read(fd, yastik, MAX)) > 0)
        write(fd, yastik, nb);
     exit(0);
  }else
     /* Bu kesimi "ata sunucu" görev çalıştırır */
     close(fd);
  }
}
```

Çizim 9.14. Koşut İşleme Olanak Veren Satır Yankılama Sunucu Programı

Bu yolla yaratılan her sunucu oğul görev, ayrı bir istemci programa hizmet vermekte, hizmet gereksinimi bittiğinde (istemci programa `CTRL D` damgası girilince) `exit(0)` komutu ile, sistemdeki varlığına son vermektedir.

`signal(SIGCHLD,SIG_IGN)` komutu, sunucu programa, `exit(0)` komutunu karşılamak ve bu yolla, sonlanan oğul görevin, sistemde hortlak *(zombie)* durumuna düşmesini engellemek üzere eklenmiştir.

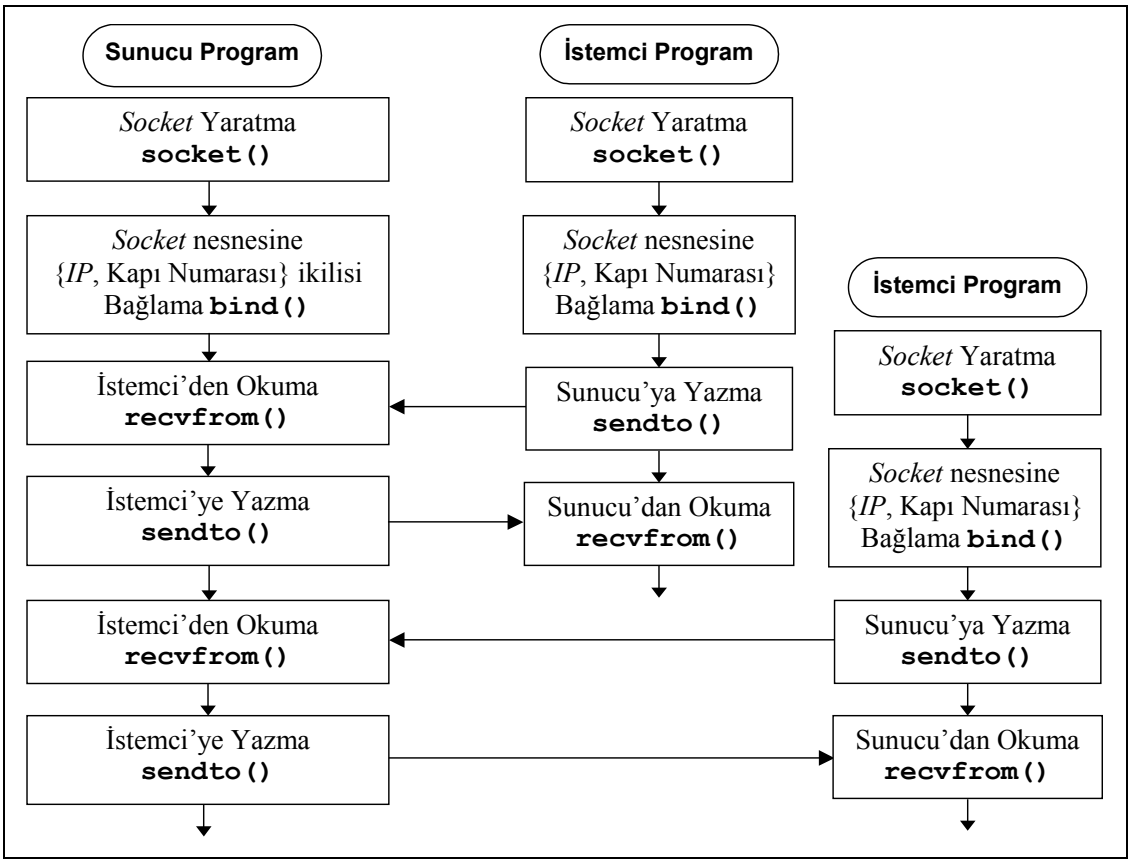
UNIX'te görevlerin sonlanmasına yarayan `exit()` komutu, ata göreve, bu durumu bildiren, *signal* adlı bir ileti göndermektedir. Ata görev, `signal()` komutu aracılığıyla bu iletiyi almadan oğul görevin, *PCB (Process Control Block)* adlı görev iskeleti silinmemekte ve bu nedenle sistemdeki varlığı tam olarak son bulamamaktadır. Eğer ata görev düzeyinde `signal()` komutuna yer verilmezse sonlanmak isteyen oğul görev(ler), *UNIX* tanımları gereği, "görev iskeletine sahip bitmiş görev" ya da hortlak *(zombie)* durumuna düşmektedir. Sistemde bu biçimde varlığı süren görevler, ancak ata görevleri sonlandığında sistemden tümüyle silinmektedir. Bu durum, işletim sistemince görev iskeletleri için öngörülen veri yapılarını gereksiz yere işgal etme gibi olumsuz bir sonuç doğurmaktadır. Örnek sunucu program hiç sonlanmayan bir program olarak öngörüldüğünden, `signal(SIGCHLD,SIG_IGN)` komutunu içermemesi durumunda bu olumsuzluk burada daha da belirginlik kazanmaktadır.

UNIX'te her görevin mutlaka bir ata görevi bulunur. Sistem açılırken işletime alınacak tüm görevler (bu arada *shell* görevini yaratan `getty`) *init* adlı bir görev aracılığıyla (`fork()` komutu kullanılarak) yaratılırlar. Bu nedenle, işletimin herhangi bir anında, tüm görevlerin "ilk" atası *init* adlı bu görevdir. Eğer bir görev, yarattığı oğul görev(ler) sonlanmadan biterse, başka bir deyişle, oğul görev(ler), yetim *(orphan)* durumuna düşerse, *UNIX* kuralları gereği, *init* adlı görevi ata görev olarak alırlar. Bu durumda, işlettikleri `exit()` komutları bu ilk ataya yönlendirilerek hortlak durumuna düşmeleri, sistem tarafından engellenmiş olur.

Sunucu programın, her istemci program için ayrı bir görev olarak işletiliyor olması, `listen()` komutu ile belirlenen istemci kuyruk boyunu anlamsız kılmaktadır. Bu nedenle, yeni sunucu programda sözkonusu kuyruk boyu için bir ile yetinilmiştir.

9.2.8. İstemci/Sunucu Programlarda *Connectionless* Modun Kullanımı

İstemci ve sunucu programlarca gönderilen ve alınan iletilerin sıralarının bozulması bir sakınca oluşturmuyorsa ya da ileti sıra denetim işlevini, uygulama kendisi zaten üstleniyorsa, ulaşım katmanı düzeyinde *TCP* kesimi yerine, sıra denetimi yapmayan, daha hızlı *UDP* kesiminin kullanılabileceği daha önce açıklanmıştı. *TCP* yerine *UDP* katmanının kullanılması durumunda, sunucu ve istemci programların yapıları Çizim 9.15'de gösterildiği biçimde olmaktadır.



Çizim 9.15. *Connectionless* Modda, İstemci ve Sunucu Program Yapıları

Bu yeni yapıda, sunucu program, istemciler tarafından bilinen bir kapı numarasını bağlayacağı bir *socket* yaratıp bu kapıya gönderilen ve hatasız alınan datagramlara ilişkin yanıt datagramlar göndermekle yetinmektedir. Başka bir deyişle, sunucu program düzeyinde, oturum tabanlı sıra denetimi için gerekli `accept()` sistem çağrısına yer verilmemektedir. Bu son çağrının sunucu programda yer almayışı, datagram alış-verişinde, `accept()` sistem çağrısınca geri döndürülen bağlantı göstergesine (`fd`) gereksinim duyan `read()` ve `write()` komutlarının kullanımını olanaksız kılmaktadır. *UNIX*'te, *connectionless* modda, bunların yerine kullanılacak:

```
sendto  (sock, buf, buf_length, flags, address, address_length);
recvfrom(sock, buf, buf_length, flags, address, &address_length);
```

gibi, iki sistem çağrısına yer verilmektedir. Burada:

`sock`	`socket()` çağrısının döndürdüğü gösterge *(rendezvous descriptor),*
`buf`	Aktarıma konu olan veri yastığı göstergesi,
`buf_length`	Aktarıma konu olan veri yastığı boyu,
`flags`	Özel seçenek göstergesi, genelde 0,
`address`	`sendto()` bağlamında verinin gönderileceği hedef uygulama, `recvfrom()` bağlamında verinin alınacağı kaynak uygulama,
`address_length`	Uygulama adresinin boyudur.

`sendto()`, `recvfrom()` sistem çağrıları, istemciye özgü bağlantı göstergesi taşımadığından, kaynak ya da hedef uygulamanın kimliğini (*IP* ve Kapı numaralarını) taşımaktadır. Bu yolla, sunucu uygulama, değişik istemcilerden gelen datagramları ayrıştırma olanağı bulabilmektedir. Bu bilgiler ışığında, örnek satır yankılama sunucu ve istemci programların, *connectionless* moddaki görünümleri, Çizim 9.16 ve Çizim 9.17'de verilmiştir.

```
/* Connectionless modda çalışan örnek sunucu program */

#include <sys/types.h>
#include <sys/socket.h>
#include <netinet/in.h>
#include <stdio.h>
#include <netdb.h>

#define MAX 80
#define SUNUCU_KAPISI 1490

main()

{

char yastik[MAX];
int nb, sock;
unsigned long istemci_boyu;
struct sockaddr_in sunucu, istemci;

sunucu.sin_family = AF_INET;
sunucu.sin_addr.s_addr = INADDR_ANY;
sunucu.sin_port = htons(SUNUCU_KAPISI);

sock = socket(AF_INET, SOCK_DGRAM, 0);

bind(sock, (struct sockaddr *)&sunucu, sizeof sunucu);

istemci_boyu = sizeof istemci;

while(1) {
  while((nb = recvfrom(sock, yastik, MAX, 0,
              (struct sockadd *)&istemci, &istemci_boyu))>0)

     sendto(sock, yastik, nb, 0, (struct sockaddr *)&istemci,
     sizeof istemci);
}

}
```

Çizim 9.16. *Connectionless* Modda Satır Yankılama Yapan Sunucu Program

```
/* Connectionless modda çalışan örnek istemci program */

#include <sys/types.h>
#include <sys/socket.h>
#include <netinet/in.h>
#include <stdio.h>
#include <netdb.h>

#define MAX 80
#define SUNUCU_KAPISI 1490

main(int argc, char *argv[])
{
int nb, sock;
unsigned long sunucu_boyu;
char yastik[MAX];
char sunucu_adi[MAX];
struct sockaddr_in sunucu, istemci;
struct hostent *s_bilgi;

istemci.sin_family = AF_INET;
istemci.sin_addr.s_addr = INADDR_ANY;
istemci.sin_port = 0;   /*Kapı Numarasını sistem belirlesin*/

sock = socket(AF_INET, SOCK_DGRAM, 0);
bind(sock, (struct sockaddr *)&istemci, sizeof istemci);

strcpy(sunucu_adi, argv[1]);
s_bilgi = gethostbyname(sunucu_adi);
sunucu.sin_family = s_bilgi->h_addrtype;
memcpy(&sunucu.sin_addr, s_bilgi->h_addr, s_bilgi->h_length);
sunucu.sin_port = htons(SUNUCU_KAPISI);

sunucu_boyu = sizeof sunucu;

while((nb = read(0, yastik, MAX)) > 0){
   sendto(sock,yastik,nb,0,(struct sockaddr *)&sunucu,
          sizeof sunucu);
   nb = recvfrom(sock,yastik,MAX,0,(struct sockaddr *)&sunucu,
                 &sunucu_boyu);
   write(1, yastik, nb);
}
close(sock);
}
```

Çizim 9.17. *Connectionless* Modda Satır Yankılama İsteyen İstemci Program

Connectionless modda satır yankılama amaçlı sunucu ve istemci programların işletimi, sunucu programın `defne`, istemci programın da `yunus` ve `liza` adlı iki farklı bilgisayar üzerinde, eşanlı çalıştırıldığı varsayılarak, aşağıda örneklenmiştir:

```
defne> sunucu&
PID 24679
```

```
yunus> istemci defne
aaaaaaaaaaaaa
aaaaaaaaaaaaa
bbbbbbbbbbbbbbbbbbbbb
bbbbbbbbbbbbbbbbbbbbb
CTRL D
yunus>

liza> istemci defne
ccccccccccccccccccccccccc
ccccccccccccccccccccccccc
eeeeeeeeeeeeeeeeeeeee
eeeeeeeeeeeeeeeeeeeee
CTRL D
liza>
```

Bu işletim örneğinden de anlaşılacağı üzere, sunucu programın *connectionless* moddaki yapısı, birden çok istemciye koşut hizmet verme olanağını, *connection oriented* modda yapılanın tersine, herhangi bir ek düzenlemeye gerek kalmaksızın, içinde taşıyabilmektedir.

İstemci-sunucu yaklaşımını gerçekleştirmede kullanılan *socket* düzeneği, bu amaçla var olan tek gerçekleştirim seçeneği değildir. Bunun yanı sıra, Ulaşım Katmanı Arabirimi *(TLI Transport Layer Interface)* ve Uzaktan Yordam Çağırma *(RPC Remote Procedure Call)* olarak adlandırılan iki başka seçenek daha mevcuttur. Uzaktan Yordam Çağırma, *socket* düzeneğine göre daha üst düzey bir gerçekleştirim aracıdır. Bu araçtan, Birlikte Çalışan Görevler adlı Dördüncü Bölümde kısaca söz edilmiştir. *RPC*, izleyen kesimde açıklanan *NFS* yazılımınca da kullanılan bir araçtır.

9.3. Dağıtılmış Kütük Yönetim Sistemi *(NFS)*

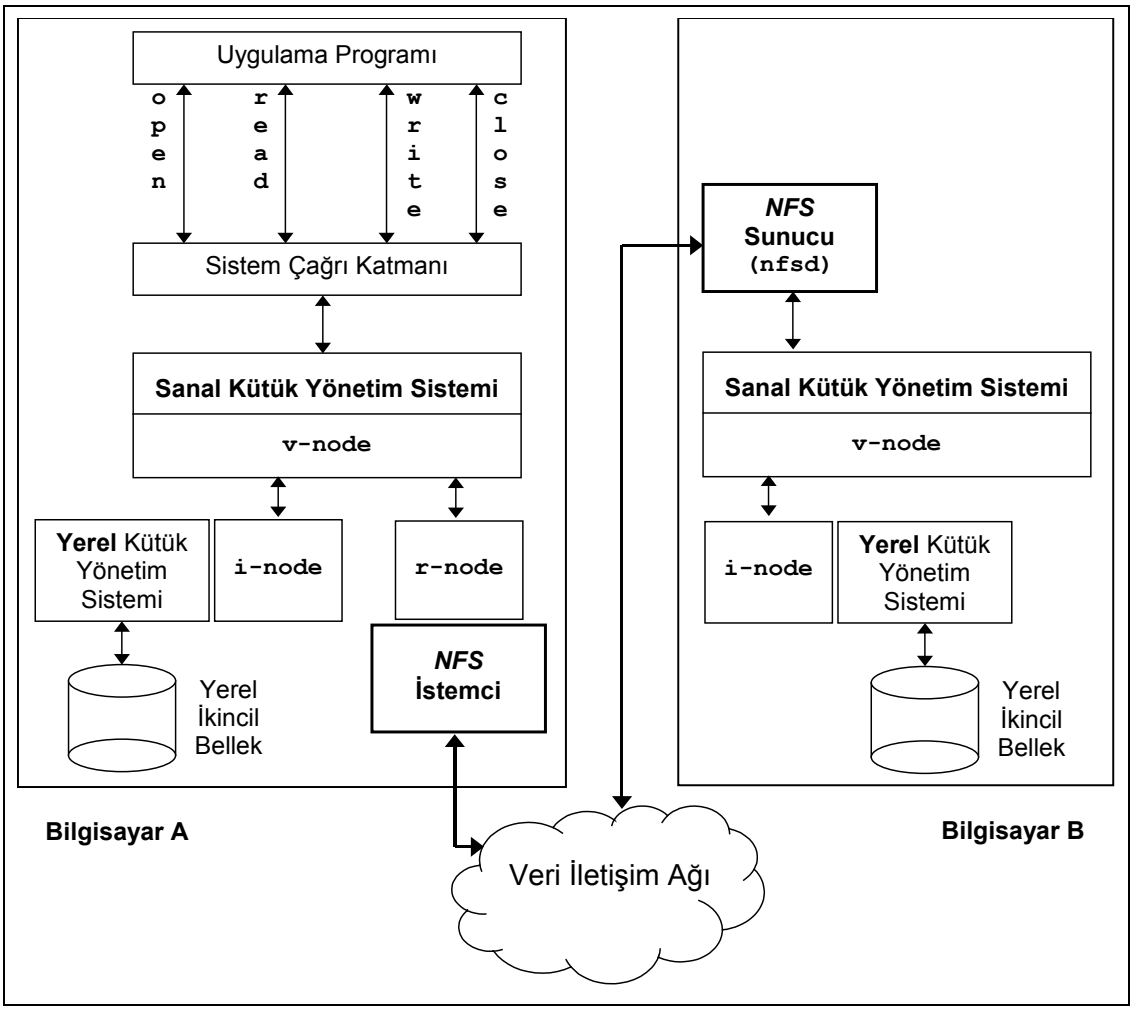
Dokuzuncu Konu başlığı altında, şimdiye değin, kullanıcıların, kendi bilgisayarlarıyla aynı ağ içinde yer alan başka bilgisayarların "işlem" kapasitelerinden, dolaylı olarak nasıl yararlandıkları, *socket* düzeneğine dayalı olarak açıklanıp örneklenmiştir. İzleyen kesimde, bu kez, kullanıcıların, başka bilgisayar sistemleri üzerinde yer alan "kütük"lere nasıl erişebildikleri ve bu yolla, ağ içinde verileri nasıl paylaştıkları, *UNIX* bağlamında açıklanacaktır:

9.3.1. Ağ Düzeyi Kütük Paylaşımı

UNIX işletim sistemi, ağ düzeyi kütük paylaşım hizmetini, yerel kütük yönetim sisteminin *(UFS Unix File System)* genişletilmiş bir biçimi olan ve *NFS Network File System*[36] olarak adlandırılan dağıtılmış bir kütük yönetim sistemine dayalı olarak vermektedir. Bu sistemde, uygulama programlarının kütük işlemlerine dönük sistem çağrıları, sanal kütük yönetim sistemi *(VFS Virtual File System)* olarak adlandırılan bir

[36] *NFS Network File System, Sun Microsystems Firması tarafından, iş istasyonları arası, ağ düzeyi kütük ve yazıcı paylaşımı amacıyla geliştirilmiştir.*

kesime yönlendirilmektedir. Bu kesim, üzerinde işlem yapılmak istenen (açılan) tüm kütüklere ilişkin, (`i-node` benzeri) `v-node` *(virtual i-node)* adlı bir veri yapısı tutmaktadır. Bu veri yapısı, ilgili kütüğün, yerel ya da uzak, ne tür bir kütük olduğunu belirlemeye yaramaktadır. Eğer sözkonusu kütük, uzak nitelikli (başka bir bilgisayardaki) bir kütük ise, `v-node` adlı veri yapısı, bu kütüğe erişim için gerekli tüm bilgileri de tutmaktadır. Bu sayede, bir kütüğe ilişkin `read()`, `write()` gibi işlemler, ya yerel ya da uzak kütük yönetim sistemine yönlendirilebilmektedir.

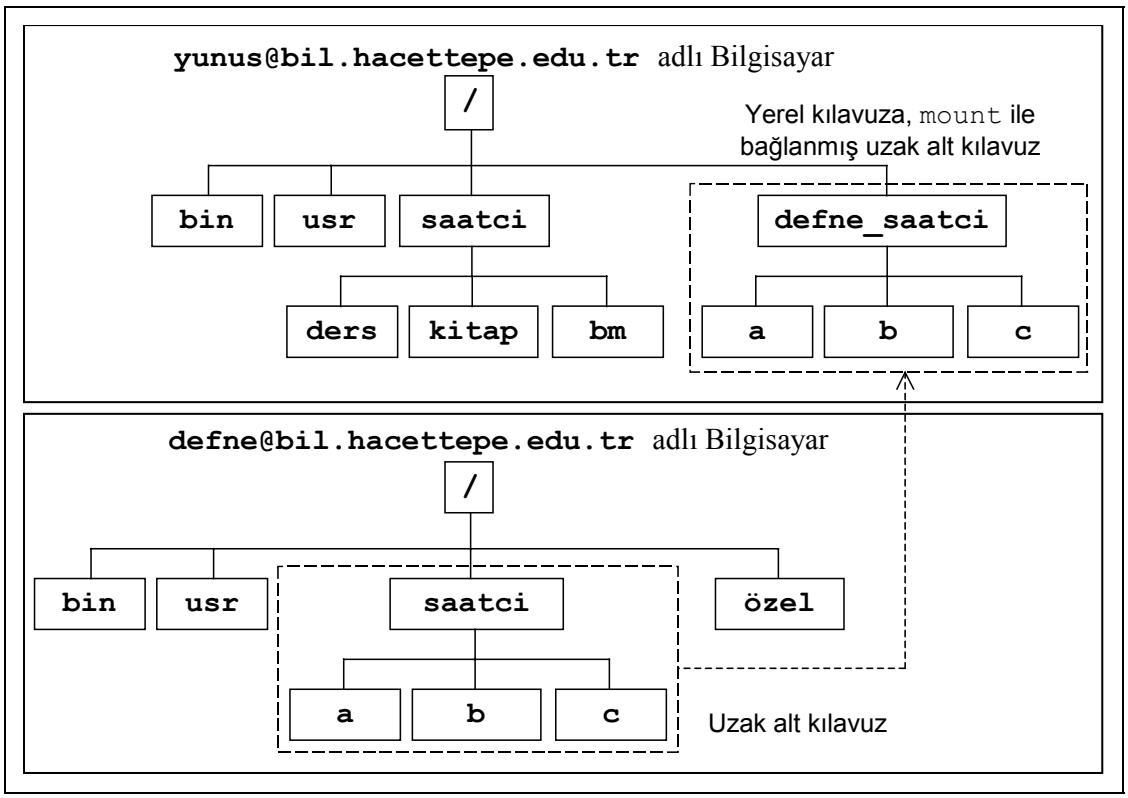


Çizim 9.18. *NFS* Bağlamında Uzak Kütük İşlemlerinin Ele Alınışı

Uzak kütük yönetim sistemine ilişkin işlemler, istemci-sunucu yaklaşımına dayalı olarak gerçekleştirilmektedir. Sanal kütük yönetim sistemi, uzak kütük işlemleri için, "*NFS* İstemci" adlı bir görevden yararlanmaktadır. Bu görev, işleme konu olan kütüğün yer aldığı bilgisayar sistemindeki "*NFS* Sunucu (`nfsd`)" adlı bir görevle işbirliği içinde, uzak kütük işlemlerinin gereğini yerine getirmektedir (Çizim 9.18).

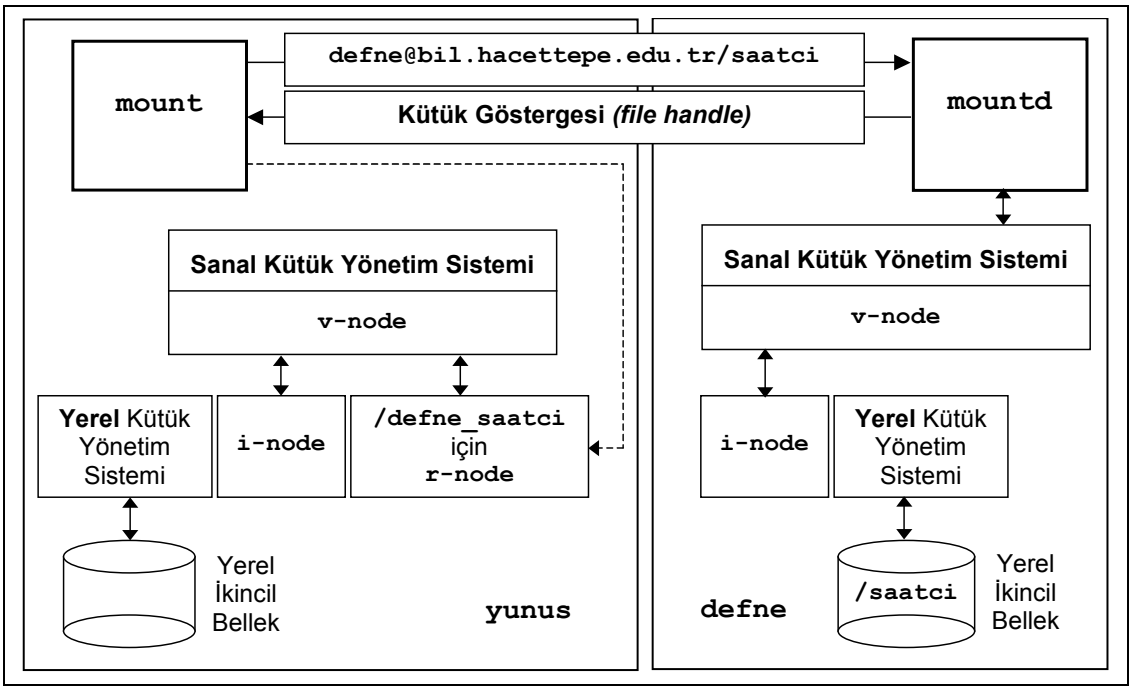
NFS bağlamında, kütük işlemleri, uzak ve yakın kütük ayrımını konu etmektedir. İşletim sisteminin kolay kullanım hizmet ilkesi gereğince, bu ayrımın uygulama

programlarına taşınması uygun değildir. Bu nedenle *NFS*, kullanıcılara, ağ içi değişik bilgisayar sistemleri üzerinde sakladıkları kütüklerini, kendi yerel bilgisayarında saklanan kütükler gibi gösterme olanağını sunmaktadır. Bunun yapılabilmesi, `mount` adlı bir istemci program aracılığıyla mümkün olmaktadır. Kullanıcılar, bu programı kullanarak, uzak bir bilgisayar üzerinde tanımlı herhangi bir alt kılavuzu yerel bir alt kılavuz altına bağlayabilmekte ve bu yapıldıktan sonra, o alt kılavuz altında yer alan (uzak) kütüklere yerel (yakın) kütükler gibi erişebilmektedirler (Çizim 9.19).



Çizim 9.19. Uzak Alt Kılavuzun Yerel Kılavuz İçinde Sanal Konumu

`mount` programı, uzak ve bununla ilgili yakın alt kılavuz kimliklerini argüman olarak almaktadır. Uzak alt kılavuz kimliği, (`defne@bil.hacettepe.edu.tr/saatci` gibi) {ilgili bilgisayar adı; alt kılavuz adı} ikilisinden oluşmaktadır. `mount`, uzak alt kılavuz adından ilgili bilgisayar kimliğini (`defne@bil.hacettepe.edu.tr`) elde etmekte ve bu bilgisayar üzerinde çalışan `mountd` *(mount deamon)* adlı sunucu programdan, ilgili uzak alt kılavuz (`/saatci`) için bir kütük göstergesi *(file handle)* sağlayıp bunu, adı (örneğin `/defne_saatci` olarak) verilen yakın bir alt kılavuzla ilişkilendirmektedir. Bu bağlamda, sanal yakın alt kılavuza ilişkin `v-node` yapısı, elde edilen kütük göstergesini de taşıyan ve `open()`, `read()`, `write()` gibi izleyen kütük işlemlerinde başvurulacak bir `r-node` yapısı ile ilişkilendirilmektedir. Bu aşamadan sonra, yerel kılavuza, özel bir adla bağlanan uzak alt kılavuz altındaki kütüklere erişim, kullanıcı için, *NFS* sayesinde, diğer yerel kütüklere erişimden farksız olmaktadır (Çizim 9.20).

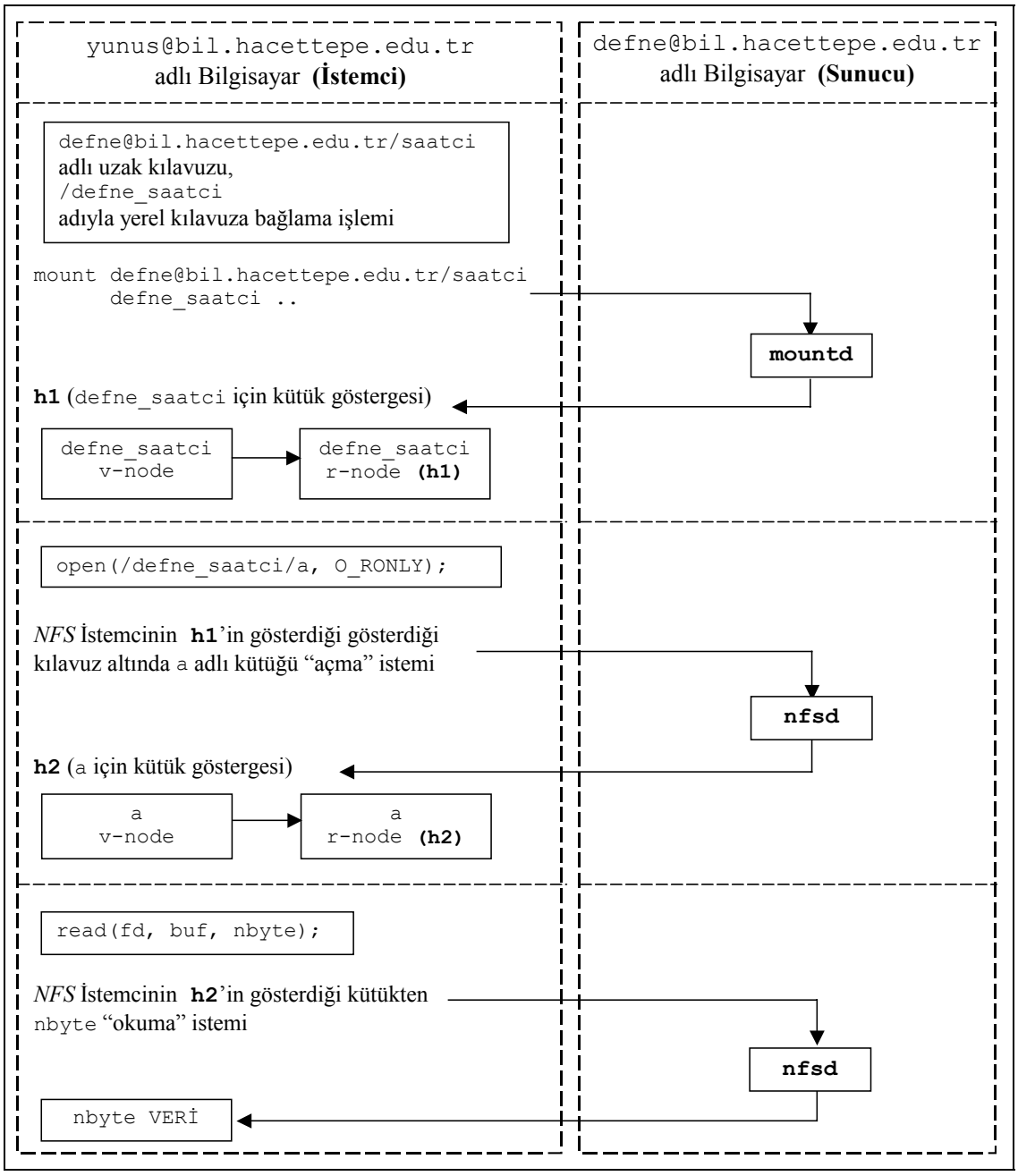


Çizim 9.20. Uzak bir Alt Kılavuzun Yerel Kılavuza `mount` programı ile Bağlanması

`mount` programı kullanılarak yerel kılavuza bağlanan uzak kütüklere ilişkin `open()` ve `read()` sistem çağrılarının işletim adımları, aşağıda açıklandığı biçimde olmaktadır:

Bir uygulama programı tarafından işletilen `open(/defne_saatci/a,O_RDWR)` sistem çağrısı, sanal kütük yönetim sistemine yönlendirilmekte ve ilgili kütüğe ilişkin (`a`) bir `v-node` yaratılmaktadır. Sözkonusu kütüğün, uzak nitelikli bir kılavuz altında yer aldığı anlaşıldığında, başka bir deyişle, sözkonusu kılavuzun `v-node` yapısı, bir `r-node` yapısını gösterdiğinden, sanal kütük yönetim sistemi, *NFS* İstemcisine başvurmaktadır. *NFS* İstemcisi, uzak bilgisayar *NFS* Sunucusuna (`nfsd`), "kütük açma" istemi yönlendirmekte ve sunucudan, bu kez, açılan uzak kütükle (`a` ile) ilgili bir kütük göstergesi (*file handle)* elde etmektedir. *NFS* İstemcisi, elde ettiği bu göstergeyi de taşıyan bir `r-node` yaratarak bunun, açılmak istenen kütük için sanal kütük yönetim sistemince yaratılan `v-node` ile ilişkilendirilmesini sağlamaktadır. Sözkonusu uzak kütükle (`a` ile) ilgili, `read()`, `write()`, `close()` gibi sonraki işlemler, sanal kütük yönetim sistemince, bu `v-node` kullanılarak yürütülmektedir. Başka bir deyişle, uygulama programına döndürülen kütük göstergesi (*file descriptor* `fd`), bu yapının göstergesi olmaktadır.

Açma işlemi sonrasında, örneğin, `read(fd,buf,nbyte)` komutu işletilmek istenirse, bu komutun argüman olarak taşıdığı `v-node` göstergesinden (`fd`) `r-node` göstergesi elde edilmekte ve okuma işlemi, yine *NFS* İstemcisine yönlendirilmektedir. *NFS* İstemcisi, `r-node` içinde saklanan ve daha önce uzak bilgisayar sisteminden elde ettiği kütük göstergesini kullanarak, bu işlemi *NFS* Sunucusundan talep edip gerçekleştirmektedir (Çizim 9.21).

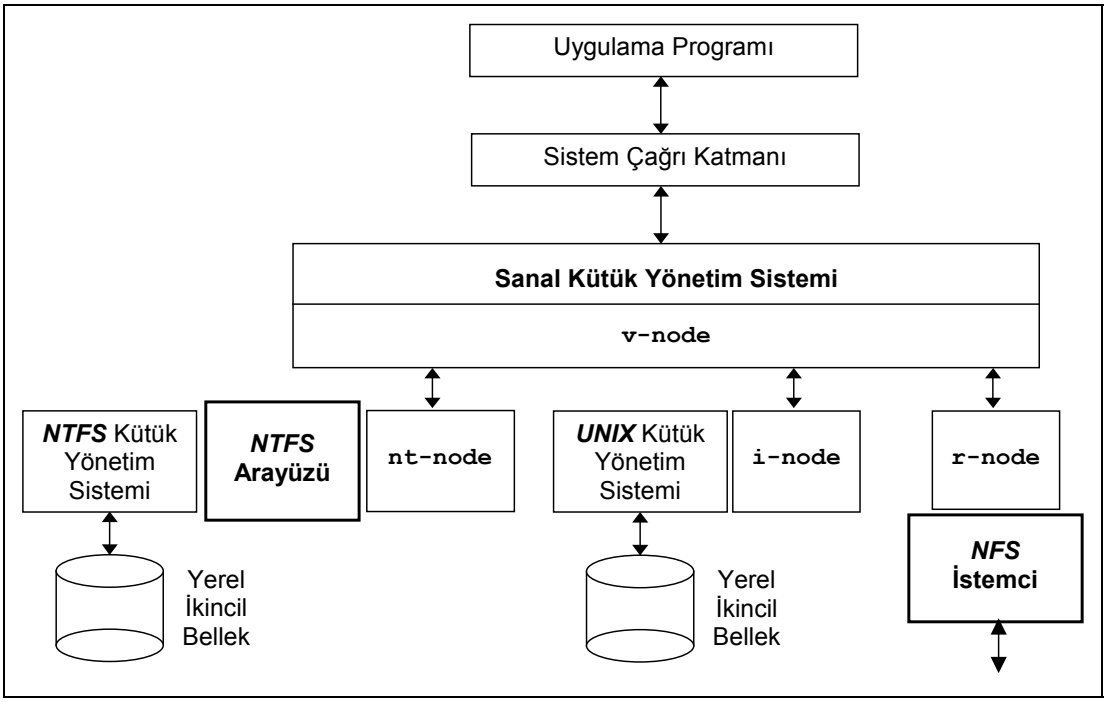


Çizim 9.21. Uzak Kütüklere ilişkin `open()` ve `read()` Sistem Çağrılarının Ele Alınışı

9.3.2. Farklı Kütük Yönetim Sistemlerinin Bütünleşmesi

NFS, uygulama programlarına, yerel kütük yönetim sistemiyle aynı nitelikte, uzak kütük yönetim sistemlerinden hizmet alabilmenin yanı sıra, gerek yerel gerekse uzak, farklı kütük yönetim sistemlerini de kullanabilme olanağı yaratmaktadır. *NFS* kütük yönetim sisteminin yapısı açıklanırken, bunun *VFS Virtual File System* adlı bir alt kesim içerdiği, bu kesimin açılan her kütük için bir `v-node` yarattığı, bu yapının, açılan kütüğün yerel

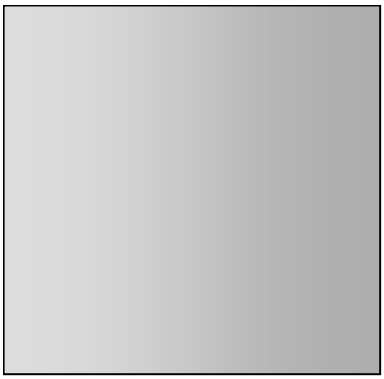
ya da uzak olmasına göre ya bir `i-node` ya da bir `r-node` ile ilişkilendirildiği, sözkonusu kütük yerel nitelikli bir kütük ise yerel kütük yönetim sistemine, uzak nitelikli bir kütük ise de *NFS* İstemcisi adlı bir arayüze başvurulduğu belirtilmişti. *NFS* yapısında böyle bir arayüzün öngörülmüş olması, *VFS Virtual File System* adlı alt kesim üzerinden, *UNIX* işletim sisteminkinden farklı, başka kütük yönetim sistemlerine erişimi de mümkün kılmakta ve programlara, *UNIX* ortamında, farklı kütük yönetim sistemleri altında yer alan kütükleri kullanabilme olanağını yaratmaktadır.



Çizim 9.22. *NFS-VFS* ile Farklı Kütük Yönetim Sistemlerine Erişim

Bunun yapılabilmesi için, her farklı kütük yönetim sistemi için, *NFS* istemci benzeri bir arayüzün sistemde yer alması gerekmektedir. Uygulama programlarının bu sistemler içinde tanımlı kütükleri kullanabilmeleri için, öncelikle, uzak kütüklerde olduğu gibi, ilgili alt kılavuzun, `mount` programı kullanılarak *UNIX* kılavuz sistemine bağlanması ve bu yolla, sanal kütük yönetim sistemi düzeyinde bununla ilgili bir `v-node` yapısının yaratılması gerekmektedir. Bu sayede, sözkonusu alt kılavuz altında yer alan bir kütüğe erişimde, bunun, farklı bir sisteme yönelik kütük olduğunun algılanması ve işlem istemlerinin, bu sisteme ilişkin, özel bir arayüze (örneğin *Windows NTFS* Arayüzüne) yönlendirilmesi mümkün olabilmektedir (Çizim 9.22).

Dağıtılmış kütük yönetim sisteminin yapısına dönük açıklanan bu yaklaşım, *Windows* işletim sistemince de benimsenen bir yaklaşımdır. Bu sayede, *Windows* ortamında, hem "ağ komşuları" olarak tanımlanan bilgisayarlar düzeyinde saklanan kütüklere, hem de aynı bilgisayar üzerinde yer alan, örneğin *FAT32* ya da *NTFS (NT File System)* adlı farklı kütük yönetim sistemlerine ilişkin kütüklere erişim mümkün olabilmektedir.



İ Ş L E T İ M S İ S T E M L E R İ

ABEL, P: *IBM-PC Assembly Language and Programming*, Englewood Cliffs, N.J.: Prentice Hall International, 1991.

BASHE, C.J., JOHNSON, L.R., PALMER, J.H., PUGH, E.W: *IBM's Early Computers*, MIT Press, Cambridge MA., 1986.

BEN-ARI, M: *Principles of Concurrent Programming*, Englewood Cliffs, N.J.: Prentice Hall International, 1982.

BROWN, C: *UNIX Distributed Programming*, Prentice Hall International, 1994.

COMER, D.E: *Internetworking with TCP/IP: Volume I: Principles, Protocols and Architecture*, Prentice Hall International, 1995.

CORNES, P: *The LINUX A-Z*, Prentice Hall International, 1997.

EGAN, J.I., TEIXEIRA, T.J: *Writing a UNIX Device Driver*: 2/e, John Wiley and Sons Inc., 1992.

HALSALL, F: *Data Communications, Computer Networks and Open Systems*: 4/e, Addison-Wesley, 1996.

INTEL Corp: *Intel 486 Microprocessor Programmer's Reference Manual*, Intel: Osborn: McGraw-Hill, Newyork, 1990.

PHAM, T.Q., GARG, P.K: *Multithreaded Programming with Windows NT*, Prentice Hall: PTR, 1995.

ROCHKIND, M.J: *Advanced UNIX Programming*, Englewood Cliffs, N.J.: Prentice Hall International, 1985.

STALLINGS, W: *Operating Systems: Internals and Design Principles*: 4/e, Prentice Hall International, 2001.

STEVENS, R: *TCP/IP Illustrated: Volume I: The Protocols*, Addison-Wesley, 1993.

STRAUSS, E: *80386 Technical Reference*, Brady Books, Newyork, 1987.

SWITZER, R: *Operating Systems: A Practical Approach,* Prentice Hall International, 1993.

TANENBAUM, A.S., WOODHULL, A: *Operating Systems: Design and Implementation: 2/e*, Prentice Hall International, 1997.

TANENBAUM, A: *Modern Operating Systems: 2/e*, Prentice Hall International, 2001.

TANENBAUM, A.S., STEEN, M: *Distributed Systems: Principles and Paradigms*, Prentice Hall International, 2002.

WAKERLY, J.F: *Microcomputer Architecture and Programming: The 68000 Family,* John Wiley and Sons Inc., 1989.

SÖZLÜK

İ Ş L E T İ M S İ S T E M L E R İ

A

adres *(address)*: Komut kodları ile verilerin bir program içinde ya da ana bellekte bulunduğu konumun sırası ya da kimliği. Komut kodları ile verilerin, yer aldıkları programın başına göreli konumları mantıksal adresleri *(logical addresses)*; aynı kod ve verilerin ana bellekte yerleştikleri sözcüklerin bellek başına göreli konumları ise fiziksel adresleri *(physical addresses)* oluşturur. Örn. Ana bellekte sözcük adresi, diskte disk öbek adresi, komut adresi.

adres yolu *(address bus)*: İşleyicinin ana bellek sözcükleri ve giriş/çıkış arabirimi yazmaçlarını adreslemede kullandığı im iletim hatları. Örn. 80386 işleyicisinin adres yolu 32 hattan oluşur.

adres dönüştürme *(address translation)* Bellek yönetimi kapsamında, mantıksal adreslerden fiziksel adreslere geçiş süreci.

adresleme *(addressing)*: İşleyicinin, içeriğine erişmek amacıyla, bir ana bellek sözcüğü ya da bir giriş/çıkış arabirimi yazmacını seçme süreci.

ağ *(network)*: Birden çok düğüm ve bunlar arasında kurulan bağlantıların oluşturduğu yapı. Örn. Bilgisayar ağı, telefon ağı.

ağ düzeyi birlikte işlem *(distributed concurrent processing)*: Görev göçü yoluyla, görevlerin, sunuldukları bilgisayardan farklı bilgisayarlara taşınarak işletilmesi.

ağaç *(tree)*: Ağaç benzeri dallanmaları mantıksal olarak temsil eden yapı ya da veri yapısı. Örn. Ağaç yapılı kılavuz kütük yapısı.

ağırlık *(significance)*: Bir sayının değeri hesaplanırken bu sayıyı oluşturan rakamların çarpıldığı basamak taban değeri. Örn. En büyük ve en küçük ağırlıklı basamaklar *(the most and the least significant digits)*.

akış çizeneği *(flowchart)*: Algoritma ya da çözüm yolunun çizimsel gösterimi. Akış çizgesi olarak da adlandırılır.

alan adı sistemi *(DNS Domain Name System)*: Internet içinde, bir bilgisayarın simgesel adından *IP* numarasına geçiş sistemi.

alındı *(acknowledgement)*: Bir isteme ya da gönderiye ilişkin olumlu yanıt. Örn. Kesilme istemi alındısı, ileti alındısı.

amaç program *(object program)*: Derleme sonrası elde edilmiş, program başına göreli adres değerleri taşıyan ikili kod dizisi.

ana işlem birimi *(central processing unit)*: Bir bilgisayar sisteminde efendi konumundaki işleyici. Örn. Ana işlem birimi ve yardımcı işleyiciler.

anahtarlama *(switching)*: Ana işlem biriminin bir göreve atanması. Ana işlem birimi yazmaçlarının görev iskeletinde saklanan içeriklerle günlenmesi. Bir görevin ana işlem biriminini kullanmaya başlaması.

anlayış *(intelligence)*: Dış denetim ve izlemeye gerek bırakmaksızın davranabilme yeteneği. Örn. Anlayışlı arabirim.

arabirim *(interface)*: Giriş / çıkış sürücüleri ile ana bellek arası veri aktarımlarınında kullanılan programlanır elektronik çevrim. Örn. Disk arabirimi, terminal arabirimi.

ardıl iletişim *(serial communication)*: Gönderici ile alıcı arasında aktarılan damga kodunu oluşturan bitlerin, aynı iletişim hattı kullanılarak zaman içinde, ardarda aktarıldığı iletişim biçimi. Seri iletişim de denmektedir.

argüman *(argument)*: Bir yordamın diğer bir yordama aktardığı değer ya da adres.

aritmetik mantık birimi *(arithmetic logic unit)*: Ana işlem birimi içinde ikili toplama, çarpma, bölme gibi aritmetiksel işlemler ile tümleme, *ve, ya da* gibi mantıksal işlemleri yerine getiren birim. İki giriş bir de çıkışı yastığı bulunduğundan, çizimle gösterimlerde V harfi gibi çizilir.

aşma *(overrun)*: Gönderilen bir damga kodu daha okunmadan ikincisinin gönderilmesi ve birinci kodun ezilmesi durumu.

ata görev *(parent process)*: Yaratılanlar için yaratan görev.

atama *(allocation)*: Bir kaynağın kullanım hakkını bir göreve verme. Örn. Bellek kesimi atama, Ana işlem biriminin bir göreve atanması.

aynalama tekniği *(mirroring)*: Yedekleme amacıyla verilerin, sistemli biçimde iki ya da daha çok sürücüye koşut yazılması.

aygıt *(device)*: Bir bilgisayar sisteminde, arabirimi aracılığıyla, ana işlem birimi ana bellek ikilisine bağlanan giriş/çıkış sürücüsü.

aygıt sürücü *(device driver)*: Çekirdek katman düzeyi, ilgili arabirim ana bellek arası veri aktarımını denetleyen sürücü program.

ayrıcalık *(privilege)*: Verilere erişimde, işletim sistemi işlevlerinden (hizmetlerinden) yararlanmada sahip olunan özel hak. Örn. Ayrıcalıklı kullanıcı, Ayrıcalıklı görev.

B

bağ *(link)*: izleyen öge göstergesi. Örn. Bağ alanı, bağ sayacı.

bağlam *(context)*: Bir görevin işletim bütünlüğünü sağlayan çalışma ortamı.

bağlam anahtarlama *(context switching)*: Ana işlem birimi yazmaç takımının, işletimine geçeceği görevin iskeletindeki bilgilerle günlenmesi.

bağlayıcı *(linker)*: Ayrı ayrı derlenmiş yordamları, bütün bir programa dönüştürerek yükleme aşamasına hazırlayan sistem programı.

basamak *(digit)*: Bir sayıyı oluşturan rakamların konumları. Örn. 4 basamaklı sayı, yüzler basamağı.

başarım *(performance)*: İstenen yönde elde edilen sonuç. Örn. Sistem başarımı. Algoritma başarımı.

birleşen *(component)*: Bütünü oluşturan ögelerin herbiri. Örn. Bilgisayar birleşenleri. İşletim sistemi birleşenleri.

birlikte çalışan görevler *(concurrent processes)*: İşletimleri birlikte sürdürülen görevler.

bitişken *(contiguous)*: Arada boşluk kalmaksızın birbirine değen. Örn. Bitişken bellek alanları, bitişken disk öbekleri.

bitiştirme *(compaction)*: Ana bellekte ya da disk sürücüler üzerinde bir program ya da kütüğe atanmış alanların, arada boşluk kalmayacak biçimde yanyana getirilmesi.

boy *(size, length)*: bit, bayt ya da sözcük türünden uzunluk. Örn. Bellek boyu, sözcük boyu.

bölüm *(partition)*: Programlara atanan bellek parçası.

bölümlü bellek yönetimi *(partitioned memory management)* Ana belleği programlara, parçalar halinde atayan bellek yönetimi. Örn. Değişmez bölümlü, değişken bölümlü bellek yönetimi *(Static, dynamic partitioned memory management)*.

bölüşülür kaynak *(shared resource)*: Birden çok göreve eşanlı olarak atanabilen kaynak. Örn. Yazıcılar bölüşülür kaynaklar değildir.

bütünlük *(integrity)*: Bir bütün olarak korumayla ilgili. Örn. Veri bütünlüğü.

büyük ağırlıklı bayt önce yerleşimi *(big endian)*: Bir sözcüğü oluşturan baytların, ana bellekte, büyük ağırlıklı bayttan başlayarak yerleşmesi.

büyük boy bilgisayar *(mainframe computer)*: Yüksek işlem hızına, çok büyük ana bellek ve disk sığasına sahip, genişleyebilir çok kullanıcılı bilgisayar.

Ç

çekirdek katman *(kernel)*: İşletim sisteminin donanımın ayrıntısına en yakın kesimi.

çerçeve *(frame)*: Veri bağlantı katmanının kullandığı veri birimi.

çevrim-içi *(on line)*: Giriş verilerinin bilgisayar sistemine dolaysız aktarıldığı bilişim uygulaması. Örn. Çevrim-içi bankacılık işlemleri.

çıkış *(output)*: Verilerin arabirimler üstünden (dış ortama) kullanıcı ortamına ya da ikincil belleklere aktarılması.

çift yönlü *(duplex)*: iki yönlü iletişime izin veren. Örn. Yarı çift yönlü, tam çift

yönlü iletişim *(half duplex, full duplex communication)*.

çizelge *(table)*: Belirli sayıda satır ve sütundan oluşan yapı; tablo.

çizenek *(diagram)*: Çizimsel bir gösterim biçimi. Örn. Zaman çizeneği *(timing diagram)*, durum çizeneği *(state diagram)*.

çizge *(graph)*: Çizimsel bir gösterim biçimi. Örn. Döngüsüz yönlü çizge *(acyclic directed graph)*

çok görevli işlem *(multitasking)*: Kullanıcıların, birden çok görevi, birlikte işletime sokabilmelerine olanak veren işlem türü.

çok iş düzeni *(multiprogramming)*: Bir işin işletimi tamamlanmadan diğer işlerin de işletime alınabildiği işletim düzeni.

çok işleyicili sistem *(multiprocessing system)*: Efendi konumunda birden çok işleyici içeren bilgisayar sistemi.

çok kullanıcılı sistem *(multiuser system)*: Aynı anda birden çok kullanıcıya hizmet verebilen bilgisayar sistemi.

çoklama *(multiplexing)*: Aynı fiziksel iletişim kanalını, aynı anda, birden çok alıcı-verici çifti arasında kullanabilmeye olanak veren teknik. Örn. Sıklık bölümlü çoklama, zaman bölümlü çoklama *(frequency division multiplexing, time division multiplexing)*.

çoklayıcı kanal *(multiplexor channel)*: Sıklık bölümlü ya da zaman bölümlü çoklama yöntemlerinden birini kullanarak, aynı anda, aynı fiziksel iletişim ortamı üzerinden birden çok arabirime hizmet verebilen kanal.

D

dağıtılmış işlem *(distributed processing)* Değişik bilgisayar sistemleri üzerine dağılmış veri ve işlem birimlerini bir bütün olarak kullanan işlem türü.

damga *(character)*: Harf, rakam, noktalama ve denetim işaretlerinin genel adı. Örn. A damgası.

damga dizgisi *(character string)*: Bir sıra damga. Örn. "terim sözlüğü" damga dizgisi.

datagram *(datagram)*: Veri iletişim yazılımı ulaşım katmanının kullandığı veri birimi

değişim belirteci *(changed bit, dirty bit)*: Görüntü bellek düzeninde, sayfa ya da kesimlere yazma yapıldığını gösteren bir durum belirteci.

derleme *(compilation)*: Kaynak programları amaç programlara dönüştürme.

derleyici *(compiler)*: Kaynak programları amaç programlara dönüştüren sistem programı. Örn. *Pascal*, *C* derleyicileri.

devingen *(dynamic)*: Devinimli, hareketli, değişebilen, değişime açık. Örn. Devingen adres dönüştürme *(dynamic address translation)*.

diske taşıma *(swapping)*: Kimi ana bellek alanlarını, daha öncelikli programlara yer açmak üzere diske taşıma.

dizgi *(string)*: damgalardan oluşan doğrusal dizi.

dizi *(array)*: Verilerin satır, satır-sütun gibi düzenlenmiş biçimi.

dizin *(index)*: Dizi ögelerine erişimi sağlayan gösterge ya da göstergeler bütünü.

dizin yazmacı *(index register)*: Dizinli adreslemede dizin değerini tutan ana işlem birimi yazmacı.

dizinli adresleme *(indexed addressing)*: Bir dizin değerine göre adresleme.

doğrudan adresleme *(direct addressing)*: Erişilecek veri ya da veri küme adresinin doğrudan verildiği adresleme biçimi.

doğrudan bellek erişim denetleme birimi *(direct memory access control unit)*: Arabirim - ana bellek arası veri aktarımlarını gerçekleştiren yardımcı işleyici.

dolaylı adresleme *(indirect addressing)*: Erişilecek veri ya da veri küme adresinin, ek erişim ya da hesaplama sonucu elde edildiği adresleme biçimi.

dönüş bekleme süresi *(rotational delay)*: Diskte, ize eriştikten sonra ilgili sektörün okuma-yazma kafası altından geçmesinin beklendiği süre.

döngüsel kaynak bekleme *(cyclic wait*): Görevlerin, kaynak beklerken kilitlenmelerine neden olan bir durum.

durgun *(static)*: Devinimsiz, hareketsiz, değişimsiz.

durum yazmacı *(status register)*: Veri aktarım işlemlerinin izlendiği arabirim yazmacı.

E

efendi - köle ilişkisi *(master - slave relationship)*: ana ve yardımcı birim ilişkisi.

elde biti (*carry bit*): İkili toplama sonucunda işlem sözcük uzunluğunu aşan bit.

engellenir kesilme *(maskable interrupt) :* Denetim birimine maske bitiyle denetlenerek ulaşan kesilme türü.

engellenemez kesilme *(non-maskable interrupt):* Denetim birimine dolaysız ulaşan kesilme türü.

erişim *(access)*: Verilere, okuma ya da yazma amacıyla erişme. Örn. Bellek sözcüklerine erişim, diske erişim.

erişim belirteci *(accessed flag/bit)*: Görüntü bellek düzeninde, sayfa ya da kesimlere okuma ya da yazma amacıyla erişim yapıldığını gösteren bir durum belirteci.

erişim hakkı *(access right)*: Nesnelere erişimi denetlemek üzere, okuma, okuma-yazma, işletme biçiminde kodlanan bilgi.

eşlik biti *(parity bit)*: İletişim hatalarını yakalayabilmek amacıyla damga bitlerine eklenen artık bit. Örn. Çift eşlik biti, tek eşlik biti.

eşlik hatası *(parity error)*: Eşlik biti aracılığıyla saptanan iletişim hatası.

etkileşimli işlem *(interactive processing*): Çok kullanıcılı bir bilgisayar sisteminde, kullanıcıların sistemden, terminaller aracılığıyla, yalnız başına

hizmet alıyormuş izlenimini edindikleri işlem türü.

G

geçit *(gate)*: mantıksal bir işlemi yerine getiren çevrim. Örn. *ve* geçiti *ya da* geçiti.

gerçek zamanlı işlem *(real time processing)*: Yanıt süresine bir üst sınırın konabildiği, etkileşimli işlemin özel bir türü.

giriş *(input)***:** Verilerin, arabirimler aracılığıyla kullanıcı ortamından ya da ikincil belleklerden ana belleğe aktarılması.

giriş / çıkış *(input / output)***:** Verilerin ana işlem birimi - ana bellek ikilisi ile bu ikilinin dışındaki ortam arasında aktarımına ilişkin. Örn. Giriş / çıkış birimi.

giriş / çıkış kapısı *(input / output port)***:** Giriş/çıkış adres evreninde bir adrese sahip, bağımsız olarak erişilen arabirim yazmaç ya da yastığı.

görev denetim öbeği *(process control block)*: Görev iskeletinin diğer adı.

görev göçü *(process migration)*: Ağ içi bilgisayarların işletim kapasitelerini paylaştırmak üzere, işletim sırasında, bir görevi başka bir sisteme taşıma ve işletimini orada sürdürme

görev iskeleti *(task skeleton)*: İşletim bütünlüğünün korunması amacıyla her görev için tutulan ve görevi simgeleyen veri yapısı.

görev yönetimi *(process management)*: Ana işlem biriminin yönetimi.

görevler arası iletişim *(interprocess communication)*: Birlikte çalışan görevlerin karşılıklı ileti alış verişi.

görevler arası zamanuyumlama *(interprocess synchronization)*: Birlikte çalışan görevlerin işletim akışlarını karşılıklı olarak denetlemeleri.

görüntü *(virtual)*: Öyleymiş izlenimi edinilen. Örn. Görüntü bellek *(virtual memory)*

gösterge *(pointer)*: Çoğunlukla, adres değeri taşıyan yazmaç ya da değişken. Örn. Yığıt göstergesi.

grafik kullanıcı arayüzü *(Graphical User Interface-GUI)*: İşletim sistemi kabuk katmanı komutlarının ekranda ikonlar ile simgelenmesine ve tıklanarak çalıştırılmasına olanak veren sistem yazılımı.

güdüm yazmacı *(command register)*: Veri aktarımlarını gerçekleştirmek üzere arabirimin programlanmasına olanak veren arabirim yazmacı.

günleme *(updating)*: değişiklik yapma.

güvenlik *(security)*: Bir bilgisayar sisteminde saklanan verilerin gizliliğinin, bütünlüğünün ve kullanılabilirliğinin sağlanması.

I-İ

IP: *UNIX* işletim sisteminin iletişim alt kesiminin ağ katmanı.

ikidurumlu *(flip / flop)*: *0* ya da *1* durumlarından birinde bulunan ve bir denetim imi aracılığıyla bir durumdan diğerine geçebilen mantıksal çevrim.

ikili *(binary)*: *0* ya da *1* değerlerini alabilen.

ikincil bellek *(secondary memory)*: Ana bellek dışında, disk, disket, mıknatıslı şerit gibi, elektrik kesintilerinde içeriklerini yitirmeyen bellekler.

ileti *(message)*: Alıcı ile gönderici arasında aktarılan bayt dizisi.

im *(signal)*: İşaret. Fiziksel bir olayı simgeleyen zamana bağımlı değer. Örn. Kesilme istem imi.

imleç *(cursor)*: Girilen damganın ekranda görüntülendiği konum.

istem *(request)*: Talep. Örn. Doğrudan bellek erişim istemi.

istemci *(client)*: Aynı ya da farklı bir bilgisayar üzerinde çalışan, sunucu nitelikli bir programdan hizmet alan program. Örn. *ftp* istemci.

istemci-sunucu paradigması *(client-server Paradigm)*: Bir ağ içinde bütünleşmiş bilgisayar sistemleri üzerinde çalışan kimi programların, aynı ya da farklı bilgisayarlar üzerinde çalışan diğer programlara hizmet üretmesini öngören yaklaşım. Örn. *Socket* düzeneği, istemci-sunucu paradigmasını gerçekleştirmeye olanak veren bir düzenektir.

iş *(job)*: Kullanıcıların, sistemden bir bütün olarak işletilmesini istedikleri program, komut demeti.

iş denetim dili *(job control language)*: İş tanımlamada yararlanılan sisteme özel dil.

iş istasyonu *work station)*: Grafik nitelikli ekrana sahip, büyük boy bir bilgisayar sistemine bağlanarak da çalışabilen gelişmiş kişisel bilgisayar.

iş yönetimi *(job management)*: Toplu işlem kuyruğundaki işlerin görev tanımlarını yaparak hazır görevler kuyruğuna bağlayan işletim sistemi kesimi.

işleç *(operator)*: İşlenenler üzerinde yürütülecek işlemi simgeleyen, +, -, /, * gibi damga.

işlem *(operation)*: Bir ya da daha çok işlenen üzerinde yürütülen eylem.

işlenen *(operand)*: Üzerinde işlem yapılan öge. Örn. Komut işleneni.

işletmen *(operator)*: Bir bilgisayar sisteminin işletiminden en alt düzeyde sorumlu kişi.

iz *(track)*: Disk üzerinde verilerin saklandığı iç içe çemberlere verilen ad.

iz *(path)*: Bir kütüğün bulunduğu konumu belirleyen bilgi. Kütük kimliği içinde kütük adına kadar olan kesim.

K

kabuk katman *(shell)*: İşletim sisteminin kullanıcıyla etkileşim kurulan, üst soyutlama düzeyi işlemlerin yürütüldüğü kesimi. Sistem komut yorumlama katmanı.

kanal *(channel)*: Bir bilgisayar sisteminde, verilerin giriş/çıkış sürücülerinden ana belleğe aktarımını sağlayan yardımcı işleyici. Örn. Hızlı giriş/çıkış kanalı.

karşılıklı dışlama *(mutual exclusion)*: Ortak kaynağa erişim sağlayan görevin aynı kaynağa erişmek isteyen diğer görevlerin işletimini engellemesi.

karşılıklı tıkanma *(mutual blocking)*: Ortak kaynağa erişim sağlayan iki ya da daha çok görevin işletimlerini karşılıklı olarak engellemeleri.

katman *(layer)*: Bir yazılım sisteminde sistem çağrıları ile birbirinden hizmet alan sıradüzensel yapıdaki alt kesimler. Örn. Çekirdek katman, *TCP* katmanı.

kaydırma *(shift)*: Bir sözcüğü oluşturan bitlerin herbirini, komşusunun yerine gelecek biçimde sağa, sola doğru taşıma. Örn. Kaydırma komutları.

kaynak *(resource)*: Görevlerin kullanımına sunulan ana işlem birimi, ana bellek, giriş / çikiş birimi gibi donanım; kütük, sistem yazılımı gibi yazılım nitelikli ögeler. Örn. Kaynak paylaşımı.

kaynak *(source)*: Çıkma noktası. Örn. İletiyi üreten kaynak görev, iletinin yollandığı *hedef (destination)* görev.

kaynak program *(source program)*: Bir derleyici ya da yorumlayıcı aracılığıyla amaç programa dönüşecek üst düzey/ simgesel program.

kesen algoritma *(preemptive algorithm)*: Görevlerin, işletimlerini istemleri dışında kesen algoritma.

kesen yönetim *(preemptive scheduling)*: İşletimi, görevlerden kaynaklanmayan nedenlerle de kesmeye olanak veren görev yönetimi. Örn. Uyarı tabanlı görev yönetimi *(event driven scheduling)* kesen bir yönetim biçimidir.

kesilme *(interrupt)*: İşleyicinin, donanım birimlerinden kaynaklanan dış uyarılar sonucu, rasgele bir anda, işletmekte olduğu programı, ileride geri dönmek üzere keserek uyarıyla ilişkili özel bir yordamın işletimine geçmesi.

kesilme alındısı *(interrupt acknowledgement)*: Kesilme istemine ilişkin olumlu yanıt.

kesilme istem imi *(interrupt request signal)*: İşleyici dışındaki birimlerden işleyiciye ulaşan ve kesilme istemini simgeleyen im.

kesilme maskesi *(interrupt mask)*: kesilme istem iminin, işleyici denetim birimine ulaşıp ulaşmamasını denetleyen düzenek.

kesilme önceliği denetleme birimi *(interrupt priority control unit)*: Kesilme istemlerinin ele alınışında (kesilme yordamlarına iç içe sapışlarda) öncelikleri, ana işlem birimi adına gözeten yardımcı işleyici.

kesilme vektörü *(interrupt vector)*: Kesilme yordam başlangıç adreslerinin saklandığı ana bellek kesimi.

kesilme yordamı *(interrupt routine)*: Kesilme istemiyle sapılan özel hizmet yordamı.

kesim *(segment)*: Mantıksal bağımsızlık içeren program parçaları. Örn. Kod kesimi, veri kesimi, yığıt kesimi.

kesim taban yazmacı *(segment base register)*: Kesimli bellek yönetiminde, adres dönüştürme işlemleri sırasında kullanılan, kesim başlangıç adreslerinin tutulduğu yazmaç.

kesim tanım çizelge yazmacı *(segment description table register)*:

Kesimli bellek yönetiminde, adres dönüştürme işlemleri sırasında kullanılan, kesim tanım çizelgesi başlangıç adresinin tutulduğu yazmaç.

kesim tanım çizelgesi *(segment description table)*: Kesimli bellek yönetiminde, adres dönüştürme işlemleri sırasında kullanılan, kesim başlangıç adreslerinin tutulduğu çizelge.

kesim zorlama öneki *(segment overwrite prefix)*: Kesimlemenin yapıldığı sistemlerde işlenecek kesimi belirleme sözde komutu.

kesimleme *(segmentation)*: Program adres evrenini, mantıksal bağımsızlık içeren parçalar biçiminde düşünme.

kesimli bellek yönetimi (*segmented memory management)*: Programları ana belleğe bitişken olmayan kesimler biçiminde yükleyerek çalıştırmaya olanak veren bellek yönetimi.

kılavuz kütük *(directory)*: Bir sürücü üzerinde saklanan kütüklere erişimi sağlayan özel kütük. Örn. Kök kılavuz, alt kılavuz kütük.

kısa dönemli planlama *(short term scheduling)*: Görev yönetiminin diğer adı.

kilitleme *(locking)*: Bellek yastık alanı, tutanak, giriş/çıkış sürücüsü gibi ögelerin erişimini diğer görevlere kapama. Örn. İki aşamalı kilitleme *(two phase locking)*.

kilitlenme *(deadlock)*: Görevlerin işletimlerinin, hiçbir zaman gerçekleşmeyecek koşulları beklemeleri sonucu engellenmesi.

kilitlenmelerden korunma *(deadlock prevention)*: İşletim sistemlerinde kilitlenmeleri engellemede yararlanılan bir yaklaşım.

kilitlenmeleri önleme *(deadlock avoidance)*: İşletim sistemlerinde kilitlenmeleri engellemede yararlanılan bir yaklaşım.

kilitlenmeleri yakalama *(deadlock detection)*: İşletim sistemlerinde kilitlenmeleri engellemede yararlanılan bir yaklaşım.

kişisel bilgisayar *(personnel computer)*: Tüm kaynakları tek bir kullanıcıya atanan mikrobilgisayar.

koşut *(parallel)*: Birlikte çalışan, yanyana giden.

koşut işlem *(parallel / concurrent processing)*: Birlikte çalışan görevlerle ilgili işlem türü.

koşut iletişim *(parallel communication)*: Gönderici ile alıcı arasında aktarılan damga kodunu oluşturan bitlerin, herbirinin ayrı bir iletim hattı üzerinden aktarıldığı iletişim biçimi. Paralel iletişim de denmektedir. Örn. *Centronix*, koşut iletişim standartı.

kök kılavuz *(root directory)*: Ağaç yapılı kılavuz kütüklerde en alt düzeyde yer alıp tüm alt kılavuzlara erişimi sağlayan özel kılavuz kütük. Örn. Kök kılavuz sürücü formatlama aşamasında yaratılır ve silinemez.

komut *(instruction)*: Bir programı oluşturan en küçük öge. Örn. Simgesel komut, makina komutu.

konuma duyarlı *(location sensitive)*: Ana bellekte yer aldığı konum, önlem alınmadan değiştirilemeyen içerikle ilgili. Örn. Sapma adresleri konuma duyarlı değerlerdir.

koruma *(protection)*: Bir bilgisayar sisteminde saklanan verilerin izinsiz kullanım, günlenme ve silinmelerinin engellenmesi.

koruma alanı *(protection domain)*: Erişimi kısıtlanan ögeler kümesi.

koruma halkası *(protection ring)*: İç içe halkalar biçiminde düşünülen ve erişimi kısıtlanan ögeler kümesi.

kritik kesim *(critical section)*: görevlerin ortak kaynağa erişim yapan kesimleri.

kullanıcı *(user)*: Bilgisayar sisteminden hizmet alan kişi.

kullanıcı kodu *(user code)*: Kullanıcının bilgisayar sistemindeki tanımı ya da kimliği.

kullanıcı modu *(user mode)*: Görev işletiminde bir ayrıcalık düzeyi. Örn. Kullanıcı / işletim sistemi modu *(user / kernel mode)*.

kullanılabilirlik *(availability)*:Hizmet verme sürekliliğini gösteren özellik.

kurtarma *(recovery)*: Hatadan arındırarak yeniden kullanılır kılma.

küçük ağırlıklı bayt önce yerleşimi *(little endian)*: Bir sözcüğü oluşturan baytların, ana bellekte, küçük ağırlıklı bayttan başlayarak yerleşmesi.

kütük *(file)*: Ana bellek dışında saklanan veri kümelerine verilen genel ad. Örn. Disk kütüğü, yazıcı kütüğü.

kütük açma (*open)*: Kütük erişim bilgilerinin ana belleğe taşınma evresi.

kütük kapama (*close)*: Kütükle ilgili ana bellekte yapılan günlemeleri diske yazarak kütüğün ana bellekteki varlığına son verme.

kütük tanım / atama çizelgesi (*file description / allocation table)*: Bir sürücüyle ilgili olarak, kütüklere atanan disk öbek adres bilgilerinin tutulduğu çizelge. Örn. *MS-DOS*'ta *FAT*.

kütük yönetim sistemi *(file system)*: Kütüklerin kolay, hızlı ve güvenli kullanımı ile ikincil belleklerin verimli kullanımını sağlayan işletim sistemi birleşeni.

kuyruk *(queue)*: Görevlerin, kaynak beklerken sıralandığı yapı. Örn. Hazır görevler kuyruğu, giriş / çıkış bekler kuyruğu.

kuyruk çizeneği *(queuing diagram)*: Görevlerin, işletimleri sırasında bulunabildikleri durumları, bağlı oldukları kuyruklar ve bunlar arasındaki geçişlerle gösteren çizenek.

M-N

makina komutu *(machine code)*: İşleyicinin dolaysız işletebildiği ikili kodlanmış komut.

makina döngüsü *(machine cycle)*: Belirli sayıda saat periyodundan oluşan komut işletiminin alt evresi

mantıksal adres *(logical address)*: Kullanıcının düşüncesinde gerçekliği olan adres. Örn. Komutların program başına göreli adresleri.

mıknatıslı şerit birimi *(magnetic tape unit)*: Verilerin yedeklenmesi amacıyla kullanılan giriş / çıkış birimi.

mikroişleyici *(microprocessor)*: Çok yoğun tümleştirme *(VLSI)* tekniği kullanılarak üretilmiş işleyici. Örn. Intel 80486 mikroişleyicisi.

noktalı onlu gösterim *(dotted decimal representation)*: *IP* adresleri için kullanılan gösterim biçimi

O-Ö

oğul görev *(child process)*: Ata görevin yarattığı görev.

onaltılı *(hexadecimal)*: taban değerin 16 olduğu sayı sistemine ilişkin.

ortak anahtar *(public key)*: Şifrelemede anahtar dağıtım sorununu ortadan kaldırmak üzere öngörülen yöntemin kullandığı anahtar çiftinden, gizliliği gerçekleştirmede, şifreleme amacıyla kullanılanı.

ortak anahtar şifreleme yöntemi *(public key encryption)*: Şifrelemede anahtar dağıtım sorununu ortadan kaldırmak üzere, farklı anahtarlarla şifreleme ve şifre çözme yapan yöntem.

oturum tabanlı hizmet *(connection oriented)*: Hata ve sıra denetimi yapılarak verilen veri iletişim hizmeti.

oturumsuz hizmet *(connectionless)*: Sıra denetimi yapılmadan verilen veri iletişim hizmeti.

öbek *(block)*: Disk, mıknatıslı şerit gibi birimlerle ana bellek arasında bir seferde aktarılan veri kümesi.

ön bellek *(cache)*: Ana bellekle ana işlem birimi arasında yer alan, sıkça erişilen verilerin tutulduğu, ana belleğe göre en az 10 kat hızlı özel bellek.

öncekilik *(precedence)*: Birlikte çalışan görevlerin birbirlerine göreli işletim sıralarıyla ilgili. Örn. Öncekilik çizgesi *(precedence graph)*.

öncelik *(priority)*: Bir istem ya da görevin diğerlerine göre daha üstün işletim haklarına sahip olması.

öncelik yaşlanması *(priority aging)*: Görev önceliklerinin dönem dönem değiştirilerek düşük öncelikli görevlerin de işletilebilmesine olanak veren teknik.

öykünüm *(emulation)*: Bir donanımın diğer bir donanımın çalışma biçimini taklit etmesi, öykünmesi. Örn. Çevrim içi öykünüm *(in circuit emulation)*.

özdevimli *(automatic)*: Dış denetim gerektirmeden, kendiliğinden.

özel anahtar *(private key)*: Şifrelemede anahtar dağıtım sorununu ortadan kaldırmak üzere öngörülen yöntemin kullandığı anahtar çiftinden, gizliliği gerçekleştirmede, şifre çözme amacıyla kullanılanı.

öznitelik *(attributes)*: Bir kütüğe ilişkin özel bilgiler.

P-R

paket *(packet)*: Veri iletişim yazılımı ağ katmanının kullandığı veri birimi.

parola *(password)*: Kullanıcıların bilgisayar sistemine girişlerini denetlemede yararlanılan özel sözcük.

posta kutusu *(mail box)*: Görevler arası iletişimde iletilerin aktarıldığı özel bellek yastık alanı.

protokol *(protocol)*: Veri iletişiminde değişik bilgisayar sistemlerinin uyması gereken kurallar manzumesi.

protokol veri birimi *(protocol data unit)*: Veri iletişim yazılımının herhangi bir katmanının kullandığı veri yapısı.

rasgele erişim *(random access)*: Bir önce erişilen konumdan bağımsız olarak, bir dizinin / kütüğün herhangi bir sözcüğüne / tutanağına erişim.

S-Ş

saltık adres *(absolute address)*: Sözcüklerin ana belleğin başına göreli adresleri.

sayaç *(counter)*: İçeriği artırılıp eksiltilerek günlenen yazmaç. Örn. Program sayacı, yığıt sayacı.

sayfa *(page)*: Ana belleği ya da program adres evrenini oluşturan eşit uzunluktaki parçalar.

sayfa bellekte belirteci / biti *(page exception flag / bit)*: Görüntü bellek düzeninde erişilen mantıksal sayfanın bellekte bulunup bulunmadığını gösteren belirteç ya da bit.

sayfa çıkarma algoritması *(page replacement algorithm)*: Sayfalı görüntü bellek yönetiminde, diske taşınacak sayfayı belirleme algoritması.

sayfa içi adres *(page offset)*: Sayfa başına göreli adres.

sayfa taban yazmacı *(page base address)*: Sayfalı bellek yönetiminde, adres dönüştürme işlemleri sırasında kullanılan, sayfa başlangıç adreslerinin tutulduğu yazmaç.

sayfa tanım çizelgesi *(page description table)*: Sayfalı bellek yönetiminde, adres dönüştürme işlemleri sırasında kullanılan, sayfa başlangıç adreslerinin tutulduğu çizelge.

sayfa tanım çizelge yazmacı *(page description table register)*: Sayfalı bellek yönetiminde, sayfa tanım çizelgesi başlangıç adresinin tutulduğu yazmaç.

sayfa tanım çizelgesi sınır yazmacı *(page description table bound register)*: Sayfalı bellek yönetiminde, sayfa tanım çizelgesi boyunun tutulduğu yazmaç.

sayfalı bellek yönetimi *(paged memory management)*: Parçalanmayı önlemek üzere belleği eşit uzunluktaki parçalara bölen ve görevlere bitişken yer atamayı gerektirmeyen bir yönetim biçimi.

sayfalı görüntü bellek yönetimi *(demand paging)*: Sayfalı bellek yönetimini görüntü bellek düzeninde gerçekleştiren bellek yönetimi.

seçici kanal *(selector channel)*: Aynı anda tek bir giriş / çıkış birimine hizmet verebilen kanal türü.

seçmeli giriş / çıkış programlama *(programmed IO*: Aarabirim durum yazmaçlarının sınanması yoluyla yürütülen giriş / çıkış denetimi.

sektör *(sector)*: Diskte fiziksel en küçük veri saklama birimi.

semafor *(semaphore)*: Birlikte çalışan görevlerin zamanuyumlanmasında yararlanılan özel bir tür değişken.

sığa *(capacity)*: Sığabilen büyüklük, boyut.

sınır yazmacı *(bound register)*: Bölümlü bellek yönetimlerinde bölümden taşmaları denetlemeye yarayan ana işlem birimi yazmacı.

sıradan *(ordinary)*: Sistem yönünden hiçbir ayrıcalığı ve önceliği olmayan. Örn. Sıradan kullanıcı.

sıradan erişim *(sequential access)*: Erişilecek sözcüğün / tutanağın hep bir önce erişilenden bir sonraki sözcük / tutanak olduğu erişim türü.

sıradüzen *(hierarchy)*: Önem ya da öncelik sırası.

simgesel programlama *(symbolic / assembly programming)*: Makina diline en yakın programlama türü.

silindir *(cylinder)*: Diskte, değişik yüzeylere dağılmış, aynı çapta izler bütünü.

sistem çağrısı *(system call)*: İşletim sistemlerinde, değişik işlevsel kesimlerden hizmet almada yararlanılan, yordam çağırmaya benzer düzenek.

sistem komutu *(system command)*: İşletim sisteminin kabuk katmanınca yorumlanıp çalıştırılan komutlar.

sistem programcısı *(system programmer)*: Sistem çağrılarını kullanarak işletim sistemine ilişkin programlar geliştiren, ayrıcalıklı programcı.

socket: *TCP/IP* iletişim yazılımının sistem çağrı düzeneği.

soyutlama *(abstraction)*: Düşüncede gerçekliği olan (soyut) modellerle düşünme.

soyutlama düzeyi *(abstraction level)*: Muhakemeye taban oluşturan soyut modelin gerçekliğe yakınlık / uzaklık düzeyi.

sözcük *(word)*: Ana bellekte bir seferde erişilen öge.

sözcük tür tanım eki *(word descriptor, tag)*: Sözcük içinde saklanan içerik tür kodu.

sözdizim *(syntax)*: Dilde sözcük ve tümceleri oluşturan damgaların sıralanış kuralları bütünü.

sunucu *(server)*: Aynı ya da farklı bir bilgisayar üzerinde çalışan, istemci nitelikli bir programa hizmet üreten program. Örn. *ftp* sunucu.

süreç denetimi *(process control)*: Fiziksel üretim süreçlerinin bilgisayarla denetimi.

sürücü *(drive / driver)*: Giriş / çıkış birimlerinde, verilerin fiziksel olarak yer aldığı birleşen, aygıt. Örn. Disk sürücü.

sürücü yordam *(driver)*: Giriş / çıkış birimini, en alt düzeyde denetleyen, işe koşan yordam. Örn. Aygıt sürücü.

şifreleme *(cryptography - encryption)*: Güvenlik nedeniyle, verilerin saklanırken ya da iletilirken bir anahtara göre dönüştürülmesi. Örn. Simetrik ya da ortak anahtar şifreleme yöntemi.

şifre çözme *(decryption)*: Şifrelenmiş verilerin anlamlı görünümlerine dönüştürülmesi.

T

taşma *(overflow)*: Aritmetiksel bir işlemin, kodlamanın öngörmediği bir uzunlukta sonuç üretmesi durumu.

TCP: *UNIX* işletim sisteminin iletişim alt kesiminin, hata ve sıra denetimi yapabilen ulaşım katmanı.

TCP/IP: *UNIX* işletim sisteminin iletişim alt kesiminin adı. Internet Protokolu.

tek iş düzeni *(monoprogramming)*: Tüm bilgisayar kaynaklarının, aynı anda tek bir iş için kullanıldığı işletim düzeni.

toplu işlem *(batch processing)*: Sisteme sunulan işlerin biriktirilip topluca işletime alınması.

toplu işlem kuyruğu *(batch queue)*: Toplu işlemde işlerin, işletime alınmak üzere beklediği kuyruk.

tutanak *(record)*: Kütüğü oluşturan birim.

tüketilir kaynak *(consumable resource)*: Yastık alanları, iletiler gibi görevlerce paylaşılarak tüketilen ancak işletim sonunda varlıkları son bulan kaynaklar.

U-V

UDP: *UNIX* işletim sisteminin iletişim alt kesiminin, sıra denetimi yapmayan ulaşım katmanı.

uzaktan yordam çağırma *(remote procedure call)*: Dağıtılmış işlemde bir gerçekleştirim tekniği.

uzun dönemli planlama *(long term scheduling)*: İş yönetiminin diğer adı.

veri bağlantı katmanı *(data link layer)*: Katmanlı mimaride, fiziksel katmanın hemen üzerinde yer alan katman. Veri iletişim arabirimi sürücü yazılımı.

veri bütünlüğü *(data integrity)*: Verilerin bozulmalara karşı korunmasıyla ilgili.

veri gizliliği *(data privacy)*: Verilere sahibinin izni, denetimi ve bilgisi dışında üçüncü kişilerin erişiminin engellenmesiyle ilgili.

veri güvenliği *(data security)*: Verilerin izinsiz erişim ve bozulmalara karşı korunmasını sağlayan önlemler bütünü.

veri yolu *(data bus)*: Ana işlem birimi, ana bellek ve giriş/çıkış arabirimleri arasında verileri simgeleyen imlerin iletildiği hat demeti.

Y-Z

yastık *(buffer)*: Verilerin geçici olarak saklandığı ana bellek ya da arabirim yerel bellek alanı.

yatay erişim *(seek)*: Diskte ize erişim.

yatay erişim süresi *(seek time)*: Diskte bir ize ortalama erişim süresi.

yazılım kesilmesi *(software interrupt)*: Kimi özel makina komutlarının çalıştırılması yoluyla, donanım kesilmeleriyle ile aynı sonuçların yaratılması.

yazmaç *(register)*: Ana işlem birimi iç bellek sözcükleri.

yedekleme *(backup)*: Koruma amacıyla kütüklerin ek kopyalarını alma.

yeniden girilir program *(reentrant program)*: Birden çok görev kapsamında işletilebilen yordam.

yeniden kullanılır kaynak *(reusable resource)*: Aynı anda en çok bir göreve atanan, sınırlı bir süre sonunda serbest bırakılan ve başka bir görevin kullanımına verilen kaynak. Örn. Değişken, kütük.

yeri değişir program *(relocatable program)*: İçerdiği tüm adres bilgileri program başına göreli olan program.

yer değiştirme *(relocation)*: Bir yordamın, işletimi sırasında ana bellekte yüklendiği yerin değiştirilmesi.

yer değiştirme taban yazmacı *(relocation register)*: Yerideğişir bölümlü bellek yönetiminde, adres dönüştürme işlemlerinin taban aldığı ana işlem birimi yazmacı.

yerel *(local)*: Bulunulan ortama özel. Örn. Yerel sayfa çıkarma politikası.

yerellik *(locality)*: Bir görevin, birim sürede eriştiği sözcüklerin adreslerinin dağılımıyla tanımlanan evren.

yerellik düzeyi *(degree of locality)*: Bir görevin, birim sürede eriştiği adres evreninin boyutunun ters orantılı olarak belirlediği düzey. Erişilen adres evreni ne kadar küçük ise yerellik düzeyi o kadar yüksektir.

yetki *(capability)*: Nesneye erişim hakkı.

yetki listesi *(capability list)*: Görevlerin nesnelere hangi haklarla erişeceklerini gösteren liste.

yığıt *(stack)*: son gelen önce mantığına uygun veri yapısı.

yığıt göstergesi *(stack pointer)*: Yığıt başındaki sözcüğün adresini tutan ana işlem birimi sayacı.

yığıt başı *(top of the stack)*: Yığıta son giren değerin yer aldığı konum.

yol *(bus)*: Ana işlem birimi, ana bellek ve giriş/çıkış arabirimleri arasında adres, veri ve denetim imlerinin iletildiği iletim hatları demeti.

yol kullanım istemi *(bus request)*: Köle konumundaki işleyicilerden efendi konumundaki işleyiciye gönderilen, ortak kullanılan yolları kullanabilme istemi. Örn. Doğrudan bellek erişim denetleme biriminden ana işlem birimine gönderilen istem.

zaman dilimli *(time sliced)*: Görevlerin ana işlem birimine eşit uzunlukta sürelerle, dönerek anahtarlanması. Örn. *Round Robin* görev yöntemi.

zaman paylaşımlı *(time sharing)*: Etkileşimli işlemin eski adı.

zamanuyumlama *(synchronization)*: Birlikte çalışan görevlerin işlem akışlarının, işletim bütünlüğünü korumak üzere denetlenmesi.

zamanuyumlu iletişim *(synchronous communication)*: Gönderici alıcı arası zamanuyumunun özel zamanuyum damgalarıyla sağlandığı, aktarılan damgalara, *start, stop* gibi ek zamanuyum bitlerinin eklenmediği iletişim biçimi.

zamanuyumsuz iletişim *(asynchronous communication)*: Gönderici alıcı arası zamanuyumunun *start/stop* damgalarıyla sağlandığı iletişim biçimi.

zincirleme bağlantı *(chaining)*: Kesilme istem alındı iminin arabirimlere zincirleme olarak ulaştığı yapı.

İNGİLİZCE TÜRKÇE

İ Ş L E T İ M S İ S T E M L E R İ

A

absolute address: Saltık adres
abstraction: Soyutlama
abstraction level: Soyutlama düzeyi
access: Erişim
access right: Erişim hakkı
accessed flag/bit: Erişim belirteci / biti
acknowledgement: Alındı
acyclic directed graph: Döngüsüz yönlü çizge
address: Adres
address bus: Adres yolu
address translation:Adres dönüştürme
addressing: Adresleme
allocation: Atama
allocation table: Atama çizelgesi
arithmetic logic unit: Aritmetik mantık birimi
array: Dizi
assembly programming: Simgesel programlama
attributes: Öznitelik
automatic: Özdevimli
availability: Kullanılırlık, kullanılabilirlik

B

Backbone: Omurga
backup: Yedekleme
batch processing: Toplu işlem
batch queue: Toplu işlem kuyruğu
big endian: Büyük ağırlıklı bayt önce
binary: İkili
bit: Bit
block: Öbek
bound register: Sınır yazmacı
buffer: Yastık
bus: Yol
bus request: Yol kullanım istemi

C

cache: Ön bellek
capability: Yetki
capability list: Yetki listesi
capacity: Sığa
carry bit: Elde biti
central processing unit: Ana işlem birimi
chaining: Zincirleme
changed bit: değişim biti
channel: Kanal
character: Damga
character string: Damga dizgisi
client: İstemci, işveren
client/server: İstemci/Sunucu, İşveren/İşgören
child process: Oğul görev
close: Kütük kapama
command: güdüm, komut
command register: Güdüm yazmacı
compaction: Bitiştirme

compilation: Derleme
compiler: Derleyici
component: Birleşen
concurrent processes: Birlikte çalışan görevler
concurrent processing: Koşut işlem
confidentiality: Gizlilik
consumable resource:Tüketilir kaynak
connection: Bağlantı
connectionless: Oturumsuz
connection oriented: Oturum tabanlı
context: Bağlam
context switching:Bağlam anahtarlama
contiguous: Bitişken
counter: Sayaç
critical section: Kritik kesim
cryptography: Şifreleme
cursor: İmleç
cyclic wait: Döngüsel bekleme
cylinder: Silindir

D

data bus: Veri yolu
data integrity: Veri bütünlüğü
data privacy: Veri gizliliği
data security: Veri güvenliği
deadlock: Kilitlenme
deadlock avoidance: Kilitlenmeden sakınma
deadlock detection: Kilitlenmeyi yakalama
deadlock prevention: Kilitlenmeden korunma
decryption: Şifre çözme
degree of locality: Yerellik düzeyi, yerellik derecesi
demand paging: Sayfalı görüntü bellek yönetimi
destination: Hedef
device: Aygıt, sürücü
device driver: Aygıt Sürücü
diagram: Çizenek
digit: Basamak
direct addressing: Doğrudan adresleme
direct memory access control unit: Doğrudan bellek erişim denetleme birimi
directory: Kılavuz, kılavuz kütük
dirty bit: Değişim biti
distributed: Dağıtılmış
distributed processing: Dağıtılmış işlem
dotted: Noktalı
drive: Sürücü, aygıt (donanım)
driver: Sürücü yazılım
duplex: Çift yönlü
dynamic: Devingen

E

emulation: Öykünüm
encryption: Şifreleme
event driven scheduling: Uyarı tabanlı yönetim

F

file: Kütük
file allocation table: Kütük tanım çizelgesi, kütük atama çizelgesi
file description table: Kütük tanım çizelgesi
file system: Kütük sistemi
flip/flop: İkidurumlu
flowchart: Akış çizeneği, akış çizgesi
frequency division multiplexing: Sıklık bölümlü çoklama
full duplex: Tam çift yönlü

G-H-I

gate: Geçit
graph: Çizge
half duplex: Yarı çift yönlü
hexadecimal: Onaltılı
hierarchy: Sıradüzen
in-circuit emulation: Çevrim içi öykünüm

in-circuit emulator: Çevrim içi öykünücü
index: Dizin
index register: Dizin yazmacı
indexed addressing: Dizinli adresleme
input: Giriş
input/output: Giriş / çıkış
input/output port: Giriş / çıkış kapısı
instruction: Komut
integrity: Bütünlük
integration: Tümleme
intelligence: Anlayış
interactive processing: Etkileşimli işlem
interface: Arabirim
interprocess communication: Görevler arası iletişim
interprocess synchronization: Görevler arası zamanuyumlama
interrupt: Kesilme
interrupt acknowledgement: Kesilme alındısı
interrupt mask: Kesilme maskasi
interrupt priority control unit: Kesilme önceliği denetleme birimi
interrupt request signal: Kesilme istem imi
interrupt routine: Kesilme yordamı
interrupt vector: Kesilme vektörü

J-K-L

job: İş
job control language: İş denetim dili
job management: İş yönetimi
kernel: Çekirdek katman
kernel mode: İşletim sistemi modu
layer: Katman
least significant digit: En küçük ağırlıklı basamak
length: Boy
link: Bağ
linker: Bağlayıcı
little endian: Küçük ağırlıklı bayt önce
local: yerel
locality: yerellik
location sensitive: Konuma duyarlı
locking: Kilitleme
logical address: Mantıksal adres
long term scheduling: İş yönetimi, uzun dönemli planlama

M-N

machine code: Makina kodu
machine cycle: Makina döngüsü
magnetic tape unit: Mıknatıslı şerit birimi
mailbox: Posta kutusu
mainframe computer: Büyük boy bilgisayar
manual: El kitabı
maskable interrupt: Engellenir kesilme
master/slave relationship: Efendi / köle ilişkisi
message: İleti
microprocessor: Mikroişleyici
migration: Göç
mirroring: Aynalama
monoprogramming: Tek iş düzeni
most significant digit: En büyük ağırlıklı basamak
multilevel: Çok düzeyli
multiplexing: Çoklama
multiprocessing system: Çok işleyicili sistem
multiprogramming: Çok iş düzeni
multitasking: Çok görevli işlem
multiuser system: Çok kullanıcılı sistem
mutual blocking: Karşılıklı tıkanma
mutual exclusion: Karşılıklı dışlama
network: Ağ
nonmaskable interrupt: Engellenemez kesilme

O

Object: Nesne
object program: Amaç program
object oriented: Nesneye yönelik
on line: Çevrim içi
open: Kütük açma
operand: İşlenen
operation: İşlem
operator: İşleç, işletmen
ordinary: Sıradan
output: Çıkış
output port: Çıkış kapısı
overflow: Taşma
overrun: Aşma

P

page: Sayfa
page base address: Sayfa taban yazmacı
page description table: Sayfa tanım çizelgesi
page description table register: Sayfa tanım çizelge yazmacı
page exception flag: Sayfa bellekte biti
page offset: Sayfa içi göreli adres
page replacement algorithm: Sayfa çıkarma algoritması
paged memory management: Sayfalı bellek yönetimi
parallel: Koşut
parallel communication: Koşut iletişim
parent process: Ata görev
parity bit: Eşlik biti
parity error: Eşlik hatası
partition: Bölüm
partitioned memory management: Bölümlü bellek yönetimi
password: Parola
path: İz
performance: Başarım
personal computer: Kişisel bilgisayar
physical address: Fiziksel adres
pointer: Gösterge
precedence: Öncekilik
precedence graph: Öncekilik çizgesi
preemptive algorithm: Kesen algoritma
preemptive scheduling: Kesen yönetim
priority: Öncelik
priority aging: Öncelik yaşlanması
privilege: Ayrıcalık
privacy: Gizlilik
private: Özel
process control: Süreç denetimi
process control block: Görev denetim öbeği
programmed I/O: Seçmeli giriş/çıkış programlama.
protection: Koruma
protection domain: Koruma alanı
protection ring: Koruma halkası
public: Ortak

Q-R

queue: Kuyruk
queuing diagram: Kuyruk çizeneği
random access: Rasgele erişim
real time processing: Gerçek zamanlı işlem
record: Tutanak
recovery: Kurtarma
reentrant program: Yeniden girilir program
reference manual: Başvuru el kitabı
register: Yazmaç
relocatable program: Yeri değişir program
relocation: Yer değiştirme
relocation register: Yer değiştirme yazmacı
remote procedure call: Uzaktan yordam çağırma
request: istem

resource: kaynak
reusable: Yeniden kullanılır
root directory: Kök kılavuz
rotational delay: Döngüsel gecikme, dönüş gecikmesi
round robin scheduling: Eş öncelikli görev yönetimi

S

secondary memory: İkincil bellek
security: Güvenlik
seek: İze erişim
seek time: İze erişim süresi
segment: Kesim
segment base register: Kesim taban yazmacı
segment description table: Kesim tanım çizelgesi
segment description table register: Kesim tanım çizelge yazmacı
segment overwrite prefix: Kesim zorlama öneki
segmentation: Kesimleme
segmented memory management: Kesimli bellek yönetimi
sector: Sektör
selector channel: Seçici kanal
semaphore: Semafor
sequential access: Sıradan erişim
serial communication: Ardıl iletişim
server: Sunucu, İşgören
shared resource: Paylaşılır kaynak
shell: Kabuk katman
shift: Kaydırma
short term scheduling:Görev yönetimi
signal: İm
significance: Ağırlık
size: Boy
software interrupt: Yazılım kesilmesi
source: Kaynak
source program: Kaynak program
stack: Yığıt
stack pointer: Yığıt göstergesi
state diagram: Durum çizeneği
static: Durgun
status register: Durum yazmacı
string: Dizgi
swapper: Diske taşıma görevi
swapping: Diske taşıma
switching: Anahtarlama
symbolic: Simgesel
synchronization: Zamanuyumlama
synchronous communication: Zamanuyumlu iletişim
syntax: Sözdizim
system call: Sistem çağrısı
system command: Sistem komutu
system programmer: Sistem programcısı

T

table: Çizelge
tag: Tür tanım eki
task skeleton: Görev iskeleti
time division multiplexing: Zaman bölümlü çoklama
time sharing: Zaman paylaşımlı
time sliced: Zaman dilimli
timing diagram: Zaman çizeneği
top of the stack: Yığıt başı
track: İz
tree: Ağaç
two phase locking: İki aşamalı kilitleme

U-V-W

unit: Birim
update: Değiştirme, günleme
user: Kullanıcı
user code: Kullanıcı kodu
user mode: Kullanıcı modu
virtual: Görüntü/sanal
virtual memory: Görüntü/sanal bellek
VLSI : Çok büyük yoğunlukta tümleştirme
word: Sözcük
work station: İş istasyonu

D N

İ Ş L E T İ M S İ S T E M L E R İ