İŞLETİM SİSTEMLERİ

DERS 9 BELLEK YONETİMİ

BELLEK YONETİMİ

- Bellek önemli bir kaynaktır ve dikkatli yönetilmelidir. İşletim sistemlerinde bellek hiyerarşisini yöneten parçaya **bellek yöneticisi(memory manager) denilir.**
- Bellek yöneticisinin görevi, belleğin hangi parçalarının kullanımda olduğunu, hangi parçalarının kullanılmadığını izlemek, proseslere bellek tahsis etme (allocate) , tahsis edilen belleği geri almak (deallocate) ve belleğin çalışan prosesler için yeterli olmadığı durumlarda bellek ile disk arasındaki takas işlemlerini gerçekleştirmektir.

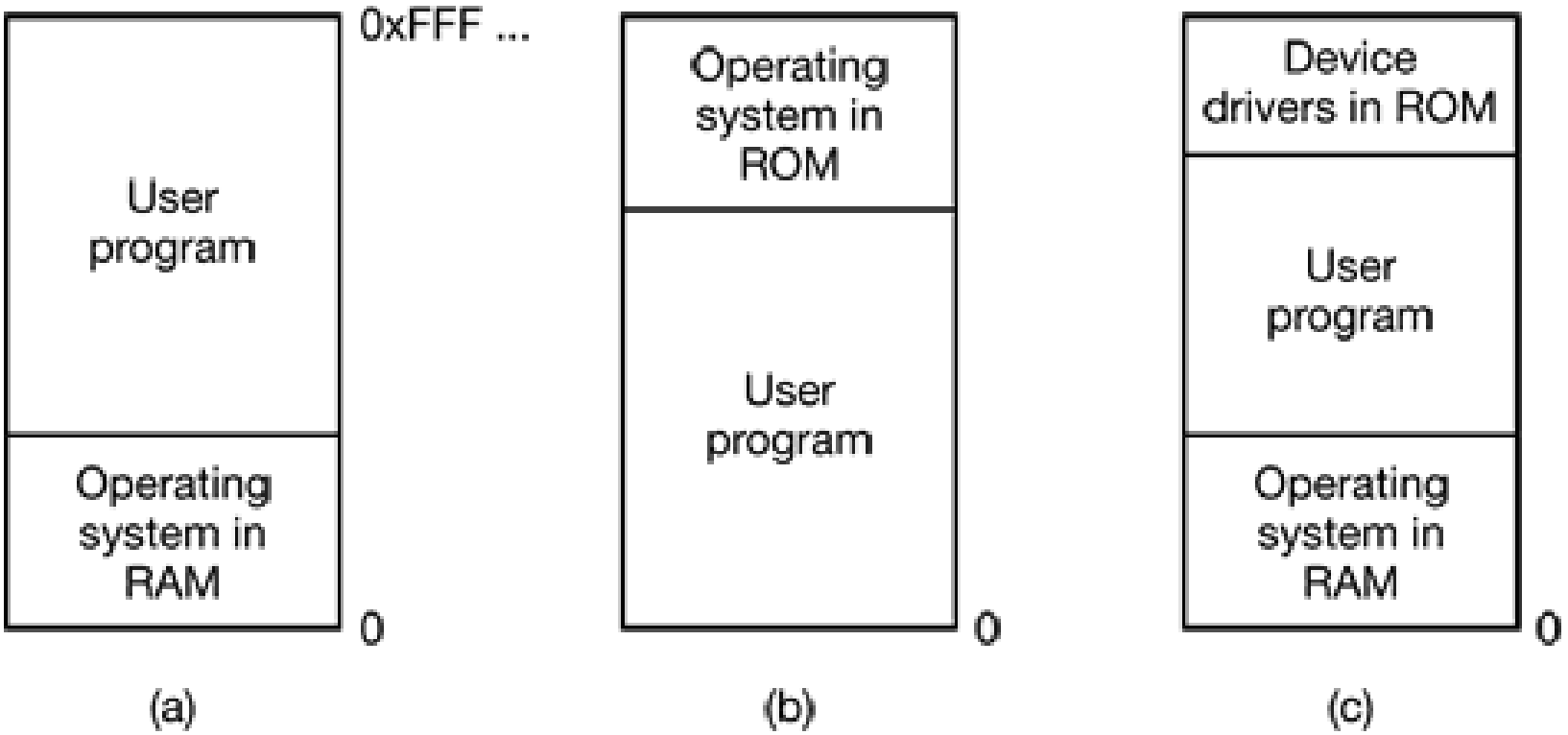
1. BASİT BELLEK YÖNETİMİ

- Bellek yönetim sistemleri iki temel sınıfta incelenebilir:
  - Çalışma zamanında süreçleri bellek ile disk arasında yer sürekli yer değiştirenler (takaslama, sayfalama yapanlar). Süreçlerin bu şekilde disk ile bellek arasında yer değiştirilmesinin nedeni, belleğin boyutunun yetersiz olmasıdır.
  - Değiştirme işlemi yapmayanlar. Bu metot ilkine göre daha basittir.

1. 1. SAYFALAMA VE TAKASLAMA YAPMAYAN TEKLİ PROGRALAMA

- En basit bellek yönetim şekli, bilgisayarda sadece bir prosesin çalıştırılmasına izin veren ve belleğin işletim sistemi ile çalışan proses arasında paylaşıldığı işletim sistemlerindeki kullanımdır.

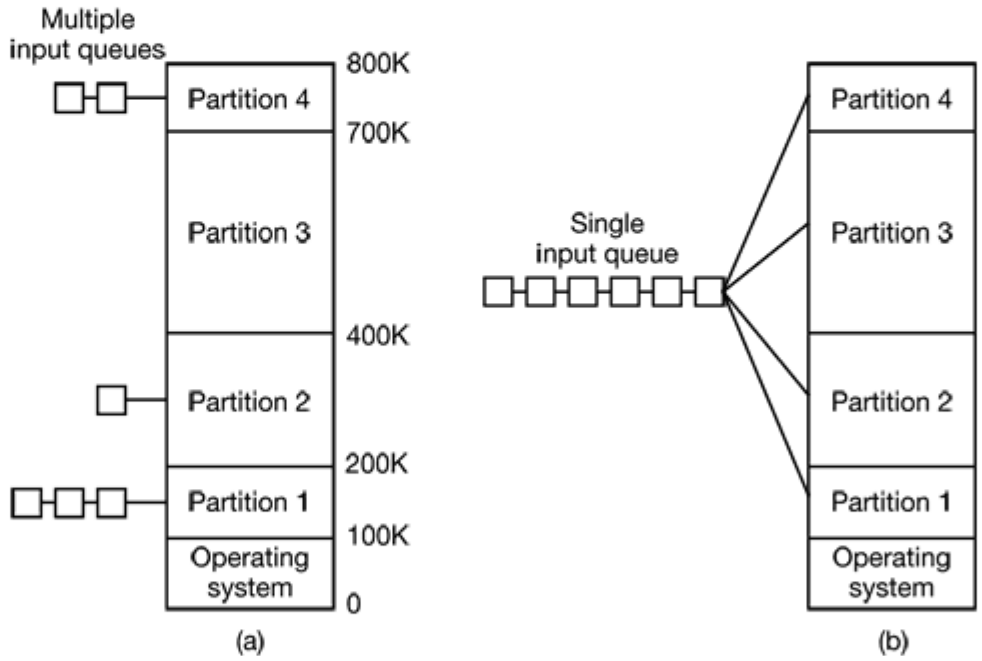
- Bu şekildeki sistemde aynı anda tek bir proses çalışır. Kullanıcı bir programı çalıştırdığında, programa ait diskteki veriler belleğe kopyalanır. Program sonlandığında işletim sistemi bir uyarı mesajı yayınlar ve yerine yeni bir prosesin başlatılmasını bekler. Yeni bir proses başlatılınca ise bu proses ait veriler bellekteki mevcut bitmiş prosesin üzerine yazılarak belleğe yüklenir.



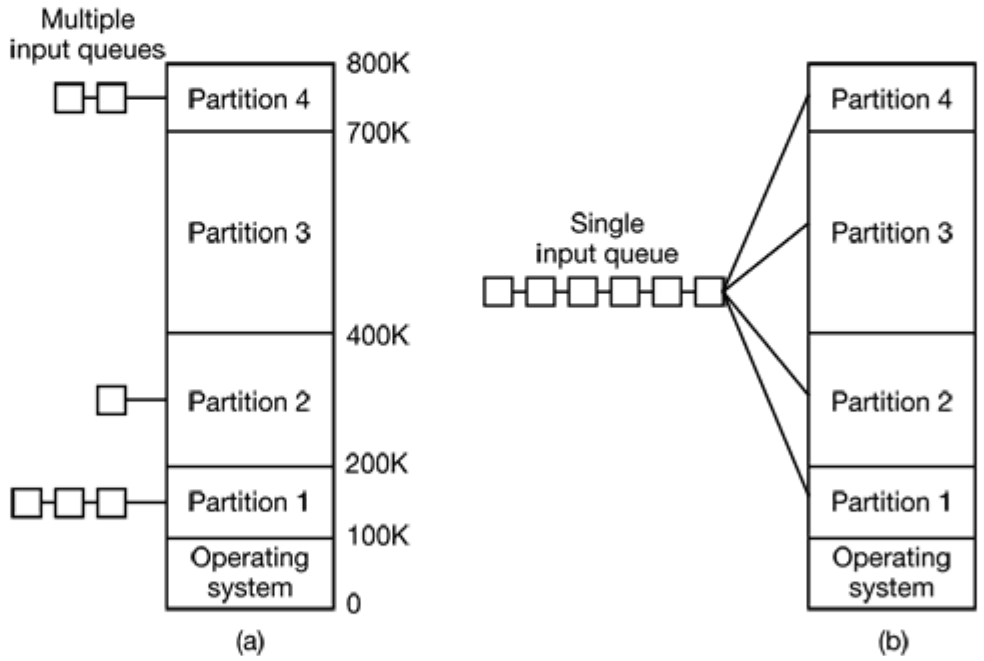
a) modeli mainframe sistemlerde ve minicomputer lerde kullanıldı.
b) modeli gömülü(embedded) sistemlerde kullanılır. Palm
c) modeli ilk kişisel bilgisayarlarda kullanılmıştır, sistemin ROM daki kısmına BIOS denilir.

1.2.SABİT BÖLÜMLÜ (PARTİTİONS) MULTİPROGRAMMİNG

- Tekli programlama sadece gömülü sistemlerde kullanılır. Günümüzdeki modern işletim sistemleri aynı anda birden fazla prosesin çalıştırılmasına olanak sağlar.
- Bellek n farklı bölüme (tercihen farklı boyutlu) ayrılarak multiprogramming sağlanır.
- Yeni bir proses başlatıldığında, kendi boyutuna uygun en küçük boyutlu bölümün kuyruğuna girer. Bölüm boyutları sabittir ve bölümün kullanılmayan alanı başka prosesler tarafından kullanılamaz.



- Şekil a’da farklı giriş kuyruklarına sahip bölümler görülmektedir. Bu kullanımın dezavantajı, küçük bölümlerin kuyrukları dolu iken büyük bölümlerin kuyruklarında proseslerin olmamasıdır. Şekilde bu durum bölüm 1 ve bölüm 3 üzerinde gösterilmiştir. Bellekte Bölüm 3 alanında büyük bir boşluk var iken Bölüm 1’ e girmek için bekleyen birçok prosesler vardır.

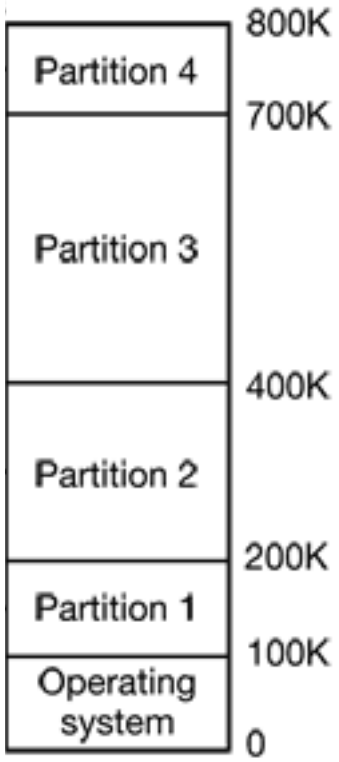


- Şekil b'de ise alternatif bir kullanım gösterilmiş ve bölümler için ayrık kuyruk oluşturma yerine belleğe tek bir kuyruk oluşturma yoluna gidilmiştir. Bir bölüm boşaldığında kuyruktaki ilk boyutu uygun prosese bölüm tahsis edilmektedir.
- Bu yöntem OS/360 da kullanılmıştır.

(RELOCATION) VE KORUMA(PROTECTION)

- Multiprogramming'de yer değiştirme ve koruma birer problemdir. Farklı prosesler bellekte farklı adreslerde çalıştırılırlar. Programın bağlama işlemi(link) yapıldığında, bağlayıcının programdaki kullanıcı prosedürlerinin, kütüphane prosedürlerinin, ana programın adreslerini bilmesi gerekmektedir. Program kodlarını bu adreslere göre düzenleyebilmesi için programın bellekte hangi adrese konulduğunu bilmesi gereklidir.

- Örneğin, bağlayıcının oluşturduğu ikili(binary) programın ilk komutu 100 numaralı bellekteki bir prosedürü çalıştırsın. Eğer bu program Partion 1'e yüklenmiş ise program 100 numaralı (işletim sistemine ait) mutlak adrese gidecek ve doğru çalışmayacaktır. Aslında gitmesi gereken adres 100K+100 dür. Eğer program Partition 2'ye yüklemiş ise, gidilmesi gereken adres 200K + 100'dür. Bu durum yer değiştirme problemi olarak bilinir.
- Bir programın bellekteki herhangi bir alana yazmasını ya da bir alanı okumasını engelleyen bir mekanizma yoktur. Çok kullanıcılı ve çoklu programlamalı (multiprogramming) sistemlerde bir prosesin diğer proseslere ait alanlara erişimi istenmeyen bir durumdur ve bu durum koruma problemi olarak adlandırılır.

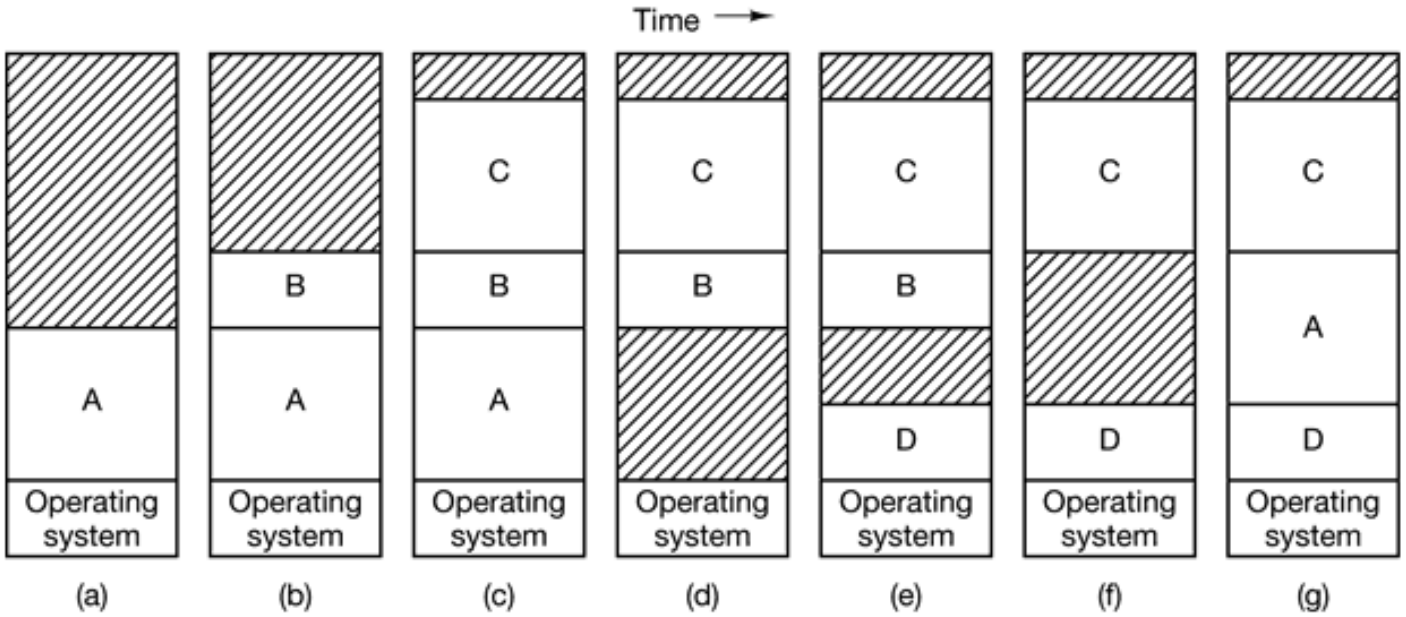


- Hem yer değiştirme hem de koruma probleminin çözümüne yönelik geliştirilen çözüm: bilgisayarlara **base** ve **limit** adı verilen iki yazmaç eklenmesidir. Bir proses işlemciye anahtarlandığı anda base yazmacına prosesin bulunduğu partion'ın başlangıç adresi atanır. Limit yazmacına ise ilgili partion'ın uzunluğu atanır. Yani base yazmacının değeri 100K olarak atanmış ise 100 numaralı adrese yapılan çağrı 100K+100 olarak değiştirilir. Oluşan yeni adres limit yazmacındaki değer ile partion dışını taşıp taşmadığı konusunda kontrol edilir. Böyle proseslere ait bölümlerin korunması sağlanmış olur.

2. SWAPPING (PROSES DEĞİŞİMİ)

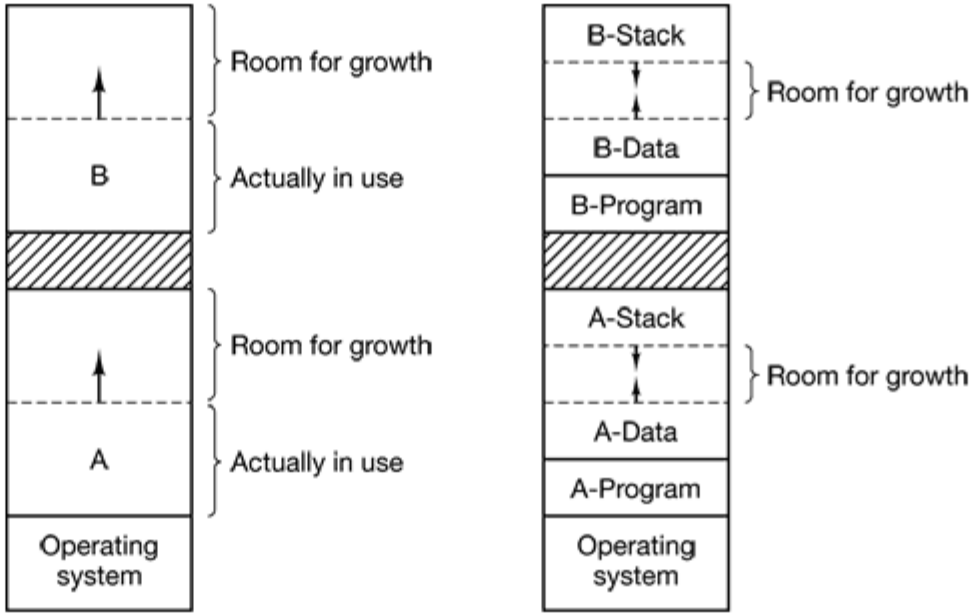
- Batch sistemlerde sabit bölümlü bellek yapısı kullanımı mümkündür. Bir proses belleğe girdiği zaman görevini sonlandırıncaya kadar bellekte kalır.
- Fakat zaman paylaşımlı ve grafik arayüzüne sahip kişisle bilgisayarlarda durum farklıdır. Bazen bellekte aktif çalışan bütün prosesler için yeterli yer olmayabilir. Bu durumda bellekte yer bulamayan proseslerin disk üzerinde tutulması ve zamanı gelince belleğe dinamik olarak yüklemesi gerekir.
- Burada bellek yönetimi ile ilgili iki genel yaklaşım kullanılır: **Takaslama** (**Swapping) ve Sanal Bellek (Virtual Memory).**

- Başlangıçta sadece A prosesi belleğe yüklemiştir. Daha sonra sırası ile *B* ve *C* prosesleri diskten belleğe takaslanmıştır. Sonra A prosesi bellekten diskte takaslanmış ve yerine D prosesi girmiştir. B prosesi sonlandıktan sonra tekrar A prosesi belleğe takaslanmıştır.



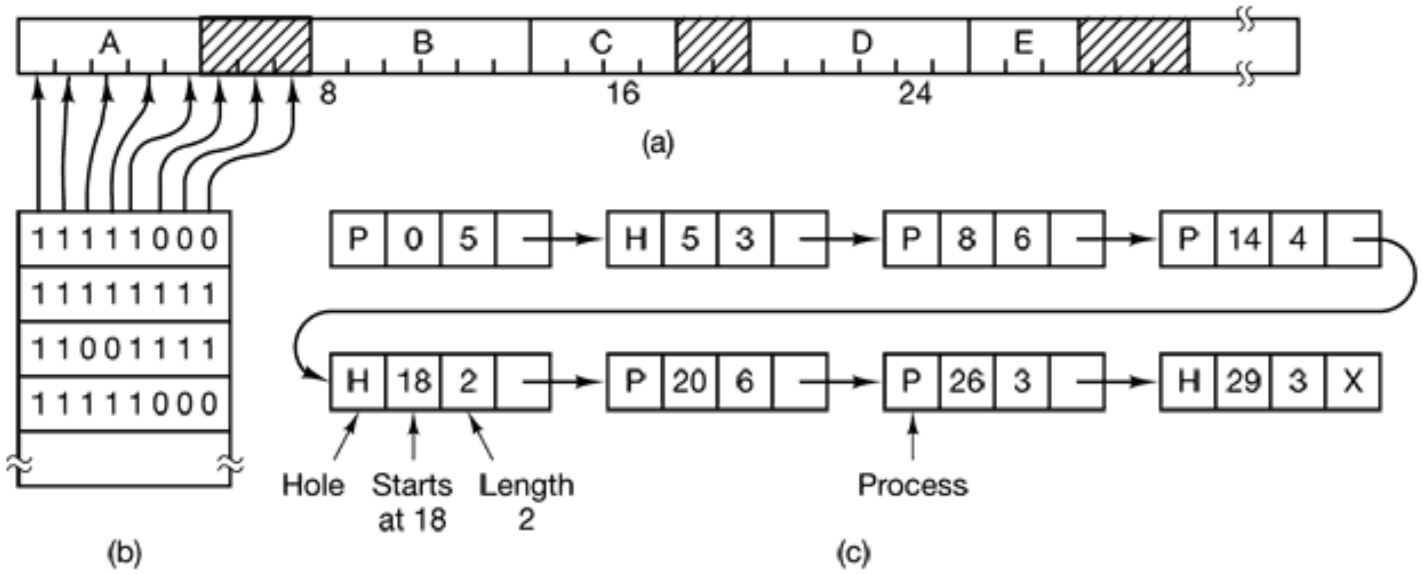
- Takaslamada bellekteki bölümlerin sayısı, boyutları ve konumları zamanla değişmektedir. Halbuki bu veriler sabit bölümlü bellek yönetiminde sabitti. Belleğin sabit bölümlere ayrılmaması ile daha esnek bir bellek kullanımı gerçekleştirilebiliyor ve bellek daha verimli kullanılmış oluyor. Bölümlerin boyutları sabit olmadığı için bellek bazen büyük, bazen de küçük birçok parçaya ayrılabilmektedir. Bu durumda belleğin boş-dolu kısımlarını izlemek bellek yönetimi açısından daha karmaşık hale gelmektedir.

- Birçok prosesin boyutu çalışma zamanında genişlemektedir. Proseslere bellek tahsisi yapılırken ihtiyaç duydukları alanlardan daha fazla alan tahsisi yapılır. Böylece proseslere genişleme alanı bırakılmış olur. Böyle proseslere ekstra alanlar tahsis edilmese ve genişlemeye çalışan her bir prosesin bellekten diske takaslanması gerekir ve bellek yönetimini daha da zorlaştırır. Ayrıca çalışma zamanında bellekten diskte bir proses takaslandığında belleğe ait ekstra genişleme alanları değil prosesin orijinal kısmı diske gönderilir.



2.1. BİTMAP İLE BELLEK YÖNETİMİ

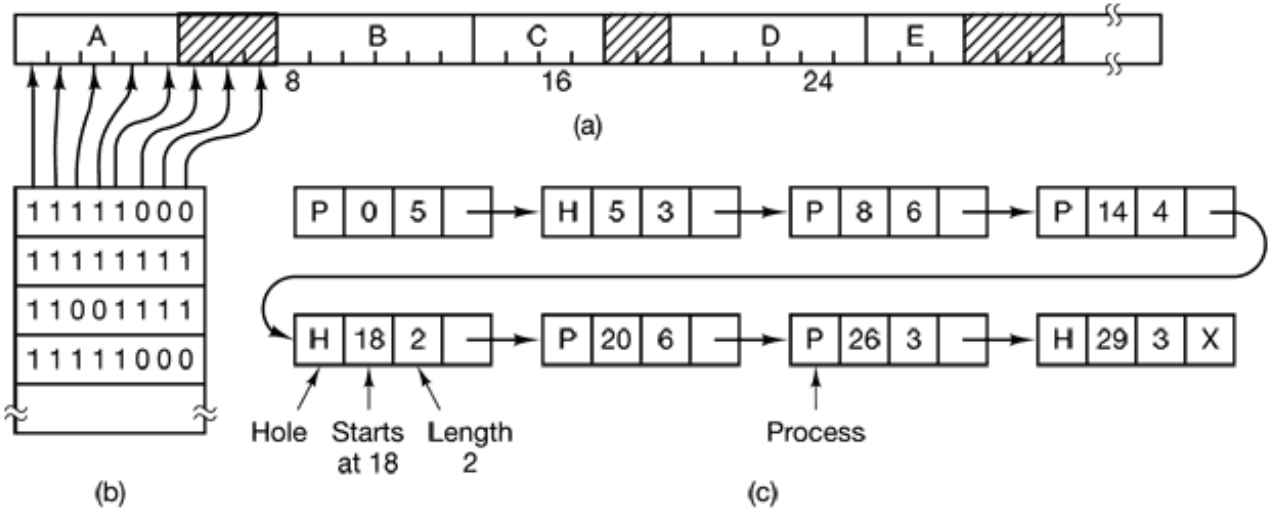
- Bellek dinamik olarak tahsis ediliyorsa, işletim sistemi bu işi yönetmelidir. Genel olarak bellek yönetimi için iki yapı kullanılır:
  1. Bitmap
  2. Listeler
- Bitmap yapısı kullanıldığında bellek boyutları birbirine eşit küçük birimlere ayrılır. Bu birimlerin boyutları birkaç byte olabileceği gibi birkaç KB da olabilir. Bitmap de her bir bellek birimi bir bit ile temsil edilir. Eğer bu bir 1 ise bellekteki ilgili birim dolu, 0 ise boş anlamına gelir.



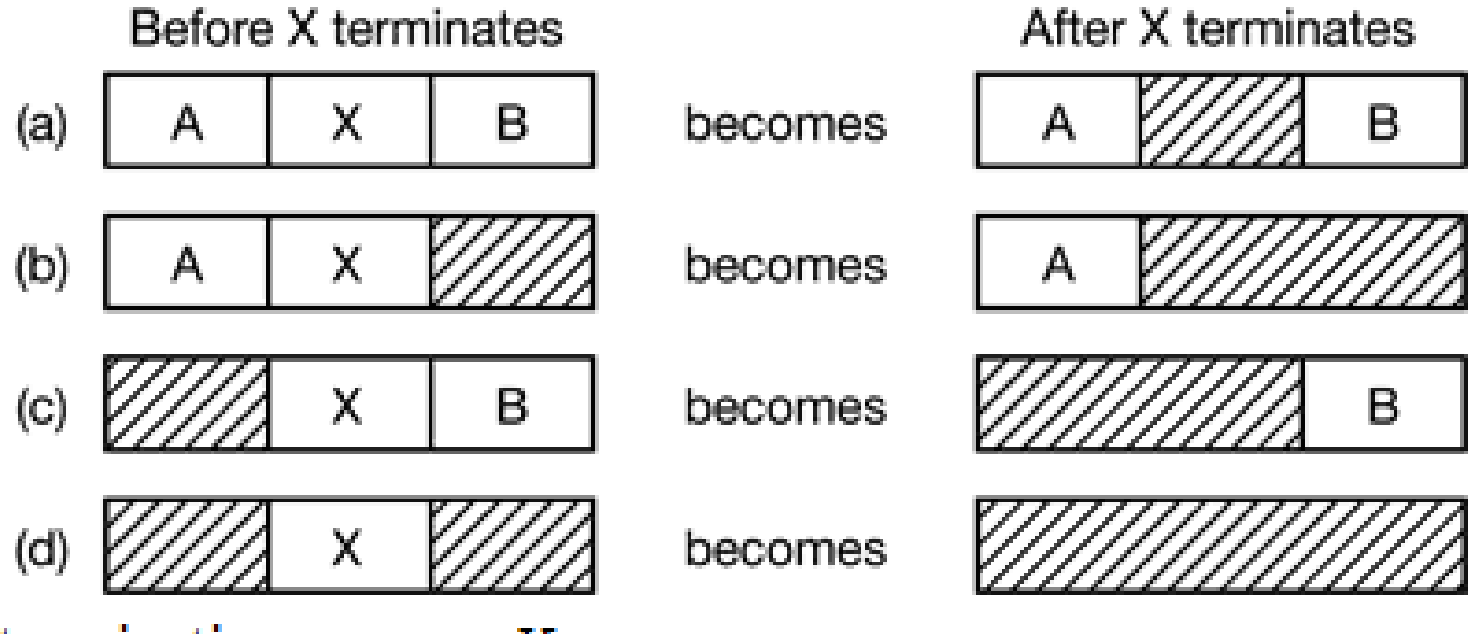
- Şekil a'da 5 prosesin (A,B,C,D,E) ve 3 boşluğun bulunduğu bir bellek kesiti verilmiştir. Taralı alanlar boşlukları ifade etmektedir. Şekil b'de ise resimdeki bellek kesitine ait bitmap verilmiştir.
- Belleğin parçalara ayrıldığı bellek birimlerinin boyutu işletim sistemlerinde bir tasarım problemidir.

2.2 BAĞLI LİSTELER İLE BELLEK YÖNETİMİ

- Bellek yönetiminde kullanılan diğer bir yağı ise bağlı listelerdir. Liste her biri boşluğu ya da bir prosesi temsil eden düğümlerden oluşmaktadır. Düğümlerdeki ilk alan P ise düğüm bir prosesi, H ise boşluğu temsil etmektedir. İkinci alan düğümün başlangıç noktasını, üçüncü alanı düğümün uzunluğunu ve son alanı ise bir sonraki düğümü işaret eden bir işaretçiyi tutar



- Aşağıdaki örnekte, birbirine komşu düğümlerden X prosesi sonlandığında bellekteki liste yapısında oluşan farklı kombinasyonlar gösterilmiştir



- Yeni bir proses oluşturulduğunda ya da mevcut proses diskten Ram’e yükleneceği sırada bellekte yer tahsisinde kullanılan çeşitli algoritmalar vardır:

1. İlk algoritma ilk uygun yer algoritmasıdır (**first fit)**. Bellek yöneticisi yer talebinde bulunan proses için gerekli olan alanı belleğin başından itibaren tarama yaparak arar. İlk bulduğu yeterli boşluğu bu belleğe tahsis eder. Eğer bulunan boşluk belleğin ihtiyaç duyduğundan daha büyük ise bu düğüm iki parçaya ayrılır. Oluşturulan yeni düğümlerden birincisine ilgili proses yerleştirilir, ikinci düğüm ise boşluk olarak kalır. First fit algoritması genel olarak hızlı bir algoritmadır.

2. İlk uygun yer algoritması benzer olan diğer bir yöntem ise **sonraki uygun yer (next fit)** algoritmasıdır. İlk sefer çalıştığında first fit ile aynı şekilde çalışır. Bellek talebinde bulunan proses ilk uygun boşluğa yerleştirilir ve taranan en son nokta kaydedilir. Next fit, first fit'den farklı olarak yeni bir proses için alan tahsisi yapılacağı zaman bu kalınan noktadan itibaren bellekte boş yere taraması yapar. First fit ise her seferinde baştan belleği taramaya başlar. Fakat ikinci ve daha sonraki proses
algoritma en son bulduğu uygun yer bilgisini saklar.
Bir sonraki aramada bağlı listenin başından değil,
saklamış olduğu düğümden sona doğru aramaya başlar.
Yapılan simülasyonlarda Firts Fit'in Next Fit'den biraz daha iyi performans gösterdiği görülmüştür.
3. Diğer bir algoritma ise **en uygun yer (best fit)** algoritmasıdır. Best fit her bir bellek tahsisinde bütün belleği baştan sona tarar ve talepte bulunan prosesin ihtiyacına en uygun boşluğu bulmaya çalışır. Böylece bellekteki boşlukları çok parçalamadan daha verimli bir kullanımı hedefler. Fakat bu algoritmalar için en yavaş çalışan algoritma next fit'dir.

3. SANAL BELLEK

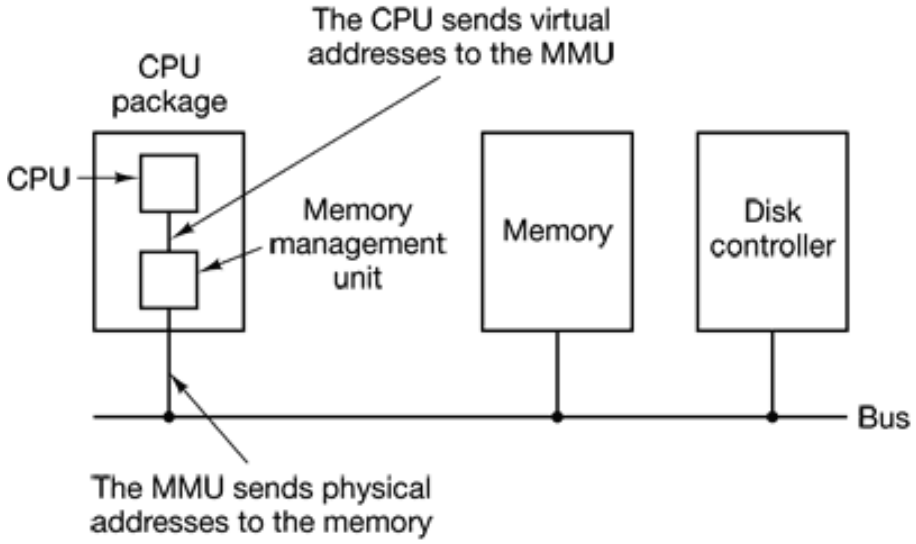
- Bilgisayarlarda kullanılan programların boyutlarının belleğin boyutunu aşması ile sanal bellek kullanımına başlanmıştır. İşletim sistemi programın aktif olarak çalışan kısımlarını bellekte tutarken diğer kısmını ise diskte tutmaktadır. Örneğin 16 MB'lik bir programı 4MB'lık Ram'e sahip bir bilgisayarda çalıştırmak istediğimizde prosesin anlık olarak ihtiyaç duyduğu 4MB'lık kısmı bellekte diğer 12MB'lık kısmı diskte tutulur. Çalışan proses programın diskteki bulunan kısmana ihtiyaç duyduğunda Ram'deki kısım diske gönderilip, ihtiyaç duyulan kısım ise Ram'e getirilir.
- Multiprogramming sistemlerde de sanal bellek kullanılabilir. Bir proses diskten verinin getirilmesini beklerken bloklanır ve işlemci başka bir proses anahtarlanır. Bloklanan prosese ait veri diskten Ram'e transfer edildiğinde proses hazır duruma geçer. Virtual memory can also work in a multiprogramming system.

3.1. PAGİNG

- Sanal bellek sistemleri genellikle *sayfalama (paging)* adı verilen bir teknik kullanır. Bilgisayarda her bellek bölgesinin bir adresi vardır. Programda şu şekilde bir komut (instruction) çalıştırdığında:
  MOV REG,1000

  Bu komut 1000. adresteki veriyi REG yazmacına kopyalar (ya da tersi).

- Program tarafından oluşturulan adreslere **sanal adres (virtual addresss)** denir. Sanal bellek kullanıldığında, sanal adresler direk bellek yolunda kullanılmaz. Bunun yerine Bellek Yönetim Birimi (Memory Management Unit) kullanılır. Bellek Yönetim Birimi programdan gelen sanal adresi fiziksel adrese (bellek üzerindeki adrese) dönüştürür.

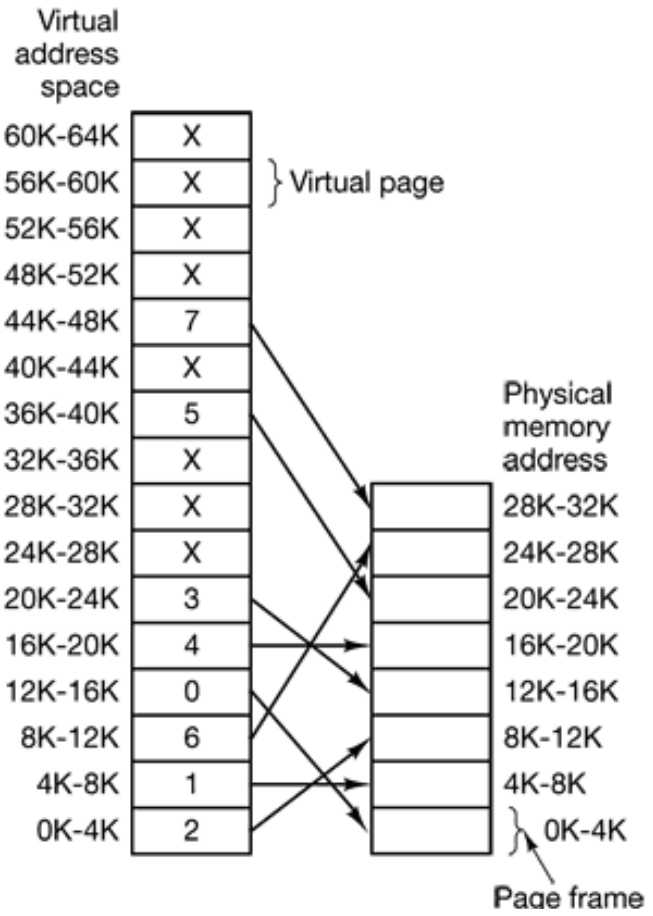


- Örneğin 16-bitlik adres kullanan bir program 0’dan 64K’ya kadar adresleme yapabilir. Bu alan (0’dan 64K’ya) sanal adres alanıdır. Aynı zamanda 64K’lık bu program 32 KB’lik bir belleğe sahip bir bilgisayarda çalıştırılmak istensin. Dolayısı ile programın tamamı belleğe sığamayacaktır. Bu durumda programın tamamı diskte bulunacak ve ihtiyaç duyulan kısımlar belleğe transfer edilecektir.
- Sanal adres alanı adı sayfa olan birimlere ayrılmıştır. Bellekte sayfalar karşılık gelen alanlara sayfa çerçevesi **(page frame)** adı verilmektedir. Sayfa ve sayfa çerçevelerinin boyutları eşit olmalıdır. Bu örneğimizde sayfa ve safya çerçevelerinin boyutu 4KB olsun. Dolayısı ise sanal bellek alanı 64/4=16 sayfadan, bellek ise 32/4=8 sayfa çerçevesinden oluşuyor. Ram ile disk arasındaki veri transferi her zaman sayfa boyutunda gerçekleşmektedir.

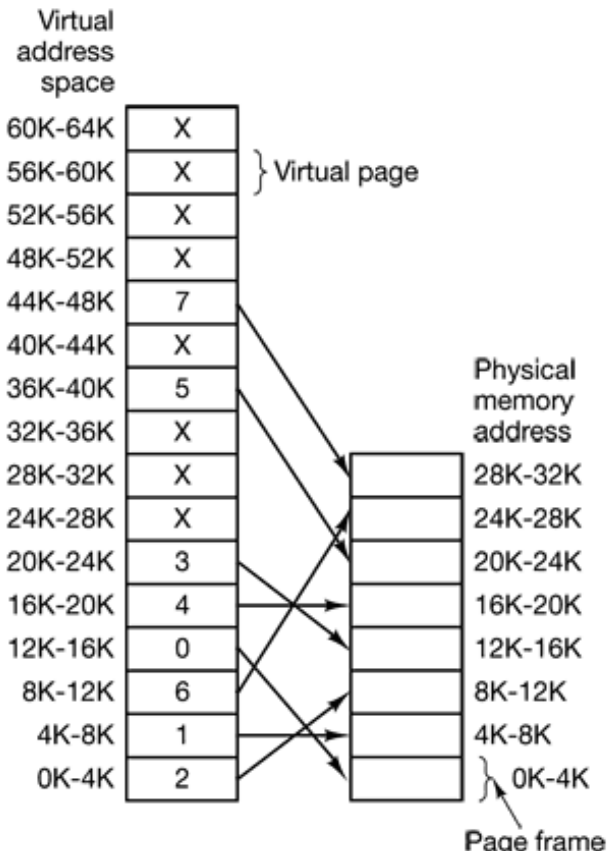
- Örneğin program 0. adrese ulaşmak istesin.

  MOV REG,0

  0 adresi bellek yönetim birimine gönderilir. Bellek yönetim birimi bu sanal adresin (0'dan 4095'e) bellek üzerinde 2. sayfa çerçevesine (8192 to 12287) haritalandığı görülmektedir. Böylece 0 numaralı adres 8192'ye dönüştürülmüş ve veri yoluna konulmuştur. Bellek yönetim biriminin yaptığı dönüşümden habersizdir ve sadece 8192 nolu adresten veri okunması/yazılması işlemini yürütür.

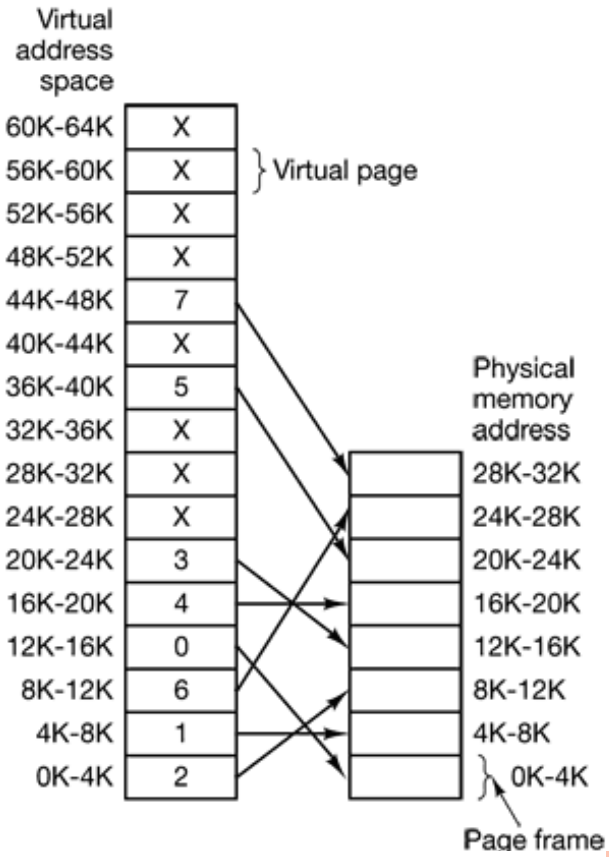


- 16 sanal sayfaya karşılık bellekte 8 sayfa çerçevesi olduğu için programa ait sayfalardan 8 tanesi belleğe yüklenebilir. Şekilde de görüldüğü gibi sanal bellek uzayında yanında çarpı işareti olanlar belleğe yüklenmeyenler, yanına numara yazanlar ise o numaralı sayfa çerçevesine yüklenen sayfalardır. Gerçek donanım üzerinse VAR/YOK biti ile hangi sayfaların bellekte olup olmadığı bilgisi tutulmaktadır.



- Eğer program şöyle bir komut icra ederse; MOV REG,32780

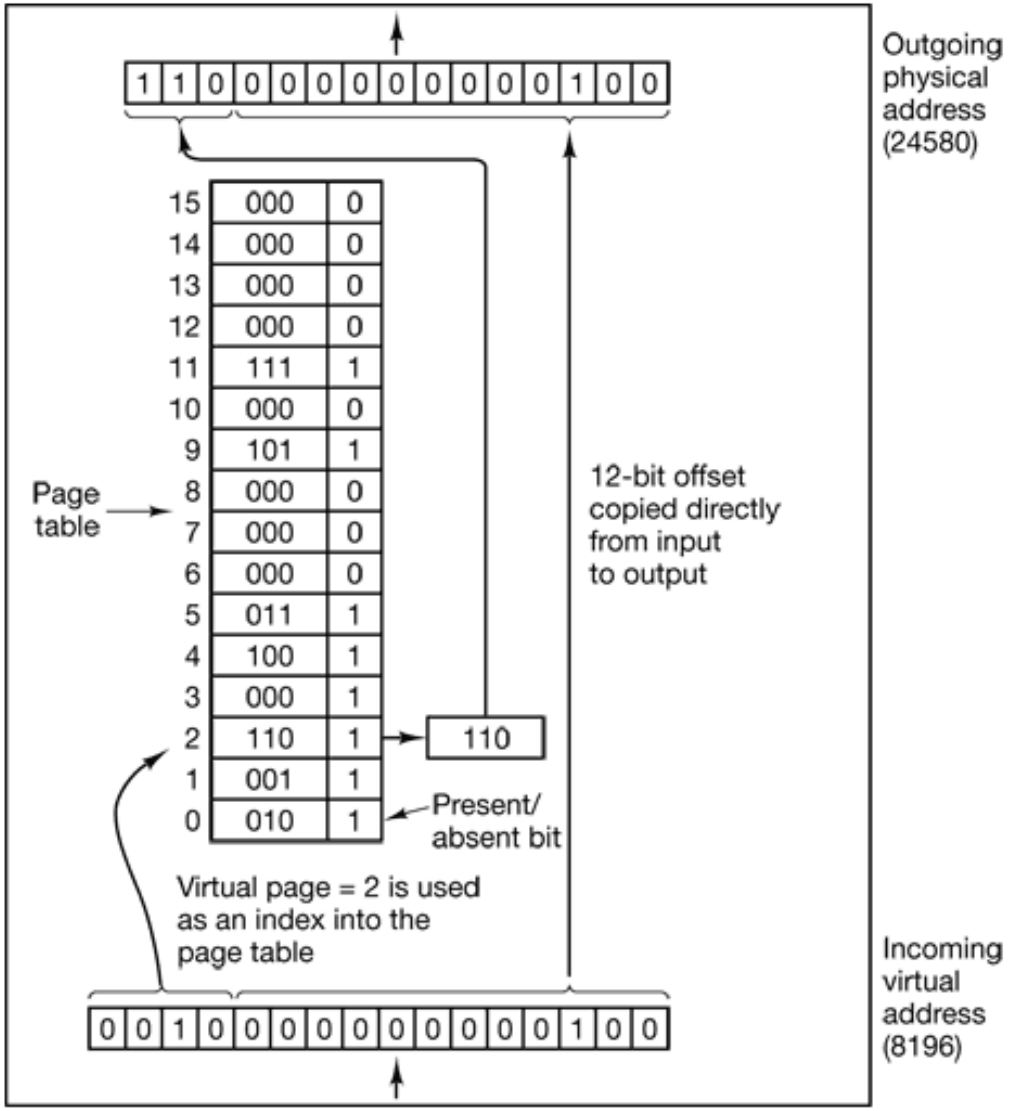
  8. sayfadaki (başlangıcı 32768 olan sayfa) 12. adresteki veriyi kopyalamak ister. Bellek yönetim birimi 8. sayfanın bellekte olmadığını fark eder ve işletim sistemine bir sayfa hatası gönderir. İşletim sistemi bellekteki en az kullanılan sayfalardan birini diske göndererek yer açar. Oluşan boşluğa 8. sayfayı yerleştirir ve sayfa haritasında gerekli güncellemeleri yapar.



SAYFA TABLOLARI (PAGE TABLES)

- Sanal adresin ilk sıradaki bitleri sanal sayfa numaralarını daha sonrakiler ise ofseti ifade eder. Örneğin 16 bitlik bir adres ve sayfa boyutunun 4KB olduğu bir sistemde, ilk 4 bit sanal bellekteki 16 sayfayı adresleme için kullanılırken, diğer 12 bit ise 4KB (0-4095)'lık sayfa içindeki ofset adresini adresler. Sayfa boyutları değiştikçe adresleme yapısı da değişir.

- Sanal sayfa (virtual page) numarası sayfa tablosundaki sayfaların indeksidir.
- Sayfa tabloları sanal sayfaların bellekteki sayfa çerçeveleri ile ilişkilendirmesini sağlarlar. Matematiksel olarak konuşursak; sayfa tablosu bir fonksiyon, sayfa numarası bir parametre ve bellekteki sayfa çerçevesi ise bu fonksiyonun sonucudur.

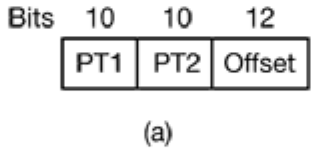


- Sayfa boyutu 4 KB olan ve 32 bitlik adres uzayında 1 milyon sayfa, 64 bitlik bir adres uzayında ise bu rakamdan çok daha büyük bir sayfa sayısı karşımıza çıkar. Sanal adres uzayında 1 milyon sayfa demek sayfa tablosunda 1 milyon kayıt olması anlamına gelir ve her bir prosesin kendi sayfa tablosunu tuttuğunu düşünürsek, bellekte sayfalara ulaşımın çok uzun zaman alacağını tahmin etmek hiç de zor olmaz.

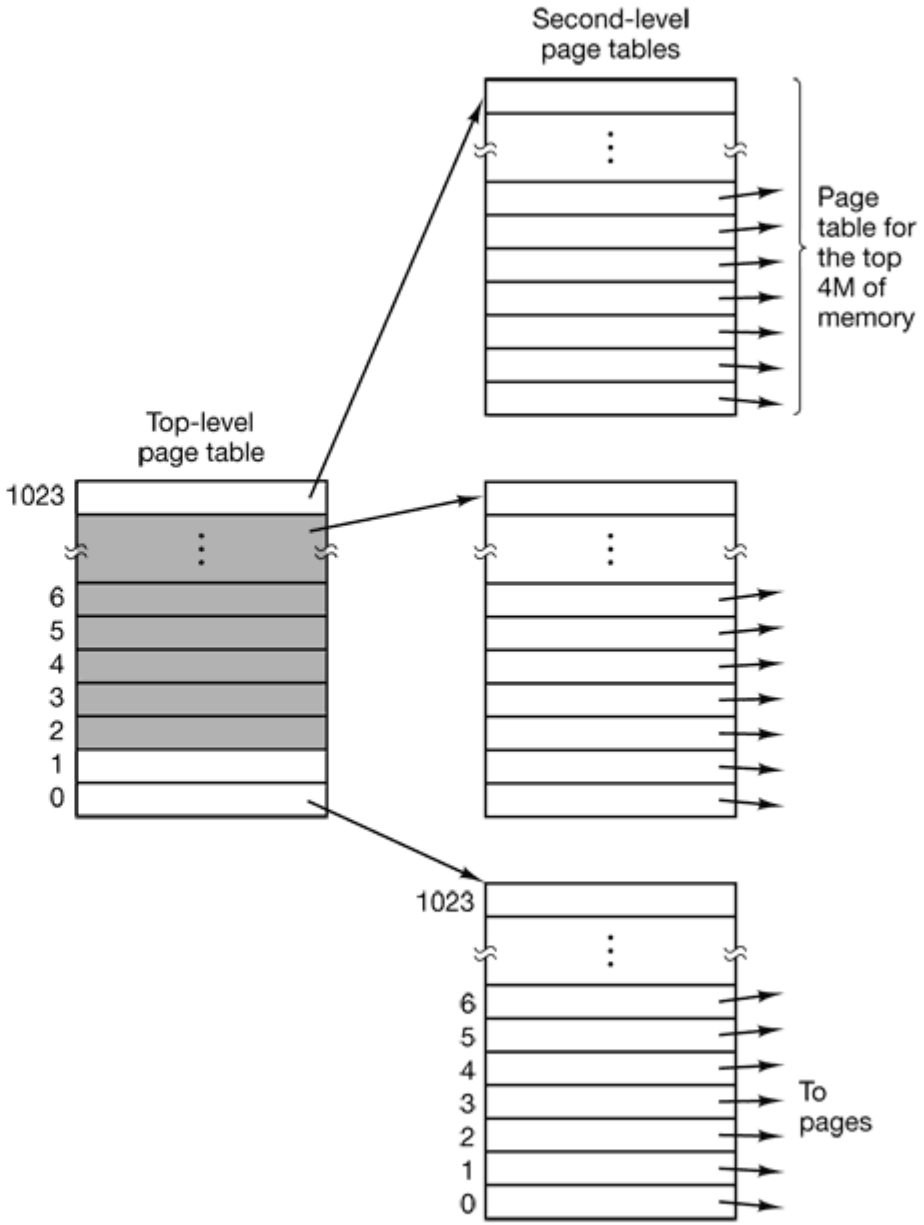
ÇOK KATMANLI SAYFA TABLOLARI

- Bütün sayfa tabloları bellekte tutulduğu ve sayfa tablolarının boyutu çok büyük olduğu için çoğu bilgisayar çok katmanı sayfa tablolarını kullanır. Basit bir örnek ile 4KB sayfa boyutuna sahip ve 32 bitlik sanal adres alanı olan bir bilgisayarda, adres uzayı 10 bitlik PT1 alanı, 10 bitlik PT2 alanı ve 12 bitlik offset alanı olmak üzere üç alana ayrılır. 12 bitlik bir offset alanı olduğu için 2^{20} adet sanal sayfa vardır.

- Bu mimarinin önemi bir prosese ait bütün sayfa tablolarını hepsini aynı anda bellekte tutmamasıdır.

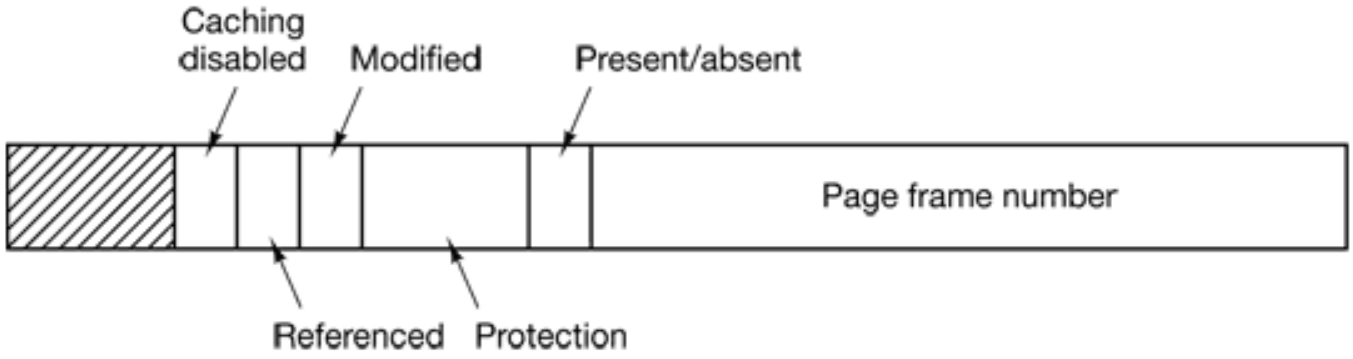


- The two-level page table system can be expanded to three, four, or more levels. Additional levels give more flexibility, but it is doubtful that the additional complexity is worth it beyond three levels.

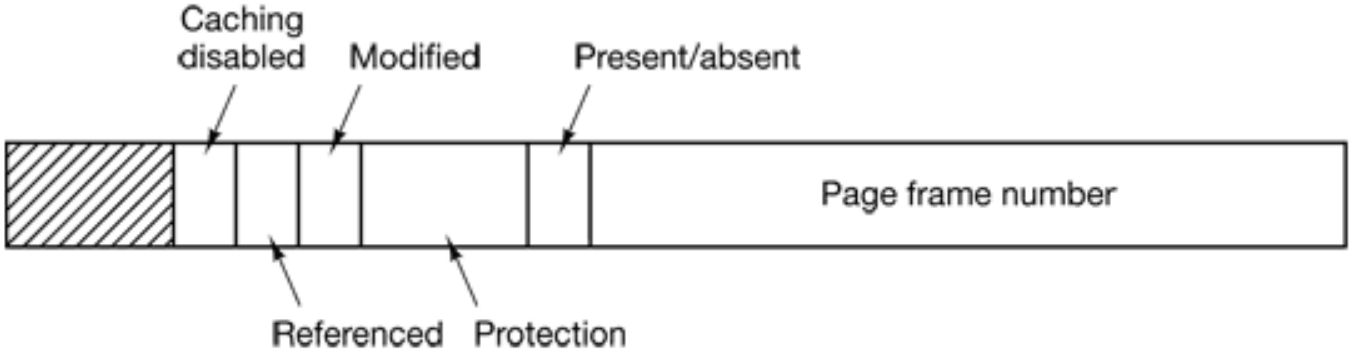


SAYFA TABLOLARINDAKİ KAYITLARIN YAPISI

- Boyutu bilgisayardan bilgisayara farklılık gösterebilir fakat genellikle 32 bit'dir ve kabaca içerisindeki bilgiler aynıdır. En önemli alanı sayfa çerçeve numarasıdır. Sağdan sola doğru devam edersek VAR/YOK (Present/Absent) biti 1 ise aranan sayfa bellektedir. Fakat bu değer 0 ise bu sayfaya ulaşılmaya çalışılınca sayfa hatası verilir çünkü sayfa bellekte yoktur.



- Sıradaki koruma (*Protection)* biti en basit hali ile tek bir bitten oluşur ve bu bitin 1 olması yalnız okunabilir, 0 olması ise hem yazılıp hem de okunabilir bir sayfa olması anlamına gelir.
- *Modified* ve *Referenced* bitleri sayfa kullanımlarını takip ederler. Eğer sayfaya bişey yazılmış ise Modified biti 1 yapılır. Bu durumda sayfa dirty sayfa olur ve sayfada yapılan değişikliğin disk üzerindeki orijinal sayfaya da yansıtılması gerekir.
- Bir sayfa kullanılmaya başladığı anda *Referenced* biti 1 yapılır.
- Sayfa tablolarında sayfaların diskteki adresleri tutulmaz.



TLBS—TRANSLATİON LOOKASİDE BUFFERS

- **TLB** sanal adresten fiziksel adrese dönüşümü sayfa tablolarını kullanmadan yapan küçük bir donanımdır.
- Sayfa tablolarının küçük bir parçası ağırlıklı okunmakta, kalan kısmı nadiren okunmaktadır

Valid	Virtual page	Modified	Protection	Page frame
1	140	1	RW	31
1	20	0	R X	38
1	130	1	RW	29
1	129	1	RW	62
1	19	0	R X	50
1	21	0	R X	45
1	860	1	RW	14
1	861	1	RW	75

- Bellek Yönetim Birimine sanal bir adres geldiğinde, TLB'de aranan sayfa ile ilgili bir eşleşme varmı öncelikle ona bakılır. Eğer aranan kayıt TLB içinde var ise ve koruma bitinde de herhangi bir engel yok ise, sayfa tablolarına ihtiyaç duyulmadan ilgili sayfa çerçevesinin adresi TLB'den alınır.
- Eğer aranan sayfa TLB'de bulunamaz ise, Bellek Yönetim Birimi bu sefer sayfa tablolarını kullanmak zorunda kalır. İlgili sayfa belleğe yüklenince bu sayfanın adresi TBL'ye eklenir.

SAYFA DEĞİŞTİRME ALGORİTMALARI

- Sistemde bir sayfa hatası (page fault) oluştuğunda, işletim sistemi bellekten bir sayfayı kaldırmak için seçmelidir. Seçilen ve bellekten çıkarılan sayfa yerine, yeni sayfa gelecektir. Eğer bellekten kaldırılacak olan sayfa bellekte iken değiştirilmiş ise, diskteki kopyanın güncel olması için tekrar bellekten diske yazılmalıdır. Değiştirilmemiş ise diske yapılmasına gerek yoktur.

- Her sayfa hatasında yoğun olarak kullanılmayan sayfanın çıkarılması, sistemin performansını arttırır. Yoğun kullanılan bir sayfa çıkarılırsa çok kısa bir süre sonra bu sayfa tekrar istenebilir.

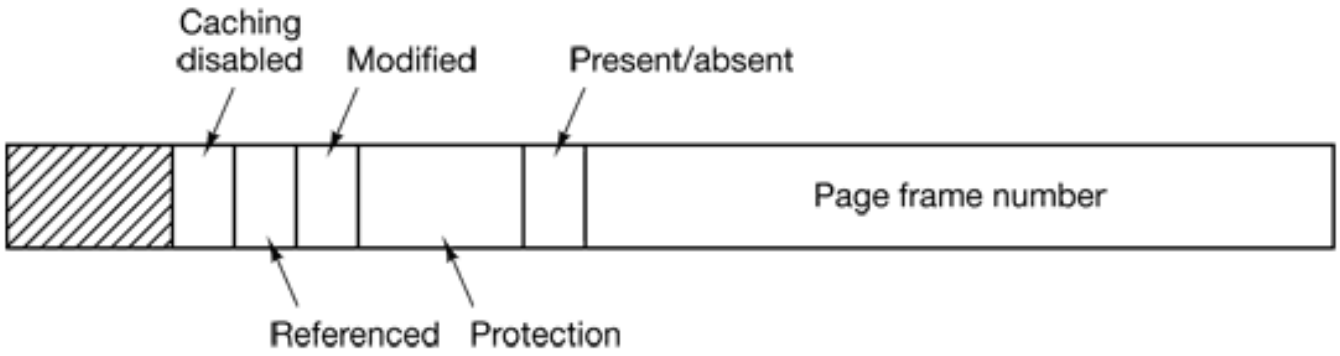
- Sayfa yer değiştirme algoritmaları bilgisayarda farklı alanlarda kullanılmaktadır. (Örn: cache bellekte). Algoritma bellek için de önbellek için de aynı mantık ile çalışır. Tek farklılık bellekte bu işlem süre olarak milisaniyeler alırken, önbellekte nanosaniyeler alır.

OPTİMAL SAYFA DEĞİŞTİRME ALGORİTMASI

- En iyi sayfa yer değiştirme algoritması kolaylıkla tanımlanır fakat uygulanamaz. Algoritma şöyle çalışır: Sayfa hatasının görüldüğü anda, bellekte bir grup sayfa bulunmaktadır. Bu sayfalardan bir tanesine bir sonraki komutta(instruction) başvurulacaktır. Bu komutu içermeyen sayfalara belki de 10,100,1000…komut sonrasına kadar erişim gerçekleşmeyecektir.
  Bu algoritmada her sayfa kendisi kullanılıncaya kadar kaç komut geçecek ise o sayı ile etiketlenir.
- Sayfa hatası oluştuğunda bu sayfalardan etiket numarası en yüksek olan sayfa en geç kullanılacak sayfa olduğu için diske gönderilmek üzere seçilir.
- Bu algoritmanın uygulanabilirliğinin olmamasının nedeni: işletim sisteminin sayfa hatası olduğunda sayfalara kaç komut sonra ulaşılacağını bilememesidir.

SON ZAMANLARDA KULLANILMAYAN SAYFANIN DEĞİŞTİRİLMESİ ALGORİTMASI

- İşletim sisteminin hangi sayfaların kullanılıp kullanılmadığı belirlemek açısından, her sayfa için 2 bit kullanılmaktadır. R (Referenced) kullanılıyor, M(Modified)değiştirildi bitleri. R biti bu sayfaya erişildiği zaman 1 lenir. M de bu sayfaya yazma gerçekleştiği zaman 1 lenir.



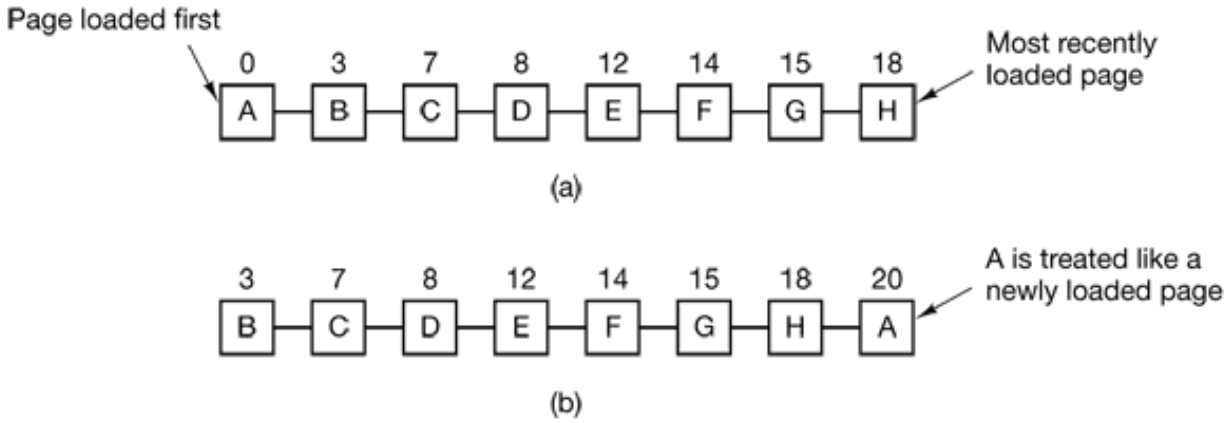
- Bir sayfa hatası meydana geldiğinde, işletim sistemi
  sayfaların R ve M bitlerine göre sayfaları dört gruba ayırır:
  - Sınıf 0: Referans edilmemiş, Değiştirilmemiş = 0 0
  - Sınıf 1: Referans edilmemiş, Değiştirilmiş= 0 1
  - Sınıf 2: Referans edilmiş, Değiştirilmemiş= 1 0
  - Sınıf 3: Referans edilmiş, Değiştirilmiş = 1 1
- Bu algoritması boş olmayan en düşük numaralı sınıflardan bir sayfayı rastgele seçer.
- Bu algoritmaya göre değiştirilmiş fakat referans edilmemiş bir sayfa referans edilmiş (kullanımda olan) fakat değiştirilmemiş sayfaya göre daha önce bellekten çıkarılır.

THE FİRST-IN, FİRST-OUT (FIFO) SAYFA DEĞİŞTİRME ALGORİTMASI

- İşletim sistemi bellekteki sayfaları geliş sırasına göre bağlı listede tutar. Listenin başında ilk gelen sayfa, sonunda da en son gelen sayfa bulunur. Bir sayfa hatası oluştuğunda her zaman listenin başındaki sayfa çıkarılır ve yeni gelen sayfa sona eklenir. FIFO en çok kullanılan sayfaları da bellekten gönderdiği için çok kullanılan bir algoritma değildir.

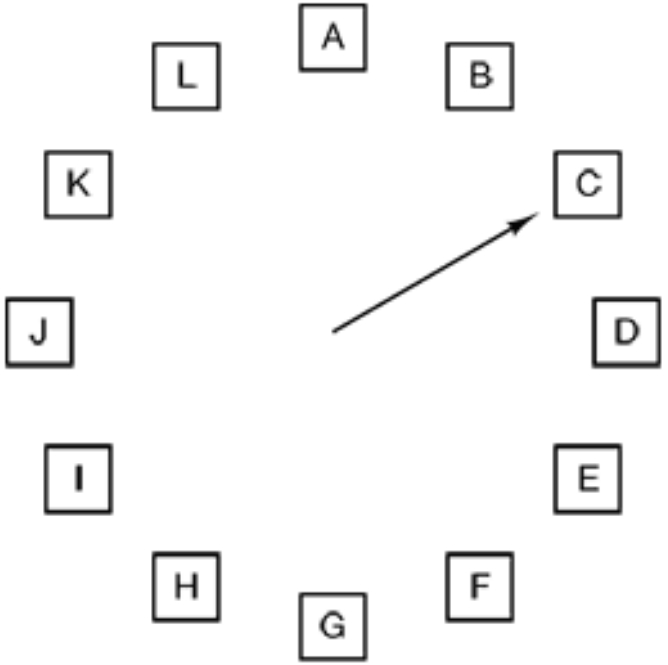
İKİNCİ ŞANS SAYFA DEĞİŞTİRME ALGORİTMASI

- FIFO üzerinde başı değişiklikler ile bu algoritma elde edilmiştir. Sayfa hatası oluştuğunda eski (listenin başındaki) sayfaların R bitlerinin kontrolü ile yoğun kullanılan bir sayfanın çıkartılması engellenir. Eğer R=0 ise bu sayfa hem eski hem de kullanılmayan bir sayfa olduğu için çıkartılabilir. Eğer R biti 1 ise bu bit 0'lanır ve bu sayfa listenin sonuna eklenir ve yüklenme zamanı güncellenir.



SAAT SAYFA DEĞİŞTİRME ALGORİTMASI

- İkinci şans algoritması makul bir algoritma olmasına rağmen, sayfaların liste üzerinde devamlı yer değiştirmesi işlemleri sisteme ek yük getirmektedir.Daha iyi bir yaklaşım, tüm sayfaları bir saat biçimli dairesel listede tutmaktır. Saatin kolu en eski sayfayı işaret eder. Sayfa hatası olduğunda saat kolunun gösterdiği sayfa araştırılır. R ye göre bir eylem



When a page fault occurs, the page the hand is pointing to is inspected. The action taken depends on the R bit:
R = 0: Evict the page
R = 1: Clear R and advance hand

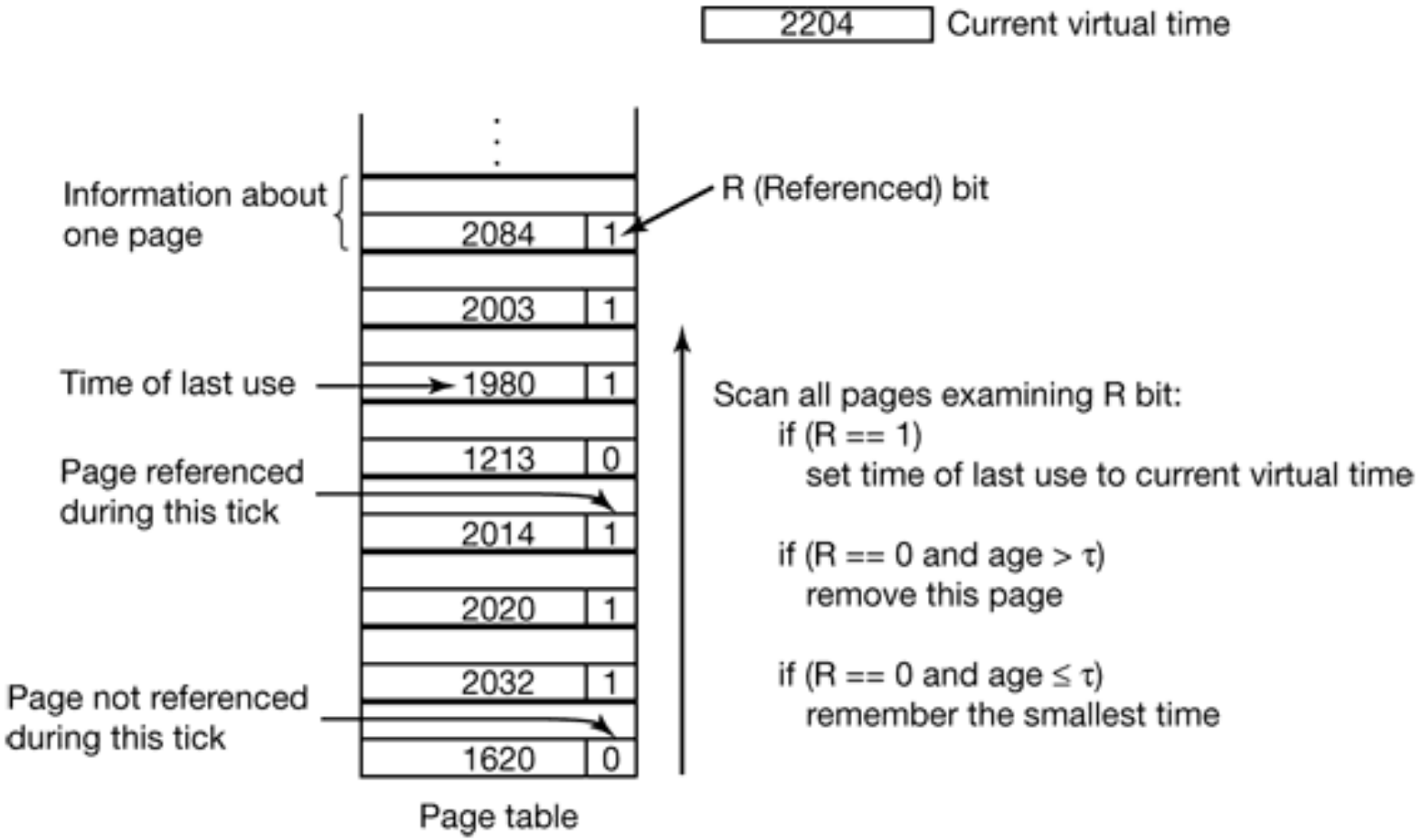
EN YAKIN ZAMANDA KULLANILAN SAYFA DEĞİŞTİRME ALGORİTMASI

- Optimal algoritmaya benzer bir algoritmadır. Ana fikri son zamanlarda kullanılan sayfalar büyük olasılıkla sonraki birkaç komutta da kullanılacaktır. Tam tersi kullanılmayan sayfalar da uzun bir zaman daha kullanılmayacaktır. Bir sayfa hatası olduğunda en uzun zamandır kullanılmayan sayfa diske gönderilir.

- Bu metodun donanımsal çözümü için donanımda 64 bitlik C sayacı (counter) kullanılır. Her komuttan sonra kullanılan sayfanın C sayacı bir arttırılsın. Bunun için sayfa tablosundaki her kayıtta ilgili sayfanın sayaç değerini tutan 64 bitlik alanları olsun. Bir sayfa kullanıldığında bu sayaç değeri 1 arttırılsın.
- Bir sayfa hatası görüldüğünde, işletim sistemi sayfa tablosundaki tüm sayaçları tarar ve en küçük olanı bulur. Bu sayfa en uzun zamandır erişilmeyen sayfadır ve gönderilecek sayfa olarak seçilir.

ÇALIŞMA KÜMESİ (WORKİNG SET) SAYFA DEĞİŞTİRME ALGORİTMASI

- Bir prosesin çalışması esnasında anlık olarak ihtiyaç duyduğu sayfalara o prosesin **çalışma kümesi** adı verilir. Eğer bir proses çalışırken bu çalışma kümesi bellekte olursa, ilgili proses bir sonraki aşamaya geçinceye kadar sayfa hatası almayacaktır.
- Bu sebeple birçok sayfalama sistemi proseslerin çalışma kümesindeki sayfaları proses çalışmadan hemen önce belleğe yüklemeyi hedefler. Böylece sayfa hataları minimum düzeye indirilmeye çalıştırılır. Ayrıca proseslerin çalışma kümeleri zamanla değişir. Çalışma kümesini belirlemek için değişik yöntemler geliştirilmiştir.


2204
Current virtual time
Information about one page
R (Referenced) bit
2084
1
2003
1
Time of last use
1980
1
1213
0
Page referenced during this tick
2014
1
2020
1
2032
1
Page not referenced during this tick
1620
0
Page table
Scan all pages examining R bit:
if (R == 1)
set time of last use to current virtual time
if (R == 0 and age > τ)
remove this page
if (R == 0 and age ≤ τ)
remember the smallest time

Sorular?