

Giriş

- Modern bilgisayar sistemlerinin çalışmasında **hafıza merkezi role sahiptir.**
- **Hafıza**, her birisi kendi adresine sahip olan **çok sayıda byte alanına sahiptir.**
- **CPU**, **program counter (PC)** değerine göre **hafızadan bir komutu fetch eder.**
- Hafızadan alınan komutlar, **bir veya birden fazla parametre için hafıza erişimi (operand)** gerektirebilirler.
- Komutun çalıştırılmasından sonra **elde edilen sonuç hafızaya tekrar yazılabilir.**

3

Giriş

Temel donanım yapısı

- **Main memory ve general purpose register'lar**, CPU tarafından adreslenen **genel amaçlı kayıt alanlarıdır.**
- Makine komutlarında **hafıza adresini parametre (operand) olarak alan komutlar vardır**, ancak **disk adresini alan komutlar yoktur.**
- **CPU'nun ihtiyaç duyduğu veri veya komut hafızada değilse, öncelikle hafızaya alınmalıdır.**
- CPU içerisindeki **register'lara genellikle bir cycle ile erişilebilmektedir.**
- **Hafızaya erişim** bus üzerinden yapılır ve **register'a göre oldukça uzun süre gerektirir.**
- Hafıza ile CPU arasına çok daha **hızlı ve CPU'ya yakın** bir saklama alanı oluşturulur (**cache**).

4

Giriş

Temel donanım yapısı

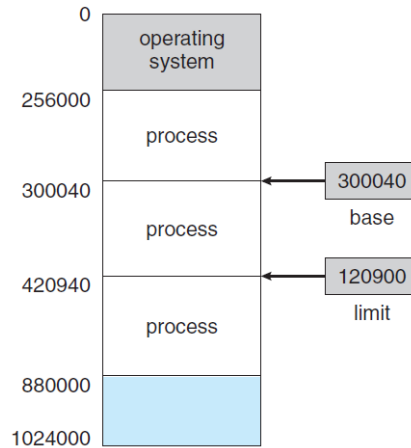
- Diğer kullanıcı process'lerin bir process'e ayrılan alana erişiminin engellenmesi gereklidir.
- Çok kullanıcıli sistemlerde **bir kullanıcı process'ine başka kullanıcının erişiminin de engellenmesi gereklidir.**
- **Bu tür koruma işleri donanımsal düzeyde yapılır.** İşletim sistemi düzeyinde yapıldığında performans düşer.
- **Her process kendisine ait ayrı bir hafıza alanına sahiptir.**
- Böylelikle, process'ler birbirinden ayrılmış olur ve birden fazla process eşzamanlı çalıştırılabilir.

5

Giriş

Temel donanım yapısı

- Bir process için ayrılan alanın **başlangıç adresi (base register)** ve **boyutu (limit register)** belirlenmelidir.

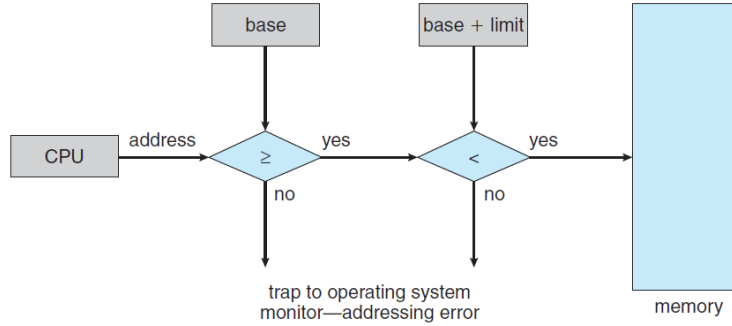


6

Giriş

Temel donanım yapısı

- Hafıza alanının korunması donanımla gerçekleştirilebilir.
- Kullanıcı modunda istek yapılan her hafıza adresinin **base** ile **base+limit** aralığında olduğu kontrol edilir.



- İstenen adres **process için ayrılan alanın dışında ise hata üretilir.**

Giriş

Adres binding

- Bir program disk üzerinde binary dosya olarak bulunur.
- Bir programın çalıştırılabilmesi için hafızaya alınması gereklidir.
- Bir process disk üzerinden hafızaya alınmak için kuyruğa alınır (input queue).
- Bir process hafızaya yerleştikten sonra komutları çalıştırır veya hafızadaki veri üzerinde işlem yapar.
- Process'in çalışması tamamlandığında kullandığı hafıza alanı boşaltılır.
- Kullanıcı programı çalışmadan önce ve çalışması süresince farklı aşamalardan ve/veya durumlardan geçer.

Giriş

Adres binding

- **Kaynak programda adres genellikle semboliktir (count).**
- **Compiler** bu adresleri yeniden yerleştirilebilir (**relocatable**) adreslere dönüştürür (**Örn.: program başlangıcından itibaren 14.byte**).
- Linkage editör veya loader, bu adresleri mutlak (absolute) adreslere dönüştürür (**Örn.: 74014**).

9

Giriş

Adres binding

- Komutların veya verilerin hafıza adreslerine bağlanması (binding) farklı şekillerde olabilir:
 - **Compile time:** Compile aşamasında kodun hafızada yerleşeceği yer bilinirse **mutlak code (absolute code)** oluşturulabilir. Yerleşeceği hafıza alanı değişirse yeniden compile edilmesi gereklidir. (MS-DOS işletim sistemi .com programlarını bu şekilde çalıştırır.)
 - **Load time:** Compile aşamasında programın yerleşeceği yer bilinmiyorsa, derleyici yeniden yerleştirilebilir (**relocatable code**) kod oluşturur.
 - **Execution time:** Eğer program çalışması sırasında bir segment'ten başka bir segment'e geçerse, adres binding **run-time**'da yapılır.

10

Giriş

Mantıksal ve fiziksel adres alanı

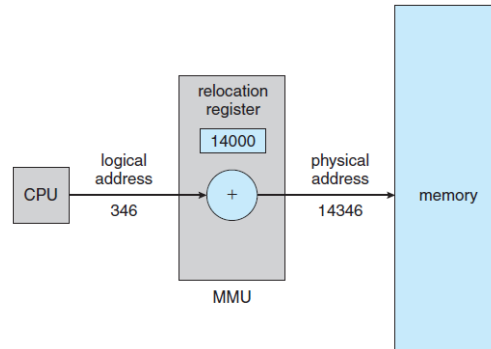
- CPU tarafından oluşturulan adres **mantıksal adres (logical address)** olarak adlandırılır.
- Hafıza biriminin gördüğü adres **fiziksel adres (physical address)** olarak adlandırılır.
- **Compile-time** veya **load-time** adres binding işlemleri **mantıksal** veya **fiziksel** adres üretir.
- **Execution-time** adres binding ise **sanal adrestir (virtual address)** (**page number + offset**).
- Run-time'da sanal adresin fiziksel adrese dönüştürülmesi **donanım bileşeni (memory-management-unit, MMU)** tarafından yapılır.
- **Base-register** fiziksel adrese dönüştürme için kullanılan **donanım bileşenidir**.

11

Giriş

Mantıksal ve fiziksel adres alanı

- **Base-register** (relocation register) değeri, **CPU'nun hesapladığı adrese** eklenir.



- Kullanıcı programı hiçbir zaman fiziksel adresi bilmez.
- Mantıksal adres **[0, 0+max]** aralığında, fiziksel adres **[R, R+max]** aralığındadır.

12

Giriş

Dynamic loading

- Bir programın tamamı hafızaya yüklenmez gerektiğinde modül (blok) halinde yüklenir (dynamic loading).
- **Önce main program hafızaya yüklenir ve çalıştırılır.**
- Bir program parçası (routine) çalışırken başka rutin'i çağırdığında, hafızada yüklü değilse loader tarafından yüklenir.
- **Çok büyük boyuttaki programların çalıştırılması için hafıza yönetimi açısından fayda sağlar.**
- Sık kullanılmayan rutin'lerin (hata yordamları) hafızada sürekli bulunmasını engeller.

13

Giriş

Dynamic linking ve paylaşılan kütüphaneler

- **Dinamik bağlanan kütüphaneler** sistem kütüphaneleridir (dil kütüphanesi) ve **kullanıcı programına çalışırken bağlanır.**
- Bazı işletim sistemleri statik bağlamayı destekler ve binary programa loader tarafından bağlanır.
- Her kütüphane için küçük bir **kod parçası (stub)** yükleneceği uygun hafıza alanını gösterir.
- Tüm programlar aynı kütüphaneyi kullanır.
- **Kütüphanelerde yapılacak güncellemeler tüm kullanıcı programlarına kolaylıkla yansıtılır.**

14

Konular

- Giriş
- **Swapping**
- Bitişik hafıza atama
- Segmentation
- Paging

15

Swapping

- Bir process çalışmak için hafızada olmak zorundadır.
- Bir process geçici olarak **diske (backing store)** aktarılabilir ve tekrar hafızaya alınabilir (**swapping**).
- **Ready queue** (hazır kuyruğu), CPU'da çalıştırılmak üzere bekleyen process'leri tutar.
- **CPU scheduler** bir process'i çalıştırmaya karar verdiğinde dispatcher'ı çağırır.
- Dispatcher, çalışacak process'in hazır kuyruğunda olup olmadığını kontrol eder ve kuyrukta ise çalıştırır.
- Kuyrukta değilse ve **hafızada yeterli yer yoksa başka bir process'i hafızadan atar (swap out) ve istenen process'i yükler (swap in)**.

16

Swapping

- İki process'in yer değiştirmesi context-switch işlemini gerektirir ve uzun süre alır.
- 100 MB'lık bir process'in 50MB/sn hızındaki bir diske kaydedilmesi için 2sn gerekir. İki process'in yer değiştirmesi 4sn süre alır.
- Process'lerin **dinamik hafıza gereksinimleri** için **request_memory()** ve **release_memory()** sistem çağrıları kullanılır.
- Bir process'in **swap out** yapılabilmesi için **tüm işlemlerini bitirmesi zorunludur**.
- Bir process I/O kuyruğunda bekliyorsa veya başka bir işlem sonucunu bekliyorsa swap out yapılamaz.
- Modern işletim sistemleri **hafıza eşik değerinin altına düşmeden swapping yapmaz**.

17

Swapping

Mobil sistemlerde swapping

- Mobil sistemler swapping işlemini desteklemez.
- Mobil cihazlar kalıcı saklama birimi olarak **hard disk yerine flash bellek kullanır**.
- Flash belleklerde yazma sayısı limiti vardır.
- Mobil cihazlarda, **main memory ile flash bellek arasındaki throughput değeri düşüktür**.
- Apple **iOS** işletim sistemi, **uygulamalardan hafızayı boşaltmasını ister**.
- Read-only veri sistemden atılır ve sonra flash bellekten tekrar yüklenir.
- Değişebilen veriler (stack) hafızadan atılmaz.
- **Android** işletim sistemi, **yeterli hafıza alanı yoksa bir process'i sonlandırır ve durum bilgisini flash belleğe kaydeder**.

18

Konular

- Giriş
- Swapping
- Bitişik hafıza atama
- Segmentation
- Paging

19

Bitişik hafıza atama

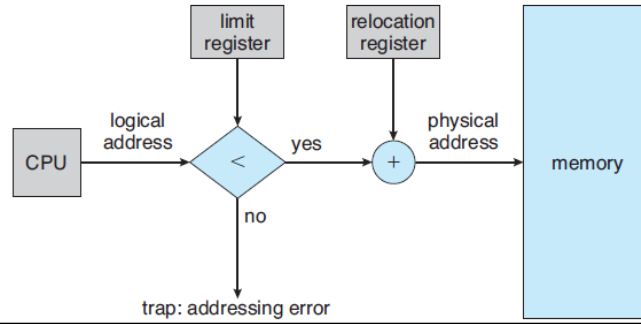
- **Main memory** hem **işletim sistemini** hem de **kullanıcı programlarını** **yerleştirmek zorundadır.**
- Bitişik hafıza atama yönteminde **hafıza iki parçaya ayrılır:**
 - İşletim sisteminin yerleştiği kısım
 - Kullanıcı process'lerinin yerleştiği kısım
- İşletim sistemi **hafızanın başlangıcına** veya **sonuna yerleşebilir.**
- Interrupt vector table düşük adrese veya yüksek adrese yerleştirilebilir.
- İşletim sistemi ile interrupt vector tablosu genellikle aynı tarafa yerleştirilir.
- **Bitişik hafıza atama yönteminde bir process için ayrılan alan tek bölümden oluşur** ve sonraki process için ayrılan yere kadar devam edebilir.

20

Bitişik hafıza atama

Hafıza alanı koruma

- Bir process'in sahip olmadığı hafıza alanına erişimini engellemek gerekir.
- Sistemde **relocation register** ve **limit register** ile her process'e kendisine **ait hafıza alanı ayrılabilir**.
- CPU tarafından istenen her adresin [**base, base + limit**] arasında olup olmadığı kontrol edilir.



21

Bitişik hafıza atama

Hafıza alanı atama

- Bir process'e **hafıza alanı atama** işletim sistemine göre **farklı şekillerde yapılabilir**.
- Hafıza çok sayıda **sabit boyutta** küçük **parçaya ayrılabilir** (**fixed-sized partitions**) ve **her parça bir process'i içerebilir** (**multiple partition**).
- Multiprogramming sistemlerde **eşzamanlı çalışan program sayısı partition sayısına bağlıdır**.
- Bir **partition boşaldığında**, hazır kuyruğunda **bekleyen bir process seçilerek partition atanır**.
- IBM OS/360 işletim sistemi kullanmıştır günümüzde kullanılmamaktadır.
- Değişken parçalı (**variable-partition**) yönteminde **işletim sistemi hafızanın boş ve dolu olan parçalarını bir tabloda tutar**.
- Bu yöntemde, **her process'e farklı boyutta parça ayrılabilir**.

22

Bitişik hafıza atama

Hafıza alanı atama

- Bir process sisteme girdiğinde **ihtiyaç duyacağı kadar hafıza alanı ayrılabilirse hafızaya yüklenir** ve CPU'yu beklemeye başlar.
- Bir **process sonlandığında** ise ayrılan **hafıza alanı serbest bırakılır**.
- Herhangi bir anda, işletim sisteminde **kullanılabilir hafıza blokları listesi** ile **process'lerin giriş kuyruğu kümeleri** vardır.
- Kuyruğun başındaki process için **kullanılabilir yeterli alan yoksa beklenir** veya **kuyruktaki process'ler taranarak boş alana uygun olan varsa seçilir**.
- **Hafızaki boş alanların birleştirilmesi, process'e uygun alanın oluşturulması, serbest bırakılan alanların birleştirilmesi** işlemleri **dynamic storage allocation problem** olarak adlandırılır.

23

Bitişik hafıza atama

Hafıza alanı atama

- Dynamic storage allocation problemi için **3 farklı çözüm kullanılabilir**:
 - **First fit**: Yeterli boyuttaki **ilk boş alan atanır** ve listede kalan kısım aranmaz.
 - **Best fit**: Yeterli boyutta alanların **en küçüğü seçilir**. **Tüm liste aranır**.
 - **Worst fit**: Yeterli boyuttaki alanların **en büyüğü seçilir**. **Tüm liste aranır**.
- Simülasyonlarda, **alan atama süresinin first fit ile, hafıza alanı kullanma verimliliğinin best fit ile daha iyi olduğu** görülmüştür.
- **First fit** yöntemi best fit ve worst fit'e göre **daha kısa sürede atama gerçekleştirmektedir**.

24

Bitişik hafıza atama

Fragmentation

- Process'ler hafızaya yüklenirken ve atılırken **hafıza alanları sürekli parçalanır (fragmentation)**.
- Bir process için yeterli alan olabilir, ancak bunlar **küçük parçalar halinde dağılmış durumda olabilir**.
- **En kötü durumda her iki process arasında boş kısım olabilir**.
- First fit ile yapılan **istatistiksel analize göre, N tane kullanılmış blok için N/2 tane boş blok oluşur**.
- Bu durumda hafızanın 1/3 kısmı kullanılamaz. **Buna %50 kuralı (50-percent rule) denir**.
- Fragmentation çözümünde küçük bloklar yer değiştirilerek büyük blok elde edilir (**fazla süre gerektirir**).
- **Segmentation** ve **paging** yaklaşımları fragmentation çözümünde etkindir.

25

Konular

- Giriş
- Swapping
- Bitişik hafıza atama
- **Segmentation**
- Paging

26

Segmentation

- **Segmentation** yaklaşımda, her segment bir isme ve uzunluğa sahiptir.
- Bir **mantıksal adres**, **segment adı ile offset** (segment içerisindeki konumu) **değerini belirler**.

<segment number (ad), offset>

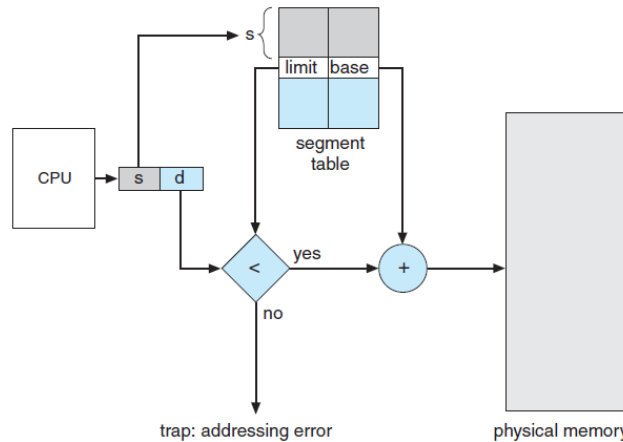
- Bir C derleyicisi aşağıdakiler için ayrı ayrı segment oluşturabilir:
 - Program kodu
 - Global değişkenler
 - Heap (nesneler yerleştirilir)
 - Stack (thread'ler kullanır, lokal değişkenler, call/return)
 - Standart C kütüphanesi
- Derleme sırasında **derleyici segment atamalarını gerçekleştirir**.

27

Segmentation

Segment adresleme donanımı

- Segment ve offset adresiyle **iki boyutlu adresleme yapılır**.
- **Hafıza adresleri tek boyutludur** ve dönüştürme işlemi gereklidir.

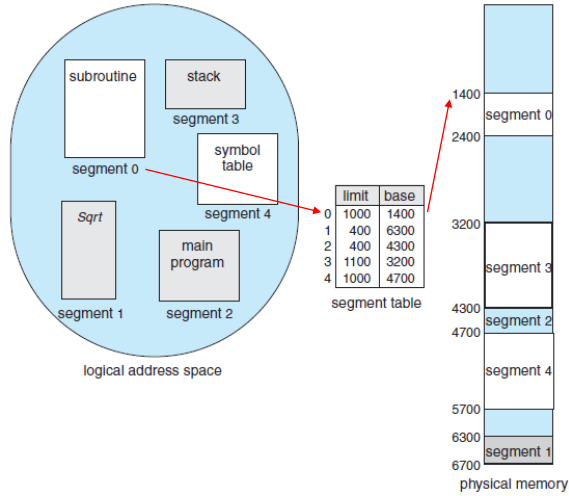


28

Segmentation

Segment adresleme donanımı

- Şekilde 5 segment vardır ve aşağıdaki gibi yerleştirilmiştir.



29

Konular

- Giriş
- Swapping
- Bitişik hafıza atama
- Segmentation
- **Paging**

30

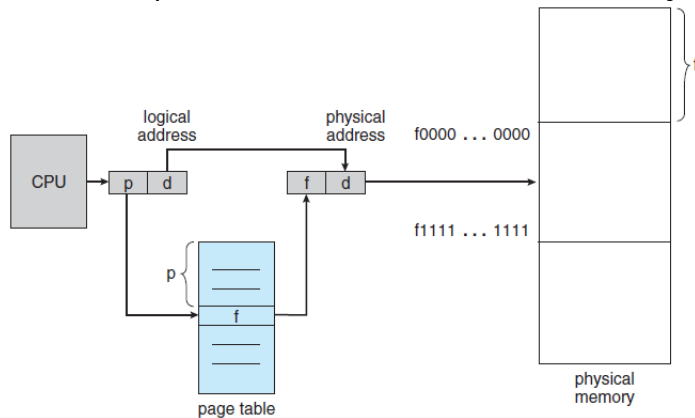
Paging

- Segmentation ile bir process'e atanan fiziksel adres alanının bitişik olmamasına izin verilir.
- **Paging** ile segmentation'da olduğu gibi process'lere bitişik olmayan hafıza adresleri atanabilir.
- Paging yönteminde,
 - Fiziksel hafıza **frame** adı verilen küçük bloklara bölünür.
 - Mantıksal hafıza ise aynı boyutta **page** adı verilen bloklara bölünür.
- Bir process çalıştırılacağı zaman **kaynak kodu diskten alınarak hafızadaki frame'lere yerleştirilir.**
- Mantıksal adres alanı ile fiziksel adres alanı birbirinden ayrıştırılmış durumdadır.

31

Paging

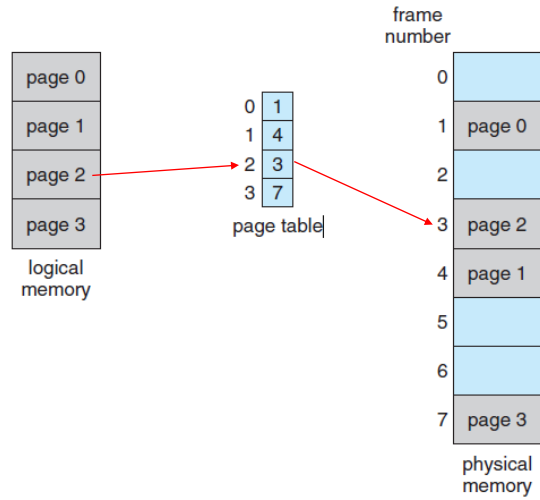
- CPU tarafından oluşturulan **her adres iki parçaya ayrılır: page number(p) ve page offset(d).**
- **Page number**, **page table** içerisindeki indeks değeri için kullanılır.
- **Page table**, her sayfanın fiziksel hafızadaki **base adresini** içerir.



32

Paging

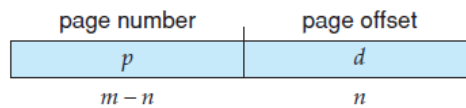
- Page size donanım tarafından mikroişlemci mimarisinde belirlenir.
- Page size 512 byte ile 1 GB arasında olabilir.



33

Paging

- Mantıksal adres boyutu 2^m , fiziksel adres (offset) boyutu 2^n byte ise, **sayfa numarası** için soldaki **m-n bit**, **offset değeri** için sağdaki **n bit** alınır.
- Aşağıda örnek mantıksal adres görülmektedir.



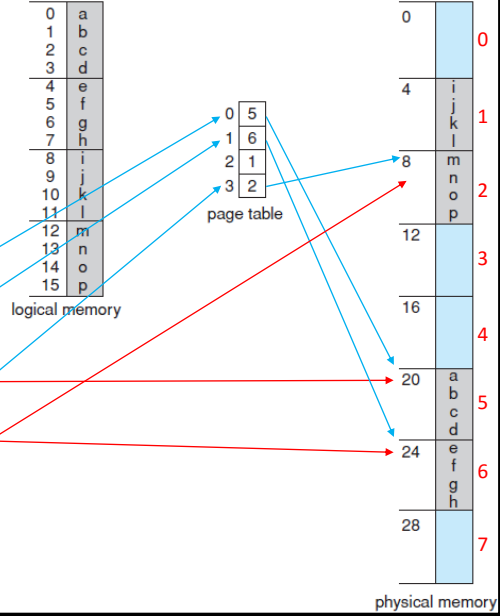
- p page table içindeki **indeks**, d ise sayfadaki **displacement** değeridir.

34

Paging

- Fiziksel hafızada **8 sayfa** var ve toplam 32 byte boyutundadır.
- **Page size = 4 byte**
- **Mantıksal adres boyutu m = 4 bit**
- **Offset adresi n = 2 bit**
- Mantıksal adres = 0 için **page = 0** ve **offset adres = 0** olur.
- Mantıksal adres = **0000**
- Mantıksal adres = **0100**

Mantıksal adres = 13 (**1101**)
Fiziksel adres = $(2 \times 4) + 1 = 9$



Paging

- **Paging ile fragmentation oluşabilir.**

Page size = 2048 byte

Process size = 72766 byte

Gerekli alan = 35 sayfa + **1086 byte**

1086 byte 36. sayfaya yerleştirilir.

Kullanılmayan alan 2048 – 1086 = 962 byte

- **36. sayfada 962 byte boş alan kalır.**
- En kötü durumda 1 byte kalır ve ayrı sayfaya yerleştirilir.
- Boş alan 2048 – 1 = 2047 byte olur.

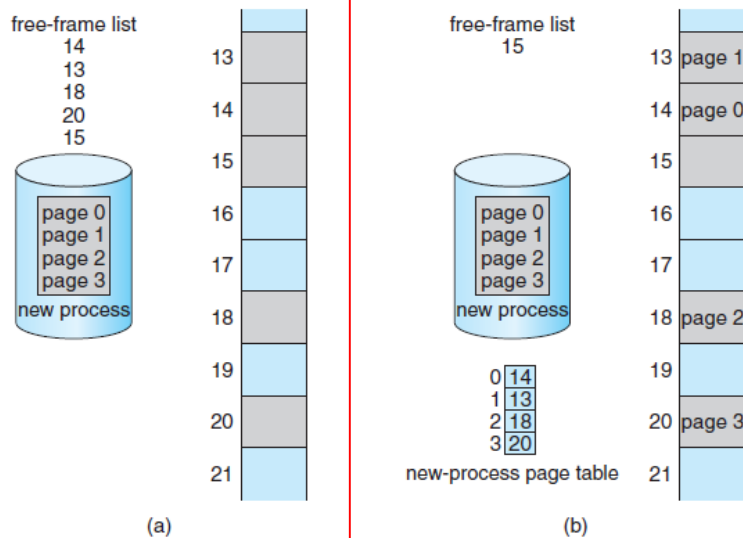
Paging

- 32-bit CPU'da genellikle **32-bit ile page table adresi verilir.**
- 2^{32} adet fiziksel page frame bulunur.
- **Bir frame boyutu 4 KB (2^{12}) ise,**
Toplam adreslenebilir fiziksel hafıza = $2^{32} * 2^{12} = 2^{44}$ olur (16 TB).
- Bir process sisteme çalışmak için geldiğinde, gerekli sayfa sayısı belirlenir.
- **Process'in her sayfası bir frame'e ihtiyaç duyar.**
- Toplam n sayfa varsa, en az n tane frame boş olmalıdır.
- Her sayfa bir frame'e yerleştirilir ve **frame numarası page table'a kaydedilir.**
- **Programcı process'in adresini tek ve bitişik olarak görür.** Frame eşleştirmesini işletim sistemi yapar.

37

Paging

- Şekilde bir process'e ait sayfaların hafızaya yerleştirilmesi görülmektedir.



38

Paging

Translation look-aside buffer (TLB)

- Her işletim sistemi page table saklamak için kendine özgü yöntem kullanır.
- Bazı işletim sistemleri **her process için ayrı page table kullanır**.
- **Her page table için pointer** ayrı bir register'da tutulur.
- Her page table için **bir grup register oluşturulur**.
- **Modern bilgisayarlarda page table çok büyüktür** (Örn.: 1 milyon giriş).
- Bu durumda **page table için register oluşturulması mantıklı değildir**.
- Page table'ın **hafızada tutulması** halinde, **her adres değişikliğinde hafıza erişimi gerekli olur (performans düşer)**.
- Mikroişlemcilerde, **küçük boyutta ve hızlı donanımsal önbellek (translation look-aside buffer)** ile bu problem çözülür.

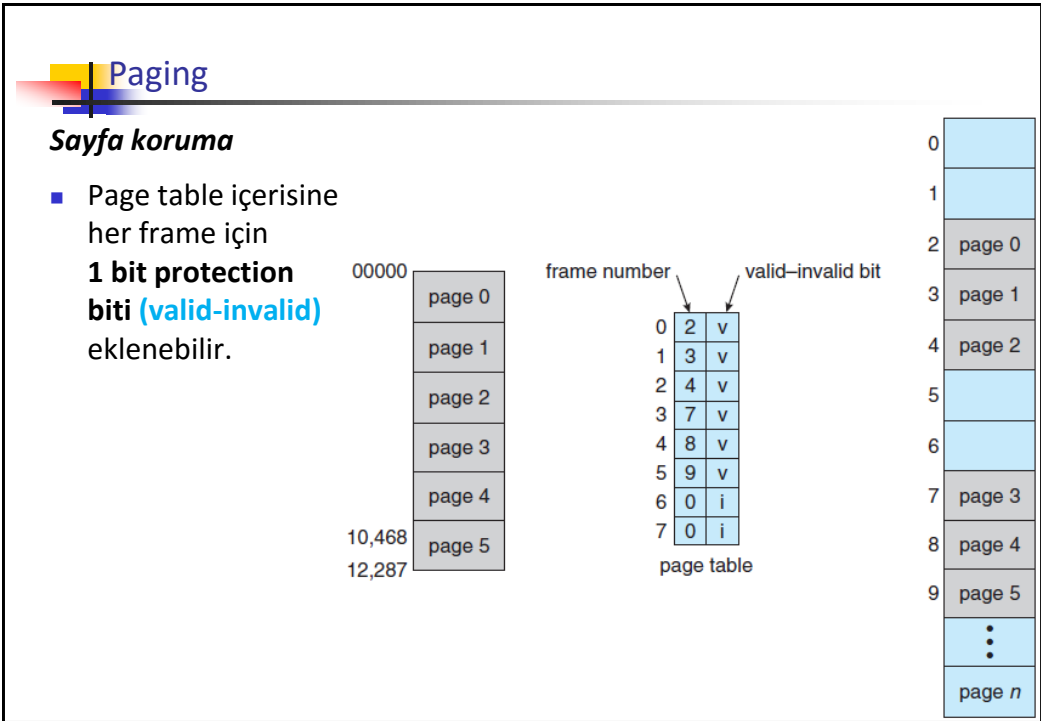
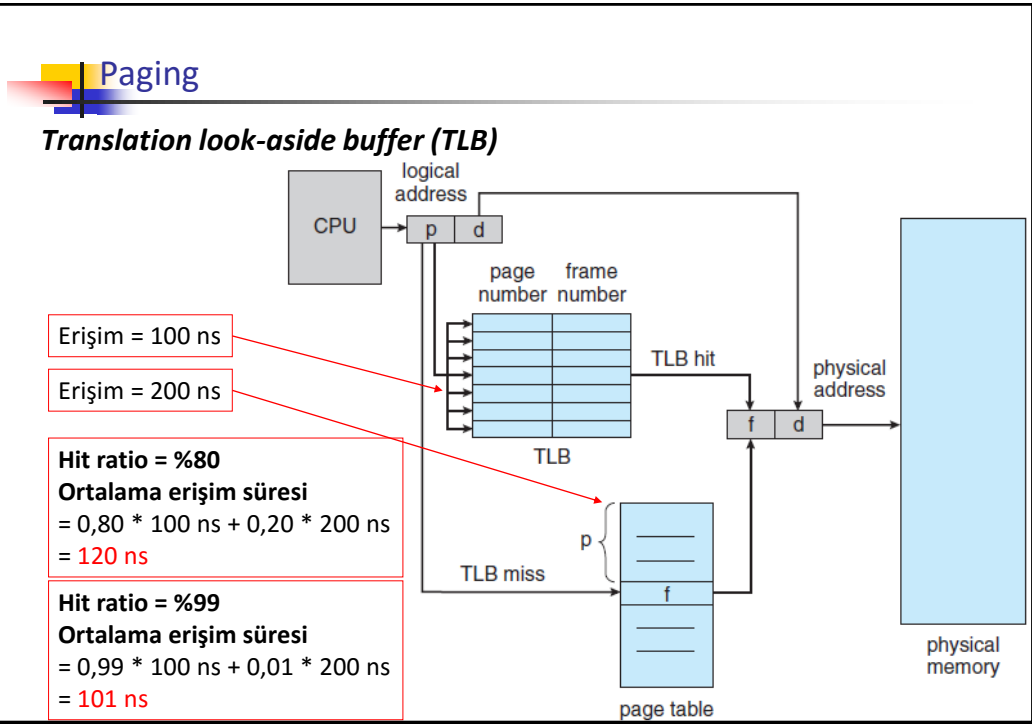
39

Paging

Translation look-aside buffer (TLB)

- TLB içerisindeki **her giriş** satırı **page number** ile **frame number** değerlerini tutar.
- Bir mantıksal adres geldiğinde, **page number tüm TLB içerisinde aranır (full associative)**.
- Page number değeri bulunursa, **ilgili satırdaki frame number değeri alınarak hafızada ilgili sayfaya gidilir**.
- TLB içerisinde bulunamayan page number için page table'a gidilir.
- Page table'dan alınan frame number ile page number değeri TLB'ye kaydedilir.
- **TLB dolu ise replacement algoritması (least recently used, round robin, random)** ile seçilen satır silinerek yerine yazılır.

40



Paging

Sayfa paylaşımı

- Birden fazla kullanıcı aynı sayfayı paylaşıp kullanabilir.
- Şekilde 3 process aynı text editörünü farklı verilerle kullanmaktadır.
- Kodun değişmez olması (reentrant / pure code) gereklidir.

