Giriş

- **Sanal bellek (virtual memory) yöntemi, process'lerin tamamının hafızaya yüklenmeden çalıştırılmasına izin verir.**
- Günümüzdeki programlar fiziksel hafızanın kapasitesinden daha büyük olabilmektedir.
- **Sanal bellek, programcıların hafıza limitlerinden soyutlanmasını sağlar.**
- Sanal bellek, dosyaların paylaşımını ve paylaşılmış hafıza oluşturulmasını kolaylaştırır.
- Bir programın çalışması için tamamının hafızaya yüklenmesine ihtiyaç yoktur:
  - **Nadiren hata yapan programlarda hata yönetimi kodlarının yüklenmesine gerek yoktur.**
  - **Dizi değişkenlerinin aktif kullanılan boyutları nadiren 10 * 10'dan büyük olmaktadır.**

3

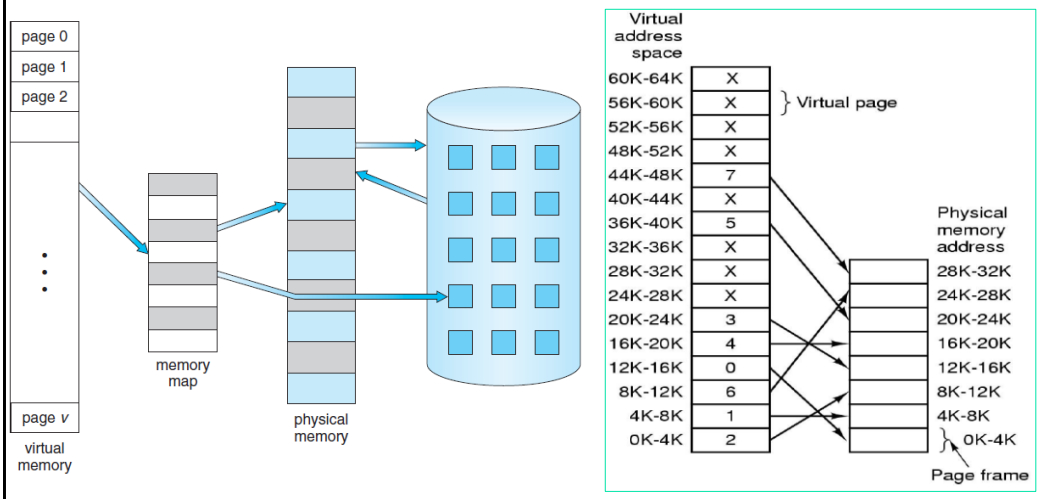
Giriş

- **Bir programın tamamının çalışması gerektiğinde bile, tümü aynı anda gerekmeyebilir.**
- Bir programın hafızaya parçalı bir şekilde alınarak çalıştırılması aşağıdaki faydaları sağlar:
  - **Bir programın toplam boyutu fiziksel hafızanın kapasitesinden fazla olabilmektedir.**
  - **Her kullanıcı programı, aynı anda küçük fiziksel hafıza alanı kullandığından, çok sayıda program eş zamanlı çalıştırılabilir.**
- Sanal bellek, programcı için hafıza alanı limitini ortadan kaldırır.

4

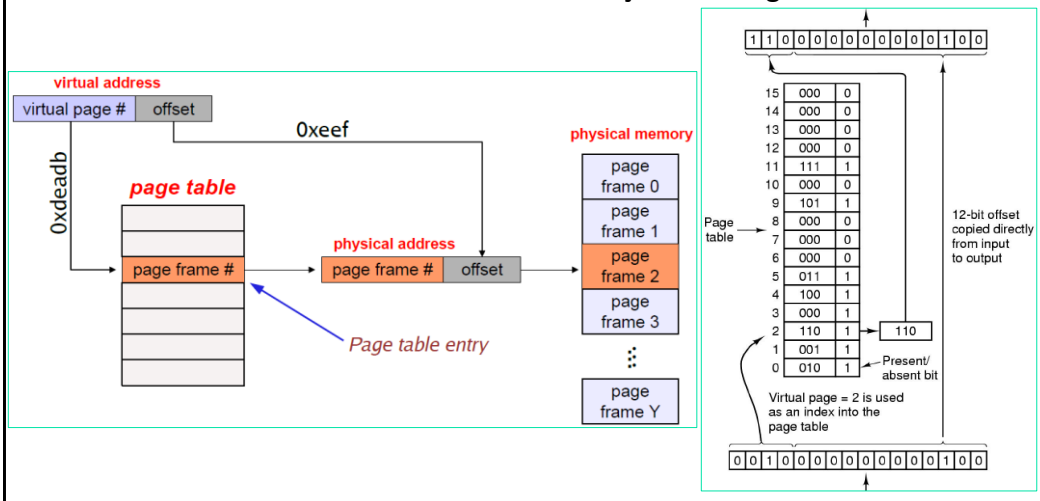
Giriş

- Sanal bellek, **programcıya fiziksel hafızadan çok büyük bir alan sağlar.**



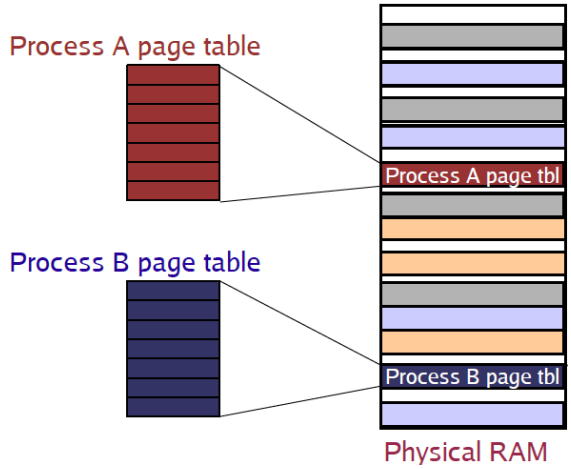
Giriş

- Sanal bellek adresinin fiziksel adrese dönüştürülmesi gereklidir.



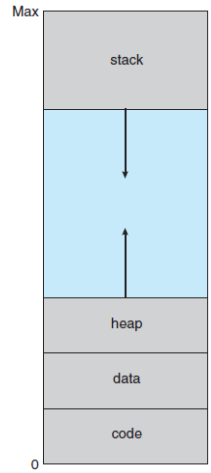
Giriş

- Process'ler için page table fiziksel hafızada saklanır**.**



Giriş

- Bir pocess'in sanal adres alanı, hafızaya nasıl yüklendiğini gösteren mantıksal bir görünümüdür.
- Bir process'in **mantıksal adresi 0 ile başlar ve bitişik bir alandır.**



Giriş

- **Sanal bellekteki heap ve stack alanları dinamik olarak büyüyebilmektedir.**
- Heap veya stack alanı büyürken fiziksel hafızada daha fazla sayfanın kullanımı gerekir.
- Sanal bellek, **process'ler arasında dosya ve hafızanın sayfalar aracılığıyla paylaşılmasını sağlar.**
  - **Sistem kütüphaneleri çok sayıda process arasında paylaştırılabilir.** Her process, kütüphaneleri kendi sanal belleklerinin parçası olarak görürler, ancak fiziksel hafızadaki sayfalar paylaştırılır.
  - **Process'ler hafızayı paylaşabilir.** Sanal bellek, bir process'in hafızada diğer process'lerle paylaşabileceği bölge oluşturmasına izin verir.
  - **fork() sistem çağrısı ile paylaşılmış sayfalar oluşturulabilir.** Process oluşturulma süresini kısaltır.

9

Konular

- Giriş
- Demand paging
- Copy-on-write
- Page replacement
- Allocation of frames
- Thrashing

10

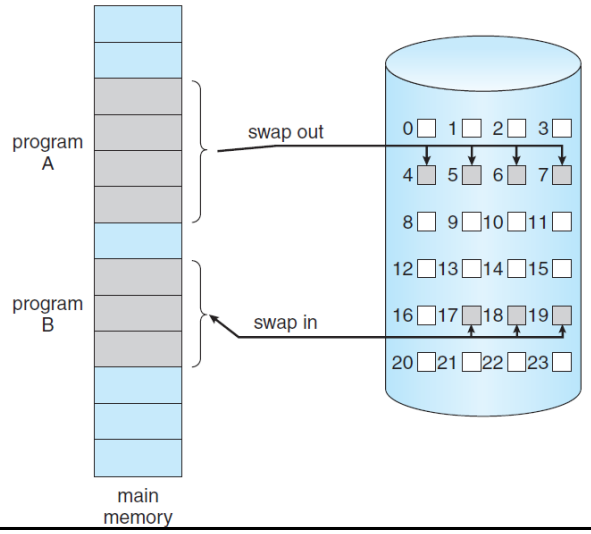
Demand paging

- **Programlara ait sayfaların ihtiyaç olduğunda yüklenmesine demand paging denilmektedir.**
- Demand paging yöntemi **sanal bellek sistemlerinde yaygın kullanılmaktadır.**
- Demand paging ile programın çalışması süresince **kullanılmayan sayfalar fiziksel hafızaya yüklenmez.**
- Demand paging sistemi **disk üzerindeki process'lerin hafızaya swapping ile yüklenmesini gerçekleştirir.**
- **Bir process içindeki bir sayfa gerekmedikçe hafızaya yüklenmez (lazy swapper).**
- **Pager process içindeki sayfaların yüklenmesini gerçekleştirir.**

11

Demand paging

- Hafızadaki **sayfaların bitişik disk aralığına** aktarımı şekilde görülmektedir.



12

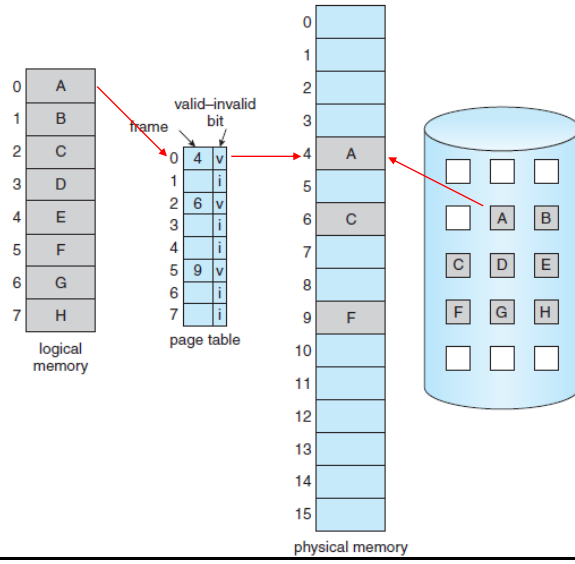
Demand paging

- Bir process swap in yapıldığında, **pager, swap out oluncaya kadar hangi sayfaların kullanılacağını tahmin eder.**
- Process'in tamamını yüklemek yerine, **gerekli sayfalar hafızaya yüklenir.**
- **Böylelikle gerekli hafıza alanı ve swap süresi azaltılmış olur.**
- Bir sayfanın hafızada mı yoksa diskte mi olduğunu tutmak için **donanımsal bileşen gerekir.**
- **valid** ve **invalid** şeklinde **bir bit** sayfanın bulunduğu yeri (hafızada olup olmadığı) **belirlemek için kullanılabilir.**

13

Demand paging

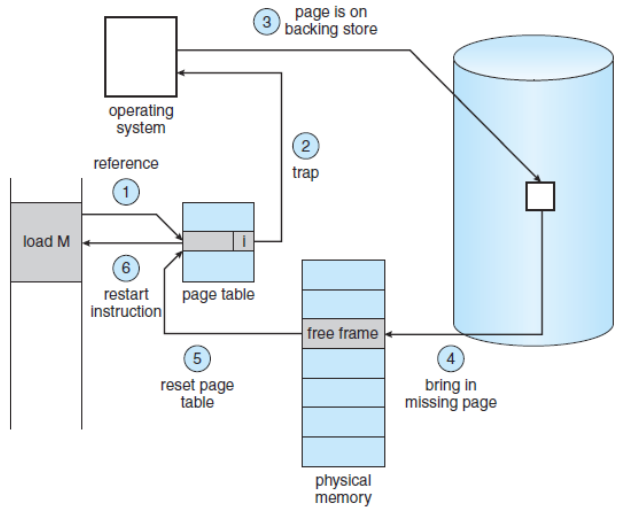
- **Invalid olan sayfaya process hiç erişmezse, invalid olmasının etkisi olmaz.**



14

Demand paging

- **Hafızada olmayan (invalid)** bir sayfaya erişime page fault (sayfa hatası) denir. İşletim sistemi sayfayı hafızaya aktarır.



15

Demand paging

- Bir sayfa hafızada bulunamadığında aşağıdaki işlemler gerçekleştirilir:
  - İstenen bloğun hafızada olmadığı belirlenir.
  - Çalışmakta olan process kesilir.
  - Hafızada boş bir frame belirlenir.
  - Disk üzerinden istenen sayfa hafızadaki boş frame'e aktarılır.
  - Sayfa tablosu değiştirilerek ilgili sayfanın hafızaya yüklendiği belirtilir.
  - Kesilen instruction ile process çalışmaya devam eder.
- Bir process başladığında hiçbir sayfa hafızada olmayabilir. **Process hemen page fault üretir ve bu sayfa hafızaya alınır.**
- Bir sayfanın ihtiyaç duyulmadığı sürece hafızaya alınmamasına pure demand paging denir.
- **Bazı programlar bir instruction ile çok sayıda sayfaya erişebilirler (bir instruction çok data) ve** çok sayıda page fault oluşur**.**

16

Demand paging

Demand paging yönteminin performansı

- Demand paging bilgisayar performansını önemli oranda etkiler.
- **Page fault hiç olmadığı durumda**, **efektif erişim süresi hafızaya erişim süresine eşittir.**
- Günümüzdeki bilgisayarlarda hafızaya erişim süresi **(ma) 10 ns-200 ns** arasındadır.
- **Page fault olma olasılığı** (p), **0 ≤ p ≤ 1** aralığındadır.

  Efektif erişim süresi = (1 - p) * ma + (p * page fault süresi)
- Page fault olması durumunda aşağıdaki temel 3 işlem yapılır:
  - **Page fault interrupt başlatılması**
  - **Sayfanın okunup hafızaya aktarılması**
  - **Process'in restart edilmesi**

17

Demand paging

Demand paging yönteminin performansı

- **Page fault olması durumunda yapılan işler yaklaşık 8 milisaniye alır.**

  Efektif erişim süresi = (1-p) * 200ns + p * 8ms

  = (1-p) * 200ns + p * 8.000.000ns

  = 200 – 200*p + 8.000.000*p

  = 200 + 7.999.800 * p
- **%1 page fault olursa** efektif erişim süresi yaklaşık 8,2 mikrosaniye olur.

18

Demand paging

Demand paging yönteminin performansı

- **%1 page fault olursa** efektif erişim süresi 40 kat yavaşlar (8,2 mikrosaniye / 200 nanosaniye = 41).
- Efektif erişim süresi artışını %10'un altına düşürmemek için page fault oranını %0,0000025'in altında tutmak gerekir.

$$220 > 200 + 7{,}999{,}800 \times p,$$
$$20 > 7{,}999{,}800 \times p,$$
$$p < 0.0000025.$$

19

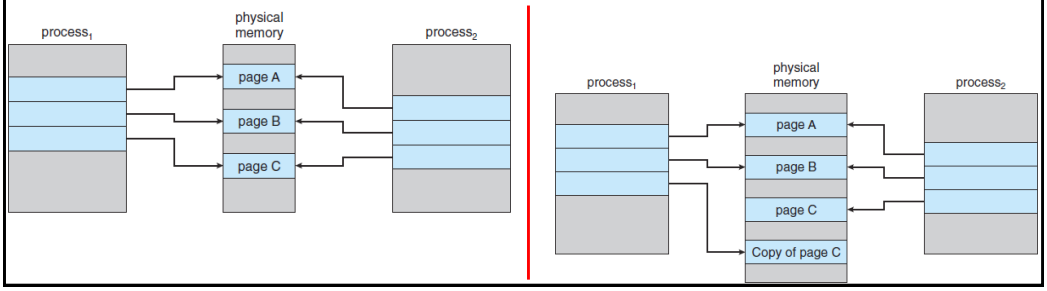
Konular

- Giriş
- Demand paging
- Copy-on-write
- Page replacement
- Allocation of frames
- Thrashing

20

Copy-on-write

- **fork() sistem çağrısı ile bir child process (parent process'in kopyası) oluşturulmaktadır.**
- Child process, parent process'in sahip olduğu sayfaların tümünün kopyalanmasına ihtiyaç duymayabilir (hemen exec() çağrısı başlatabilir.).
- Bu durumda, parent process'e ait sayfalar ortak kullanılır ve üzerinde **değişiklik yapılacağı zaman kopyası oluşturulur (copy-on-write)**.
- Sayfa C'yi değiştirmeden önceki ve sonraki durumlar aşağıdadır.



Konular

- Giriş
- Demand paging
- Copy-on-write
- Page replacement
- Allocation of frames
- Thrashing

Page replacement

- **Her sayfa ilk çağrıldığında bir kez page fault oluşur.**
- Bir process 10 sayfadan oluşuyorsa, **çalışması sırasında genellikle yarısını kullanır.**
- Bu yüzden, demand paging I/O gereksinimini azaltır.
- **Multiprogramming ile daha çok process'i çalıştırabiliriz (over-allocating).**
- Process'lerden bazıları tüm sayfaları kullanmak isteyebilir.
- İşletim sistemi **yeni sayfa için mevcut sayfalardan birisini swap out yapabilir (page replacement).**

23

Page replacement

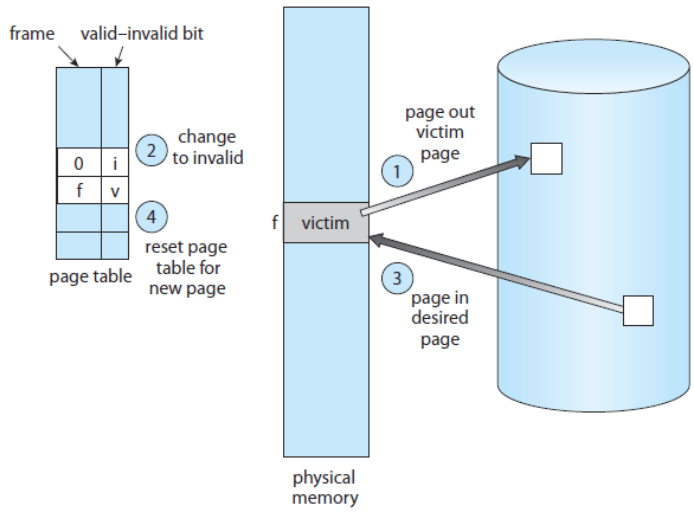
Temel page replacement algoritması

- Eğer boş frame yoksa **şu anda kullanılmayan bir frame seçilir ve swap out (disk swap space'e yazılır) yapılır.**
- Swap out yapılan frame için page table invalid yapılır.
  - **İstenen sayfa disk üzerinde bulunur.**
    - Boş frame varsa, bu frame kullanılır.
    - Boş frame yoksa, page-replacement algoritması ile bir frame seçilerek kullanılır.
    - Seçilen frame disk'e yazılır ve page table değiştirilir.
  - **Disk'ten istenen sayfa okunarak bu frame'e yazılır ve page table değiştirilir.**
  - **Page fault olan process çalışmaya devem eder.**

24

Page replacement

Temel page replacement algoritması



25

Page replacement

Temel page replacement algoritması

- Page fault olması halinde boş frame yoksa, **iki kez page fault süresi kadar beklenir (swap out ve swap in).**
- Hafızada kaldığı süre içerisinde **değişmeyen sayfaların hafızaya yazılmasına gerek yoktur.**
- Her sayfa için modify bit (dirty bit) ile değişip değişmediği tutulur.
- Page replacement algoritması ile **seçilen sayfa değişmişse hafızaya yazılır.**
- Bir process için **kaç tane frame'in hafızaya alınacağına frame-allocation algorithm** ile karar verilir.
- **Boş frame olmadığında ise hafızadan atılacak frame'e page-replacement algorithm** ile karar verilir.

26

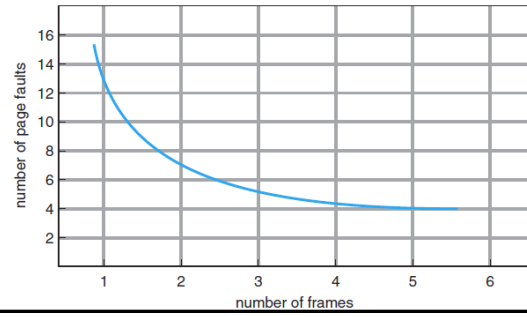
Page replacement

Temel page replacement algoritması

- Her sayfanın 100 byte olduğu durum için aşağıdaki **hafıza referansları** ve **ihtiyaç duyulan sayfalar verilmiştir.**

0100, 0432, 0101, 0612, 0102, 0103, 0104, 0101, 0611, 0102, 0103, 0104, 0101, 0610, 0102, 0103, 0104, 0101, 0609, 0102, 0105	1, 4, 1, 6, 1, 6, 1, 6, 1, 6, 1

- **Hafızaya yüklenen frame sayısı ile page fault sayısı ters orantılı değişir.**

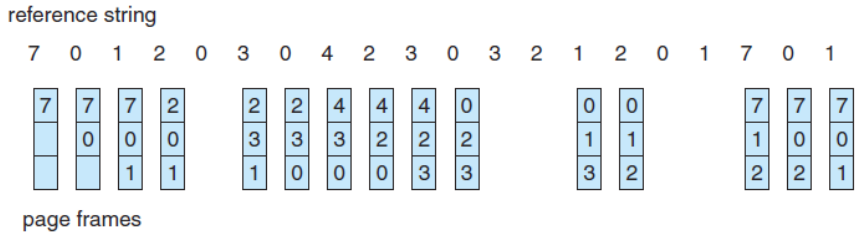


27

Page replacement

FIFO page replacement

- **FIFO algoritmasında ilk gelen frame atılarak yeni gelen buraya yazılır.**
- Şekilde process için 3 sayfa ayrılmıştır.
- Yeni gelen frame'ler en eski olan atılarak yerine yazılmaktadır.



- **Şekilde 15 page fault olmaktadır.**
- Seçilen sayfa aktif kullanılıyorsa swap out yapılması uygun değildir (o sayfa için hemen tekrar page fault oluşur).

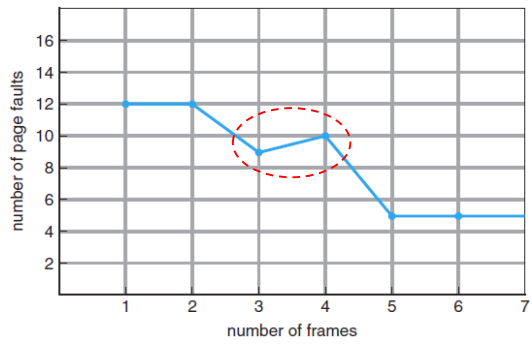
28

Page replacement

FIFO page replacement

- FIFO algoritmasında bazı durumlarda **daha fazla frame ayrılması halinde daha fazla page fault olabilmektedir** (Belady's anomaly).
- Aşağıdaki hafıza erişim serisi için page fault sayıları grafikte verilmiştir.

1, 2, 3, 4, 1, 2, 5, 1, 2, 3, 4, 5

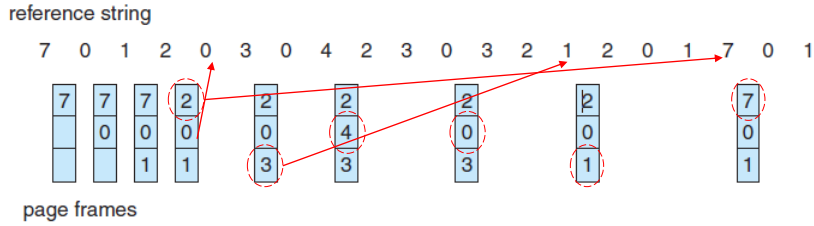


29

Page replacement

Optimal page replacement

- Optimal page replacement algoritmasında, **en uzun süre kullanılmayacak sayfa ile** yer değiştirme yapılır.
- Aşağıdaki hafıza erişim serisinde **9 page fault ile yer değiştirme yapılmaktadır.**

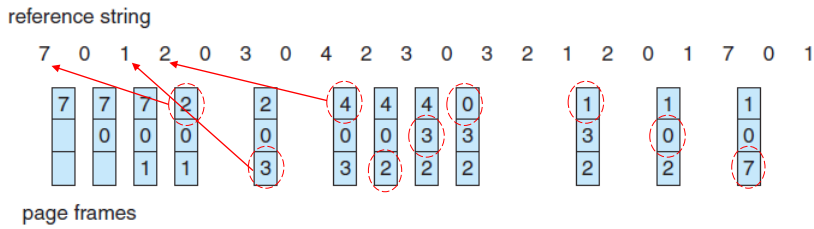


30

Page replacement

LRU page replacement

- Least recently used (LRU) algoritmasında, **en uzun süre kullanılmamış** olan sayfa atılır.
- Kullanılmama süresini tutmak için **counter kullanılabilir.**
- Kullanılan sayfa stack kullanılarak her defasında top eleman yapılır. **Stack'in en altındaki sayfa uzun süre kullanılmayan sayfadır.**

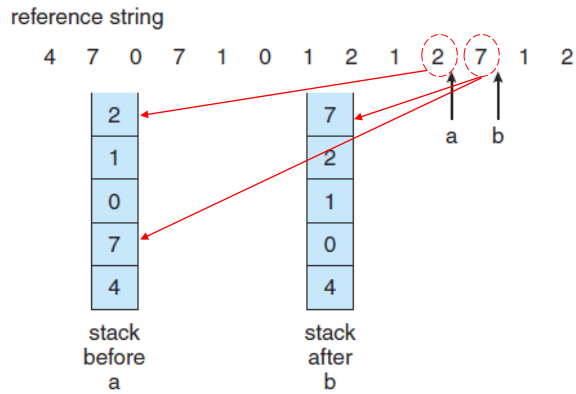


31

Page replacement

LRU page replacement

- Least recently used (LRU) algoritması için **stack kullanılabilir.**



32

Page replacement

Counting-based page replacement

- **Her sayfanın kullanılma sıklığı tutularak yer değiştirme yapılır.**
- Least frequently used (LFU) ve most-frequently used (MFU) yöntemleri kullanılabilir.
- **LFU** yönteminde, **en az sayıda kullanılan sayfa** ile yer değiştirme yapılır.
- **MFU** yönteminde, **en çok sayıda kullanılan sayfa** ile yer değiştirme yapılır.

33

Konular

- Giriş
- Demand paging
- Copy-on-write
- Page replacement
- Allocation of frames
- Thrashing

34

Allocation of frames

- **Her process için ayrılacak frame sayısı page fault oranı için önemlidir.**
- Tek kullanıcılı sistemlerde, işletim sisteminin kullandığı kısmın dışındaki tüm frame'ler process'e ayrılır.
- Tek kullanıcılı sistemlerde, tüm frame'ler dolu ise page replacement algoritması ile yer değiştirme yapılır.
- **Process'lere atanacak frame'ler için işletim sisteminin kullandığı frame'lerin dışındaki frame'ler boş (kullanılabilir) olarak ayrılır.**
- Bir process'e ihtiyaç duyduğundan daha fazlası atanmaz ve minimum frame atanmasına çalışılır.
- Bir komut kümesinde her komut sadece bir hafıza erişimi yapıyorsa, **bir frame instruction için bir frame'de data için gerekir.**
- Indirect adresleme yapılıyorsa her process için üç frame gerekebilir.
- **Gerekenden az frame atandığında, page fault oranı artar.**

35

Allocation of frames

Frame atama algoritmaları

- En kolay yöntemde, **m adet frame n process arasında eşit olarak dağıtılır (equal allocation)** (Her process m/n frame alır.).
- **Farklı process'ler farklı sayıda frame'e ihtiyaç duyarlar.**
- Her process'e atanacak frame sayısı process'in boyutuna göre belirlenebilir (proportional allocation).
- Aşağıda, ***S*** tüm process'lerin toplam boyutu, s_i i. process'in boyutudur.

$$S = \sum s_i$$

- P_i process'i için atanacak frame sayısı a_i aşağıdaki gibi hesaplanır:

$$a_i = s_i / S \times m$$

- Toplam 62 frame'in 10 ve 127 sayfa olan process'ler için frame sayıları,

  (10 / 137) * 62 = 4 ve **(127 / 137) * 62 = 57**

36

Allocation of frames

Global veya lokal atama

- Global replacement algoritmaları, **bir process'e tüm frame'ler arasından atama yapabilir.**
- **Atanacak frame başka bir process tarafından kullanılıyorsa, alınarak tahsis edilir.**
- Local replacement algoritmaları, page fault olduğunda bir process'e **kendisine atanmış frame'ler arasından birisi seçilerek tahsis edilir.**
- Global replacement algoritmalarında yüksek öncelikli process'lere düşük öncelikli process'lerin frame'leri atanabilir.
- **Local replacement algoritmalarında process'lerin frame sayıları değişmez.**
- Global replacement algoritmalarında process'ler kendi page fault oranlarını kontrol edemezler **(diğer process'ler frame alabilir!)**.

37

Konular

- Giriş
- Demand paging
- Copy-on-write
- Page replacement
- Allocation of frames
- Thrashing

38

Thrashing

- Bir process için gerekli **minimum frame sayısının altına düşüldüğünde**, **process beklemeye alınır.**
- **Bu process'in tüm frame'leri boşaltılır ve diğer process'lere tahsis edilir.**
- Bir process yeterli sayıda frame'e sahip değilse sık sık page fault olur.
- **Page fault olduğunda tüm sayfaları aktif ise birisini seçer ve yer değiştirir.**
- Ardından diğer sayfaya erişmek ister ve yeniden page fault olur. **Page fault sıklığı giderek artar.**
- Yüksek orandaki sayfalama işlemine thrashing (boşuna çalışma) denilir.
- Bir process **paging için ayırdığı süreden daha fazlasını thrashing için ayırıyorsa** thrashing yapıyor (boşuna çalışıyor) denilir.

39

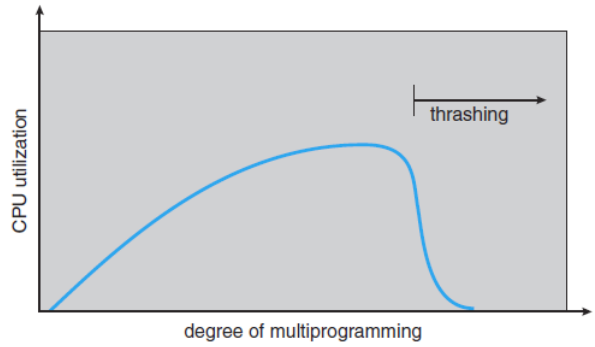
Thrashing

- **İşletim sistemi CPU doluluk oranını sürekli izler** ve **CPU doluluk oranı azaldığında** multiprogramming seviyesini artırarak **yeni process başlatır.**
- Global page replacement kullanılırsa, **diğer process'lerin sayfaları alınmaya başlar ve buna bağlı olarak page fault sıklığı artmaya başlar.**
- **Diğer process'ler yeni** sayfaya ihtiyaç duyduğunda **bu kez yeni process'e atanan sayfalar alınır.**
- Page fault arttıkça, process'ler paging biriminde sıra beklemeye başlar ve CPU hazır kuyruğu boşalır. **CPU doluluk oranı düşmeye başlar.**
- CPU scheduler, CPU'nun doluluk oranının düşmeye başladığını görür ve sürekli yeni process'ler başlatır.
- **Sonuçta multiprogramming seviyesi artar, ancak CPU doluluk oranı düşer, thrashing artar ve throughput düşer.**

40

Thrashing

- Şekilde multiprogramming seviyesi ile CPU doluluk oranı görülmektedir.



- Belirli bir noktada, **CPU doluluk oranını artırmak ve thrashing'i durdurmak için**, **multiprogramming seviyesini düşürmek** gereklidir.

41

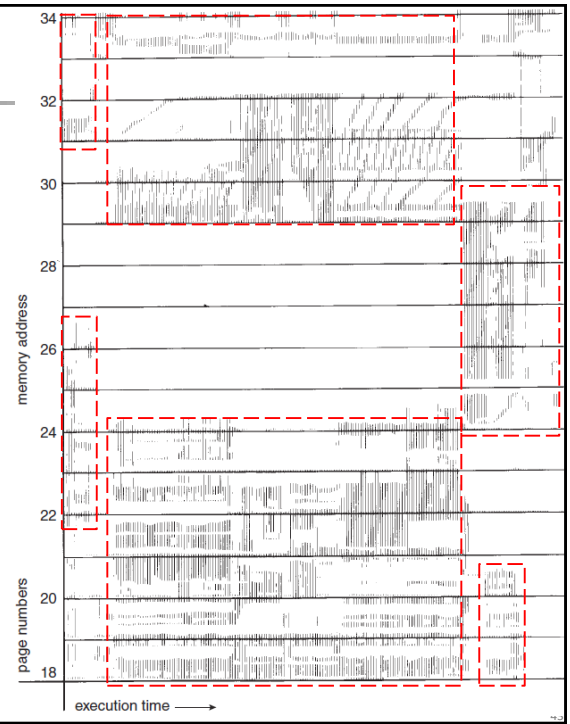
Thrashing

- Local replacement (priority replacement) algoritması kullanılarak thrashing azaltılabilir.
- Local replacement algoritmasında, **bir process thrashing yapmaya başlarsa diğer process'lerden frame almasına izin verilmez.**
- Thrashing yapan process'ler zamanın büyük kısmını paging biriminde geçirirler ve efektif erişim süreleri düşer.
- **Thrashing yapmayan process'ler içinde (paging biriminin kuyruğu büyüdüğü için) efektif erişim süresi düşer.**
- Thrashing'i engellemek için her process'e ihtiyaç duyacağı kadar frame atamak gereklidir.
- Bir process'in herhangi bir anda ihtiyaç duyacağı sayfalar locality model ile belirlenir.
- **Bir process, locality'den locality'e geçiş yaparak çalışır.**

42

Thrashing

- Locality bölgeleri büyük veya küçük, uzun süreli veya kısa süreli olabilir.



Thrashing

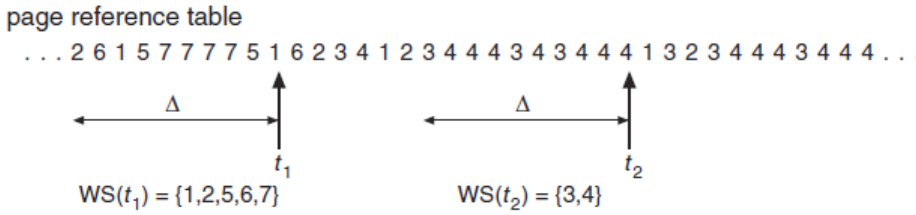
Working-set model

- Working-set modeli working-set penceresini tanımlamak için **Δ** parametresini kullanır.
- En son kullanılan sayfalar **Δ** ile gösterilir.
- Eğer bir sayfa aktif ise working-set içerisindedir.
- Bir sayfa artık kullanılmıyorsa working-set içerisinden çıkartılır.
- **Δ** çok küçük seçilirse, locality tam olarak belirlenemez.
- **Δ** çok büyük seçilirse, çok sayıda farklı locality overlap olur.
- **Δ** sonsuz alınırsa, working-set, process çalışırken kullanılan tüm sayfalar olur.

Thrashing

Working-set model

- Şekilde, **Δ = 10** alınırsa ***t1*** **anında** working-set {1,2,5,6,7}olur.
- ***t2*** **anında** working-set {3,4} olur.

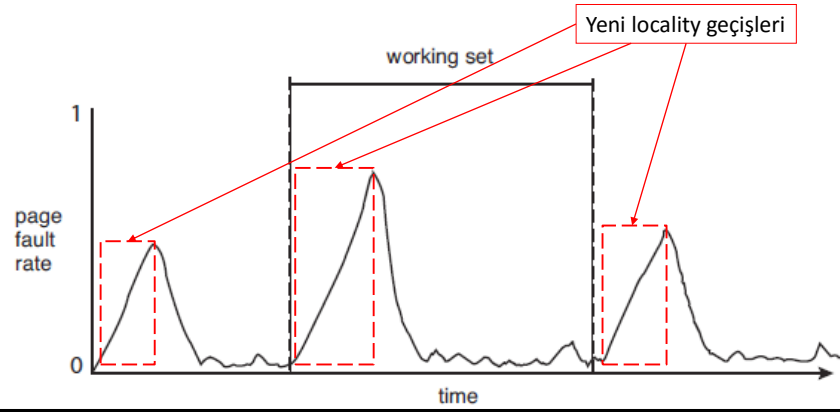


45

Thrashing

Working-set model

- Şekilde, working-set ile page fault oranı ilişkisi verilmiştir.
- Yeni bir locality'ye geçildiğinde page-fault oranı yükselmekte ve o bölgede çalışma devam ederken page-fault oranı düşmektedir.

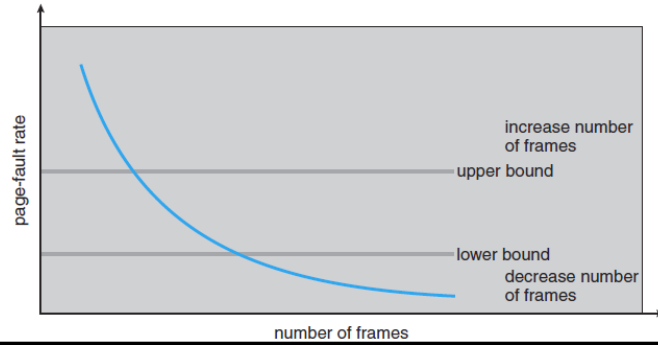


46

Thrashing

Page-fault frequency

- Page-fault frequency yaklaşımında, **page-fault oranına göre frame sayısı artırılır veya azaltılır.**
- Page-fault oranı yüksekse frame sayısı artırılır, düşükse frame azaltılır.
- Belirlenecek üst ve alt limitlerin dışına çıkıldığında, frame sayısı değiştirilir.



47