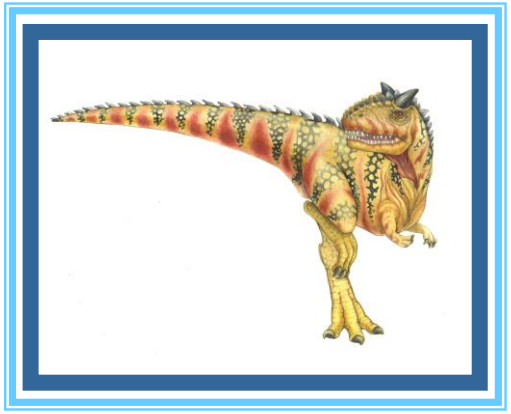
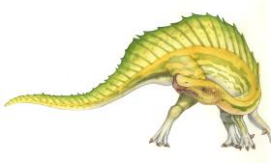
Bölüm 10: Sanal Bellek (Virtual Memory)



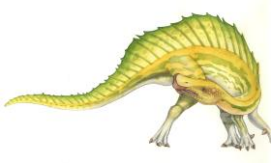
Anahat

- Arka plan
- Talebe göre Sayfalama (Demand Paging)
- Yazma üzerine kopyala
- Sayfa Yer Değiştirme (Page Replacement)
- Çerçevelerin Tahsisi
- Dayak (Thrashing)
- Bellek haritalı Dosyalar
- Çekirdek Belleği Ayırma
- Diğer Hususlar
- İşletim Sistemi Örnekleri



Hedef

- Sanal belleği tanımlamak ve faydalarını tarif etmek.
- Talebe göre sayfalama kullanarak sayfaların belleğe nasıl yüklendiğini görmek
- FIFO, optimal ve LRU sayfa yer değiştirme algoritmalarını uygulamak
- Bir prosesin çalışma kümesini tanımlamak ve program yerelliğiyle nasıl bir ilişkisi olduğunu açıklamak
- Linux, Windows 10 ve Solaris'in sanal belleği nasıl yönettiğini açıklamak
- C programlama dilinde sanal bellek yöneticisi simülasyonu tasarlamak



Arka plan

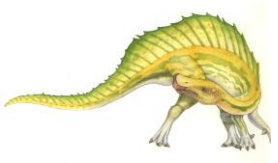
- Kodun yürütülmesi için bellekte olması gerekir, ancak programın tamamı nadiren kullanılır
  - Hata kodu, gereksiz rutinler, büyük veri yapıları
- Tüm program kodu aynı anda gerekli değil
- Kısmen yüklenen programın çalışabilme yeteneği
  - Program fiziksel bellek ile sınırlandırılmış alana bağlı değil
  - Her program çalışırken daha az bellek alır -> daha fazla program aynı anda çalışır
    - Cevap süresinde veya tamamlanma süresinde bir artış olmadan CPU kullanım oranının ve çıkışın (throughput) artması
  - Programları yüklemek veya bellekle değiştirmek için daha az G/Ç gerekmesi -> her kullanıcı programı daha hızlı çalışır

Sanal bellek

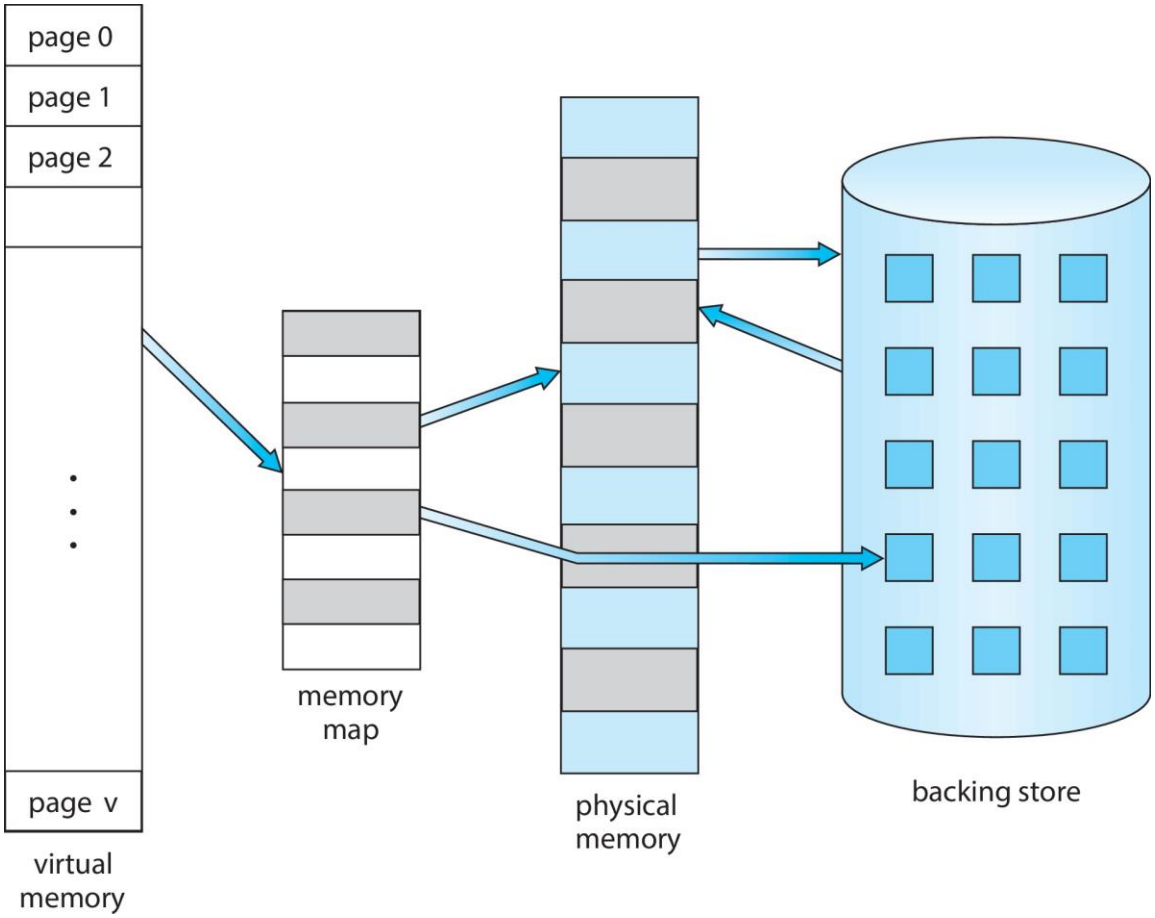
- **Sanal bellek** – kullanıcının mantıksal belleğinin fiziksel bellekten ayrılması
  - Yürütme için programın sadece bir kısmının bellekte olması gerekir
  - Mantıksal adres alanı fiziksel adres alanından çok daha büyük olabilir
  - Adres boşlukları birden fazla proses tarafından paylaşılabilir
  - Daha verimli proses oluşturma
  - Daha fazla programın eş zamanlı çalışabilmesi
  - Prosesleri yüklemek veya takas yapmak için daha az G/Ç gerekli

Sanal bellek (devam)

- **Sanal adres alanı** – prosesin bellekte nasıl tutulduğunun mantıksal görünümü
  - Genellikle adres 0'dan başlar, ard arda adresler boş kısmın sonuna kadar
  - Bu arada, fiziksel bellek sayfa çerçeveleri halinde
  - MMU birimi mantıksalı fiziksele haritalar
- Sanal bellek aşağıdaki yaklaşımlar ile uyarlanabilir:
  - Talebe göre sayfalama
  - Talebe göre segmentasyon

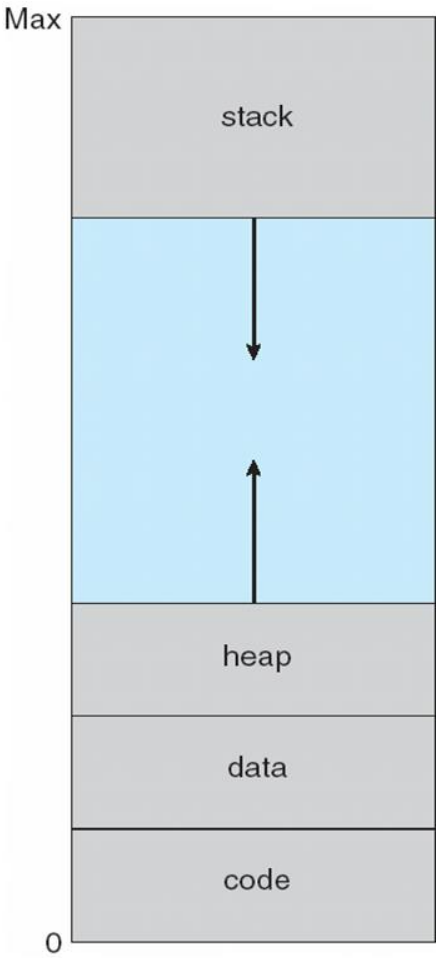


Fiziksel Bellekten Daha Büyük Sanal Bellek

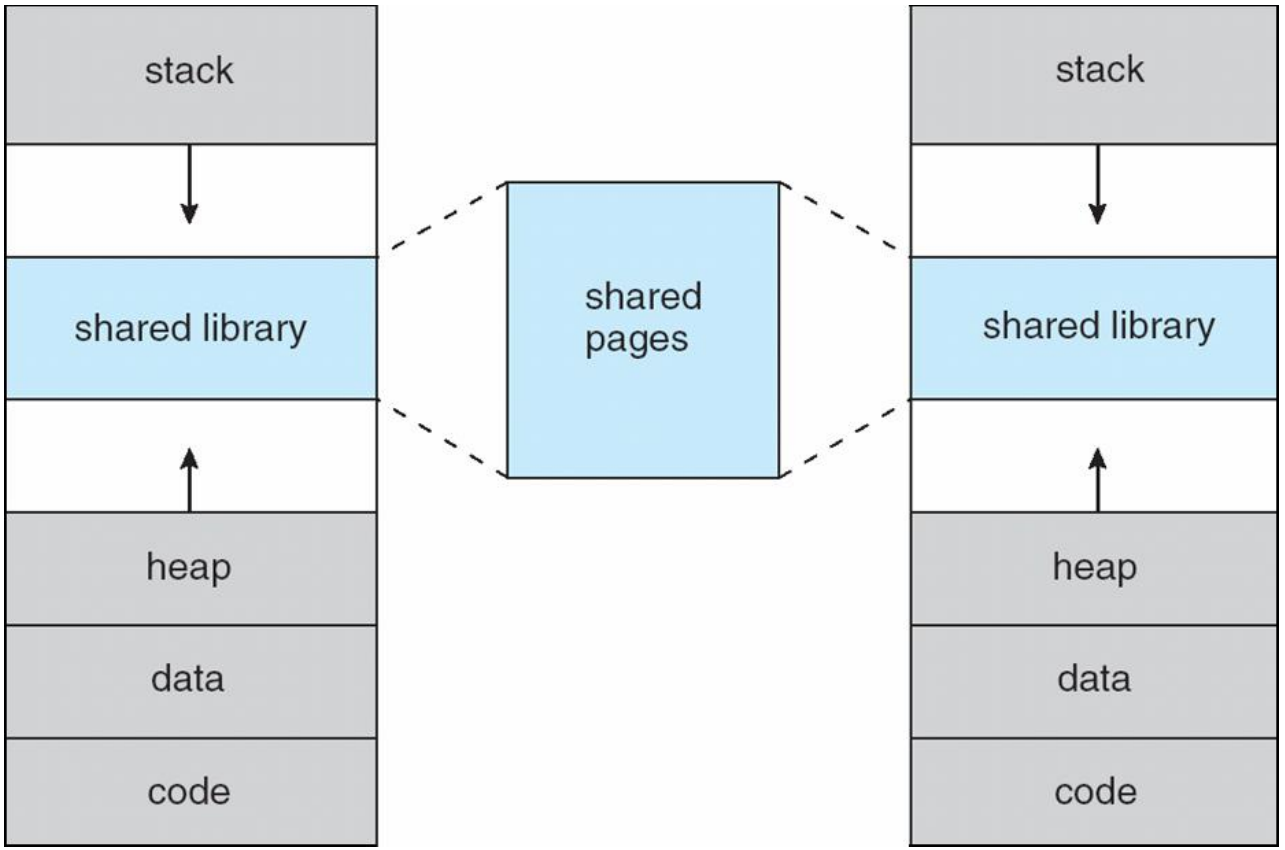


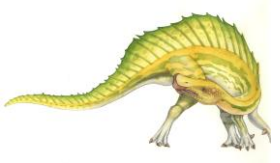
Sanal adres Alanı

- Genellikle yığının(stack) mantıksal adres alanı Max mantıksal adresinden başlayacak şekilde tasarlanır ve yığıt (heap)büyürken küçülür
  - Adres alanı kullanımını en üst seviyeye çıkarır
  - İkisi arasındaki kullanılmayan adres alanı boşluktur(hole)
    - Yığın veya yığıt yeni bir sayfaya taşmadıkça yeni fiziksel bellek gerekmez
- **Aralıklı** adres boşluklarının büyüme için kullanılmasını sağlar, dinamik bağlantılı kütüphaneler, vb.
- Sistem kütüphaneleri sanal adres alanına haritalama yoluyla paylaşılır
- Paylaşımlı bellek sayfaları sanal adres alanına haritalayarak gerçekleştirilir
- Sayfalar **`fork()`** sırasında paylaşılabilir
  - proses oluşturma sürecini hızlandırır



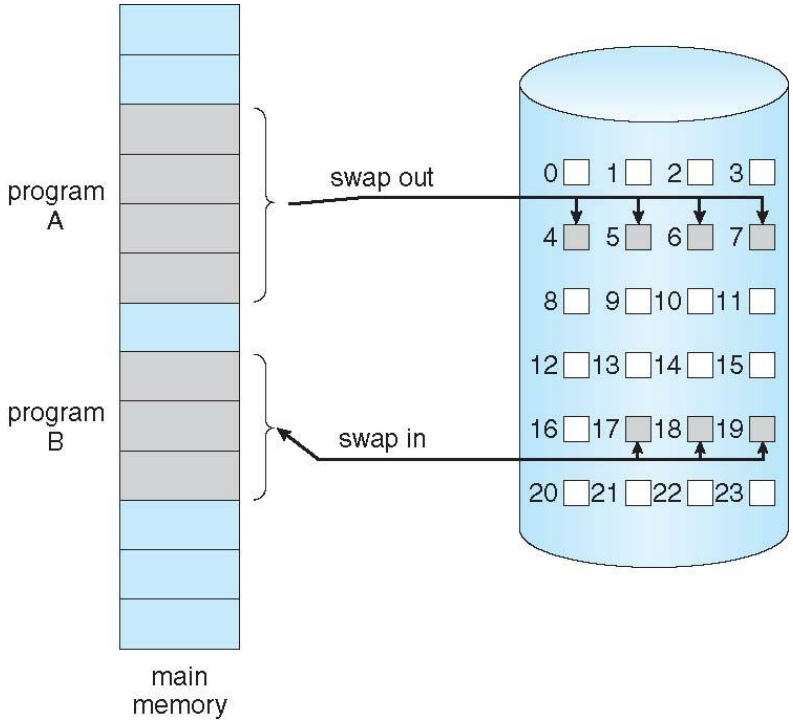
Sanal Bellek Kullanarak Paylaşılan Kitaplık





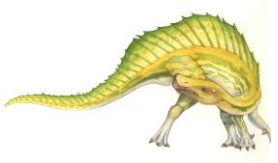
Talebe göre Sayfalama

- Yükleme zamanında tüm proses belleğe yüklenebilir
- Veya yalnızca ihtiyaç duyulduğunda bir sayfa belleğe getirilir
  - Daha az G/Ç gerekli, gereksiz G/Ç yapılmaz
  - Daha az bellek gerekli
  - Daha hızlı cevap
  - Daha fazla kullanıcı
- Takas ile birlikte sayfalama sistemine benzer (sağdaki diyagram)
- geçersiz referans ⇒ iptal
  - bellekte değil ⇒ belleğe getir
- **Yavaş takasçı** –gerekli olmadığı sürece asla bir sayfayı belleğe getirmez
  - Takasçı sayfalar ile ilgilenir = **Sayfacı**



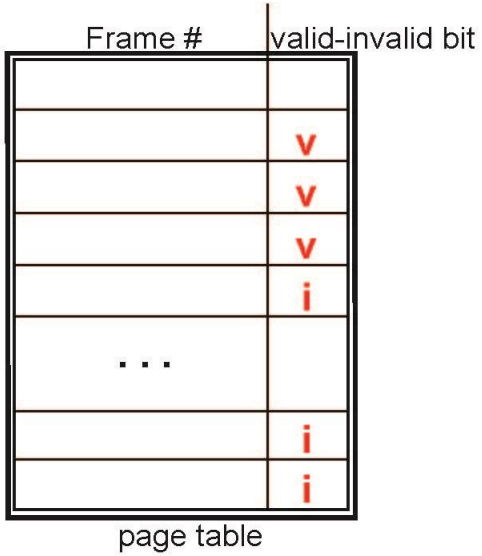
Temel Kavramlar

- Takas ile birlikte, sayfacı hangi sayfaların değiştirileceğini tahmin eder
- Sayfalar nasıl belirlenir?
- Talep göre sayfalamayı uygulamak için yeni MMU işlevi gerekir
- Eğer sayfalar gerekli ve zaten **bellekteyse**
  - Talebe göre olmayan sayfalamadan farkı yok
- Eğer sayfa gerekli ve bellekte değilse
  - Sayfayı tespit etmeli ve diskten belleğe yüklemeli
    - Program davranışını değiştirmeden
    - Programcının kodu değiştirmesine gerek kalmadan
- Geçerli(valid –v)-geçersiz(invalid-i) bitli sayfa tablosunu kullan (bkz. Ana Bellek bölümü)



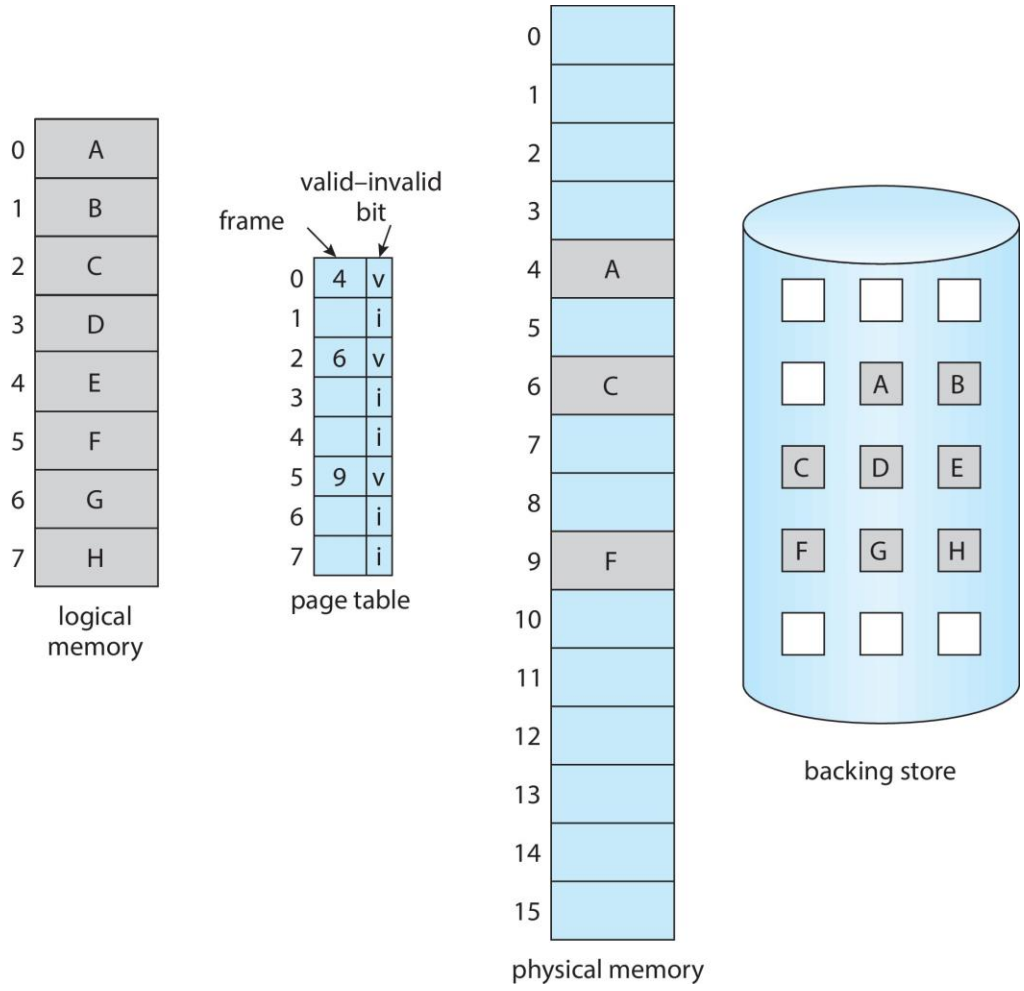
Geçerli Geçersiz Bitli Sayfa Tablosu

- Her sayfa tablo girdisi bir geçerli-geçersiz biti ile ilişkilendirilir (**v** ⇒ bellekte, **i** ⇒ bellekte değil, diskte)
- Başlangıçta tüm girdilerin geçerli-geçersiz biti **i** olarak belirlenir
- Sayfa tablosu anlık görüntüsü örneği:



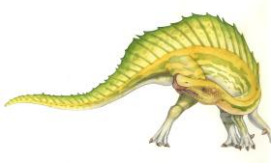
- MMU adres dönüşümü sırasında, sayfa tablosunda geçerli-geçersiz biti **i** ise ⇒ sayfa hatası

Bazı Sayfalar Olmadığında Sayfa Tablosu Ana Bellekte

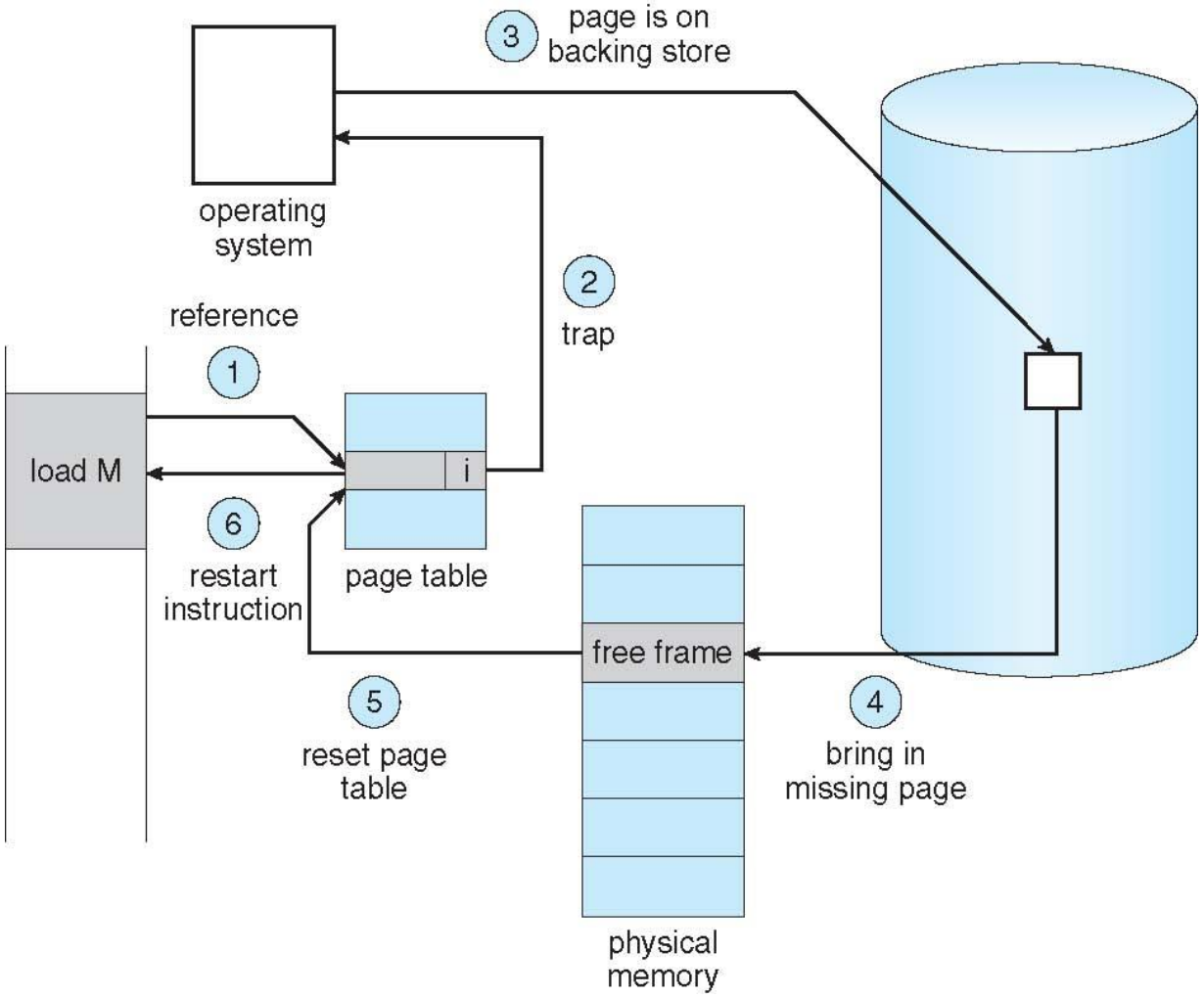


Sayfa Hatalarını İşleme Adımları

1. Bir sayfaya ilk kez başvuru yapılırsa, işletim sistemi hata yakalar
   - Sayfa hatası
2. İşletim sistemi karar vermek için başka bir tabloya bakar:
   - Geçersiz başvuru ⇒ iptal
   - Sadece bellekte değil (adım 3'e gidin)
3. Boş bir çerçeve bul (ya yoksa?)
4. Zamanlanmış disk işlemi ile sayfayı çerçeveye takas yap
5. Sayfayı şimdi bellekte olduğunu belirtmek için tabloları resetle
   Doğrulama bitini = **v** olarak ata
6. Sayfa hatasına neden olan talimatı (instruction)yeniden başlat



Sayfa Hatalarını İşleme Adımları (devam)



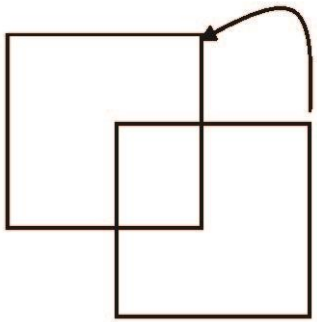
Talebe göre Sayfalama Türleri

- **Saf talebe göre sayfalama:** prosesi bellekte sayfalar olmadan başlat
  - OS, talimat işaretçisini prosesin ilk talimatına ayarlar,

  bellekte yerleşik olmayan sayfa -> sayfa hatası
  - Ve ilk erişimdeki diğer tüm proses sayfaları için
- Aslında, belirli bir talimat birden çok sayfaya erişebilir -> birden çok sayfa hatası
  - Bellekten 2 sayıyı toplayan ve sonucu belleğe geri depolayan talimatı alıp çözme
- Talebe göre sayfalama için gereken donanım desteği
  - Geçerli / geçersiz bitli sayfa tablosu
  - İkincil bellek (**takas alanına sahip** takas aygıtı)
  - Talimat yeniden başlatma

Talimatı Yeniden Başlatma

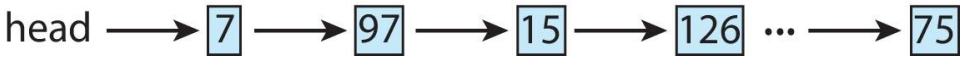
- Birkaç farklı konuma erişebilecek bir talimat düşünün
  - Blok hareket



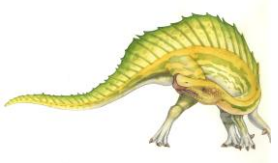
  - Konumu otomatik artırma / azaltma
  - Tüm operasyon yeniden başlatılmalı mı?
    - Kaynak ve hedef çakışıyorsa ne olur?

Boş Çerçeveler Listesi

- Bir sayfa hatası oluştuğunda, işletim sistemi diskten istenen sayfayı ana belleğe getirmelidir.
- Çoğu işletim sistemi, **boş çerçeveler listesi** tutar -- bu tür istekleri karşılamak için boş çerçeveler havuzu.

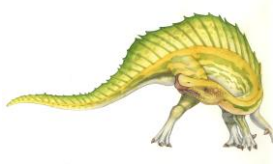


- İşletim sistemi genellikle boş çerçeveleri **isteğe bağlı sıfır doldurma** olarak bilinen bir teknik kullanarak ayırır -- çerçevelerin içeriği tahsis edilmeden önce sıfır ile doldurulur
- Bir sistem başlatıldığında, kullanılabilir tüm bellek hücreleri boş çerçeve listesine yerleştirilir.



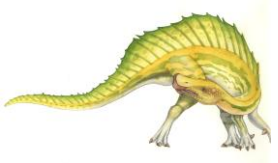
Talebe göre Sayfalama Aşamaları – Kötü Durumda

1. İşletim sistemine hata bildirimi
2. Kullanıcı kaydedicilerini ve proses durumunu kaydetme
3. Kesmenin bir sayfa hatası olduğunu belirleme
4. Sayfa referansının legal olup olmadığını kontrol etme ve diskteki sayfanın konumunu belirleme
5. Diskten boş bir çerçeveye okuma konusu:
   a) Okuma isteği servis edilene kadar bu aygıt için kuyrukta bekle
   b) Aygıtı arama ve/veya gecikme süresini bekle
   c) Sayfanın boş bir çerçeveye aktarılmasını başlat



Talebe göre Sayfalama aşamaları (devam)

6. Beklerken, CPU'yu başka bir kullanıcıya tahsis edin
7. Disk G/Ç alt sisteminden bir kesme al (G/Ç tamamlandı)
8. Kaydedicileri ve işlem durumunu diğer kullanıcı için kaydetme
9. Kesmenin diskten olduğunu belirleme
10. Sayfa tablosu ve diğer tabloları sayfanın artık bellekte olduğunu göstermek için doğrula
11. CPU'nun bu işleme yeniden tahsis edilmesini bekle
12. Kullanıcı kaydedicileri, proses durumu ve yeni sayfa tablosunu geri yükle ve sonra kesilen talimata devam et



Talebe göre Sayfalama Performansı

- Üç ana etkinlik
  - Kesmeyi yönet - dikkatli kodlama gerekli birkaç yüz talimat anlamına gelir
  - Sayfayı diskten gir– çok zaman alır
  - İşlemi yeniden başlat – yine de sadece küçük bir süre
- Sayfa Hata Oranı $0 \le p \le 1$
  - eğer $p = 0$ sayfa hatası yok
  - eğer $p = 1$, her başvuru bir hatadır
- Etkili Erişim Süresi (EAT)

  EAT = (1 – p) x bellek erişimi

  + P (sayfa hatası maliyeti

  + sayfayı dışa takas

  + sayfayı içe takas)

Talebe göre Sayfalama Örneği

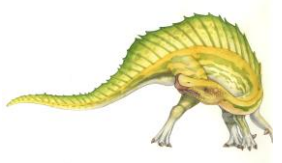
- Bellek erişim süresi = 200 nanosaniye
- Ortalama sayfa hatası servis süresi = 8 milisaniye
- EAT = (1 – p) x 200 + p (8 milisaniye)

  = (1 – p) x 200 + p x 8.000.000

  = 200 + p x 7.999.800
- 1.000 üzerinden bir erişim bir sayfa hatasına neden oluyorsa,

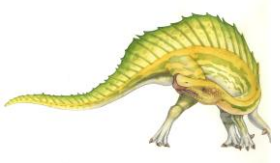
  EAT = 8.2 mikrosaniye.

  Bu 40 kat yavaşlama manasına gelir!
- Performans düşümü < yüzde 10 istersek
  - 220 > 200 + 7.999.800 x p
    20 > 7.999.800 x p
  - p < .0000025
    - her 400.000 bellek erişiminde bir sayfa hatası



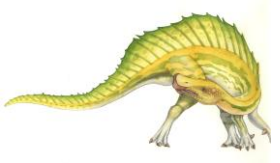
Talebe göre Sayfalama Optimizasyonları

- Aynı cihazda olsa bile takas alanı G/Ç, normal dosya sistemi G/Ç'den daha hızlıdır
  - Takas daha büyük parçalar halinde tahsis edilir; dosya sisteminden daha az yönetim gerekli
- Proses yükleme esnasında tüm proses görüntüsünü takas alanına kopyala
  - Daha sonra sayfalar takas alanından transfer edilir
  - Eski BSD Unix'te kullanılır
- Diskteki program kayıtlarından sayfayı talep et, ancak çerçeveyi boşa çıkarırken sayfayı diske takas yapmak yerine atmak yerine at
  - Solaris ve şuan ki BSD de kullanılır
  - Hala takas alanına yazmak gerekir
    - Bir dosyayla ilişkili olmayan sayfalar (yığın ve yığıt gibi) – **Anonim Bellek**
    - Bellekte değiştirilmiş ancak henüz dosya sistemine yazılmayan sayfalar
- Mobil sistemler
  - Genellikle takası desteklemez
  - Bunun yerine, dosya sisteminden sayfa talep edilir ve salt okunur sayfaları geri alır (kod gibi)

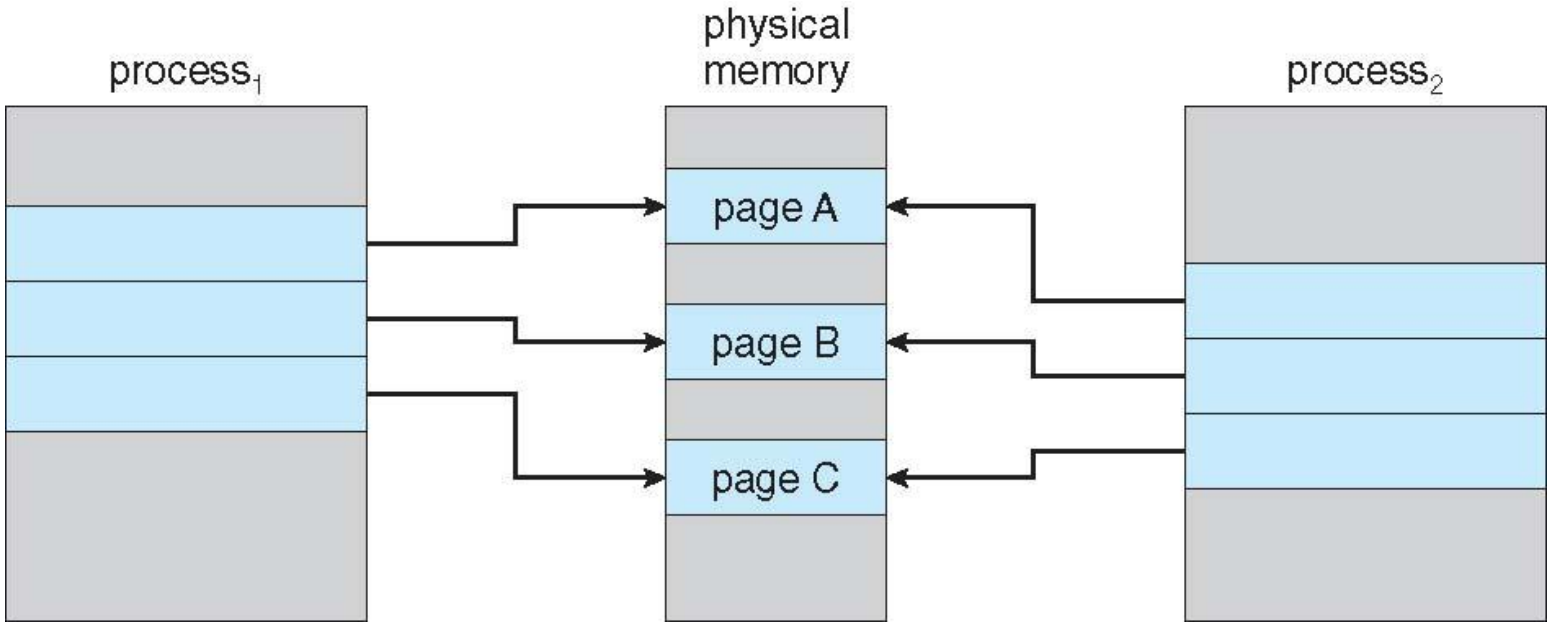


Yazma üzerine kopyalama(Copy-on-Write)

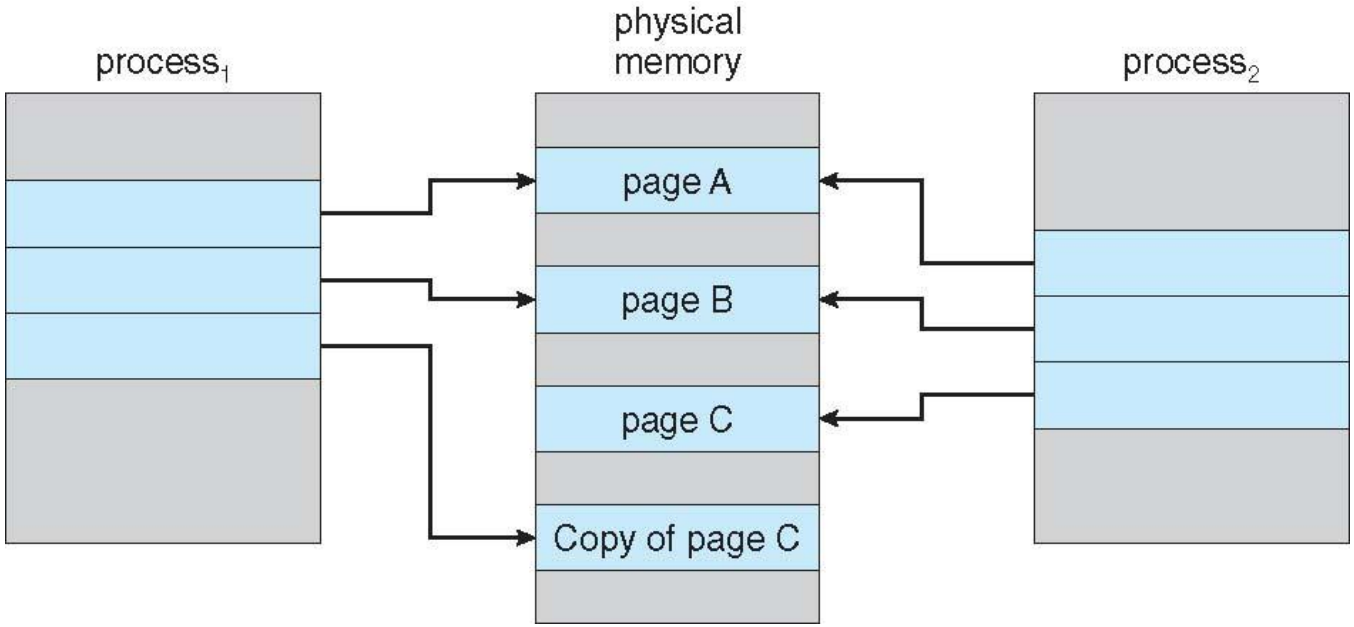
- **Yazma üzerine kopyalama** (Copy-on-Write- COW) hem ebeveyn hem de çocuk proseslerin başlangıçta bellekteki aynı sayfaları ***paylaş*masını** sağlar.
  - Her iki proses de paylaşılan bir sayfayı değiştirirse, sayfa ancak o zaman kopyalanır
- COW sadece değiştirilmiş sayfaların kopyalanması nedeniyle daha verimli proses oluşturma sağlar
- Genel olarak, boştaki sayfalar **isteğe bağlı sıfır doldurma havuzundan** alınır
  - Havuz her zaman hızlı talep sayfası işleyişi için boşta çerçevelere sahip olmalıdır
    - Sayfa hatasıyla ilgili diğer işlemlerin yanı sıra bir çerçeveyi de serbest bırakmaya ihtiyaç duyulmasın
  - Bir sayfayı tahsis etmeden önce neden sıfır ile doldurulur?
- `fork()` sistem çağrısının bir sürümü olan `vfork()` askıya alınmış bir ebeveyne ve ebeveynin COW adres alanını kulllanan bir çocuğa sahiptir.
  - Çocuk prosesin `exec()` i çağırabilmesi için geliştirildi
  - Çok verimli



Proses 1 in C Sayfasını Değiştirmesinden Önce



Proses 1 in C Sayfasını Değiştirmesinden Sonra

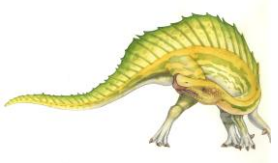


Boş Çerçeve Yoksa Ne Olur?

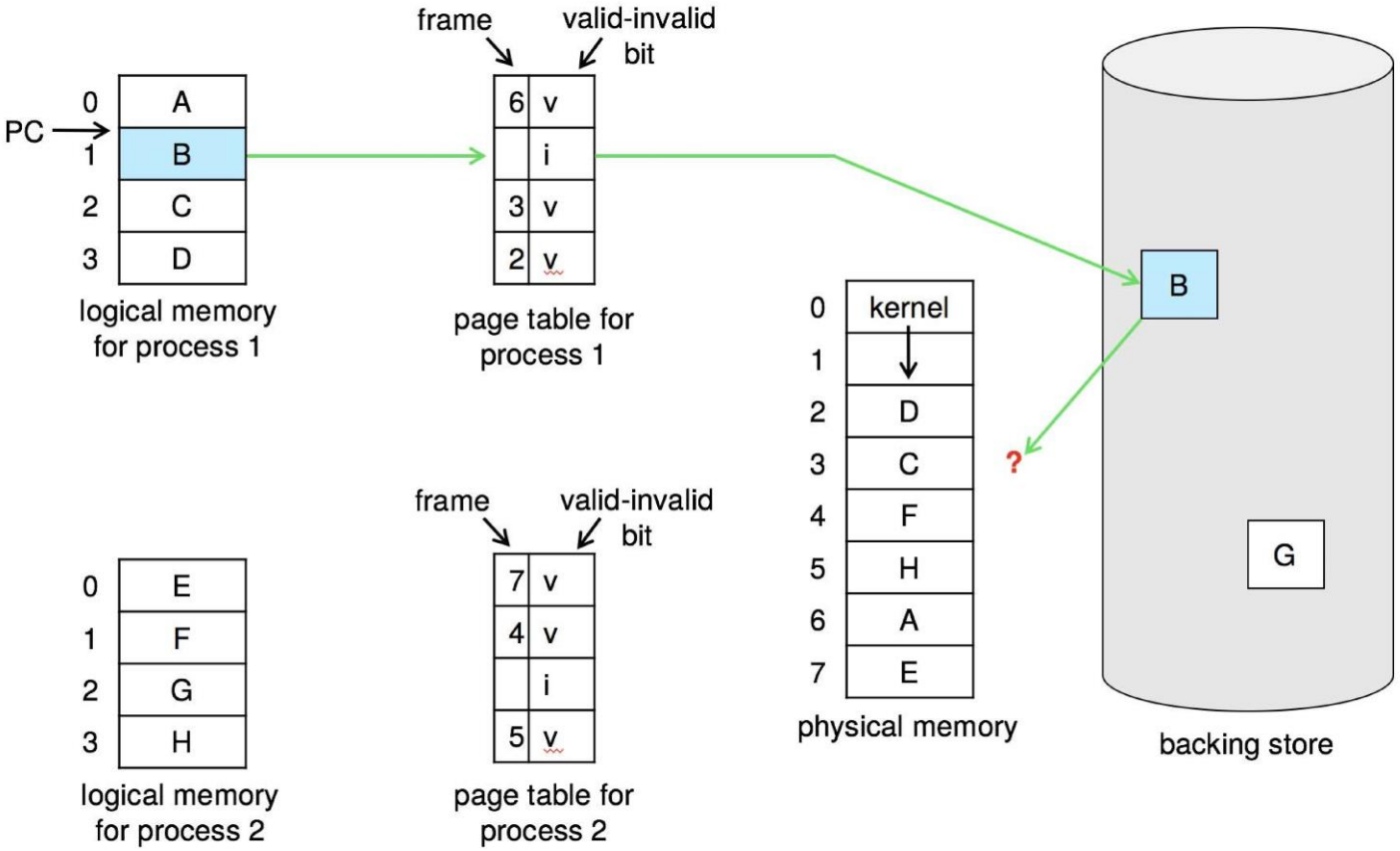
- Proses sayfaları tarafından kullanılır
- Ayrıca çekirdek, G /Ç tamponlar talep, Vb
- Her birine ne kadar ayırmak için?
- Sayfa değiştirme – bellekte bazı sayfa bulmak, ama gerçekten kullanımda değil, sayfa dışarı
  - Algoritma – sonlandırmak? takas mı? sayfayı değiştirin?
  - Performans – en az sayfa hatası yla sonuçlanacak bir algoritma istiyorum
- Aynı sayfa birkaç kez belleğe getirilebilir

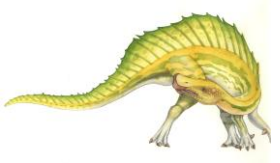
Sayfa Değiştirme

- Önle **fazla tahsisat** sayfa değiştirme içerecek şekilde sayfa hatası hizmet yordamını değiştirerek bellek
- Kullanın **Değiştirmek** (**Kirli**) **Bit** sayfa aktarımlarının genel merkezlerini azaltmak için – yalnızca değiştirilmiş sayfalar diske yazılır
- Sayfa değiştirme mantıksal bellek ve fiziksel bellek arasındaki ayrımı tamamlar – büyük sanal bellek daha küçük bir fiziksel bellekte sağlanabilir



Sayfa Değiştirme İhtiyacı



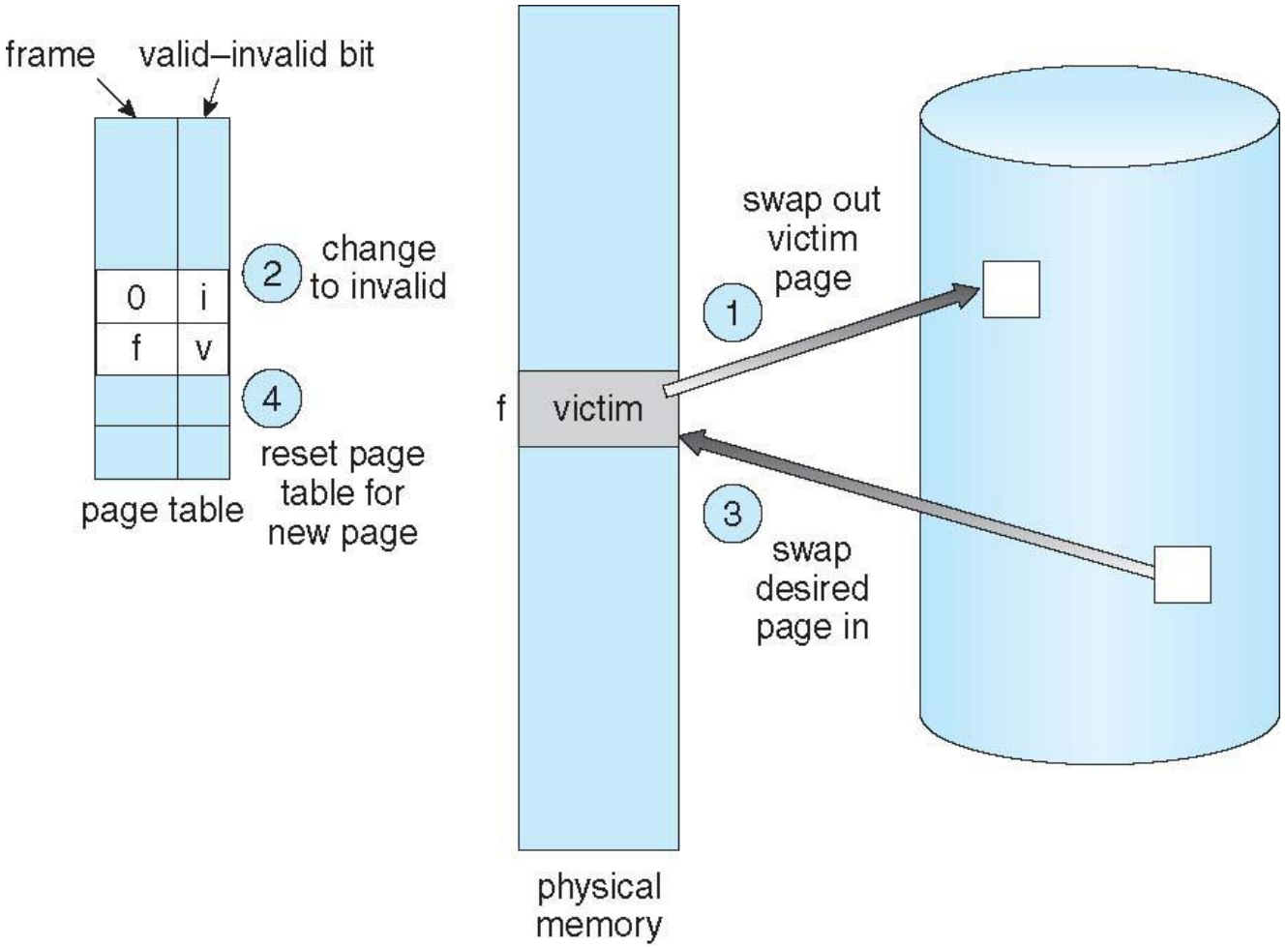


Temel Sayfa Değiştirme

1. Diskte istenilen sayfanın konumunu bulma
2. Ücretsiz bir çerçeve bulun:
   - Boş bir çerçeve varsa,
   - Boş bir çerçeve yoksa, sayfa değiştirme algoritması kullanarak **Kurban Çerçeve**
   - Kirliyse kurban çerçevesini diske yazma
3. İstenilen sayfayı (yeni) boş çerçeveye getirin; sayfa ve çerçeve tablolarını güncelleştirin
4. Bindirmeye neden olan talimatı yeniden başlatarak işleme devam edin

Not şimdi potansiyel olarak 2 sayfa transferleri sayfa hatası için - artan EAT

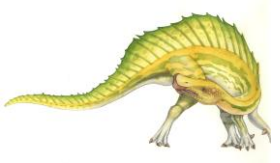
Sayfa Değiştirme



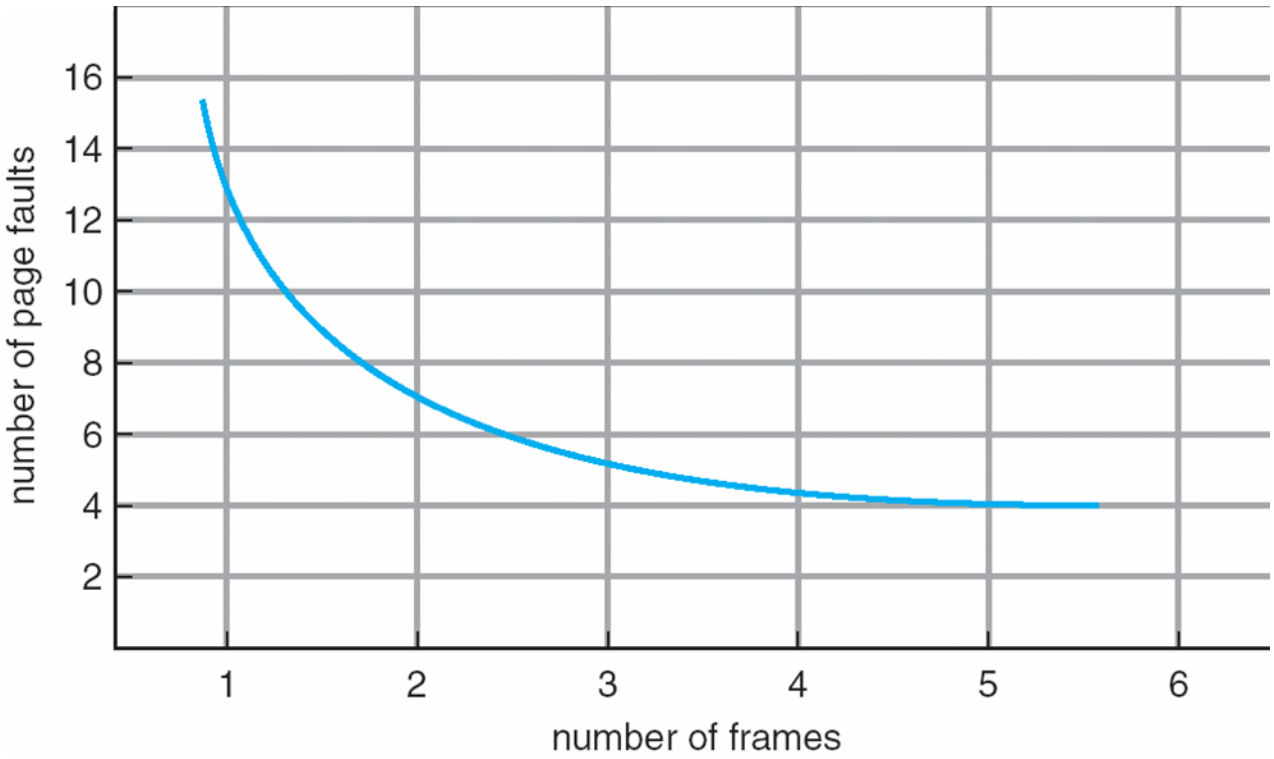
Sayfa ve Çerçeve Değiştirme Algoritmaları

- **Çerçeve ayırma algoritması** Belirler
  - Her işlem vermek için kaç kare
  - Hangi çerçeveler değiştirmek için
- **Sayfa değiştirme algoritması**
  - Hem ilk erişimhem de yeniden erişimde en düşük sayfa hata oranını istiyorum
- Algoritmayı belirli bir bellek başvurusu dizesi (başvuru dizesi) üzerinde çalıştırarak ve bu dizedeki sayfa hatalarının sayısını hesaplayarak değerlendirin
  - String sadece sayfa numaraları değil, tam adresleri
  - Aynı sayfaya yinelenen erişim sayfa hatasına neden olmaz
  - Sonuçlar kullanılabilir kare sayısına bağlıdır
- Tüm örneklerimizde, **Başvuru Dize** başvurulan sayfa numaralarının

7,0,1,2,0,3,0,4,2,3,0,3,0,3,2,1,2,0,1,7,0,1

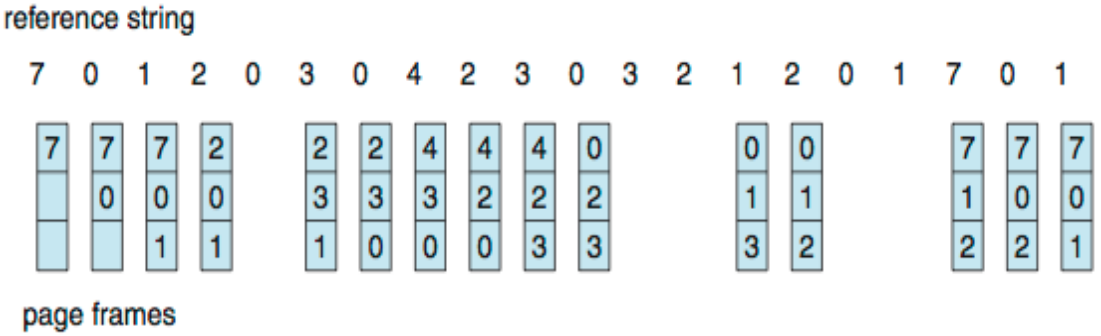


Sayfa Hatalarının Kare Sayısına Karşı Grafiği



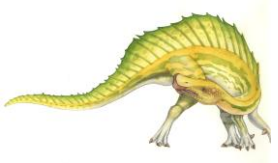
İlk İlk Çıkış (FIFO) Algoritması

- Başvuru dizesi: **7,0,1,2,0,3,0,4,2,3,0,3,0,3,2,1,2,0,1,7,0,1**
- 3 kare (her işlem de 3 sayfa bellekte olabilir)



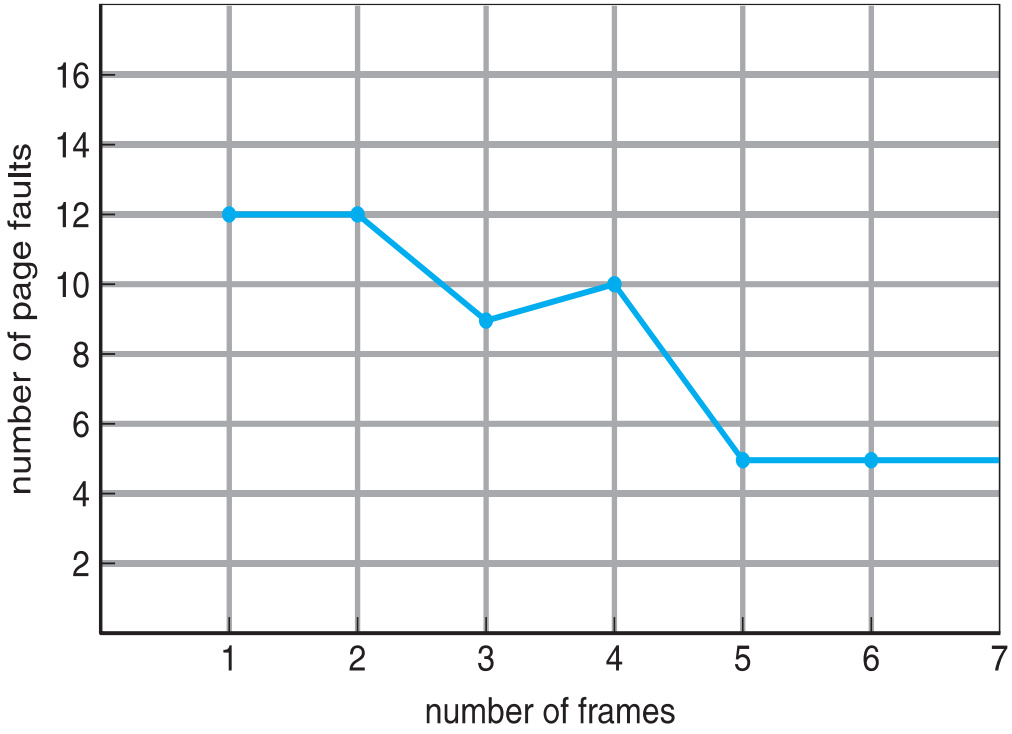
15 sayfa hataları

- Sayfaların yaşları nasıl izlenir?
  - FIFO kuyruğu kullanman



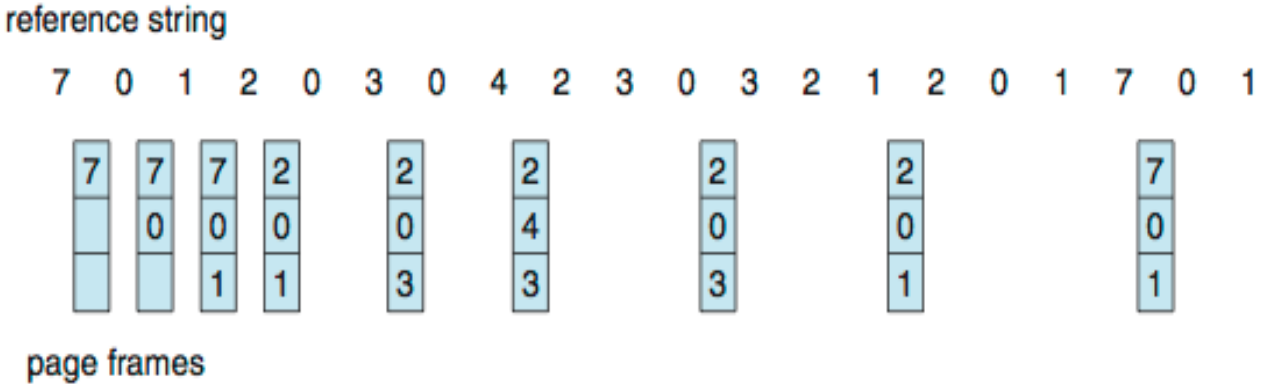
Belady's Anomali

- Dize düşünün 1,2,3,4,1,2,5,1,2,3,4,5
  - Daha fazla kare eklemek daha fazla sayfa hatasına neden olabilir!
- Belady'yi gösteren grafik's Anomali



Optimal Algoritma

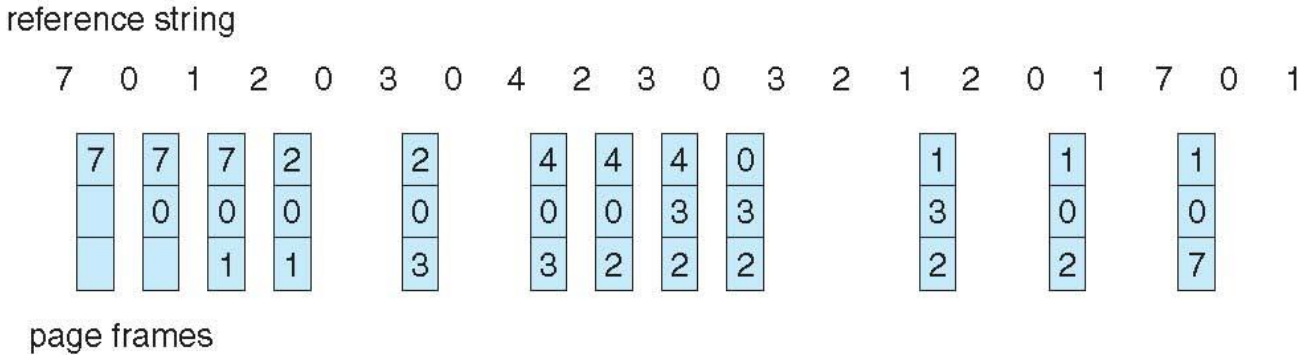
- En uzun süre kullanılmayacak sayfayı değiştirme
  - 9 örnek için en uygun
- Bunu nereden biliyorsun?
  - -bilirsiniz't geleceği okuyun
- Algoritmanızın ne kadar iyi performans gösterdiğini ölçmek için kullanılır



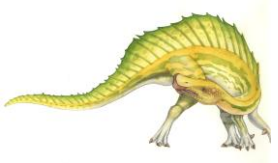
- Optimal bir örnektir **yığın algoritmaları** bu don't Belady muzdarip's Anomali

En Son Kullanılan (LRU) Algoritması

- Gelecek yerine geçmiş bilgileri kullanma
- En fazla süre içinde kullanılmayan sayfayı değiştirme
- Son kullanım süresini her sayfayla ilişkilendirme



- 12 hata – FIFO'dan daha iyi ama OPT'den daha kötü
- Genellikle iyi algoritma ve sık kullanılan
- LRU yığın algoritmaları başka bir örnektir; böylece hiçbir yokt Belady muzdarip's Anomali

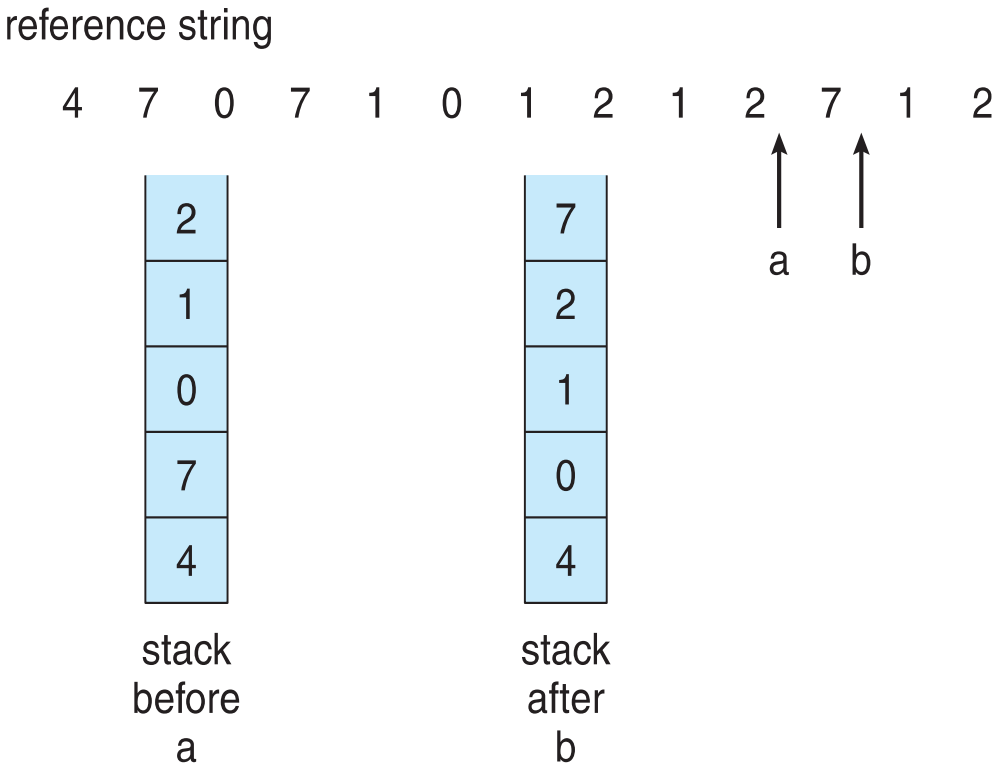


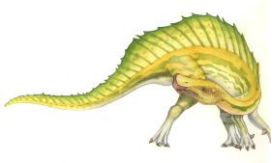
LRU Algoritma Uygulaması

- Zaman sayacı uygulaması
  - Her sayfa girişinin bir zaman sayacı değişkeni vardır; bir sayfaya bu giriş üzerinden her başvurulsa, saatin değerini zaman sayacına kopyalayın
  - Bir sayfanın değiştirilmesi gerektiğinde, en küçük değeri bulmak için sayaçlara bakın
    - Tabloda arama yapılması gereklidir
- Yığın uygulaması
  - Sayfa numaralarını çift bağlantı formunda tutun:
  - Başvurulan sayfa:
    - En üste taşıyın
    - Değiştirilmesi için 6 işaretçi gerektirir
  - Ama her güncelleme daha pahalı
  - Değiştirme için arama yok

Yığın Uygulaması

- En son sayfa başvurularını kaydetmek için yığın kullanımı



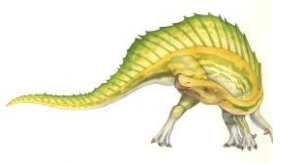


LRU Yaklaşık Algoritmalar

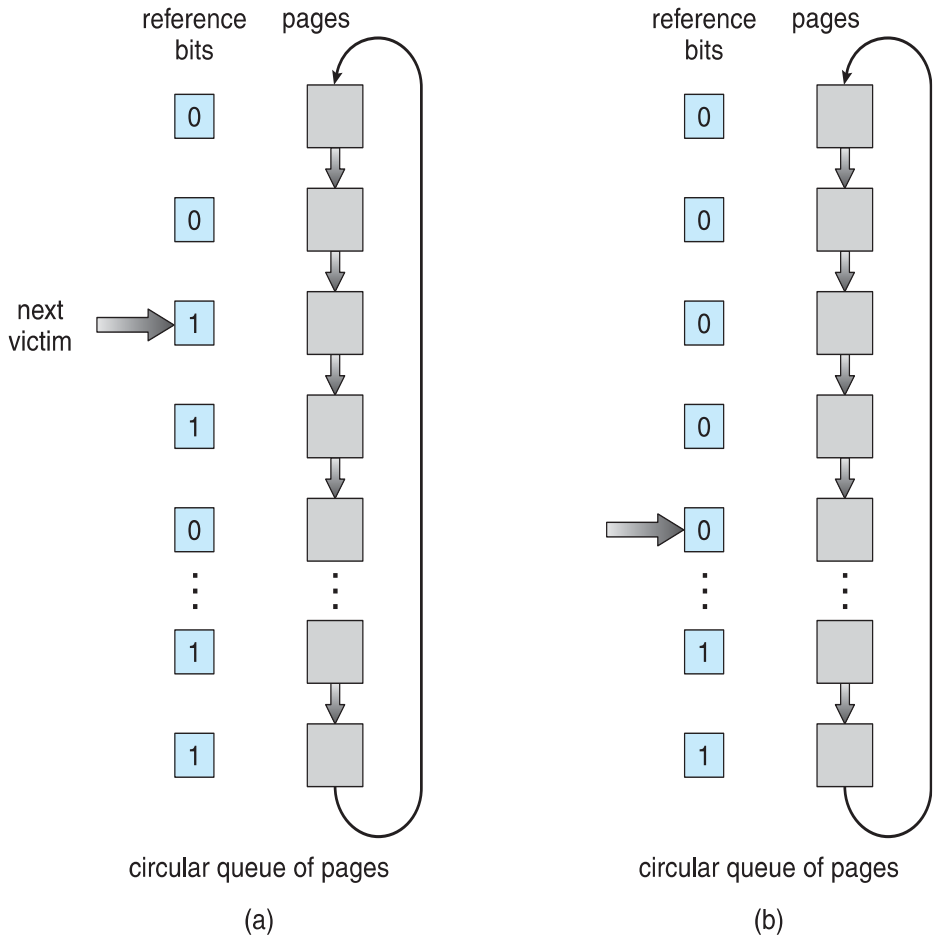
- Özel donanıma ihtiyaç duyar
- **Referans biti**
  - Her sayfa biraz ilişkilendirmek ile, başlangıçta = 0
  - Sayfa1'e ayarlanmış bit başvurulduğunda
- Herhangi bir referans biti = 0 ile değiştirin (varsa)
  - Ancak siparişi bilmiyoruz.

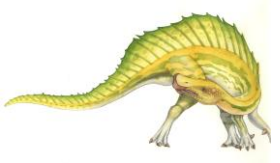
LRU Yaklaşık Algoritmalar (devam eder.)

- **İkinci şans algoritması**
  - Genellikle FIFO, artı donanım sağlanan referans biti
- **Saat** Yedek
  - Değiştirilecek sayfa
    - Referans biti = 0 -> değiştirin
    - Referans bit = 1 sonra:
      - Başvuru bitini 0'ı ayarlama, sayfayı bellekte bırakma
      - Aynı kurallara tabi olarak sonraki sayfayı değiştirme



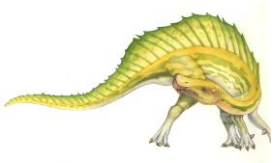
İkinci Şans Algoritması





Geliştirilmiş İkinci Şans Algoritması

- Referans bitini kullanarak algoritmayı geliştirin ve birlikte biti (varsa) değiştirin
- Sipariş edilen çifti alın (başvuru, değiştirme):
  - (0, 0) ne son zamanlarda değiştirilmiş değil kullanılan - değiştirmek için en iyi sayfa
  - (0, 1) son zamanlarda kullanılmayan ama değiştirilmiş - oldukça iyi değil, değiştirmeden önce yazmak gerekir
  - (1, 0) son zamanlarda kullanılan ama temiz - muhtemelen yakında tekrar kullanılacak
  - (1, 1) son zamanlarda kullanılan ve değiştirilmiş - muhtemelen yakında tekrar kullanılacak ve değiştirmeden önce yazmak gerekir
- Sayfa değiştirme çağrıldığında, saat düzenini kullanın, ancak dört sınıfın en düşük boş olmayan sınıfta sayfa yı kdeğiştirmeyi kullanın
  - Dairesel sırayı birkaç kez aramanız gerekebilir



Sayma Algoritmaları

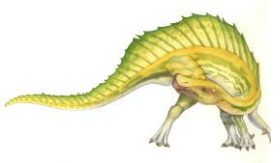
- Her sayfaya yapılan başvuru sayısının bir sayacını saklayın
  - Yaygın değil
- **Sık Kullanılan Kiralama** (**LFU**) **Algoritması**:
  - Sayfayı en küçük sayıyla değiştirir
- **En Sık Kullanılanlar** (**MFU**) **Algoritması**:
  - En küçük sayıma sahip sayfanın muhtemelen yeni getirildiği ve henüz kullanılmadığı bağımsız değişkenine dayanarak

Sayfa Arabelleğe Alma Algoritmaları

- Ücretsiz çerçevehavuzu tutun, her zaman
  - Daha sonra gerektiğinde kullanılabilir çerçeve, hata zamanında bulunamadı
  - Sayfayı ücretsiz çerçeveye okuyun ve tahliye etmek ve ücretsiz havuza eklemek için kurbanı seçin
  - Uygun olduğunda, kurbanı tahliye edin
- Büyük olasılıkla, değiştirilmiş sayfaların listesini tutma
  - Mağazayı geri alırken, sayfaları oraya yazın ve kirli olmayan
- Muhtemelen, ücretsiz çerçeve içeriğini sağlam tutun ve onlara ne olduğunu unutmayın
  - Yeniden kullanılmadan önce yeniden başvurulmuşsa, içeriği diskten yeniden yüklemenize gerek yoktur
  - Yanlış kurban çerçevesi seçilirse cezayı azaltmak için genellikle yararlıdır

Sayfa Arabelleğe Alma Algoritmaları

- Asla boş olmayan boş çerçevelerden oluşan bir havuz tutun
  - Böylece gerektiğinde bir çerçeve kullanılabilir, hata zamanında bulunamaz
  - Sayfayı ücretsiz çerçeveye okuyun ve tahliye etmek ve ücretsiz havuza eklemek için kurbanı seçin
  - Uygun olduğunda, kurbanı tahliye edin
- Büyük olasılıkla, değiştirilmiş sayfaların listesini tutma
  - Mağazayı geri alırken, sayfaları oraya yazın ve kirli olmayan
- Muhtemelen, ücretsiz çerçeve içeriğini sağlam tutun ve onlara ne olduğunu unutmayın
  - Yeniden kullanılmadan önce yeniden başvurulmuşsa, içeriği diskten yeniden yüklemenize gerek yoktur
  - Yanlış kurban çerçevesi seçilirse cezayı azaltmak için genellikle yararlıdır



Uygulamalar ve Sayfa Değiştirme

- Tüm bu algoritmalar işletim sistemi gelecekteki sayfa erişimi hakkında tahmin var
- Bazı uygulamalar daha iyi bilgiye sahip – yani, veritabanları
- Bellek yoğun uygulamalar çift arabelleğe alma neden olabilir
  - İşletim Sistemi, Sayfanın kopyalarını G/Ç arabelleği olarak bellekte tutar
  - Uygulama kendi çalışması için sayfayı bellekte tutar
- İşletim sistemi diske doğrudan erişim sağlayabilir, uygulamaların yolundan çıkmak
  - **Ham Disk** Modu
- Arabellek, kilitleme, vb. atlar.

Çerçevelerin Tahsisi

- Her sürecin ihtiyacı ***Minimum*** kare sayısı
- Örnek: IBM 370 – SS MOVE talimatını işlemek için 6 sayfa:
  - Talimat 6 bayt, 2 sayfa yayılabilir
  - Ele almak için 2 sayfa *Kaynak*
  - Ele almak için 2 sayfa *Hedef*
- ***Maksimum*** tabii ki sistemde toplam çerçeveleri
- İki büyük ayırma şeması
  - Sabit ayırma
  - Öncelik ayırma
- Birçok varyasyon

Sabit Tahsisat

- Eşit ayırma – Örneğin, 100 kare (işletim sistemi için çerçeve ayırdıktan sonra) ve 5 işlem varsa, her işleme 20 kare verin
  - Bazılarını boş çerçeve arabellek havuzu olarak tutun
- Orantılı ayırma – Sürecin büyüklüğüne göre ayırma
  - Çoklu programlama derecesi olarak dinamik, işlem boyutları değişir

- s_i = size of process p_i
- $S = \sum s_i$
- m = total number of frames
- a_i = allocation for $p_i = \frac{s_i}{S} \times m$

$m = 64$

$s_1 = 10$

$s_2 = 127$

$a_1 = \frac{10}{137} \times 62 \approx 4$

$a_2 = \frac{127}{137} \times 62 \approx 57$

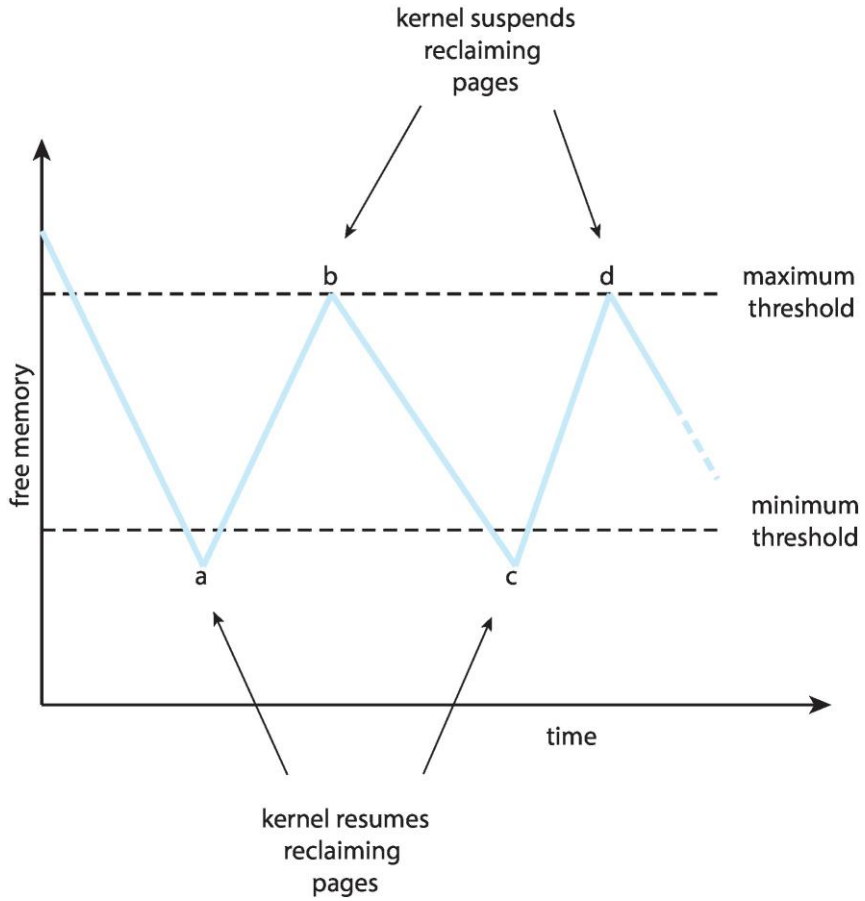
Genel ve Yerel Tahsisat

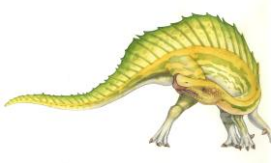
- **Küresel Yedek** – işlem tüm çerçeveler kümesinden bir yedek çerçeve seçer; bir işlem başka bir çerçeve alabilir
  - İşlem yürütme süresi büyük ölçüde değişebilir
  - Daha fazla iş fazlası çok daha yaygın olarak kullanılan
- **Yerel Yedek** – her işlem yalnızca kendi ayrılmış çerçeveler kümesinden seçer
  - İşlem başına daha tutarlı performans
  - Ama muhtemelen az kullanılan bellek
  - Bir işlemin yeterli çerçevesi yoksa ne olur?

Sayfaları Geri Alma

- Küresel sayfa değiştirme ilkesini uygulama stratejisi
- Tüm bellek istekleri, değiştirme için sayfa seçmeye başlamadan önce listenin sıfıra düşmesini beklemek yerine, serbest çerçeve listesinden karşılanır.
- Liste belirli bir eşiğin altına düştüğünde sayfa değiştirme tetiklenir.
- Bu strateji, yeni istekleri karşılamak için her zaman yeterli ücretsiz bellek olduğundan emin olmaya çalışır.

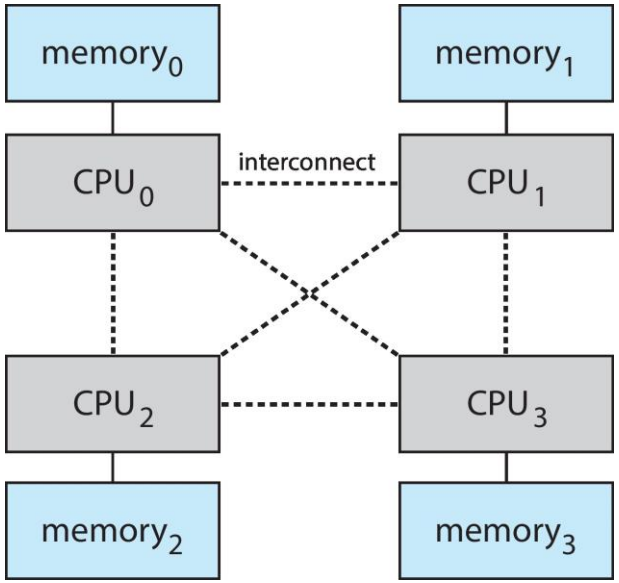
Sayfaları Geri Alma Örneği





Tekdüze Olmayan Bellek Erişimi

- Şimdiye kadar, tüm bellekeşit erişilen varsayılır
- Birçok sistem **A** – belleğe erişim hızı değişir
  - CPU'lar ve bellek içeren sistem panolarını, bir sistem veri veri çömleği üzerinden birbirine bağlı olarak göz önünde bulundurun
- NUMA çok işlemeli mimari



NUMA Erişimi (Cont.)

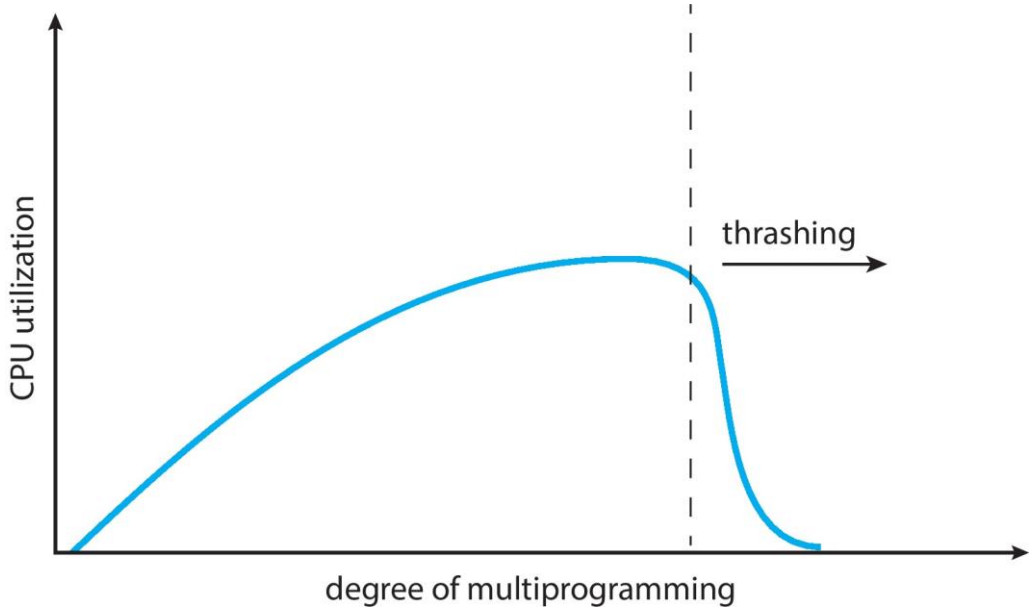
- Optimum performans, bellek tahsisinden gelir "yakın" iş parçacığının zamanlandığı CPU
  - Ve mümkün olduğunda aynı sistem panosunda iş parçacığı zamanlamak için zamanlayıcı değiştirme
  - Solaris tarafından yaratılarak çözüldü **lgroups**
    - CPU / Bellek düşük gecikme grupları izlemek için yapı
    - Zamanlamamı ve çağrı cihazımı kullandım
    - Mümkün olduğunda bir işlemin tüm iş parçacığı zamanlayın ve bu işlem için tüm bellek tahsis lgroup

Dayak

- Bir işlem yoksa "Yeterli" sayfalarda, sayfa-hata oranı çok yüksek
  - Sayfa almak için sayfa hatası
  - Varolan çerçeveyi değiştirme
  - Ama hızlı bir şekilde geri değiştirilen çerçeve gerekir
- Bu yol açar:
  - Düşük CPU kullanımı
  - Çok programlama derecesini artırmak için ihtiyacı olduğunu düşünen işletim sistemi
  - Sisteme eklenen başka bir işlem

Thrashing (Cont.)

- **Dayak**. Bir işlem sayfaları girip çıkarmakla meşguldür



Talep Sayfalama ve Thrashing

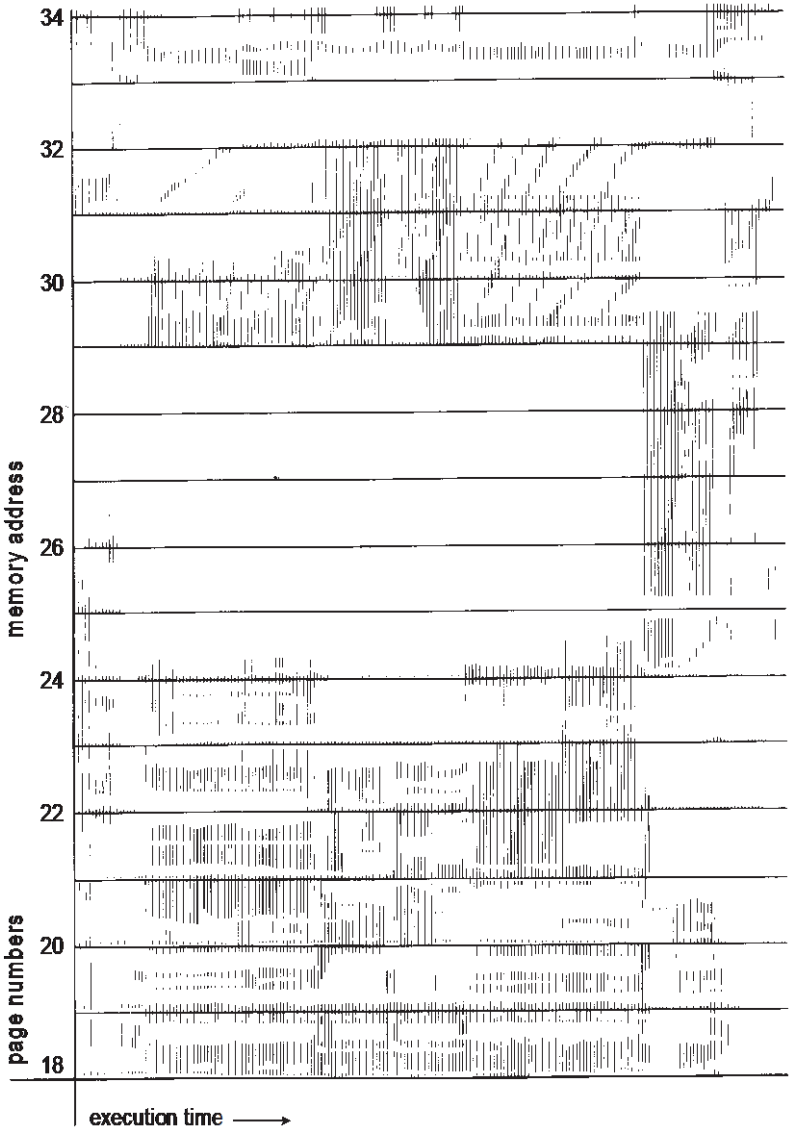
- Talep çağrıtma neden işe yarar?

  Yer Modeli

  - İşlem bir yerellikten diğerine taşınır
  - Yerellikler çakışabilir
- Dayak neden olur?

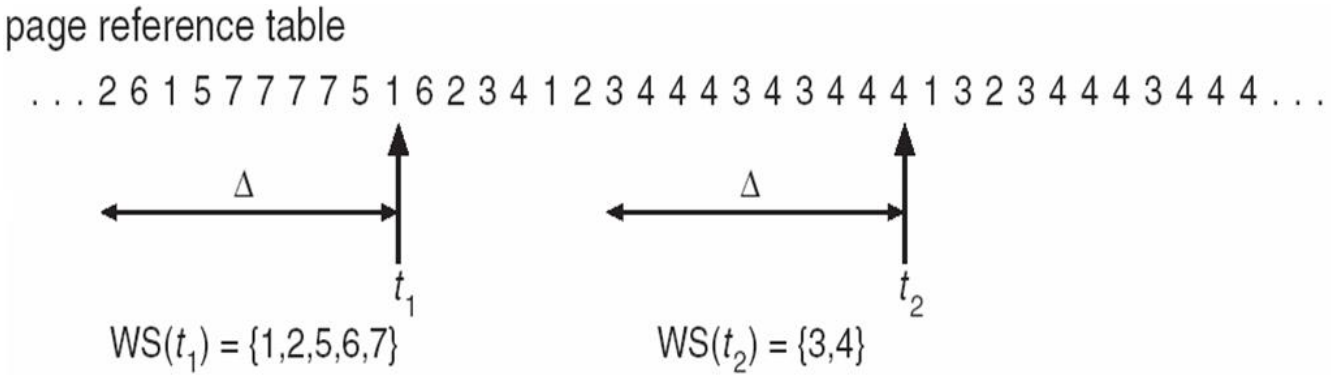
  • yerellik boyutu > toplam bellek boyutu
- Çöpatmasını önlemek için:
  - Yerelliğin boyutunu hesaplama
  - Ilkesi:
    - if > yerellik boyutu > toplam bellek boyutu→ süreçlerden birini askıya almak veya takas etmek
- Sorun: "yerellik boyutu" nasıl hesaplanır

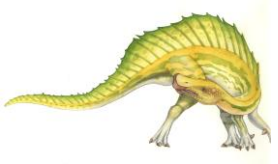
Bellek-Başvuru Desenindeki Yerellik



Çalışma Seti Modeli

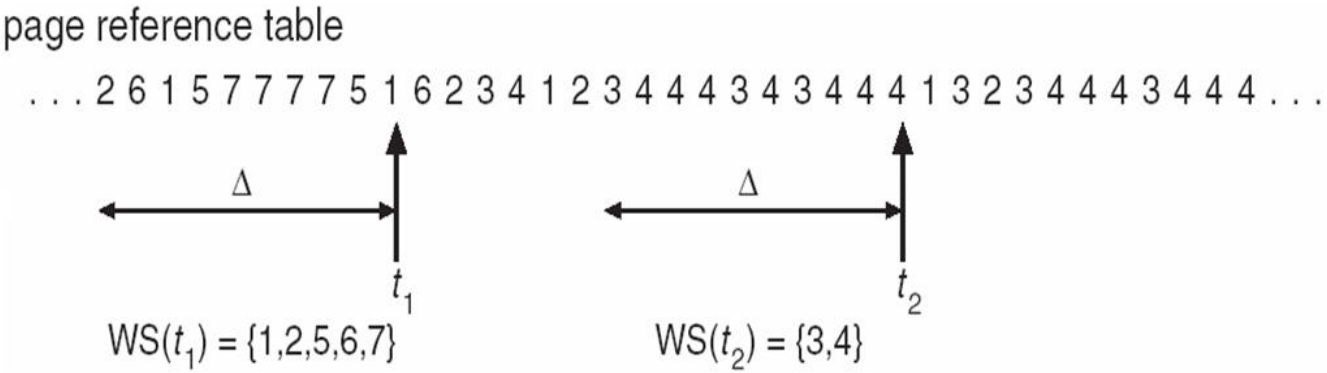
- Sabit sayıda sayfa başvuru≡ ≡ çalışma kümesi $\Delta \equiv$
  Örnek: 10.000 talimatlar
- Wss_I (Çalışma Süreci kümesi P_I) = en son Δ başvurulan toplam sayfa sayısı (zaman adedi değişir)
  - çok küçük Δ tüm yerelliği kapsamaz
  - Δ çok büyük birkaç yer kaplayacaksa
  - $\Delta = \infty \Rightarrow$ tüm programı kapsayacaksa
- Örnek



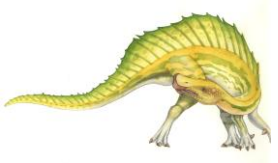


Çalışma Seti Modeli

- Sabit sayıda sayfa başvuru≡ ≡ çalışma kümesi $\Delta \equiv$
  Örnek: 10.000 talimatlar
- Wss_I (Çalışma Süreci kümesi P_I) = en son Δ başvurulan toplam sayfa sayısı (zaman adedi değişir)
  - çok küçük Δ tüm yerelliği kapsamaz
  - Δ çok büyük birkaç yer kaplayacaksa
  - $\Delta = \infty \Rightarrow$ tüm programı kapsayacaksa

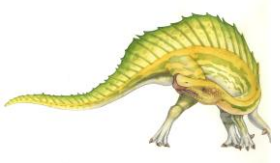


- $D = \Sigma\ Wss_I \equiv$ toplam talep çerçeveleri
  - Yerellik yaklaşık



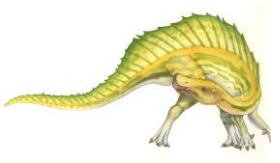
Çalışma Seti Modeli (Cont.)

- $D = \Sigma\ WSS_i \equiv$ toplam talep çerçeveleri
  - Yerellik yaklaşık
- M = toplam kare sayısı
- Eğer $D > M \Rightarrow$ Dayak
- Eğer Ilke $D >$ m, sonra askıya alma veya süreçlerden birini takas



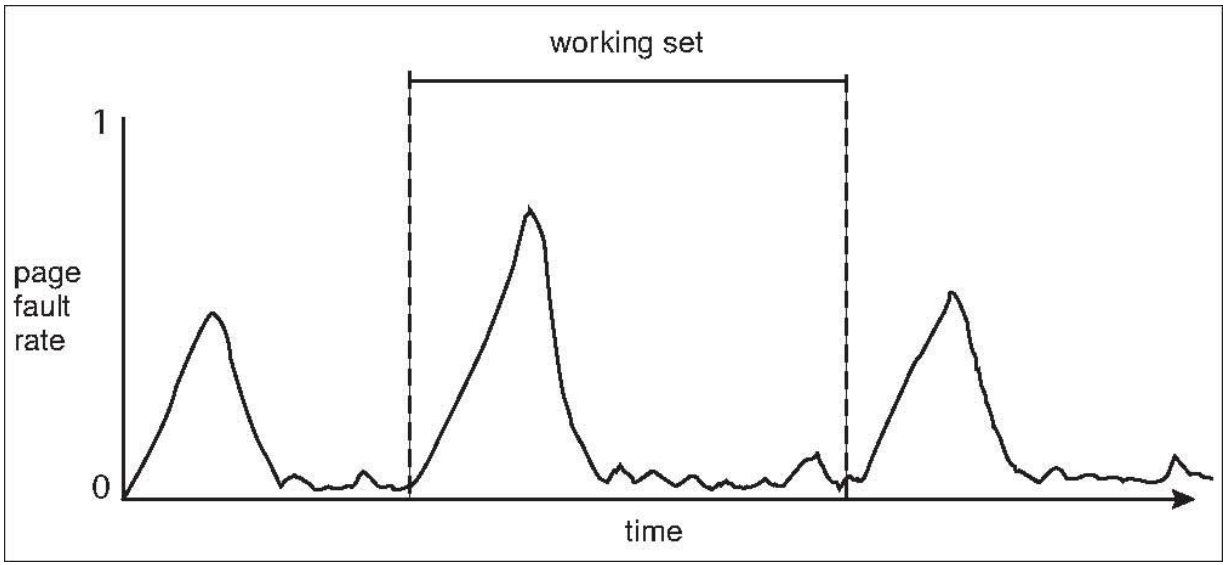
Çalışma Kümesini Takip Etmek

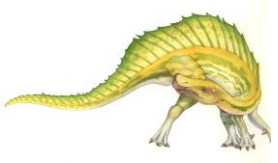
- Interval timer + bir referans biti ile yaklaşık
- Örnek: Δ = 10.000
  - Zamanlayıcı her 5000 zaman biriminden sonra kesintiye
  - Her sayfa için bellekte 2 bit tutun $I - B1_I$ Ve $B2_I$
  - Bir zamanlayıcı, başvuruyu bir B_J ve tüm referans bitlerinin değerlerini 0 olarak ayarlar
  - Eğer ya $B1_I$ Veya $B2_I$ = 1, bu Sayfa anlamına gelir I çalışma kümesinde
- Bu neden tam olarak doğru değil?
- İyileştirme = 10 bit ve her 1000 zaman biriminde kesme



Çalışma Setleri ve Sayfa Arıza Oranları

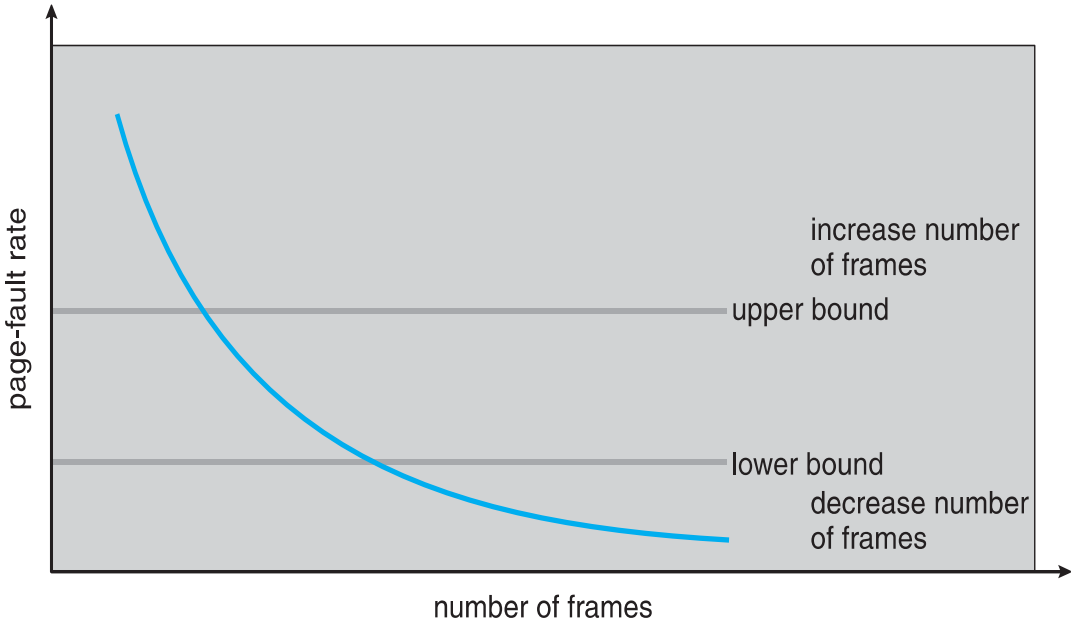
- Bir işlemin çalışma kümesi ile sayfa hata oranı arasındaki doğrudan ilişki
- Çalışma kümesi zaman içinde değişiklikleri ayarlayın
- Zaman içinde zirveler ve vadiler





Sayfa-Hata Frekans Algoritması

- WSS'den daha doğrudan yaklaşım
- Kur "Kabul edilebilir" **sayfa-hata frekansı** (**Pff**) yerel değiştirme ilkesini oranı ve kullanımı
  - Gerçek oran çok düşükse, işlem çerçeveyi kaybeder
  - Gerçek oran çok yüksekse, işlem çerçevesi kazanır



Çekirdek Belleği Ayırma

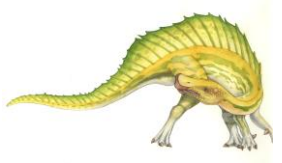
- Kullanıcı belleğinden farklı şekilde ele alındı
- Genellikle boş bellek havuzundan ayrılan
  - Çekirdek, farklı boyutlardaki yapılar için bellek ister
  - Bazı çekirdek bellek bitişik olması gerekir
    - yani, cihaz I/O için
- İki şema:
  - Buddy Sistemi
  - Levha Ayırıcı

Buddy Sistemi

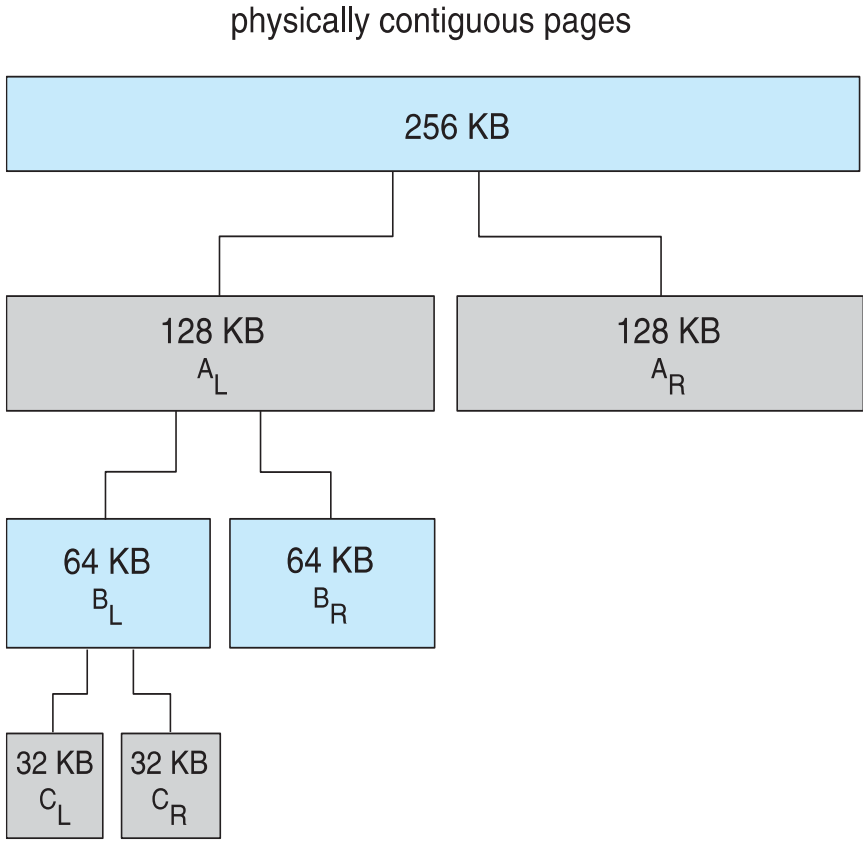
- Fiziksel olarak bitişik sayfalardan oluşan sabit boyutlu segmentten bellek ayırır
- Kullanılarak ayrılan bellek **güç-of-2 Ayırıcı**
  - 2 güç olarak boyutlandırılmış birimlerdeki istekleri karşılar
  - İstek, sonraki en yüksek 2 güce kadar yuvarlanır
  - Kullanılabilirden daha küçük ayırma gerektiğinde, geçerli öbek 2 sonraki-daha düşük güç iki arkadaş bölünmüş
    - Uygun boyutta ki parça kullanılabilir olana kadar devam edin

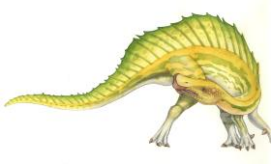
Buddy Sistem Örneği

- Varsayalım 256KB yığın kullanılabilir, çekirdek istekleri 21KB
  - A'ya Bölme$_L$ Ve A_R her biri 128KB
    - Bir daha B'ye bölündü$_L$ ve B_R ve 64KB
      - C'ye bir daha$_L$ ve C_R 32KB her - bir isteği karşılamak için kullanılan
- Avantaj – hızlı **Coalesce** kullanılmayan parçalar daha büyük bir yığın halinde
- Dezavantaj - parçalanma



Buddy Sistem Ayırıcı

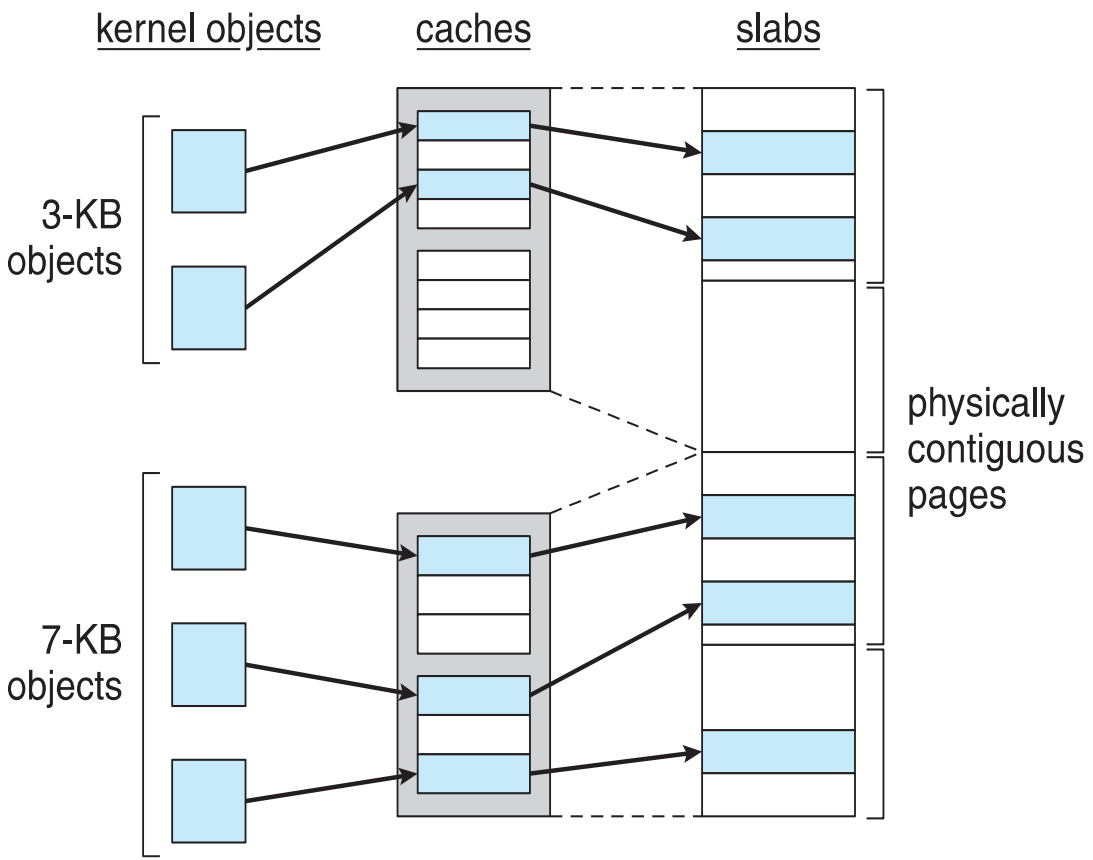


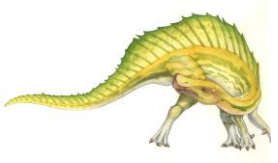


Levha Ayırıcı

- Alternatif strateji
- **Levha** bir veya daha fazla fiziksel bitişik sayfa
- **Önbellek** bir veya daha fazla levhadan oluşur
- Her benzersiz çekirdek veri yapısı için tek önbellek
  - Dolu her önbellek **Nesne** – veri yapısının anlık
- Önbellek oluşturulduğunda, olarak işaretlenmiş nesnelerle doldurulmuş **`Ücret -siz`**
- Yapılar depolandığında, nesneler **`Kullanılan`**
- Levha kullanılan nesnelerle doluysa, boş levhadan ayrılan sonraki nesne
  - Boş levha yoksa, yeni levha
- Faydaları hiçbir parçalanma, hızlı bellek isteği memnuniyeti içerir

Levha Tahsisi





Linux'ta Levha Allocator

- Örneğin, işlem tanımlayıcı türü `Yapı task_struct`
- Yaklaşık 1,7 KB bellek
- Yeni görev -> önbellekten yeni yapı ayırma
  - Mevcut ücretsiz kullanacak `Yapı task_struct`
- Levha üç olası durumda olabilir
  1. Tam – tüm kullanılan
  2. Boş – hepsi ücretsiz
  3. Kısmi – ücretsiz ve kullanılmış karışımı
- İstek üzerine, levha ayırıcı
  1. Kısmi levhada ücretsiz yapı kullanır
  2. Yoksa, boş levhadan bir tane alır
  3. Boş levha yoksa, yeni boş

Linux'ta Levha Allocator (Cont.)

- Levha Solaris başladı, şimdi çeşitli OSes hem çekirdek modu ve kullanıcı belleği için yaygın
- Linux 2.2 SLAB vardı, şimdi hem SLOB ve SLUB ayırıcılar vardır
  - Sınırlı belleğe sahip sistemler için SLOB
    - Basit BlokListesi – küçük, orta ve büyük nesneler için 3 liste nesnesi tutar
  - SLUB performans için optimize edilmiş SLAB cpu başına sıraları kaldırır, sayfa yapısında depolanan meta veriler

Diğer Hususlar

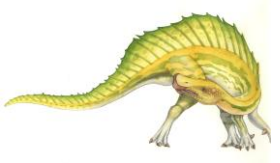
- Hazırlık
- Sayfa boyutu
- TLB erişim
- Ters sayfa tablosu
- Program yapısı
- G/Ç kilitleme ve sayfa kilitleme

Hazırlık

- İşlem başlatma işleminde oluşan çok sayıda sayfa hatasını azaltmak için
- Başvurulmadan önce bir işlemin gereksinim edeceği sayfaların tümü veya bir kısmı ön sayfa
- Ancak prepaged sayfaları kullanılmamışsa, G/Ç ve bellek boşa gitti
- Varsay -alım *S* sayfalar hazırlanır ve *A* sayfaların kullanılır
  - Soru: maliyeti ***s * A*** sayfaları kaydetmek hataları hazırlık maliyeti nden daha büyük veya daha azdır ***s * (1- A)*** gereksiz sayfalar?
  - Eğer ***A*** 0'a yakın $\Rightarrow$ hazırlık kaybeder
  - Eğer ***A*** 1'e yakın $\Rightarrow$ hazırlık galibiyetleri

Sayfa Boyutu

- Bazen işletim sistemi tasarımcılarının bir seçeneği vardır
  - Özellikle özel olarak oluşturulmuş CPU'da çalışıyorsanız
- Sayfa boyutu seçimi dikkate alınmalıdır:
  - Parçalanma
  - Sayfa tablo boyutu
  - **Çözünürlük**
  - G/Ç genel ilerli
  - Sayfa hatalarının sayısı
  - Yer
  - TLB boyutu ve etkinliği
- Her zaman güç 2, genellikle aralık 2^{12} (4.096 bayt) için 2^{22} (4.194.304 bayt)
- Ortalama olarak, zaman içinde büyüyen



TLB Erişim

- TLB Erişim - TLB'den erişilebilen bellek miktarı
- TLB Erişim = (TLB Boyutu) X (Sayfa Boyutu)
- İdeal olarak, her işlemin çalışma kümesi TLB'de depolanır
  - Aksi takdirde sayfa hataları yüksek derecede var
- Sayfa Boyutunu Artırma
  - Tüm uygulamalar büyük bir sayfa boyutu gerektirmeyen bu parçalanma bir artışa yol açabilir
- Birden Çok Sayfa Boyutu Sağlayın
  - Bu, daha büyük sayfa boyutları gerektiren uygulamaların parçalanmada bir artış olmadan bunları kullanma olanağı sağlar

Program Yapısı

- Program yapısı
  - `int[128,128] verileri;`
  - Her satır bir sayfada depolanır
  - Program 1

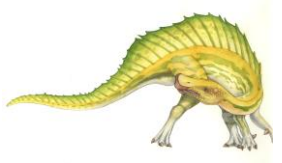
```
for (j = 0; j <128; j++)
    for (i = 0; in < 128; i++)
        veri[i,j] = 0;
```

128 x 128 = 16.384 sayfa hatası

  - Program 2

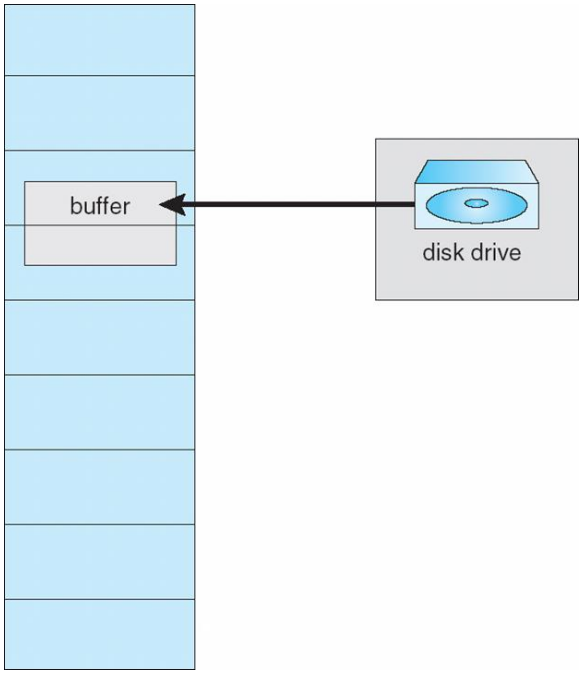
```
for (i = 0; in < 128; i++)
    for (j = 0; j < 128; j++)
        veri[i,j] = 0;
```

128 sayfa hataları



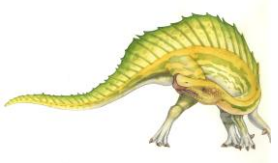
G/O kilitleme

- **I/O Kilitleme** – Sayfalar bazen belleğe kilitlenmelidir
- G/Ç'yi göz önünde bulundurun - Bir aygıttan dosyakopyalamak için kullanılan sayfalar, sayfa değiştirme algoritması tarafından tahliye için seçilmeden kilitlenmelidir
- **Sabitleme** bellek içine kilitlemek için sayfaların



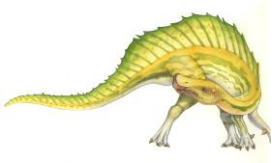
İşletim Sistemi Örnekleri

- Windows
- Solaris



Windows

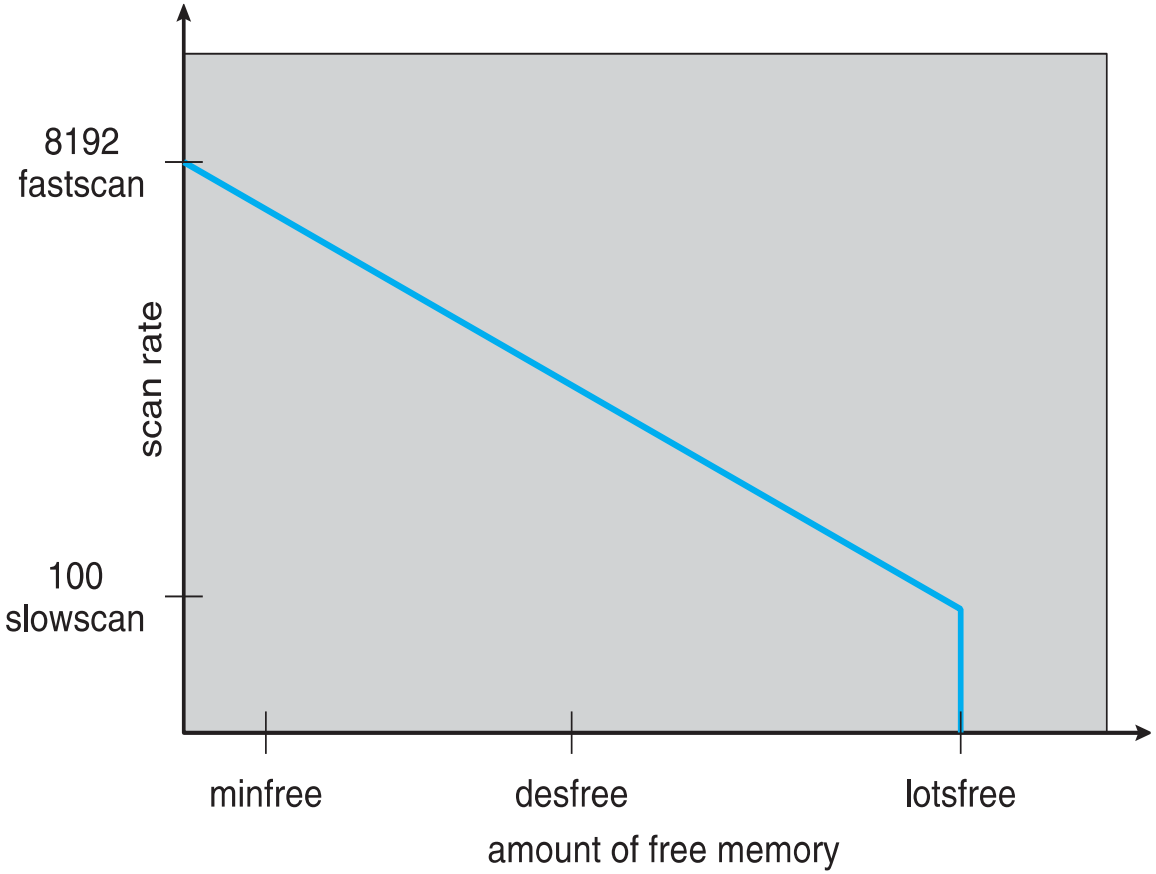
- Talep çağrılarını kullanır **Kümeleme**. Kümeleme, hata sayfasının çevresindeki sayfaları getirir
- İşlemler atanır **çalışma seti minimum** Ve **çalışma seti maksimum**
- Çalışma kümesi minimum, işlemin bellekte olması garanti edilen minimum sayfa sayısıdır
- Bir işlem, çalışma kümesi maksimumuna kadar birçok sayfa olarak atanabilir
- Sistemdeki boş bellek miktarı bir eşiğin altına düştüğünde**, otomatik çalışma seti kırpma** boş bellek miktarını geri yüklemek için gerçekleştirilir
- Çalışma kümesi kırpma, sayfaları çalışma kümesi minimumundan fazla sayfaları olan işlemlerden kaldırır



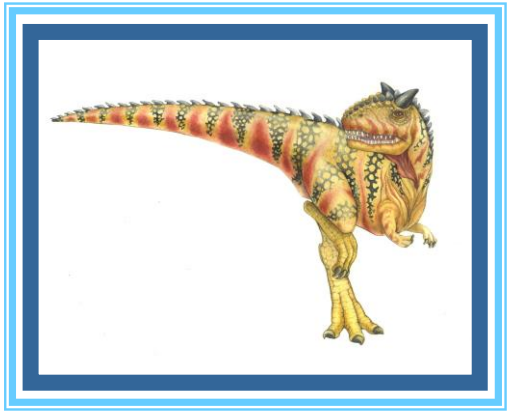
Solaris

- Hata lama işlemlerini atamak için boş sayfaların listesini tutar
- **`Lotsfree`** – eşik parametresi (boş bellek miktarı) sayfalama başlatmak için
- **`Desfree`** – artan sayfalama için eşik parametresi
- **`Minfree`** – eşik parametresi takas olmak
- Sayfalama tarafından gerçekleştirilir **`pageout`** Işlem
- **`Pageout`** değiştirilmiş saat algoritması kullanarak sayfaları tarar
- **`Tcanrate`** sayfaların taranma oranıdır. Bu aralıkları **`slowscan`** Hedef **`oruç`**
- **`Pageout`** kullanılabilir boş bellek miktarına bağlı olarak daha sık denir
- ***Öncelikli sayfalama*** işlem kodu sayfalarına öncelik verir

Solaris 2 Sayfa Tarayıcı



Bölüm 10'un Sonu

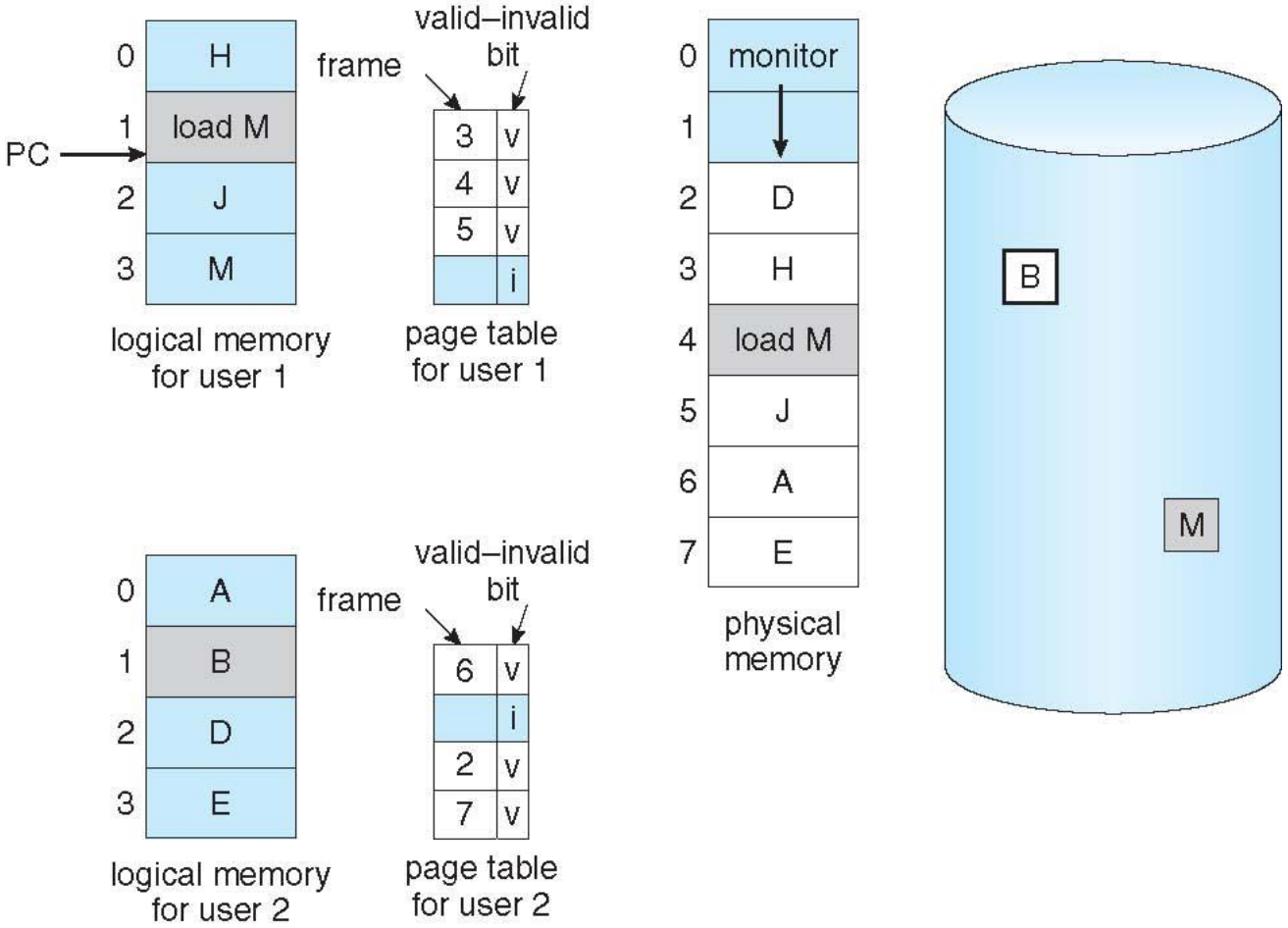


Talep Sayfalama Performansı

- Talep Sayfalama aşamaları (kötü durumda)

1. İşletim sistemine tuzak
2. Kullanıcı kayıtları ve işlem durumunu kaydetme
3. Kesmenin bir sayfa hatası olduğunu belirleme
4. Sayfa referansının yasal olup olmadığını kontrol edin ve diskteki sayfanın konumunu belirleyin
5. Diskten ücretsiz bir çerçeveye okuma sorunu:
    1. Okuma isteği servis edilene kadar bu aygıt için kuyrukta bekleyin
    2. Aygıtı arama ve/veya gecikme süresini bekleyin
    3. Sayfanın ücretsiz bir çerçeveye aktarılmasını başlatma
6. Beklerken, CPU'yu başka bir kullanıcıya tahsis edin
7. Disk G/Ç alt sisteminden bir kesme alma (G/Ç tamamlandı)
8. Kayıtları ve işlem durumunu diğer kullanıcıya kaydetme
9. Kesmenin diskten olduğunu belirleme
10. Sayfayı ve diğer tabloları doğrula, sayfanın artık bellekte olduğunu göstermek için
11. CPU'nun bu işleme yeniden ayrılmasını bekleyin
12. Kullanıcı kayıtları, işlem durumu ve yeni sayfa tablosugeri yükleme ve sonra kesintiye talimat devam

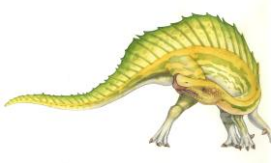
Sayfa Değiştirme İhtiyacı



Öncelik Tahsisi

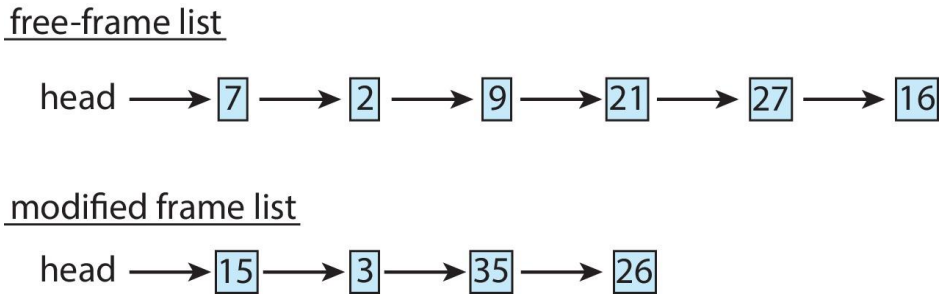
- Boyut yerine öncelikleri kullanarak orantılı ayırma şeması kullanma

- Eğer işlem P_i bir sayfa hatası oluşturur,
  - çerçevelerinden birini değiştirme için seçin
  - daha düşük öncelik numarasına sahip bir işlemden bir çerçevenin değiştirilmesi için seçin



Bellek Sıkıştırma

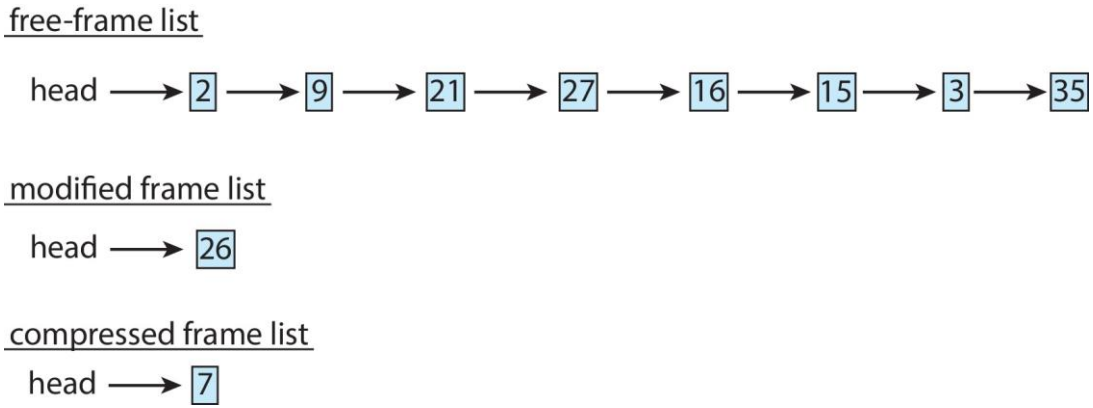
- **Bellek sıkıştırma** -- yer değiştirmek için değiştirilmiş çerçeveleri ayırmak yerine, birkaç kareyi tek bir kareye sıkıştırArak sistemin sayfaları değiştirmeye başvurmadan bellek kullanımını azaltmasını sağlıyoruz.
- 6 kareden oluşan aşağıdaki serbest çerçeve listesini göz önünde bulundurun

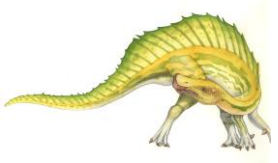


- Bu boş kare sayısının sayfa değiştirmeyi tetikleyen belirli bir eşiğin altına düştüğünü varsayalım. Değiştirme algoritması (örneğin, bir LRU yaklaşım algoritması) serbest kare listesine yerleştirmek için 15, 3, 35 ve 26 olmak üzere dört kare seçer. Bu çerçeveleri ilk olarak değiştirilmiş kare listesine yerleştirir. Genellikle, değiştirilen kare listesi sonraki boşluk takas için yazılır, çerçeveleri serbest kare listesine kullanılabilir hale. Alternatif bir strateji çerçeveleri bir dizi sıkıştırmak için{\mdash}demek, üç{\mdash}ve sıkıştırılmış sürümlerini tek bir sayfa çerçevesi olarak saklayın.

Bellek Sıkıştırma (Cont.)

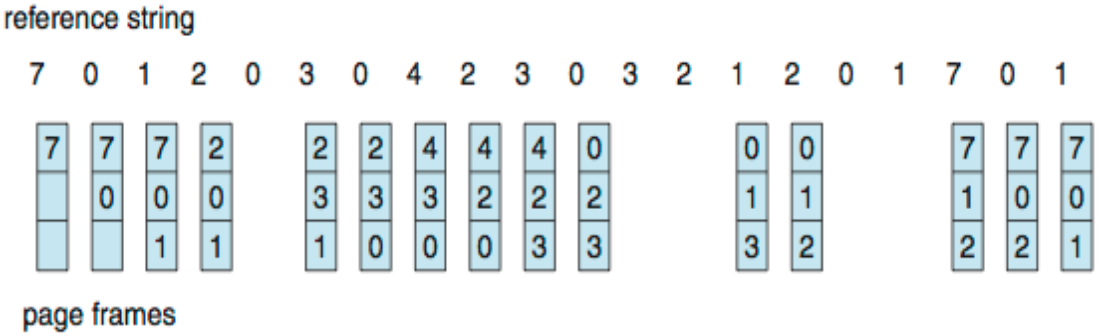
- Sayfalama nın alternatifi, **bellek sıkıştırma**.
- Yer değiştirmek için değiştirilmiş çerçeveleri ayırmak yerine, birkaç kareyi tek bir çerçeveye sıkıştırArak sistemin sayfaları değiştirmeye başvurmadan bellek kullanımını azaltmasını sağlıyoruz.





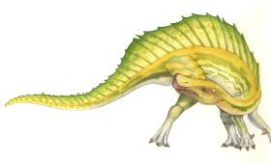
İlk İlk Çıkış (FIFO) Algoritması

- Başvuru dizesi: **7,0,1,2,0,3,0,4,2,3,0,3,0,3,2,1,2,0,1,7,0,1**
- 3 kare (her işlem de 3 sayfa bellekte olabilir)

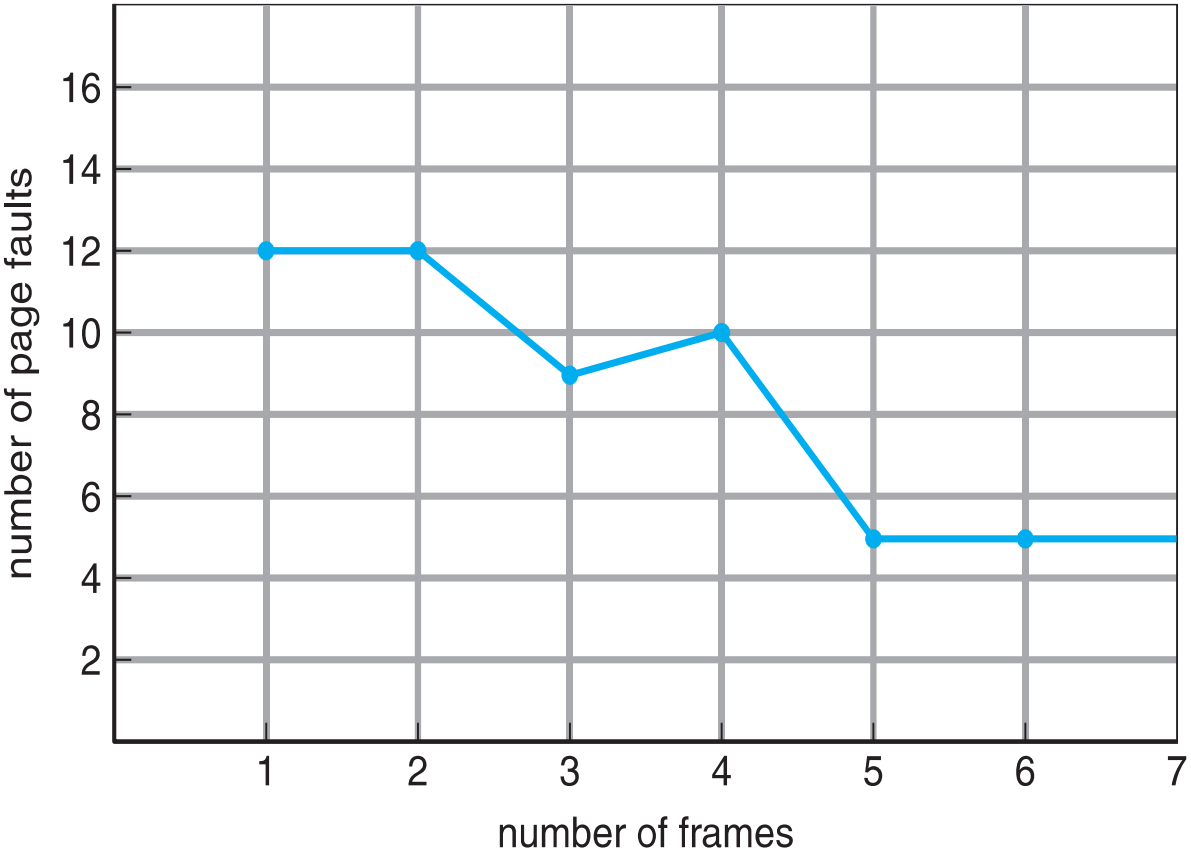


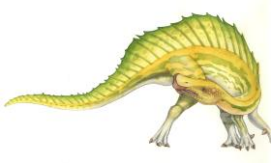
15 sayfa hataları

- Referans dizegöre değişebilir: 1,2,3,4,1,2,5,1,2,3,4,5 düşünün
  - Daha fazla kare eklemek daha fazla sayfa hatasına neden olabilir!
    - **Belady's Anomali**
- Sayfaların yaşları nasıl izlenir?
  - FIFO kuyruğu kullanman



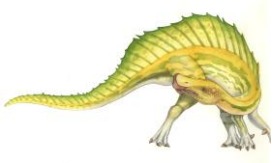
FIFO Belady Gösteren's Anomali





LRU Algoritma Uygulaması

- Karşı uygulama
  - Her sayfa girişinin bir sayacı vardır; bu giriş üzerinden her sayfa başvurulsa, saatin değerini sayacın içine kopyalayın
  - Bir sayfanın değiştirilmesi gerektiğinde, en küçük değeri bulmak için sayaçlara bakın
    - Tabloda arama yapılması gereklidir
- Yığın uygulaması
  - Sayfa numaralarını çift bağlantı formunda tutun:
  - Başvurulan sayfa:
    - En üste taşıyın
    - Değiştirilmesi için 6 işaretçi gerektirir
  - Ama her güncelleme daha pahalı
  - Değiştirme için arama yok



Çalışma Kümesini Takip Etmek

- Interval timer + bir referans biti ile yaklaşık
- Örnek: Δ = 10.000
  - Zamanlayıcı her 5000 zaman biriminden sonra kesintiye
  - Her sayfa için bellekte 2 bit tutun
  - Zamanlayıcı kopyayı kesintiye uğrattığında ve tüm başvuru bitlerinin değerlerini 0 olarak ayarladığında
  - Bellekteki bitlerden biri = 1 $\Rightarrow$ sayfa çalışma kümesinde
- Bu neden tam olarak doğru değil?
- İyileştirme = 10 bit ve her 1000 zaman biriminde kesme

Buddy Sistemi

- Fiziksel olarak bitişik sayfalardan oluşan sabit boyutlu segmentten bellek ayırır
- Kullanılarak ayrılan bellek **güç-of-2 Ayırıcı**
  - 2 güç olarak boyutlandırılmış birimlerdeki istekleri karşılar
  - İstek, sonraki en yüksek 2 güce kadar yuvarlanır
  - Kullanılabilirden daha küçük ayırma gerektiğinde, geçerli öbek 2 sonraki-daha düşük güç iki arkadaş bölünmüş
    - Uygun boyutta ki parça kullanılabilir olana kadar devam edin
- Örneğin, 256KB yığın kullanılabilir varsayalım, çekirdek istekleri 21KB
  - A'ya Bölme$_L$ Ve A_R her biri 128KB
    - Bir daha B'ye bölündü$_L$ ve B_R ve 64KB
      - C'ye bir daha$_L$ ve C_R 32KB her - bir isteği karşılamak için kullanılan
- Avantaj – hızlı **Coalesce** kullanılmayan parçalar daha büyük bir yığın halinde
- Dezavantaj - parçalanma