

İŞLETİM SİSTEMİ

- Kaynak yöneticisidir.
- Her program bu kaynaktan hak ettiği kodu zaman ve bellek alanını alır.

İŞLETİM SİSTEMİ KONSEPTLERİ

Kernel: Ana işletim sistemi programıdır (gerekirdeki)

Program: Diske kayıtlı static dosyadır.

Process: Programın koşar halidir.

Device Sürücüler: Aynı tür cihazlara aynı türde arayüze erişmeyi sağlayan

Trap: Sistem çağrıları ile kontrolün OS geçmesinden Trap olduğu zaman program kullanıcının seviyesinden yukarı ve işletim seviyesine geçen

- Paralel diskte erişim hızı daha fazladır
- Programlar bellekten kaçınılmaz Bellekten cache'ye yüklenir ordu kaçırılır

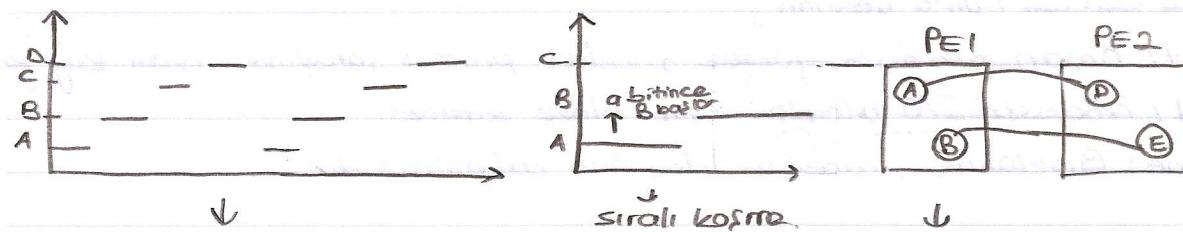
PROCESS (SÜRECLER)

Programın CPU zamanını tüketen (koşan) haline process denir.

CPU kullanılır ve cache'de koşar.

- Farklı işlemci üzerindeki paralel koşma

- Aynı işlemci üzerindeki concurrent koşma (zaman paylaşımı)



Aynı işlemci üzerinde, zaman paylaşımı parallel koşme

! Processlerin koşması için PC şarttır. Her process'e ait bir PC vardır. Değişik processler bir PC'yi kullanarak da koşabilirler.

PC içeriği process'in nerde kaldığını, nerden itibaren koşucuının belirtir.

PROCESSLERİN CPU KULLANIMI

Zaman Paylaşımı (Concurrent) Koşma: A process'in CPU'yu kullanırken switching olursa CPU kullanımı B processine geçer. PC içeriği buna bağlı değişir. Bu zaman paylaşımı koşmadır.

* Aynı işlemci üzerinde processler zaman paylaşımı koşar.

Sıralı Koşma: Bir program koşmaya başlar ve biter. Sonra diğer program koşar.

~ Zaman paylaşımı koşmadır CPU belli aralıklara bölündür ve her program bu aralıklara paylastırılır. Aynı anda tüm processler koşuyor gönülde

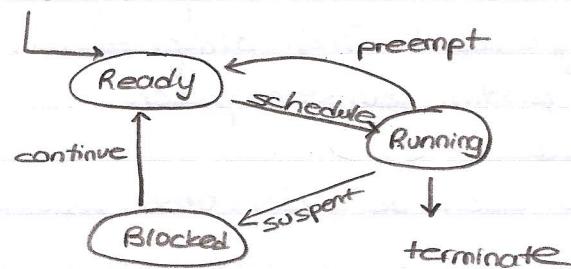
PROCESS SWITCHING (ANAHTARLAMA)

A process'inin durdurulup B'ye geçirmesidir. Bu işlemi yapan schedulerdir. PCB'ler saklanır. Bir sonraki process neye onun PCB'si eriği yoktur.

CPU'da kalan process'in gerçek döngüsünü save edip kapatık olan process'in gerçek döngüsünü restore ediyora.

* Kendini modify eden kodlar paylaşılmaz.

create



Ready: Hazırlanmış process'i tanıfeleyici sefer ve kosar hole getirin.

Running: CPU zamanı tüketen program (kod)

Blocked: Kalan process'in I/O işlemleri için askıya alınmasıdır. İşlemler bitene kadar.

Create: Prosesin oluşturulması

Terminate: Hata sonucu veya zorla prosesin sonlanması

Schedule: Hazır kuyrukundaki processlerden birini öncelikle göre sefer ve ready leri → run'a getirin

Preempt: Kalan process'in yüksek öncelikli process tarafından hazır kuy. geçebilir.

Suspend: Prosesin I/O işlemleri için bloke edilmesi

Continue: Bloklanmış prosesin bloğunun kaldırılmasıdır.

İsletim sisteminin 2 ana fonksiyonu

1- Arayüz olmak

2- Etkin ve güvenli kaynak yöneticisi olmak

MEMORY MANAGEMENT

En önemli 1. kaynak CPU, 2. kaynak bellek

TERIMLER

- contiguous : Programları belleğe ardışılı sıradır yerlestirmektedir.
- non-contiguous : Sıralı olmayan yerleştirme türü
- real memory : Gerçek, fiziksel, RAM ile sınırlı bellektir
- virtual memory : Mantıksal, sanal bellek
- logical : Software'ın bildiği şeyler
- physical : Hardware'ın bildiği şeyler
- block : Diskteki en küçük ve tipidir.

FRAGMENTATION : Bellegin küçük parçalarla bölünmesinden sonraki yaratılan bir durumdur. Fragmentation istenmez.

At idealde bellek

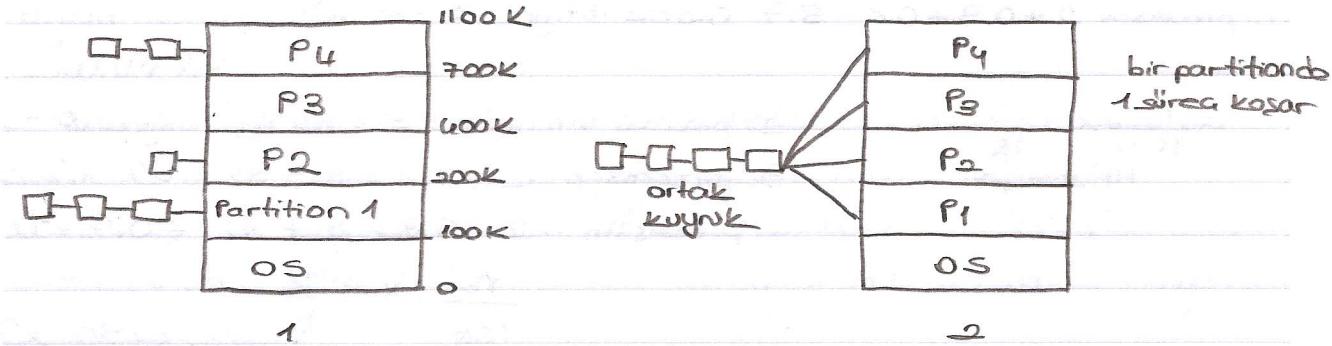
• büyük

• hızlı ve bilgilerin kalıcı olarak tutulduğu ortam olmalıdır

BASIT BELLEK YÖNETİMİ

MULTIPROGRAMMING : Çok kullanıcılı

Bellek partitionlara bölünmüştür. Bir partitionda 1 process çalışabilir



1. Bu durumda 4 process concurrent olarak çalışabiliyor. Küçük ve büyük processler ayrılmış boyutuna uygun partitionlarda oluşturulur. Anna P3 kısmına uygun process olmadığından o alan boş kalır. istenmeyen durum.

2. Bu durumda processin boyutuna bakılmaksızın sıralı koşarlanır veimsiz. Büyüklük bellek alanına kavak process atılabılır.

* Nekadar çok süreli kullanılsa CPU kullanım, o kadar artar.

* I/O yoğunluğu büyük olan processlerden daha fazla kullanılmaya.

CPU KULLANIMI

$$\text{CPU}_{\text{util}} = 1 - p^n \xrightarrow{\text{Process sayısı}} \rightarrow \text{Aynı I/O oranına sahip}$$

$\hookdownarrow \text{I/O oranı}$

$$= 1 - (0.8)^1 = 0.2$$

$$= 1 - (0.8)^2 = 0.36 \xrightarrow{\text{Process boyuna düşen işlenme kullanım}}$$

DR

İS	Başlangıç zamani	CPU zamanı ihciyacı
1	10:00	4
2	10:10	3
3	10:15	2
4	10:20	2

	1	2	3	4
I/O (başlangıç)	0.80	0.64	0.512	0.4096
CPU mesguliyet	0.20	0.36	0.488	0.5904
CPU/process	0.20	0.18	0.162	0.1476

process basına düşer

$$1 \text{ process } 0.8 = 1 - (0.8)^1 = 0.2$$

$$2 \text{ process } 0.64 = 1 - (0.8)^2 = 0.36$$

$$3 \text{ process } 0.512 = 1 - (0.8)^3 = 0.488/3$$

$$4 \text{ pro. } 0.4096 = 1 - (0.8)^4 = 0.5904$$

1.	2 dk	0.9 dk	0.8 dk	0.3	✓
2.	$\frac{2}{10} \cdot 10 = 2$	0.9 dk	0.8 dk	0.3	0.9
3.	$\frac{18}{100} \cdot 10 = 1.8$	0.8 dk	0.3	0.9	✓
4.	$\frac{16}{100} \cdot 10 = 1.6$	0.16 dk = 0.8	0.3	0.9	0.1

10.00 10.00 10.15 10.20 10.22 10.27 10.32 10:35

$$1. \text{ process} = 2 + 0.9 + 0.8 = 3.7 \text{ 0.3 dk ihtiyac var}$$

$$\frac{14}{100} \cdot x = \frac{3}{10}$$

$$14 \cdot x = 30$$

$$x \approx 2$$

$$2. \text{ process'in } 3 - 2 = 1 \text{ dk'ya ihtiyac var} = 0.1$$

$$3. \text{ process'in } 2 - 1.1 = 0.9 \text{ dk } \checkmark$$

$$4. \text{ process'in } 2 - 0.3 = 1.7 \text{ dk } - 0.9 = 0.8$$

$$\frac{16}{100} \cdot x = \frac{9}{10}$$

$$16 \cdot x = 90 \quad x \approx 5.6$$

$$2+4+2 \text{ process kaldi: } \frac{18}{100} \cdot x = \frac{1}{10} \quad 18x = 10 \quad x \approx 0.5$$

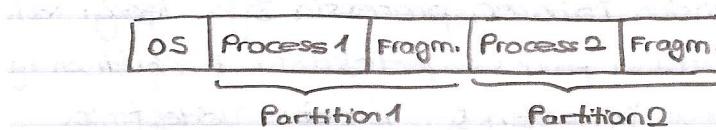
$$4. \rightarrow 1 \text{ proces kaldi: } \frac{2}{10} \cdot x = \frac{7}{10} \quad x = 3.5$$



** Eğer farklı I/O zamanı verilmemiş processler kullanılırsa
aynı zamanda CPU kullanımının gereklilikleri olursa, sonra 1 den azdır.

$$CPU_{util} = 1 - p^n \rightarrow 1 - p \cdot p \cdot p$$

SABİT PARTİTİON İLE GÖKLU PROGRAMLAMA



Partition sayısı ve boyutları user tarafından belirlenir.

- Sabit alanlara processler yerlesir. Boş kalan alanlar fragmentation olur.
- Boş alan kaybı. Deadlock fragmentation.

DİNAMİK PARTİTİON:

- Alanlar processlere için sabit değil. Yüklenme anında processe göre alanlar oluşturulur.
- Boş olan belleğe 3 process yerleştirildi. Belli süre sonra 2. process bitti. 2. processin boyutundan daha büyük boyutta ise 3. process relocate yapılıarak sola taşındı ve yer aldı. Fragmentation ortadan kalktı.

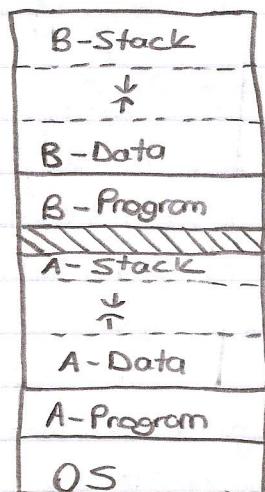
BELLEK YÖNETİMİNDE ORTAYA GİKAN PROB.

DİNAMİK PARTİTİON PROBLEMLERİ

- Fragmentation bellek alanının fazladan oluşturulup kullanılması sonucu olur. Bu relocate edilerek çözülebilir ama maliyeti yüksektir.
- Boş bellek alanlarına processler yerleştirilmelidir. Bunun için placement algoritmaları kullanılır.

SWAPPING :

Genistereye nüśart programlar için bellekte extra yer açıyor.
Genistere yörlerini aynı ise 2 programın bellek alanı paylaşılabilir.
GÖZÜM → Her program için zıt yönde genişleyen 2 adet alanının stack ve data kısmı. Çalışma anında ikisi aynı anda genişlemez. Denge sağlanır.

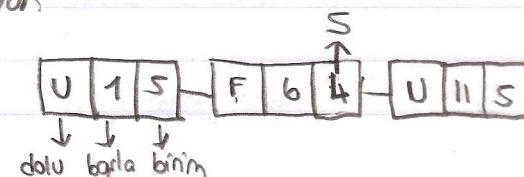
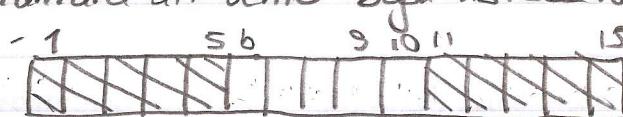


** Processler relocate edilirken dikkat etmelidir.
Bellekte hangi alanın dolu hangisinin boş olduğunu bilmemiz gereklidir. Boş olduğundan emin olmalıdır.
Yoksa ven kaybı olur.

KULLANILMAYAN BELLEK ALANI YÖNETİMİ

BIT-MAP : Bellek sabit birimler bölünür her bölmeye bir bit atonur. 0 ise boş 1 ise doludur. Çok zaman alan bir yöntemdir. Tüm alanlar taranır.

LINKED LIST : Bellekte kullanılan ve kullanılmayan alanlara ait veniler birlikte listede tutulur.



PLACEMENT (YERLEŞTİRME) ALGORİTMASI

- FIRST FIT : Bellegin en başından itibaren processin sigabileceği alan aranır. Bulunan ilk en uygun alana process yerleştirilir. En hizli en iyi.
- NEXT FIT : Bellegin başından başlanmadı. En son process yerleştirilirken alandan itibaren arama yapılır.
- BEST FIT : Bu yöntemde process en uygun alana yerleştirilmeye çalışılır. M En kötü yönünden. Çok zaman alır ve fragmentation sebebi olur.
- BUDDY SYSTEM : Beltek alanları 2^n 'nin katları şeklinde ayrılır. Fragmentasyon olur. Process tüm bellegi kaplayacaksı yerleştirilir. Eğer daha küçük ve uygun olan gelere kadar 2^i 'ye bölgü. İşibiter olan diğer boş alanlar kapatılır.

BELLEK YÖNETİMİ PROBLEMLERİ

- 1. Fragmentation yüzünden kullanılmayan alanlar var.
- 2. Programın hiç kullanmayacağı kısımları belleğe yükleniyor.
- 3. Process size fiziksel bellek alanı ile sınırlı. İstilen boyutta program yaratılamaz.

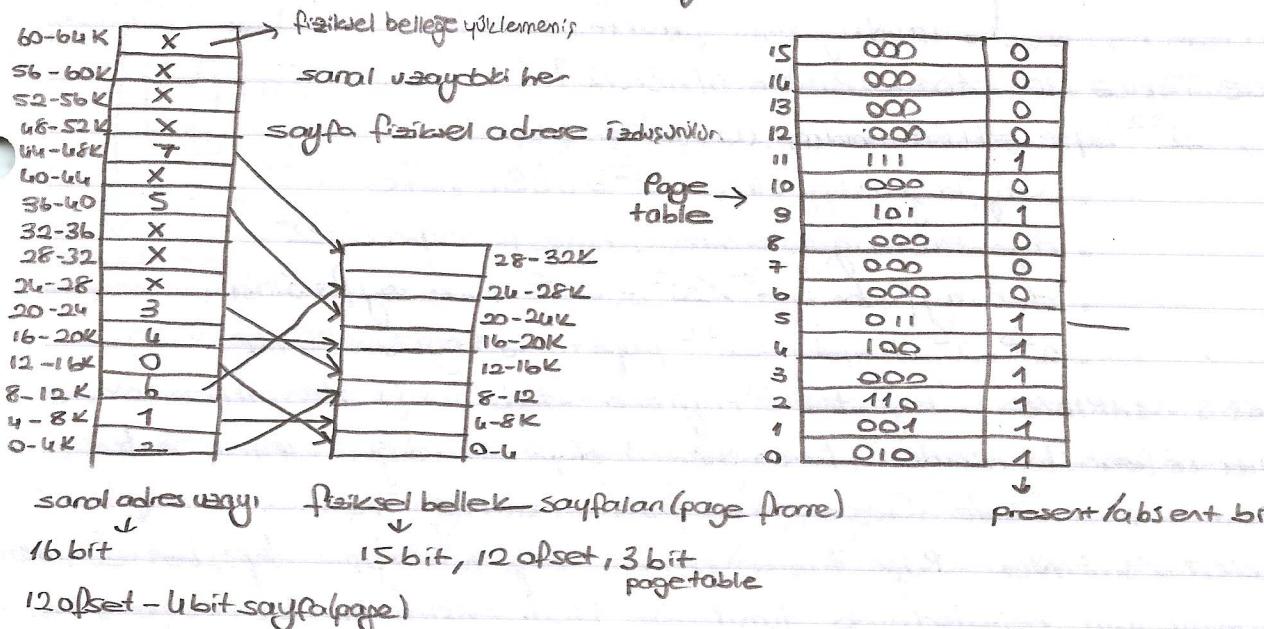
ABAZÜM ~ VIRTUAL MEMORY (tüm sonuclar)

VIRTUAL MEMORY

- Segmentli
- Sayfali
- Segmentli Sayfali Yapılan söz konusudur

VIRTUAL MEMORY

- ✓ Disk Üstündedir RAM + Disk
- ✓ Varsayılan bir bellektir
- ✓ Sanal bellek size sınırlı değil (CPU'nun adres busundaki bit) (sayısıdır. Bu adres sanal adresdir)
- .. Bir sonraki adres referanslamaların VM'de yapılır. Sonra fiziksel adrese dönüştür.



ÖR 20-24 K sanal adresi fiziksel way'a döndürülür.

32K 16K 8K 4K
 $\underbrace{0 \ 1 \ 0 \ 1}_{5' i fode ediyor} \ | \ \underbrace{0 \ 0 \ 0 \ 0 \ 0 \ 0 \ 0 \ 0 \ 0 \ 0 \ 0 \ 0}_{\text{offset 12 bit}} \ \underbrace{2^2 \ 2^1 \ 2^0}_{\sim \text{sanal adres kozuluğu (16)}}$

5' i fode ediyor.
 5'te page table içindeki 3 bitlik değer fiziksel wayda onlara 3 bit, offset girmesi gereklidir.

0 11 | 0 0 0 0 0 0 0 0 0 0 0 0 ~ fiziksel adres kozuluğu (15 bit)

24-28 K'ya bakalım

0 11 0 | 0 0 0 0 0 0 0 0 0 0 0 0

0 0 0 | 0 0 0 0 0 0 0 0 0 0 0 0 page fault, present/absent 0

Sanal bellek sayfaları → virtual page, fiziksel bellek sayfaları page frame ✓

- CPU'nun ürettiği adres sanal adreslerin

- Bellek dolduğunda ise en az kullanılan sayfalar atılır

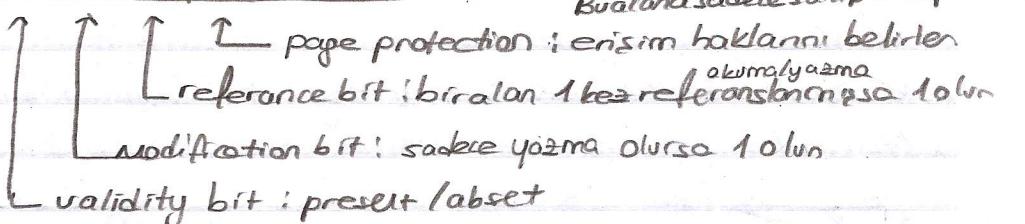
* Fragmentation tam çözülmeli. Sayfaların bölündüğünden ötürü

SAYFALI YAPI (PAGING)

- VM ve fiziksel bellek 4K'lik sayfalara bölünmüştür
- Programlar sayfalara bölünür.
- Process adresi 16 bitlik. Page + offset (4+12)

PAGE TABLE :

			Page frame
--	--	--	------------



PAGE TABLE NE KADAR BÜYÜK OLABILİR ?

$$2^{32} \text{ byte} = 4 \text{ GB adres uzayımız olsun}$$

- . sayfa genişliği (4K) $2^{12} = 4096$ byte
- . Page table genişliği 4 byte harayoruz (2^2)
- . Her sayfada $\frac{2^{32}}{2^{12}} = 2^{20}$ tane sayfa olabılır.
- . $2^{20} \cdot 2^2$ (4 byte) = 2^{22} page table boyutu. (4 MB)

ADRES MAPING : Page table register tablodaki alanlara erişir ve her referanslanan adresten geçerli olup olmadığını kontrol eder.

Present absent biti ile. Geçerli ise fiziksel adresi üretir.

DIRECT MAPING : Page table process boyutuna göre depolamakta olduğu için registerlere tutulmaz. Hizlı erişim için bellekte olmalıdır. Fakat page table çok alan kaplar. Gözüm için asağıdaki yöntemler kullanılır.

1. TWO LEVEL LOOKUP :

$$\text{Virtual adres} = 32 \text{ bit} (4 \text{ GB})$$

Directory 10bit	Page 10bit	Offset (12bit)
-----------------	------------	----------------

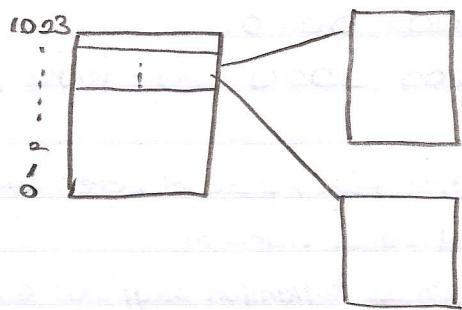
$$2^{10} = 1024 \quad 2^{10} = 1024 \quad \hookrightarrow 2^{12} = 4096 \text{ byte}$$

* Page table directory page table'a benzeyen Bellekteki sayfa tablolannın listesidir. Hangi page table'a erişeceğimizi belirten

Directory + Page → hangi page table'a erişeceğimizi belirten

Page table + offset → Fiziksel adres

PT ₁	PT ₂	Offset
10	10	12



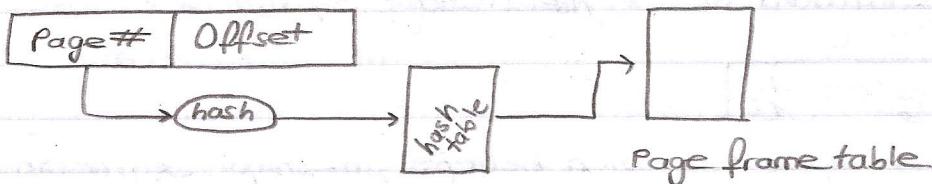
* 2. seviyedeki page table'a sadece process için gerekli alanlar yüklenir.

→ to pages

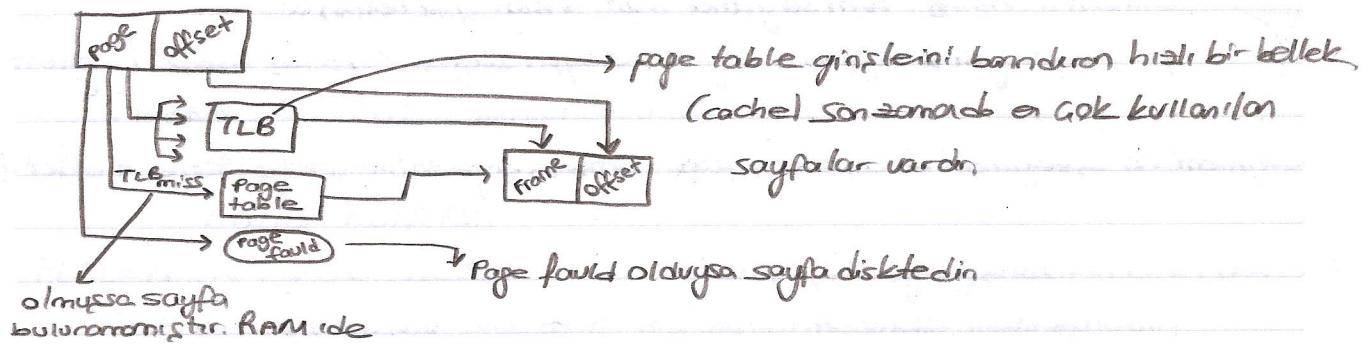
2. seviyedeki page table

2. INVERTED PAGE TABLES

- Sanal bellek için değil, fiziksel bellek için bir page table oluşturulur. Çalışma olmadan hızlı adreslenmeyi bir hash table oluşturulmustur.



3. TRANSLATION LOOKSIDE BUFFER (TLB) - Gündümüzde kullanılan



GÖZÜLEN PROBLEMLER

- Fragmentation büyük ölçüde azaldı. Yani her process'in son sayfasının yanındaki kayıp var. Her sayfa 4K boyutunda ise 2K bellekte boş kalacak.
- Programın kullanılmayan parçaların belleğe yüklenmesi.
- Process size fiziksel alana sınırlı değil. Sayfalar bellekte sıralı olmak zorunda değil.

SEGMENTLİ YAPI

Mantıksal olan program parçası için segmentli yapı oluşturabilirsin.

~~Page size~~ sabitken segmentlerin boyutları değişken olabilir.

Her segment için bir segment map table bulunur.

Segment map tableda her bir girişte segment numarası, fiziksel sep. başlangıç no ve uzunluk var.

* Segmentli yapıda istenilen uzunlukta uzay tanımlanın. Her dunum için segment oluşturulabilir. İstenilen programa istenilen alır verilebilir.

- Programları ve verileri mantıksal bağımsız adres alanlarına ayırmaya olanak sağlar. Hem paylaşımı hem de konumayı sağlanır.

SEGMENTLİ VE SAYFALI YAPI KARŞILAŞTIRILMASI

SEGMENTLİ VE SAYFALI YAPI

- Segmentasyon, sanal bellekte
- Sayfalaması, fiziksel bellekte olur.

Segmentler sayfalara bölünür. Adres usayı 3 parçadan oluşur.

segment
table

page
table

Segment	Page	Offset
18	6	10

Sadece istek yapılan segment sayfaların belleğe yükleniyor. Segmentin birden fazla segment nosu var.

Segmentli yapıda tüm sayfalar aynı anda yüklenmemiş

! process sayısı * segment sayısı * sayfa sayısı / 2 → kod fragmentation olabilir

segment no → segment descriptor → pagenumb → pagetable → page frame + offset + offset

İŞLETİM SİSTEMİ POLİTİKALARI

FETCH POLİTİKASI : Sayfanın RAM'a yüklenmesi

- DEMAND PAGING : Sayfalar ihtiyaç duyulduğu anda alının yüklenme tamamlanmadan sayfa isteği olursa page fault olusur.

- PREPAGING : İhtiyaç duyulan sayfalardan biri getirilir. Tahmin doğruya boyanlı olsa degilse yüklenme boyansı. Sayfalar bitişik (sıraltı) saklanması getirilirken daha verimlidir.

- FETCH POLİTİKASI ✓

- PLACEMENT POLİTİKASI ✓

- REPLACEMENT POLİTİKASI

- RESIDENT SET MANAGEMENT

- CLEANING POLİTİKASI

- LOAD KONTROL

PLACEMENT POLİTİKASI → Yerlestirme (nercye yerleştirilecek) (FF, BF, NA)

Fiziksel bellekte bir processin yerleştirilmesinde belirleyicidir.

Uniform : Tüm bellek uzayına erişim aynı anda ise

REPLACEMENT POLİTİKASI - sayfa atıkarma -

Fiziksel bellek dolmuşsa hangi sayfaların atıkalanıp konarının

- Page fault olmuşsa sayfa atıkılmalı

- Seçilen sayfa referanslanmışsa hâline yazılabilir

- Referanslanmamış ise modify edilir sonra hâline yazılır

- En az kullanılan sayfalar atıkılmalıdır

- Amacımız gelecekte referanslara ihtiyacılı en az olan sayfayı atıkarmak

- Optimal, NRU, FIFO, LRU, NFU, Modified NFU

Atıklanan sayfayı segerken kullanılon 2 tane scope var

~ LOCAL SCOPE : Sadece page fault olmuş sayfalar içinden kullanılma ihtiyacılı en az olanları atıkarmak

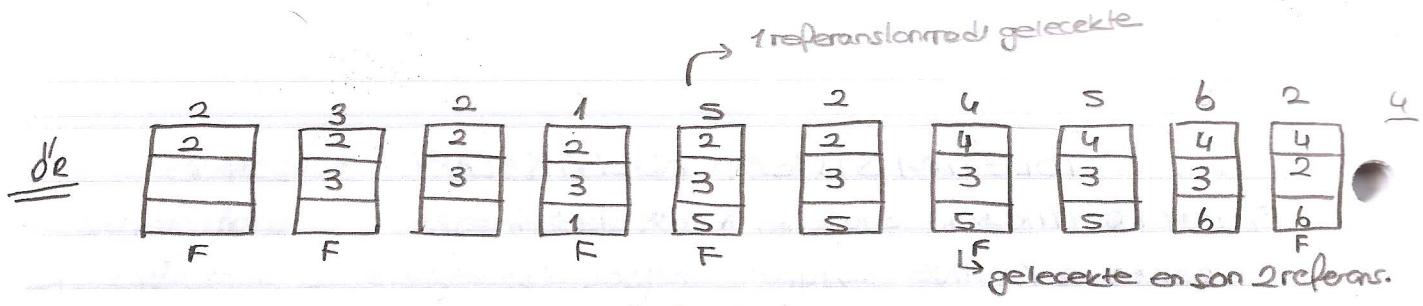
~ GLOBAL SCOPE : Tüm sayfalar içinden kullanılma ihtiyacılı en az olanları atıkarmak
GS'de bir süre genistekken diğerleri sürekli bekler. Bu yüzden kılavuzları
ve sistenden atıkılardan LS'de tüm süreçler eşit süre verilmektedir. Bu nedenle
istedikleri konu genişleyemeyeler

1. OPTIMAL REPLACEMENT ALG. → en yüksek perf.

• Gerçekleşme ihtiyacılı yoktur. Geleceğ hakkında net bilgi yok

- Gelecekte enaz referanslarda ihtiyacılı olan sayfa atıkantır

Bu algoritma diğer algoritmalarla kıyaslamak için var (referans)



2- NOT-RECENTLY USED (NRU)

Son zamanlarda kullanılmayan sayfayı ekrana Referenced (R) ve modified (M) bitlerine bakarak karar verin.

- R biti sayfa referanslarda set edilir (okuma/yazma)

- M biti sayfa modify edildiğinde set edilir (yazmadı)

class 0 : R → 0, M → 0

class 1 : RM → 0 1

class 2 : RM → 1 0

class 3 : RM → 1 1

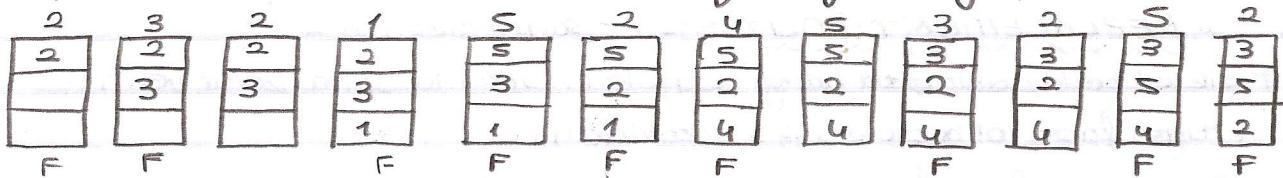
ilk gikanlır.



3- FIFO

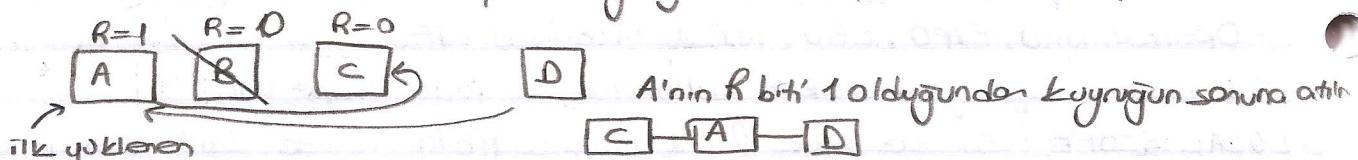
İlk önce belleğe yüklenmiş olan (en uzun süredir bellekte olan) sayfa
gikanlır fakat bu durumda sayfa tekrar geni gelebilir. (Thrashing)

Sayfa sürekli bellekte disk arasında gidiş gelir. Page fault artar.



4- SECOND CHANCE ALG.

FIFO'ya göre dizilmiş listede page fault oluşturuya gikanlaçk sayfalann
R bitine bakılır. R setlenmisse kuyruğun sonuna atılarak bir şans veriliyor.

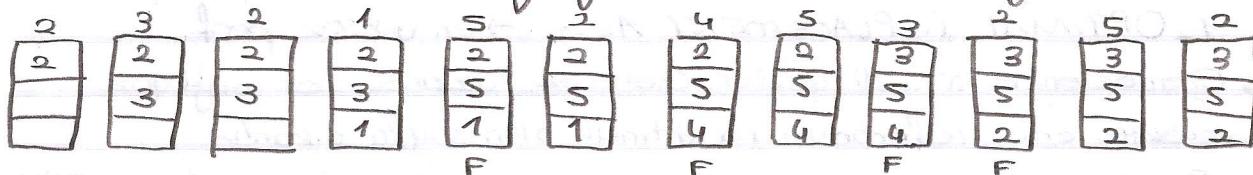


5- CLOCK ALGORİTMASI

Ok kontrol edilecek sayfanın gösterinin R setlenmemiş ise sayfa gikanlır.

6- LEAST RECENTLY USED (LRU)

Uzun zamandır kullanılmayan sayfa gikanlır. Optimuma yakın.
Geçerlilik mesri 30s. Her sayfaya art bir kullanım bilgisi tutacaksın.



HARDWARE

* Son zamanda en çok kullanılan sayfanın en az kullanılmışa doğru dizilmiş
bağlı liste bulunur. Maliyet ve çok zaman alır.

- * Her referanslanmadan sonra sonantalı/bir sayfa arttırlın
 - * Bir page fault olduğunda sunın referanslara yapılmış sayfa alır.
- SOFTWARE LRU : 2 yönler.

1) Sayfaların referanslanması şu şekilde: 0, 1, 2, 3, 2, 1, 0, 3, 2, 3

	0	1	2	3		0	1	2	3
0	0	X ⁰	1	1		0	0	0	1
1	X ¹	0	1	1		1	1	0	1
2	0	0				2	1	1	0
3	0	0				3	0	0	0

(0,0) da satır 1 yap, sütunu 0'la
 ↓
 (2,1) iin yap

2) * 6 sayfa, 5 saat arası var. Her saat arası geldiğinde R bitlerinin değeri sağa doğru kaydırarak sayfa değerine ekliyoruz.

R_{bit}H clock 0, sayfa 0-5; clock 1, sayfa 0-5; clock 2

101011 | 110010 | 110101 → R bitleri

Page 0, 10000000	11000000	11100000	Sayfalar
1, 00000000	10000000	11000000	
2, 10000000	01000000	00100000	
3, 00000000	00000000	10000000	
4, 10000000	11000000	01100000	
5, 10000000	01000000	10100000	

NFU'da ise

Process	1.	2. <u>Sayıci</u>
0	0	1
1	0	2
2	0+	1+
3	0	1
4	0	0
5	0	2

* Sayıcı değeri küçükleştiyor.

Sayfa az referanslanır.

* Ne zaman referanslandığı belli değil.

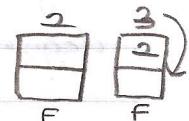
* Kaç kez en fazla sayılın

R bitleri → 101011, 110010

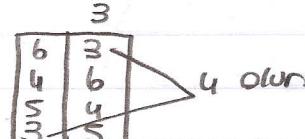
BELADY'S ANOMALY: Bir prosesin daha fazla bellek alanı verilirse (page frame) page fault ihtimali azalır. Ama öyle olmuyor.

STACK ALGORITMASI:

Page fault: Referanslanacak veninin bellekte olmaması.
 Bu algoritma sonucu page fault ve distance string oluşur.



Distance String: Page fault olduktan sonra yeni eklenenek ekrannın eski yerine takıldığı.



Referans	0	2	1	3	5	6	6	3	7	4	7	3
Stack	0	2	1	3	5	4	6	3	7	4	7	3
	0	2	1	3	5	4	6	3	7	4	7	
	0	2	1	3	5	4	6	3	3	4		
	0	2	1	3	5	4	6	6	6	6		
				0	2	1	1	5	5	5	5	
					0	2	2	1	1	1	1	
						0	0	2	2	2	2	
							0	0	0	0	0	
Page Fault:	P	P	P	P	P	P	P	P	↓	↓	↓	↓
Distance S.	∞	∞	∞	∞	∞	∞	∞	∞	4	∞	4	2

~ Distance string artarsa

P.F. azalir.

Distance string P.F. sayisini
azaltmak ian process'e ne kadar
sayfa verecegimizi yolduma olun

F → page fault sayisi

M → page frame sayici

C1 → distance stringi 1 olurken sayisi

$$\text{Page Fault Sayisini Hesaplama} \Rightarrow F_m = \left(\sum_{k=m+1}^{\infty} C_k \right)$$

$$F_2 = C_3 + C_4 + \dots + C_{\infty} = 17 \quad F_1 = C_2 + C_3 + C_4 + \dots + C_8 = 18$$

↳ 2 frame ian page fault saysi. ↳ 1 frame ian page fault saysi.

RESIDENT SET MANAGEMENT

Process'e atanacak sayfa sayisi, process'in bellekte bulunan sayfanin.

RS büyük segilirse → daha az process kosulur. PF etkisi olmaz.



Küçük segilirse → daha fazla process kosulur. CPU kullanimi artan
P. fault artar (sürecler artigindan)?

RESIDENT SET POLİTİKALARI

1. FIXED ALLOCATION : Her process'e sabit sayfa verilir.

2. VARIABLE ALLOCATION : Her process'e ihtiyac gereki farklı sayıda sayfa
ALLOCATION VE SCOPE (Local → PF icinden, Global → Tüm sayfalar)

→ Sabit Tahsis - Local Scope : Her process'e verilen page frame
sabit. Page Fault olursa sayfalar icinden alır. P Frame ↓ P. Fault ↑

→ Degişken Tahsis - Local Scope : Her process'in ihtiyacı kadar page
frame var. Page Fault olusular icinden. Her process'in resident set'i van

→ Degişken Tahsis - Global Scope : Process'lenin resident seti
degisken. Gikarma tüm sayfalar icinden olur.

* FIXED ALLOCATION - GLOBAL SCOPE olmaz. Genisteme var. Sbt olmaz

WORKING SET:

1	Routine 1
2	Routine 2
3	
4	
5	main loop
6	
7	
8	Main prog

- Programın mainde ıfibaren kosması için 4Sibit ihtiyacı var. Her kostüğünde bunları kullanır.
Bu sayfalann bellekte atılmamasını engellesek ve bellekte en hızlı kostüğü yere kayarsa program en etkin şekilde koşar. (trashing olmaz).

* Proses'in çok kullanılan sayfalara WORKING SET denir. Bu bellekte tutulmazsa process kosesmez
GERGEKLENMESİ İLE LRU (NLU) Kullanılır.

Referans biti (R) her saat kermesinde setlenir. n tanesi saat periyodu boyunca R setlenmişse bu sayfa working sete dahil edilir
CLEANING POLİTİKALARI

Modify olan sayfalari zaman zaman devreye girerek temizler. Sistem bostayken temizler. → disk kaynakları

1- DEMAND PRECLEANING : Replacement iain istek varsa temizler.

2- PRECLEANING : İstek olmadan modify olan sayfalari diske yazar (Bostayken)
PAGIN DIAMOND : Sürekli sayfalari kontrol eder precleaning yapan

LOAD KONTROL

- Sistem etkilesimli kalacak, ekilde kon process ana belleğe yüklenir?
- İşlaniye ne kadar process yüklenebilir?

PAGE SIZE



Küçükse
avantaj

- Fragmentation azalır
- kullanılmayan parçalar belleğe yüklenmez.

dekarantaj

- Program çok sayıda sayfaya ihtiyaç duyar
- Page table size artar

Büyükse

- diskten hızlı okuma yapar
- programı hızlı yükler

Optimum Page Size

$$\text{overhead} = \frac{s.e}{p} + \frac{p}{2}$$

overhead

page table

fragmentation

page size

ortalama

$s = 1000$

$e = 6 \text{ bayt}$

$p = \sqrt{2se}$

DEADLOCK

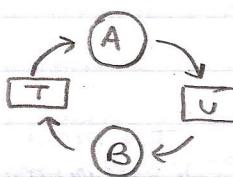
Process Deadlock : bir process hiç olmayacağı bir olay için bekliyosa deadlock olur.

Sistem Deadlock ; bir veya birden fazla process deadlock'a girmişse olur.

* Bir körmede 2 process var. Her ikisi de diğerinin üreteceği sonucu bekliyor. Bu bir deadlock sebebidir.

Bloklanmış process tek başına bloğu kaldırıramaz. Sonuç döktürse serbest kalır.

Deadlock olan process;



- hiçbir zaman kılınır

- elindeki kaynacı bırakırmaz

- uyandırılmaz

- elindeki kaynak ancak zorla alınır (p. ökünür)

A ; T'ye sahip U'yu istiyor

B ; U'ya sahip T kaynagini istiyor

* Kaynak isteği yapan process kaynacı elde edene kadar bloğunu

* System deadlock durumunda sistem yeniden başlatılır.

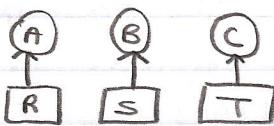
Bir sistemin hedefi her zaman deadlock'ı önlemektir.

A
(isteği) Request R
Request S
(bırak) Release R
Releases

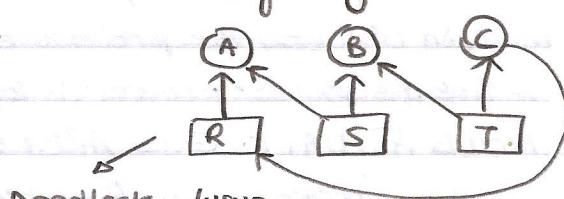
B
Request S
Request T
Releases S
Release T

C
Request T
Request R
Release T
Release R

1. A'ya R'yi ver
2. B'ye S'yi ver
3. C'ye T'yi ver.



4. A'ya S'yi ver
5. B'ye T'yi ver
6. C'ye R'yi ver



Deadlock oluşsun

* Deadlocktan kaçınmak için önce A processine gerekli kaynaklar verilir. Sonra A işi bitince kaynaklarını bırakacaktır. Diğer processler kullanıcaz.

Bir sisteme deadlock olursa er ya da geç olur. Sadece farklı kombinasyonlarda bunu ertelemis olunuz.

Proseslerin karma sıralan deadlock olma durumunu belirler. OS bu doğru sıralamayı biliyor olmaz.

* Process ölümdünde tek PCB içeriği ile tekrar başlatılmasın. OS destekli ile process'in tüm local değişkeni saklıyor (gentik atmıyor). Bu sayede process tekrar kaldığı yerden devam edebilir.

DEADLOCK KOSULLARI

- MUTUAL EX : Göz ardı ederek paylaşılmış kaynaklar exclusive kullanılmaz.

- HOLD & WAIT : 1. kaynak ele geçirilmiş 2. yi beklenme durumu

- NO PREEMPTION : Bir kaynak ele geçirilmişse o kaynak bırakılana kadar elindeki kaynacı alırsak yanlış sonuçlar üretir.

- CIRCULAR WAIT : Birbirin processler birsey yapmadan birbirini bekliyor.

NO DEADLOCK

Bu koşullardan en azından birini göz ardı edersek deadlock olmaz.

DEADLOCK YOLARI

- Deadlock önleme ✓
- Deadlock algılama ✓
- Deadlock sakınma ✓
- Deadlocktan gen döngü ✓

DEADLOCK ÖNLEME TEKNİKLERİ

- Mutual Ex : Processlerden biri ortak bellekleyken diğerini kullanamaması. Tek process'un
- Hold & Wait : 1. kaynak ele geçirilmiş 2. bekleniyor
- No Preemption : Bir kaynak elinde iken kasana kadar onutması, sonlandırmayı
- Circular Wait : Bütün kaynaklar birbirlerini bekliyor

→ M.E GÖZ ARDI EDILİRSE : Gözdekte kullanılmıyor

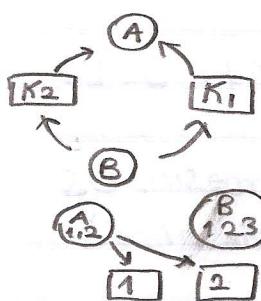
- * Kaynaklar tek process'e verme durumu. Hayati bir durum.
- * Processlere istekler gönderilir. Processler bu istekleri hizasında toplar ve kasırlar. (spooler). Kaynakları tek process'e verme hayatı oldugundan mümkün olukça tek process'e verilmesi.
- * Gözdeş sayida process'e kaynak verilmesi

→ HOLD & WAIT GÖZ ARDI EDİLİRSE (YA HEP YA HİÇ)

- * Bir process'e ya kaynakların tümünü ver ya da hiçbirini verme.
- * Vermiyorsak kaynakları ele geçirene kadar process bekler. Deadlock yok.

SORUNLAR

A - Kaynaklar boşuna kullanılıyor ve starvation van



A bütün kaynakları ele geçirmiş. Sadece K₁'i kullanıyor.

B iki kaynağı da istiyor

A binini boş yere elinde tutuyor.

hepsi aynı anda olamaz

→ A B bütün kaynakları istiyor. Ama sistemde o kaynakları kullanıp başka processler de var. Starvation olur.

- Prosesler gerek duyuulan kaynakları OS'ye bildirmeli.

- Hangi kaynakları hangi prosesin kullandığını, bilmeliyiz

AVANTAJ

- Galison bir yöntemdir. ✓
- Kodunu yazmak kolaydır. ✓

- NO-PREEMPTION : Ele geçirmezse bırakıyor sorunu, sonlanmıyor
 - Process bir kaynayı ele geçiriyor, 2. kaynayı istiyor. Eğer 2.'yi alamazsa elindeki kaynayı da bırakıyor.
 - Bırakırsa kaynakla yoptığı işler boş'a gidiyor.

AVANTAJ

- Galisir bir yöntemdir ✓
- Kaynak kullanımını diğer yöntemlere göre daha iyi ✓ → Hold & Wait
- * Sistemde 2 process var. 1. process 2 kaynayı ele geçirsin ama sadece 1. kaynak işine yanyor. 2. kaynak boş bekler process'in işi bitene kadar. Kaynak kullanımı %50.
- * Bu yöntemde 1. ele geçirip 2'yi geri vermeye, 1. kaynayı da bırakın. 1. kaynakla yoptıkları boş'a gider. İhtiyacı olan kaynak ele geçirilirse kaynak kullanımı %90, geri vermeye %50 olur.

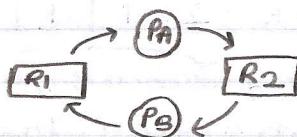
PROBLEM

- Kaynak genel alım maliyeti var. ✓
- Starvation var. ✓

Printer için bu kullanışlı değil. Kapıdan yanında 1 process çalışır, diğer yanında başka process çalışırsa sıkıntı.

- CIRCULAR WAIT : Processler birbirini beklemeye durumu

- Her kaynaya bir numara verilir. Kaynak numaraları dairesel.
- Sisteme yeni kaynak gelince ona da verilmeli.
- Processler istedikleri kaynayı artan sıradı isteyecek.



P₁ önce R₁'i almıştı. R₂'yi boşuna almış.
 P₂ R₁'i önce almış olduğunu. R₂'yi, şimdi almasa da olsa R₁'i bırakıktan sonra alır.

PROBLEM ; kaynak kullanımı artan sıradı, gerçekte ihtiyacı.

- Kodlanması zor, kaynaklara numara verilmemesi deadlock olur.

M.E → hersey havuzda toplanıp halledilir

Hold & Wait → Ya hep ya hiç (kaynak)

No - Preemption → Kaynaklar processler zorda alınır

Circular Wait → Kaynaklar numaralı

DEADLOCKTAN SAKINMA

- Deadlock'a izin verilir ancak olağın yerden sakınıyorsa.
- Her sonraki adım güvenli mi değil mi kontrol edilir.
- Prosesler kaynaklara ihtiyaç sırasına göre enisebilmeli.
- * Tek bir process'e cihazda enisme hakkı verirsek olur.
- Preemptable cihazı ise ; kaynak processin elinde zorla alınabilir.
- Non-Preemptable ise ; kaynağı zorla almaya çalışırsak process fari olur.
- * Deadlockta en önemli durum kaynağı istene sırasında bir kaynak istenir mümkünse alınır, işi bitince bırakılır. İstediği kaynak ele geçiremeyece process bloklanır.

Customer (process)	Max need	Present loan	Claim	process	swap	tekraf alınca geçer
				topluca	veren	geten
C1	800	410	390			Toplam miktar
C2	600	210	390			1000

$1000 - (410 + 210) = 380$ kalan ihtiyaçların hizbinini karşılayamaz.
Güvenli kalmadı. Sakınma yönt. en önemli sey güvenli kalmaktır.

process	topluca isteyen	plateler	Scanner	ardan	kaynaklar verilmis		kaynak daha istiyor			
					A	B	C	D	E	A
E	(6302)	toplam kaynak Miktartır			3	0	1	1	0	1
P	(5322)	tahsis edilmiş kaynaklar			0	1	0	0	0	0
A	(1020)	baz kaynak			1	1	1	0	0	3
E = P+A					1	1	0	1	0	0
					0	0	0	0	0	2

Geniye A = (1020) kalmış. Tekrar kaynak istiyor ona

Burdurumda sadece D kosar.

Sadece D kosabileceğinden ona kaynak verili. Kasma bitince kaynakları ve
** 2 process birlikte kosabiliyor mu bakılır. Kosamazsa en çok kaynak geniye
döndürerek kosulur. Amaç max kaynak tahsisi sağlayıp CPU kullanımını artırmak.

DİJKSTRA'S BANKER'S ALG.

- Elde kalan kaynak max need büyükse process losyalı.
- Bir kaynak ölümcül verilirken mutlaka geni dönmeli ve o process losyalı (safe)
- Kaynak geni dönmüyor ve process deadlock'a girdiğinde kaynak vermez (unsafe)
- En azından 1 process bile losyalık durumda değilse kaynak vermez (safe kalmalı)
- Sistem safe kalmadığa kaynak vermez.

1. Elimizdeki kaynağa göre losyalık process anyonez varsa kaynak ver

2. Kaynak durumlarnı güncelle

3. Yeni process ian 1-2 tekrarla.

OSTRICH ALGORITHM

- Unix ve Windows'ta kullanılmıştır. Deadlock yoksa basanlı olan yöntemdir.
- Alg. iyi ise iyi sonuc üretir ama sistem zamanını çok harcar. Önemli olan iyi olamaz performansla elde etmektir.
- Doğru olmayabilen ancak kolay olan algoritmları seçmek perektir.

DEADLOCK ALGILAMA (DETECTION)

- Kaynaklar numaralandırıp artan sıradı tahsis yapılırsa deadlock olmaz
- Kaynak ve process bağımlılığı belli olur.
 $E = (4, 2, 3, 1) \rightarrow \text{ihtiyaç}$ $A = (2 \ 1 \ 0 \ 0) \rightarrow \text{kalan kaynak}$

$$C = \begin{bmatrix} 0 & 0 & 1 & 0 \\ 2 & 0 & 0 & 1 \\ 0 & 1 & 2 & 0 \end{bmatrix} \quad \begin{array}{l} \xrightarrow{\text{Verilen kaynak}} \\ \xrightarrow{\text{Kullanıcı}} \end{array}$$
$$R = \begin{bmatrix} 2 & 0 & 0 & 1 \\ 1 & 0 & 1 & 0 \\ 2 & 1 & 0 & 0 \end{bmatrix} \quad \begin{array}{l} \xrightarrow{\text{Kullanılmaz}} \\ \xrightarrow{\text{Kullanılmaz}} \\ \xrightarrow{\text{Kullanılmış}} \end{array}$$

- Elimizde kalanınosta kaynak isteyen kosulun
- Deadlock olma ihtimali ve olacaksız hangi processte olaçığı önceden belirlenmelidir.
- Genelde deadlock bulma circular wait ile oluyor. Ancak pahalı. Graf

DEADLOCK KURTARMA (RECOVERY)

- Sorun aksaran prosesleri kaldırılmalıdır. Ama yaptığı işler kaybolur.
- + Sorun aksaran prosesin elinden kaynak atılır. Bu prosesin tekrar kılması için tekrar başa döndürmemiz gereklidir. Check point ile yapılabilir.

CHECK POINT:

Programa belli aralıklarda işaret kayabiliyoruz. Sistem geni yüklenme gibi. En başa değil istediğimiz check point'e abinebiliriz.

Check point sonuçları saklanmalıdır.

60 dk program düşün. 50. dk'ya check point kayıyanın. 50. dk'dan sonra bir sorun aksarsa 50. dk'ya geri döner. 10 dk kayıp.

KURTARMA

- Process'i öldürmek deadlocktan kurtulmanın en basit yoludur. Birçok sistem process'in öldürülmesini veya yeniden başlamasını destekler.
- 10'luca process deadlock'a girdiğse hangisinin öldürülmesine karar ver.
- Basitçe kurtulmak için

Process öldürülürse yaptığı işler kaybolur. Öldürmek için en zararsız olan öldürülür. Process durdurulup başlatılabilir, check point kullanılabilir.

En basit yol RE-BOOT

FILE SYSTEM

Algoritmalar ve veri yapılan sayesinde mantıksal olarak yapılan dosyalama sistemi fiziksel işleme dönüştürür yapar.

AMACI:

- Verilen depolayan bir ortam sağlar
- Veri tutarlılığı sağlar.
- Hataları minimize etmeye çalışır. Veri kayiplarını.
- Kullanıcıya standart bir arayüz sağlar.
- Güvenlik kullanıcıyı ve tüm sistemleri destekler.

Dosyalama sistemi uzun süreli bilgiler tutabilmenizi sağlar.

Depolama sisteminin hedefi RAM'deki bilgilerin gizlilikini korumak.

Kullanıcı: - Sembolik isimlerle dosyaya erişmek ister.

- Dosya yaratma, silme, değiştirme ister.
- Dosyalar arasında veri aktarımı, yedek alabilme, silineni geri alabilme
- Dosyalara erişim hakkını kontrol etmek, kendi erişim haklarını kontrol etmek

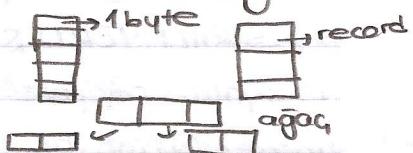
İsimlendirme

- İsim ve uzantısı : kobra.doc

- bayt dizisi (DOS, Windows, UNIX) - genellikle

- record dizisi

- ağacı yapısı



* Dosya Erisimi

- Sıralı veya rastgele olabilir.

- Sıralı Erisim: Diskin başından sonuna kadar okunur.

- Rastgele Erisim: Her okunmadı ile ya da geni rastgele girilir.

File Attribute: Dosya adı ile aynı yerde tutulur.

Yaratma, son erişim, son değiştirme

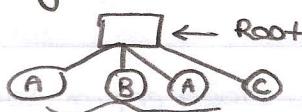
→ sistemiagnostics serindede

Dosya Operasyonları: Dosya oluşturmak, açmak, kapamak, silmek, değiştirmek

DIRECTORIES: Dizinler Dosyaların özellikleri soklandığı yerdir

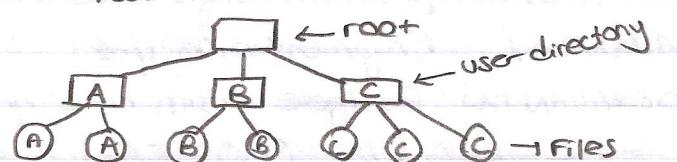
- DOS ve Windows'ta kullanılır. UNIX → i-node

Single Level Directories System



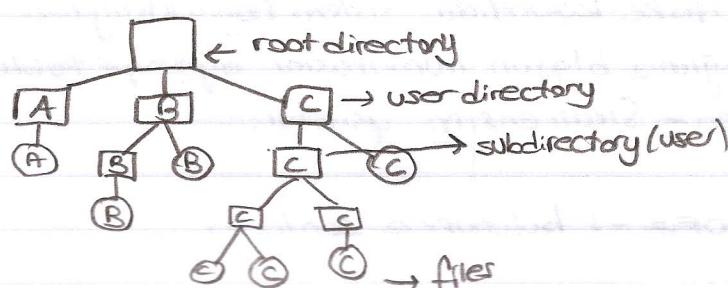
4 dosya, 3 farklı biseye alt

Two Level

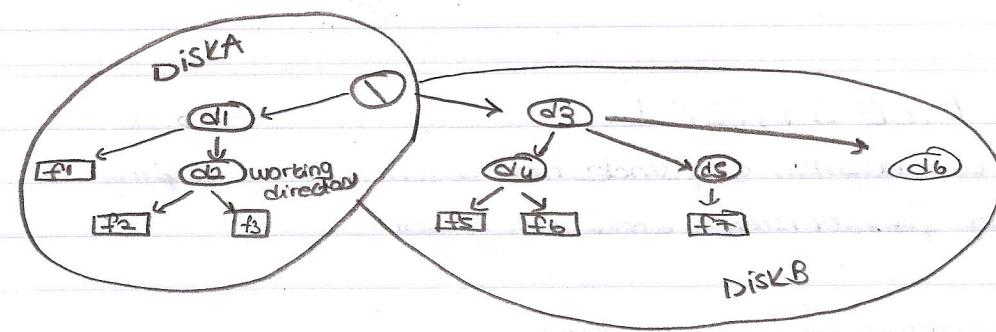


→ Files

Hierarchical Directory



→ files



Working directory : d₂ (Galison directory)

Absolute path name : f₂ : d₁/d₂/f₂ (rootten itibaren)

Relative path name : f₂ : f₂ (kullanıcının directory'si itibaren)

Bir silindrinin hem önde hem arkasında tracklar vardır. Hem önde hem arkasında okunma yapılabilir. Silindir miktar arttıkça okunan bit miktarı artar. * Diskte zaman alır disk lokasını hareket ettirmektedir.

* Contiguous allocation : Dosyaların disk serisinde olarak kaydetme

* linked list allocation : Dosyaların diske bağlı liste olarak kaydetme

* linked list yöntemiyle indeks kullanmak : DOS dosyalama sistemi → FAT

* i-nodes : UNIX

SUPER BLOCK : İlk yüklenen dosya sistemi. File system ile ilgili parametreler burada bulunur.

→ SIRALI YERLESTIRME :

Dosyalar perçeve sıralı bir şekilde kaydedilir. Dosyalar silindiğinde fragmentation olur. Bu yüzden dosya yerlestirmek zorlaşır. Relocate gerekliliği nedeniyle.

FAT (file allocation table) → dosya adı, başlangıç bloğu ve genişliği

Avantaj : Başlangıç bloğu ve uzunluk bilindiğinde tüm dosyaları eninde sıralı olduguundan kolaydır.

Desavantaj : Fragmentation ve relocation

→ BAĞLI LISTE

Dosya blokları bağlı liste şeklinde tutulur. Sıralı olmak zorunda değil. İnden

FAT → dosyanın ilk bloğunu ve adresini tutar.

AVANTAJ : Fragmentation yok.

DESAVANTAJ : Rastgele erişim çok yavaştır. Sonuncu bloğa erişim zordur.

→ LINKED LIST (DOS FAT) : Linked list ile indexli erişim

FAT diske kayıtlıdır. Sistem başlatıldığında RAM'e yüklenir ve RAM'de en fazla istedigimiz bloğun numarasını öğrenip tablodan okuma işlemini RAM'de yapmayıza sıralı erişim yaktır.

EOF = -1 bulununce sonlanır.

- DOS (Windows) FAT "linked list allocation using an index" yöntemini kullanır.
 - Tüm block pointerler FAT'in içinde saklanır.
 - Rastgele erişim daha hızlıdır.
- Günümüzde linked listte dosyanın blok numarası bir dizeki sonunda tutuluyor. Burda FAT içinde (block pointerler sondaydı).

$$16\text{ bit DOS FAT uzunluğu } (65536+2)*2 = 131076 \text{ bytes}$$

\downarrow
2¹⁶ pointer
 \downarrow
2KB
 \downarrow
block size
 \downarrow
disk size

UNIX - i-nodes

i-nodes directoryler içinde tutulur.

i-nodes düğümleri her bir dosya ile ilişkilendirilir. (Dosyanın referans eder.)

Dosyanın tutulduğu ilk 10 bloğun numaraları ve ilin bu bloklardan direkten erişilebilir.

Blok size = 1 KB 4 bayt block number için kullanılır.

$$\frac{1\text{ KB}}{4} = 256 \text{ block number tutar.}$$

$$\text{First 10 block} = 10 \times 1\text{ KB}$$

$$\text{Blok size} = 2\text{ KB} \quad \text{Single indirect} = 256 + 10 = 266 \text{ KB}$$

$$\frac{2\text{ KB}}{4} = 512 \quad \text{Double indirect} = 256 \times 256 + 266 = 65802 \text{ KB}$$

$$\text{Triple indirect} = 256 \times 256 \times 256 + 65802 = 16777216 \text{ KB} = 16\text{ GB}$$

MS-DOS FILE SYSTEM

FAT16

$$\text{Blok size} 8\text{ KB} \quad 2^{16} \times 8\text{ KB} = 512 \text{ MB}$$

$$\text{FAT-32, blok size} = 1\text{ KB}$$

$$2^{32} \times 1\text{ KB} = 4\text{ TB}$$

\downarrow
4G Pointer

Blok size arttıkça disk kullanımı verimsizleşir. Fragmentation.

Dosyalar 2KB, bloksize = 16KB seçilise 14KB boy alır.

Blok size arttıkça disk kullanımı düşer, hız artar.

Azaldıkça erişim hızı düşer.

$$\frac{2\text{ KB} \rightarrow \text{dosya boyutu}}{128 \text{ (bloksize)}} = 16 \text{ blok kullanılırsa } 2\text{ KB lik dosyayı kaydedebiliriz.}$$

* Her dosyanın i-node 1. farklıdır.