

isletim
SISTEMLERİ

Modern Operating Systems (second edition)

Andrew S. Tanenbaum

Prentice-Hall Inc.

2001

1. Introduction

2.1 Processes

2.2 Scheduling

2.3 Interprocess Communication

2.4 Deadlocks

2.5 Threads

3.1 Memory Management

3.2 Virtual Memory Management

3.3 OS Policies for Virtual Memory

4. File Systems

5. Input/Output

6. Case Studies (UNIX, DOS, WIN98, WINNT)

- Application Software (bank automation system, etc)
- System Software (data base, OS, etc)
- Physical Hardware

System Software

Uygulama yazılımları, tıpkı bir otomat ve operatör

OS (operating System)

Yapılan bütün işlemlerin alt yapısını oluşturur. Karmasık büyük bir programdır. Aynı zamanda birçok programın birleşmesiyle oluşur. İki önemli hedefi vardır,

1. Kullanıcı ile denetim arasında bir arayüz sunmak.

2. Uygulama kaynaklarının güvenli ve etkin kullanımını sağlar.

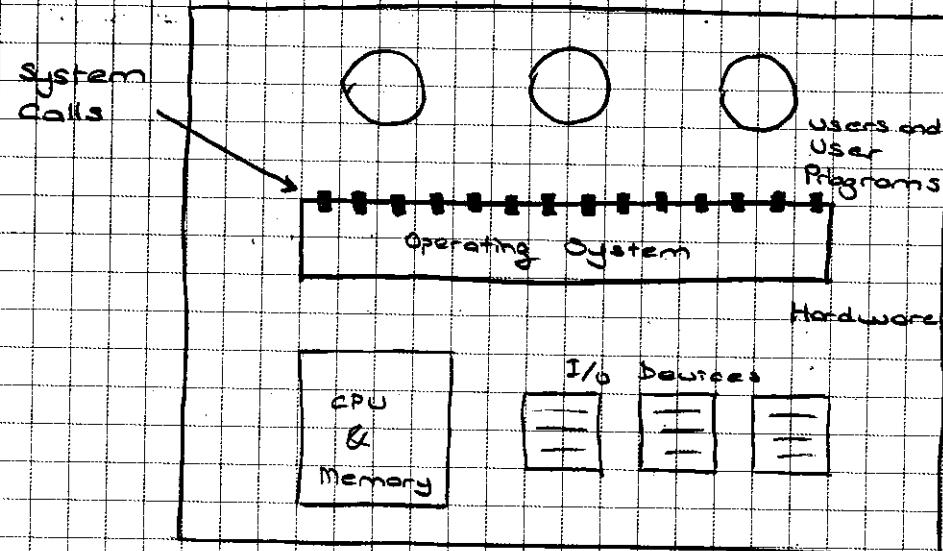
İşletim Sistemi olmasa da,

↳ Denetimin bütün detaylarını bilmesini gereklidir.

↳ Bütün denetimlerin erişebilme yeteneğine sahip olması gerekiydi.

↳ Her bir programının sevgileri yapacak komutları sahip olmali.

Operating systemin işleri:



System calls'la denetim ve yönetimi sağlanır.

İşlemci Sistemlerinin Tarihi (History):

First Generation (1945 - 1955):

Vacuum tubes kullanıldı.

Operating system yok.

Programlama wiring a plug board ile yapıldı.

Genellikle soyisli hesaplamalar için kullanıldı.

ör: trajectory computations, etc.

Second Generation (1955 - 1965):

Transistörler kullanıldı.

Fok olsalıydı.

4.0k yavuz (sistem boyutskesi hiz araligi)

Genelde batch operasyonlara uygundu. Uygunluk
Talemler bir sistem hizunda toplanip, sonra
sira ile gerekletilirlerdi.

O zamanda Fortran ve Cobol kullanılıyordu.

Kullanım alanları sınırlı oluyordu.

Fortran ve Cobol gibi yüksek seviyeli
diller kullanılmaya başlandı.

Third Generation (1965 - 1980):

Bu kuşağıın başmosaında en önemli nedensel faktörler:

Integrated circuits (small scale, entegre devreler) dir. Bu nesilde CPU'lar anakartta yer almaktadır. Bu sisteme multi-tasking kodları koymuş, Bu sisteme multi-programming ve time-sharing gibi özellikler mevcuttur.

Bu nesilde kullanılan 2 işletim sistemi sistemi vardır. Bunlar:

1. Multics OS (Dosya分享 Dosya benzeri bir sistem sistemi)

2. Original UNIX

Bu kuşak mainframe ve microcomputer kuşağı
olarak da adlandırılır.

Fourth Generation (1980 - 1990)

Network kullanımı başladı.

Bu kuşağın en önemli özelliği Large Scale Integration'dır.

Bu kuşağın en önemli bilgisayarıları PC'lerdir. Bu kuşakta Network sistemi de gelişmiştir.

Bu kuşakta kullanılan işletim sistemleri CP/M,
MS-DOS, UNIX'tır.

Now

Bu kuşakta client / server computation an plana
gelmisti.

Server bilgisayarlarında windows NT, UNIX gibi
istem sistemleri kullanılır.

Bu kuşakta SCSI (genç, hızlı entegre devreler)
kullanılarak çok gelişmiş bilgisayarlar yapılabıldı.
Ayrıca bu kuşakta bilgisayarlar Internete ve
Intranet networking ('www')'e erişebildiler.

Important Points

OS provides:

Bu bir ucgoalı orguje birimini sahiptir, tek sek servisler sunuyor.

OS:

İşletim sisteminin sağladığı servisleri yöneten sistem calls'la erişebilir.

Kullanıcılar ve programlar doğrudan hardware'a erişemez.

Sets of system calls (API's) is what program think the OS is.

Some OS Concepts

Kernel:

Bu programın birçok servislerin ilgili kodları içe
riyor. Sık sık belleğin en önemli kısmında yerles-
tirilir.

Device drivers:

Bu sürücüler I/O devices'ları organize etmek için sağ-
lırlar. Tipik olarak Kernel'in bir parçasıdır.

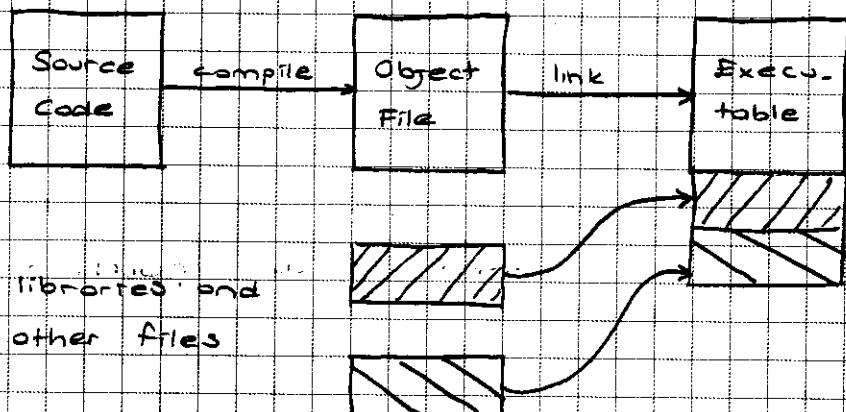
Program:

Makine kodunda static bir deye (RAM'a yüklenmemiş)

Process:

İçin edilen, kırılan bir programdır. Aynı zamanda işletim sistemi data yolları kaynaklarına sahip bir program (Kullanılmasa da işletim sistemindeki bazı kodları olmalı ve bazı dataları tutmalı).

Producing an Executable



Kullanılabilir programın içeriği mesajı

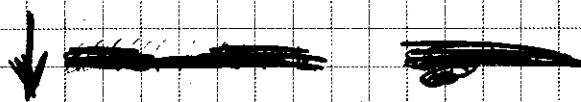
```
Ör: #include <sys/types.h>
    #include <dirent.h>
    #include <curl.h>
```

```

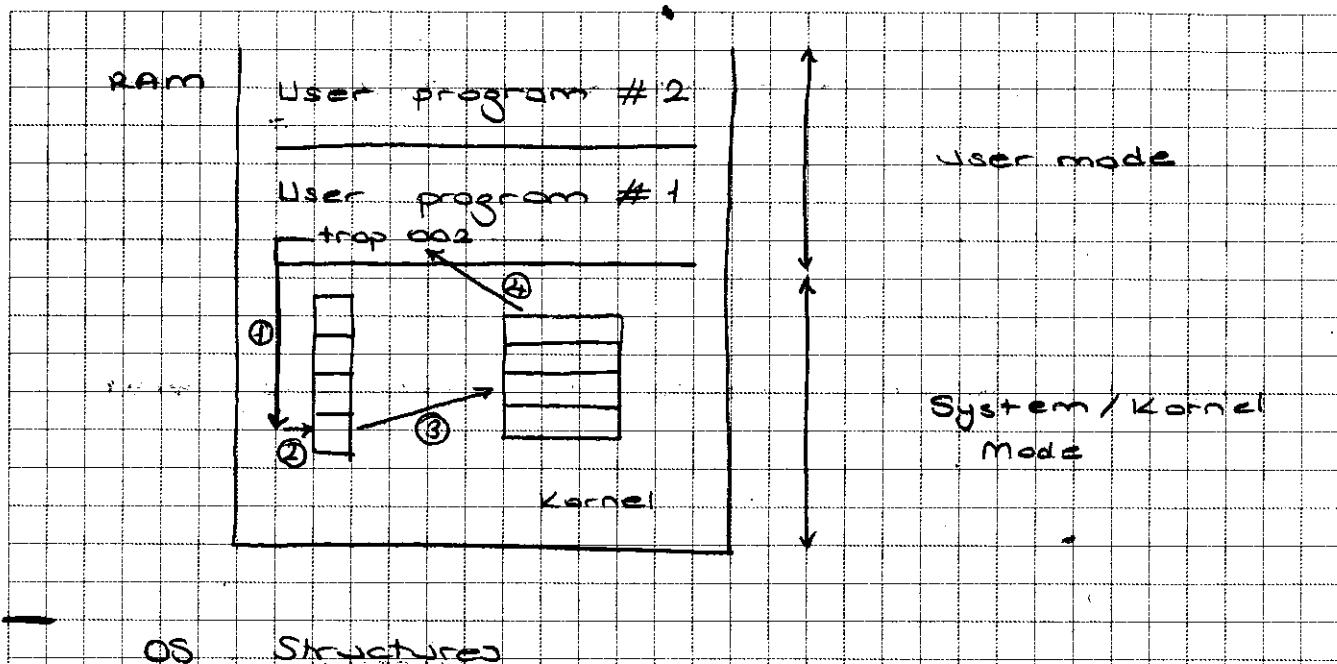
int main (int argc, char * argv[])
{
    DIR * dp;
    struct dirent * drp;
    if (argc < 1 || argc > 2)
        err_quit ("a single argument (- -) is required");
    if ((dp = opendir (argv[1])) == NULL)
        err_sys ("can't open \"%s\"", argv[1]);
    while ((drp = readdir (dp)) != NULL)
        printf ("%s\n", drp->d_name);
    closedir (dp);
    exit (0);
}

```

(İslahım sisteminin bulunduğu yerler işaretlik
yapıldır.)



- ① Program system call (trap) işlemi gerçekleştirir.
- ② Trap'a ilgili servis numaralarını belirler.
- ③ Servis yeri belirlenir ve rera edilir.
- ④ Kontrol tekrar user'in brakılır.



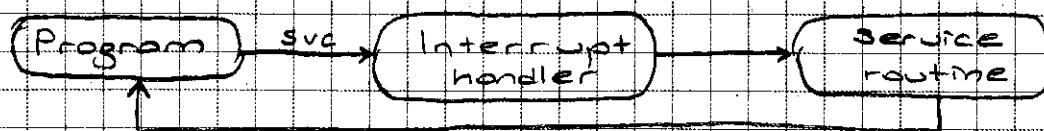
OS Structures

1. Monolithic system
2. Hierarchy of layer system
3. Virtual machine system
4. Microkernel (Client-Server) model

1. Monolithic system

Herhangi bir (yapı, yoktur, Temel alacak içinde topluluklar mis) sistem çağrıları ve bir sürü procedureller mevcuttur. Aynı zamanda bunlara dosya erişileceğiz içinde tespit edilir.

Supervisor call (svc)



İsletim sistemi 3 parçadan oluşur.

1. Operating System

2. System Call

3. Utility function

Interrupt handler arayuda gibi duranız bizi uyanırmış gibi hizmet sunuyor.

2. Layered System

5	Operator functions
4	User programs
3	I/O management
2	Operator-process communication
1	Memory management (swapping)
0	CPU management (process switching)

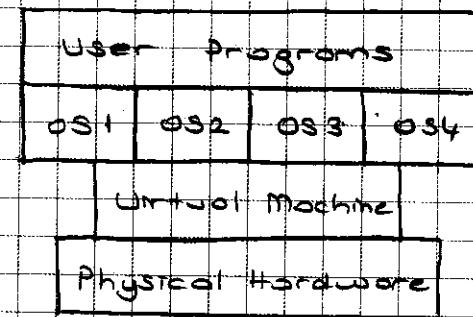
- * Tabakalar arasında iletişim semaphores'larla gerçekleştirilebilir.
- * 2. tabakada mesajların yorumlanması ve işlem kontrolü gerçekleştirilebilir.
- * 0. tabaka en önemli tabakadır.
- * Her bir tabaka özüğünü bir sonralık makine gibi yükleyip gösterirler.

Her bir tabakada en az bir process bulunur ve iştekerleri yerine getirir.

Tabakalar arasında sistem hardware ve software interruptları sayesinde sağlanır.

Her tabak kendi içindeki tabakalarla iletişimde olur.

3. Virtual Machine



Software ile oluşturulan makineyi yukarıda tanımladığımız gibi yansitıyor.

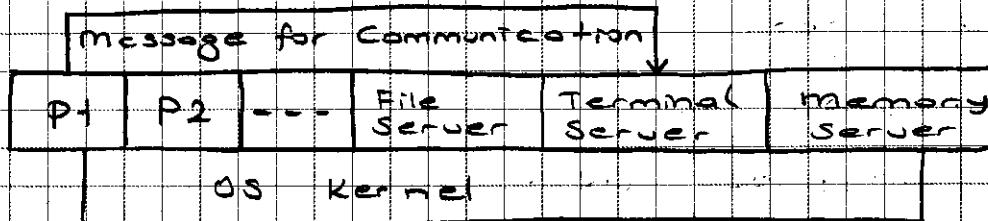
Sonuç makinede birden çok işletim sistemi kullanılır.

What is wrong?

Yazılması çok zor, çok hantal bir program.

Dönüşüme yeni bir婚纱 ekleneninde program
yeniden compile edilmeli.

4. Micro-Kernel (Client/Server) Model



Gök kisik bir isletim sistemi gerekdegi (kernel) soz konusudur B₁ basit kernelde;

1. Bellek yönetimi,

2. CPU yönetimi,

3. Inter process communication,

4. Input / Output desteği

gerçekleştiriliyor.

AK Sistemi: yeniden baslatmadan yeni cihazlar ekleyemiyorsuz.

Characteristics of Kernel

1. Mesaj geçme özgürlüğü sağlıyor.

2. Server processler rohutlikla deşifrelebilirler.

B₁ isletim sistemi etkilemiyor.

3. isletim sistemi (OS) oldukça basit. Yazımı da
haz koley.

4- Dijital sistemler geliştirme tek türde uygulanır.

5- Sunucular (örneğin File Server) rohatılıkla başka bilgisayarlara erişilebilir.

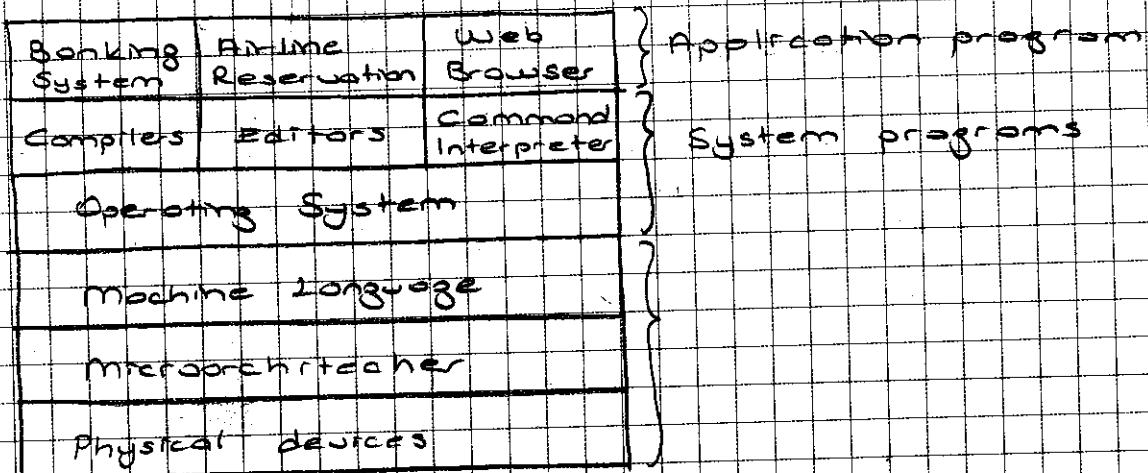
Bununla birlikte uygulanır işletim sistemleri.

— Network OS → NT

— Distributed OS

6- OS'ının işlettirdiği CPU zamanı dağılmasına göre olur.

Introduction

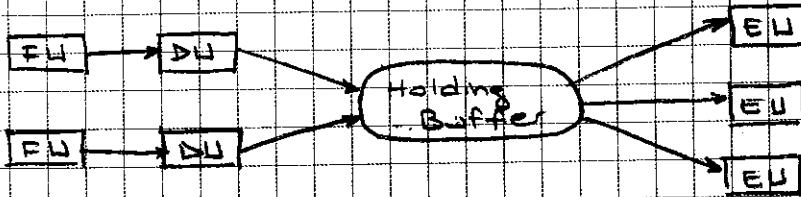


Tanımlı 2 CPU vardır: CISC + RISC

Günümüzdeki CPU'lar Fetch Unit ile bellektten emir getirir. Decode Unit sayesinde kod daha sonra (Fetch'in getirdiği kod) ve hangi kaynakların kullanılacağına

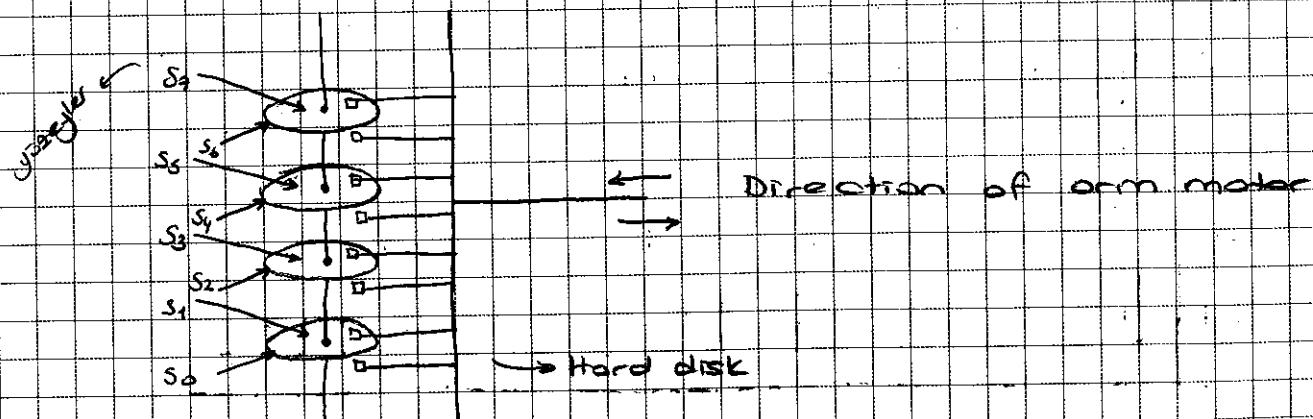
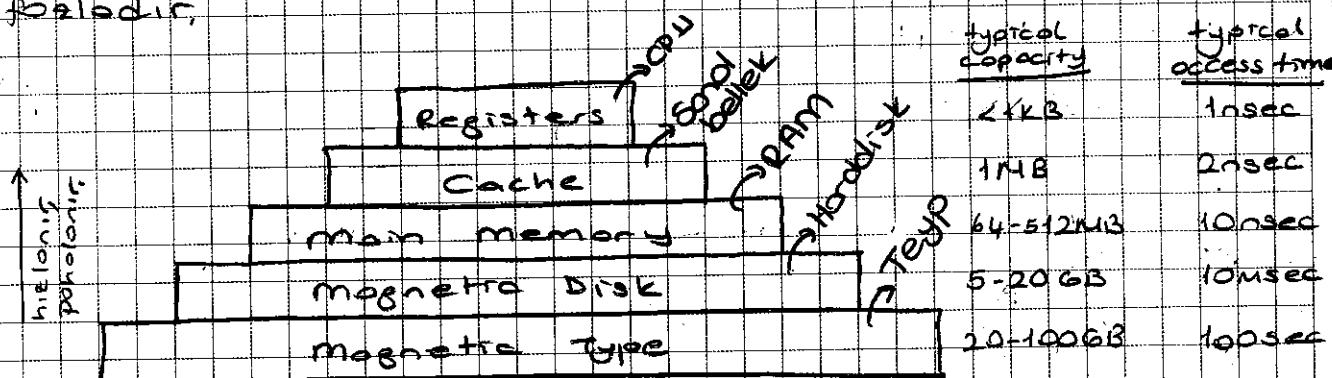
birimler.

Fetch Unit → Decode Unit → Execute Unit



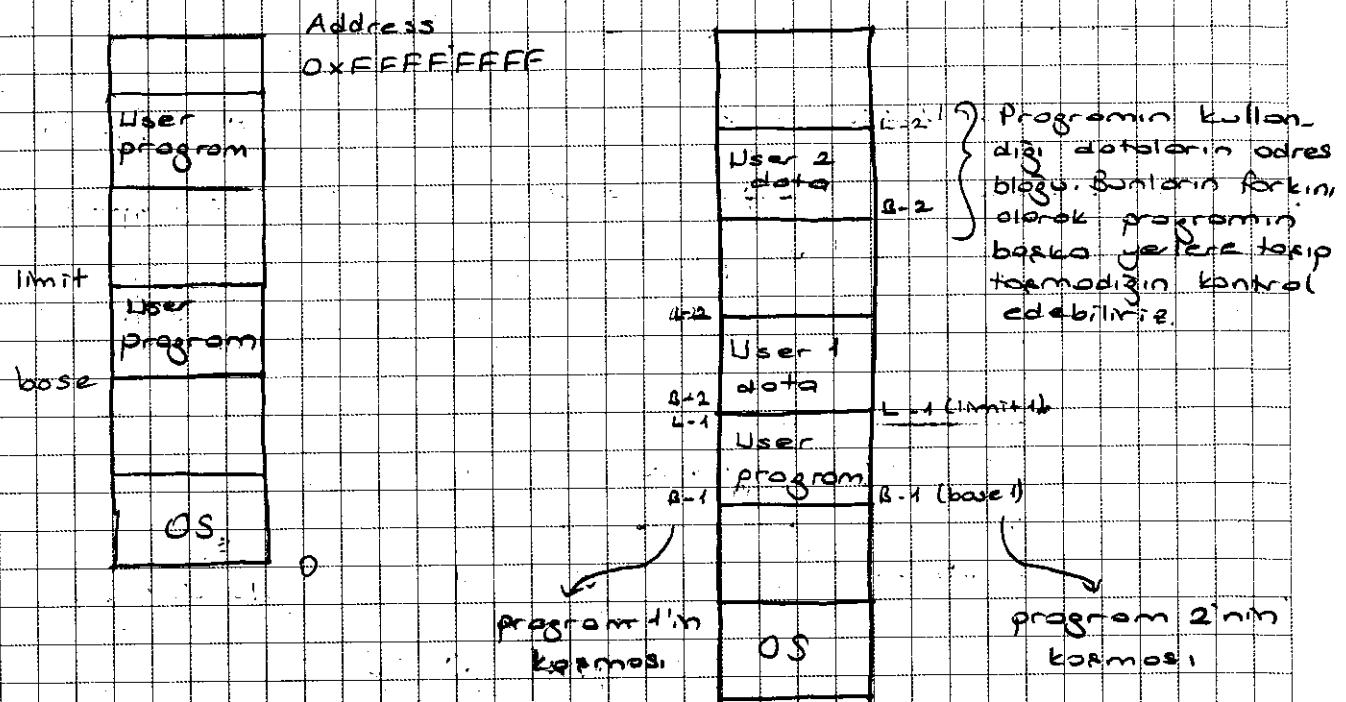
Execute unitlerin data flow'u olmasının nedeni execute istememin data flow'u zamanı almışdır.

Bu yüzden EU'lar, FU'lar ve DU'lerden data选拔dır.



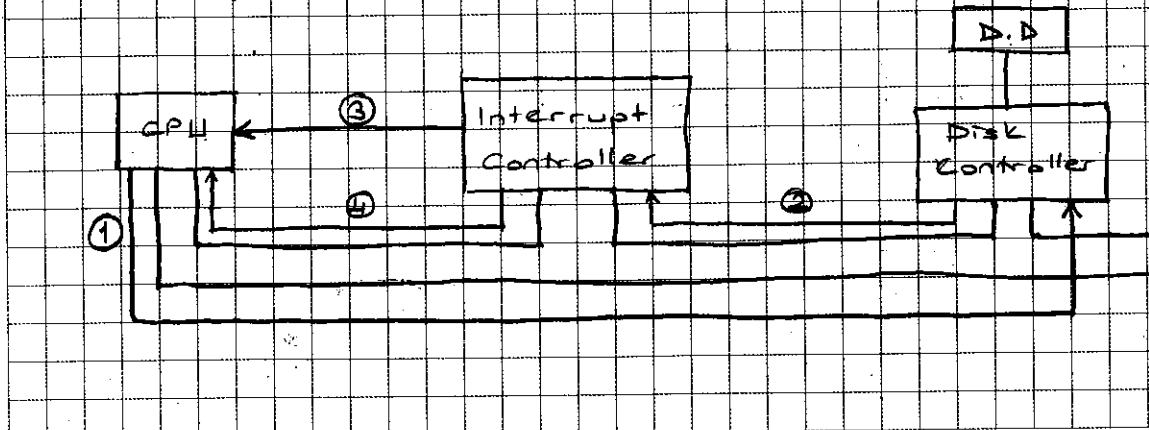
Read / Write head (1 per surface)

Bellek Haritası (RAM İsim Geneli)



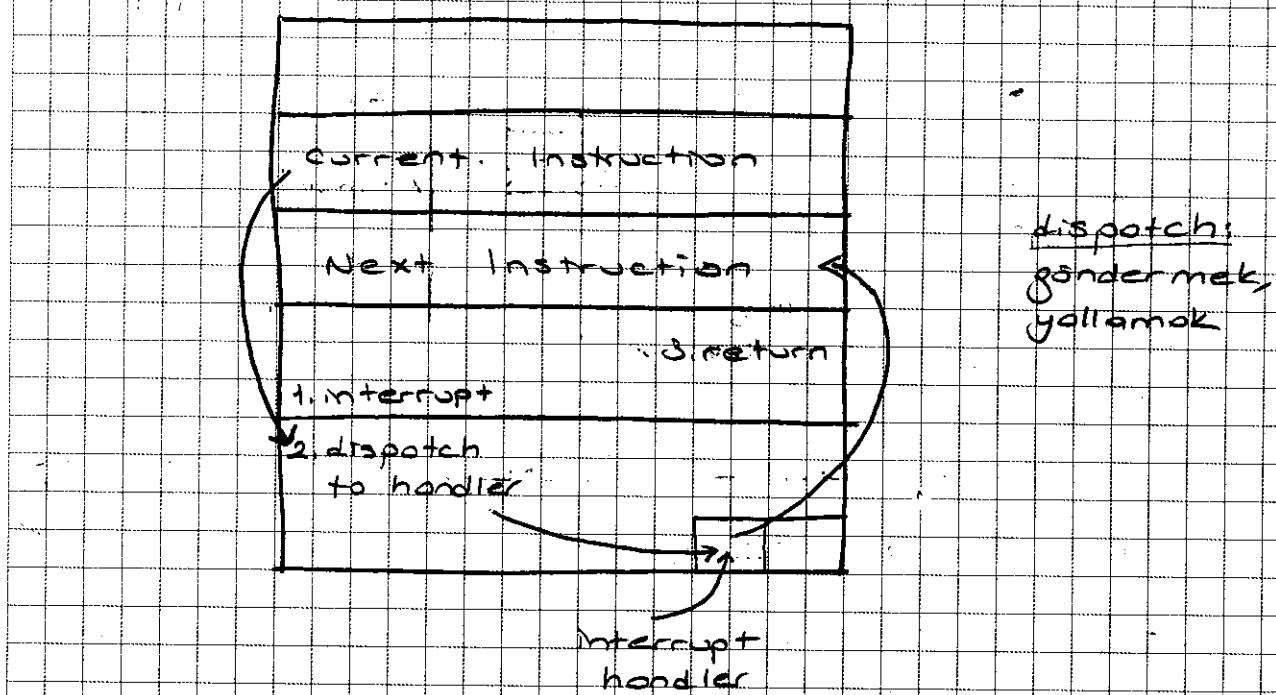
Program 1 ve program 2 körmekten B-1 ve L-1 ayırdıktan sonra program 1'in B-2, L-2, program 2'in B-2 ve L-2 kullanılır.

Input - Output Devices

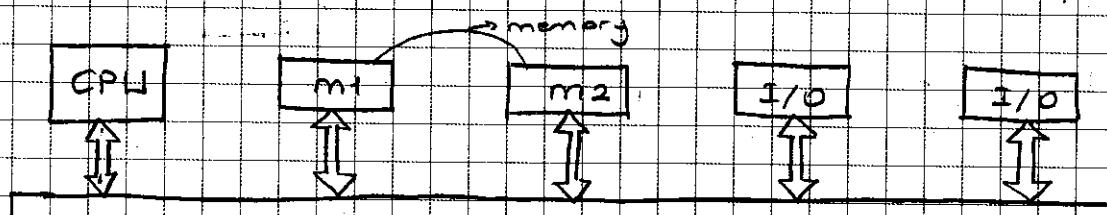


CPU, disk controllere emir veriyor, Disk controller interrupt controllere gönderiyor. Interrupt controller (3) CPU'ya olayın tamamlandığını gönderiyor, (4) özerinde CPU'ya istenilen bilgi gönderiliyor.

Interrupt Handling Process

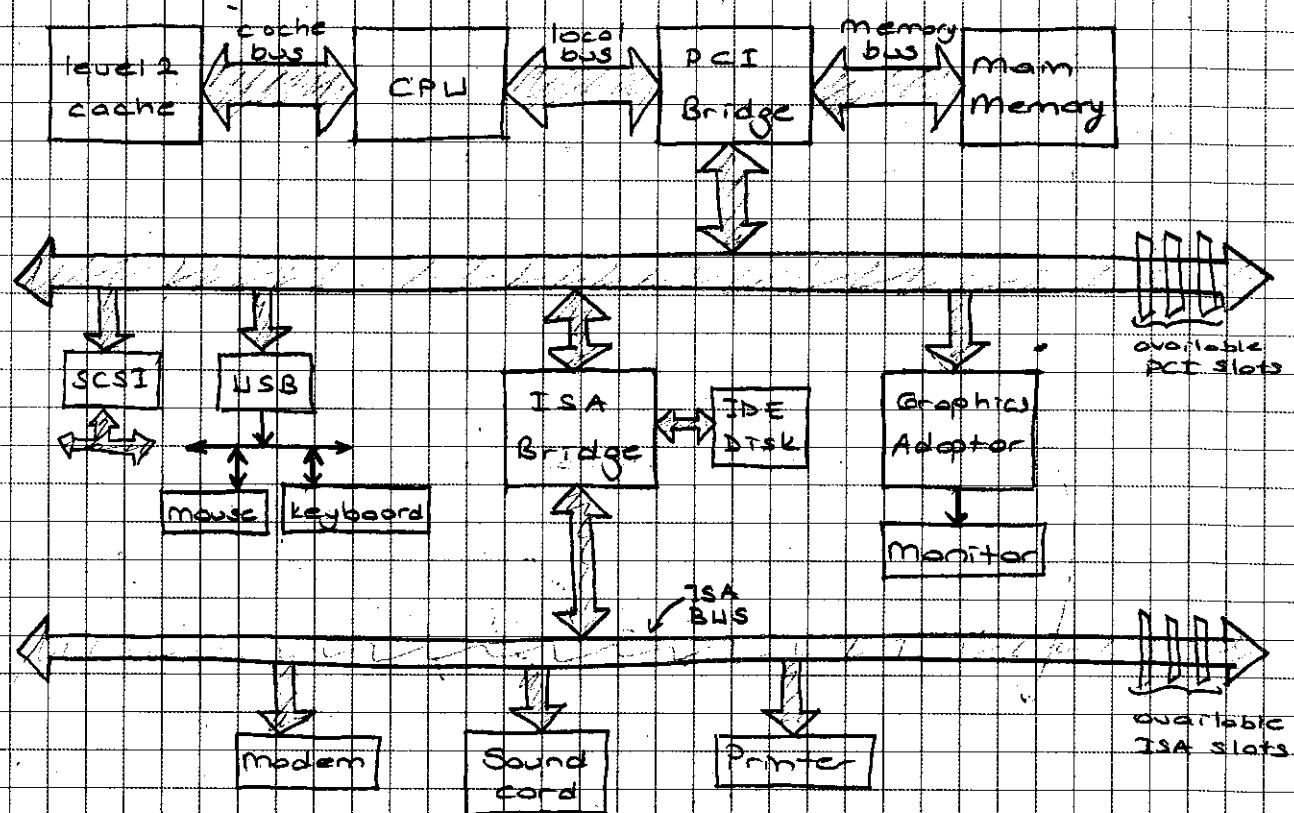


Temel Mimarisi



1980'lerin mimarisı

Gesamtdatenkette General Minimuster



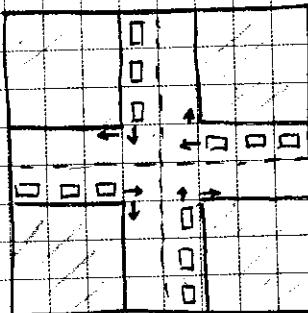
SCSI: Small Computer System Interface

USB: Universal Serial Bus

ISA: Industry Standard Architecture

PCI: Peripheral Component Interconnect
(PCIe)

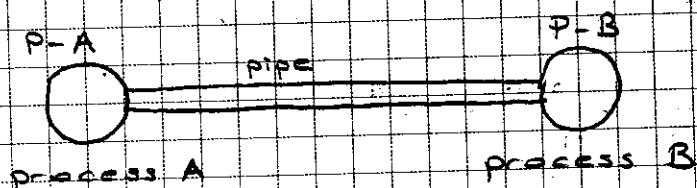
Dead Lock



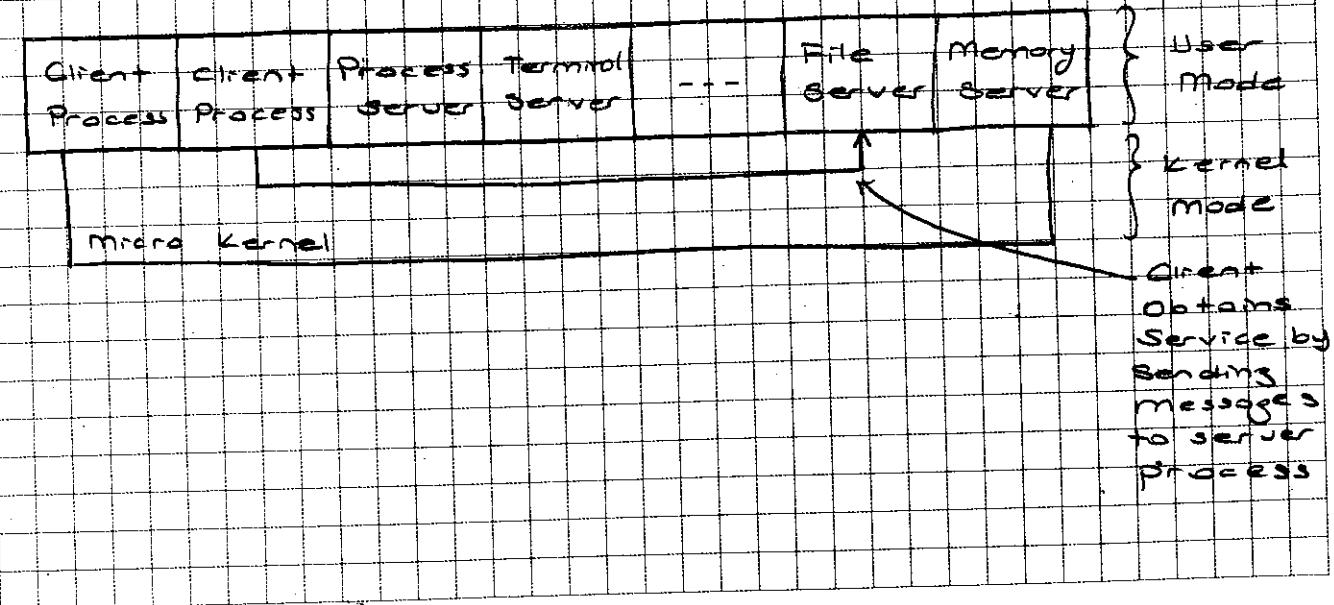
Bu aradordan biri biri yönenden uzağımına 'ye' kesişmekte olamaz.

Haberleşme

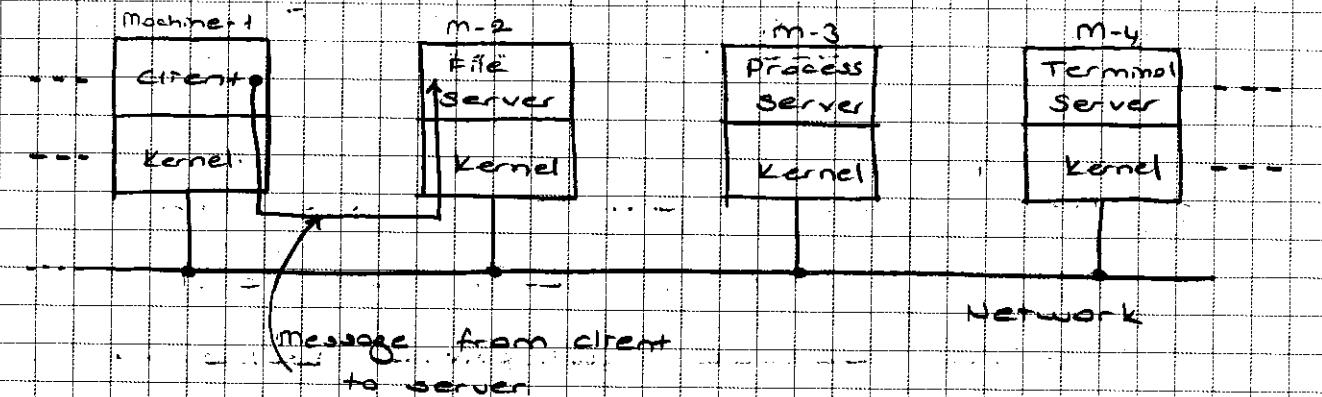
Unix'te iki process haberleşebilir.



Görmekdeki sistem sisteminin yapısı.



Distributed System



Client, M-2'nm. Kernelinden istiyor. Kernel bı isteği File Server'a ittiyor.

Düzenlik sistemlerde hedef problemleri bir makine çözüyor. İşleri matemelî oranda paylaşmamak gerekir.

Processes

- Process concept (process kavramı)
- Process scheduling
- Interprocess communication
- Dead Lock
- Threads

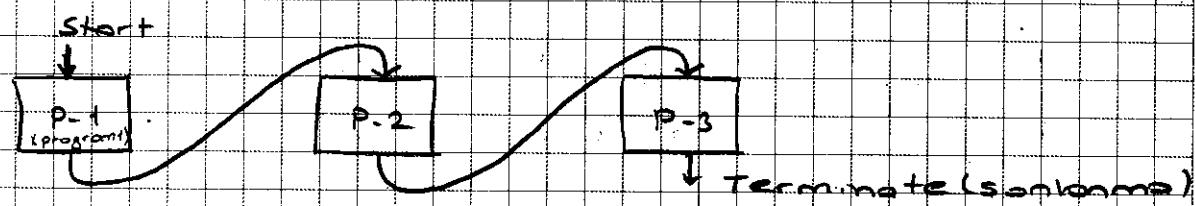
Process, (task)

What is process?

Bir programın kavutması sonuc olunan şeye process denir.

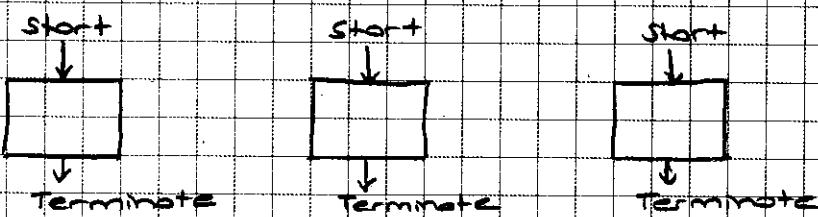
Disketde yerlesmiş program, işletim sisteminin ilgili dosyaları okuyerek ve kaynakları kullanarak program, belleğe yerleştirip kullanılabilir hale getirmesine process denir.

Sequential Execution (Sıralı Kasma)

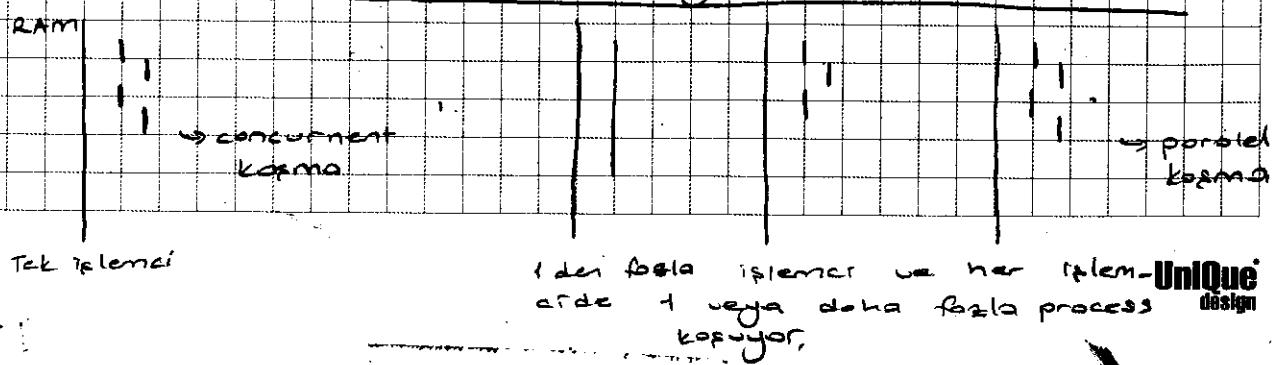


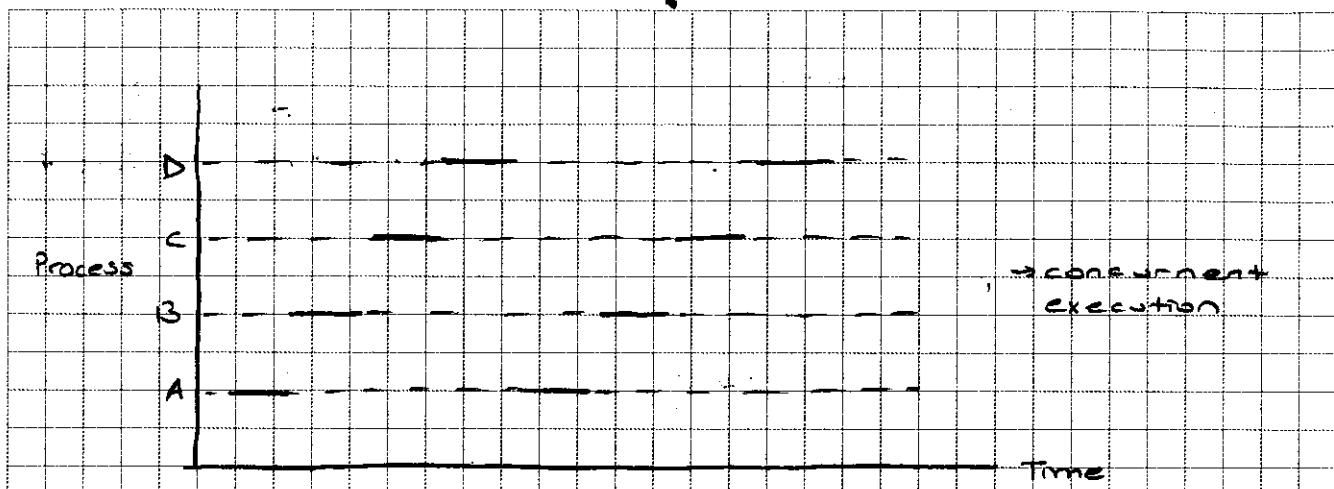
Concurrent Execution

Günümüzdeki sistemlerde kullanılır.

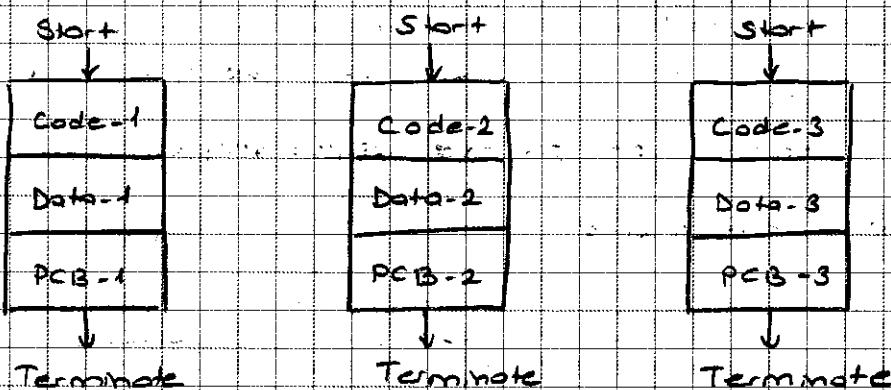


Parallel Kasma (process arası kodlar ilişmeli var)





How to Implement Concurrency?



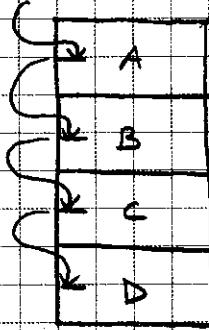
PCB (process control block): processlerin herhangi bir anda durup yeniden başlaması için gerekli olan bilgileri tutar.

Program Counter

Bir process'in bir kismi konsıdakton sonra diğerine geçilir. Bu, sekilde diğerlerinin bir kismı konsıdakton sonra bir diğerine geçilir. Son process'in bir kis-

m. kəşvildəkton sonra təkrar bəzədənərək həlinin yerindən processlərin kəşvimesinə davam edilir.

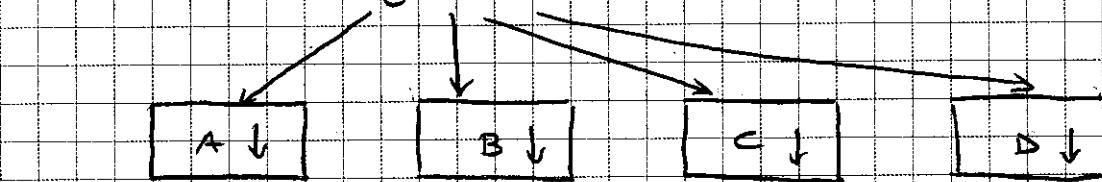
One program counter



process
switch

Program counterin içəriği sıxılış deyər, fərqli processlər arası gəzintidə saklanmır.

Four program counter



4 tane program counter olşaydı, onların deyərlərini saklmaya gərek kalmadı.

Process Table (PCB - Process Control Block)

Process table'da bulunanlar:

- Registerlərin nüvəkləri.

Günki process yeniden işlətməye başlananlığı zaman koldığı yerdəki bilgilərin bilinməsi gə-

rekr.

- Program counter
- PSW (Program Status Word)
- Process state
- Pointers (code vs data ram)
- PID (Process Identifier)
- Process priority (öncelik)
- File descriptors
- Pointer to parent process
- Pointers to all children of the process
- Hangı işlemleri gerçekleştirme konusunda bilgi

Switching Process (Proseslerin Açıktırılması)

Açıktırılması sırasında

- Save the "real world"

Prosesin bittiği bilgileri CPU'dan alıp saklanır. (Registers'in içeriği)

- Restore the "real world"

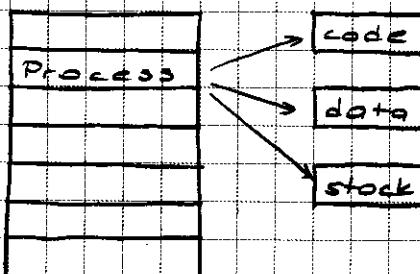
Schedule tarafından seçilen prosesin bilgileri yüklenir.

Böyledice processler arası switching geçisi yapılabilir.

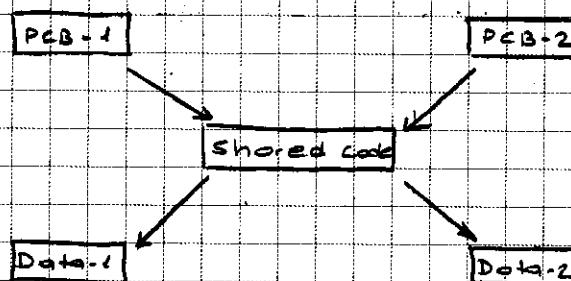
Process switching doneğim tarafların da des-teklər.

System Process Tables

Process tables



Shared code

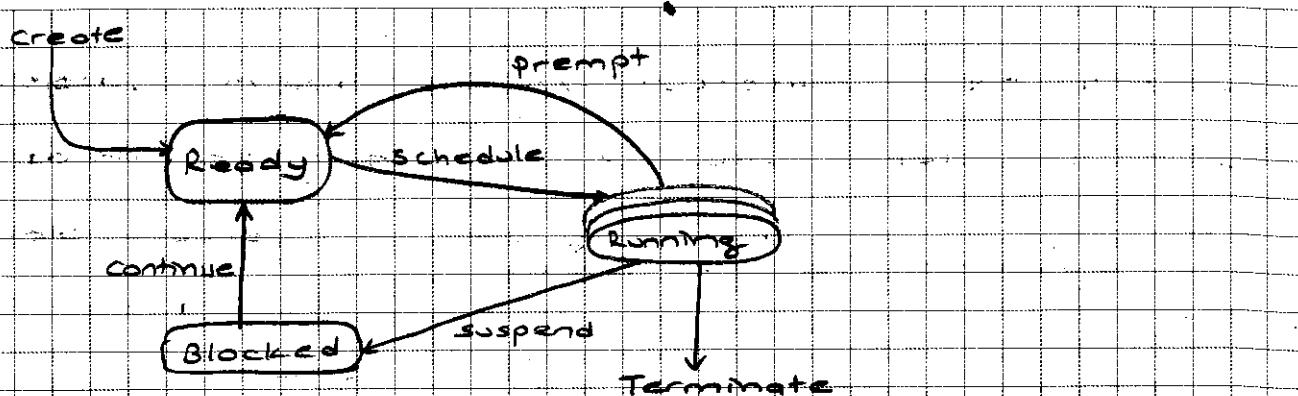


Farklı processler aynı kodu kullanabilir.

Process State

Bu process'in 3 durumu olabilir.

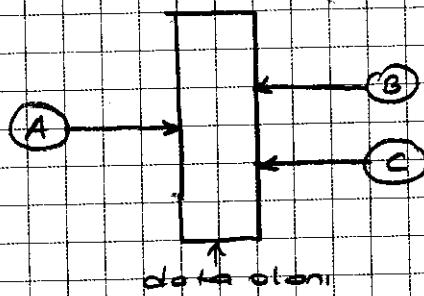
Process genel olarak bu durumlardan birinde oluraktır.



- blocked (askıya almak): Mesela bir kaynağın elle gizlmeye çalışıyordu.
- Askıya alınan ve hazır durumda kalanlar CPU zamanını tüketince,
- create: Bir process mevcut bir process tarafından veya bir servis tarafından yaratılabilir.
- terminate: Normal olarak sonlandırılması mümkün. Bir programın external olarak sonlandırılması mümkün. Bu sonlandırma bir kullanıcı tarafından yapılır. Internal olarak sonlandırma yapılır.
- schedule: Hazır rüpler arasında bir process seçilir ve kafası, tam CPU kaynakları process'e verilir.
- preempt: Bir process, diğer gelen process'ler yatkın olduğu sahip olduğunu keşfederken (preempt) hazır kaynığına geri oturabilir.

- suspend: BuPROCESSLER I/O KAYNAKLARINI KULLANIYORSA, ONI OLARAK YUKU MODUNA GEMİSSE Veya SİNKRÖNİZASYON İSTEMİNDE GEMİSSE PROCESS BLOKLANABİLİR.

Örneğin;



B processi data alanını de gemicisiye A ve C processleri bu alanı de gemicileri ve B processinin bu data alanını terk etmeye kader kusursuz beklətiñirler (blokkonclar).

- continue: Blokkonun bir processin istegi boyunca boşalmışsa (serbest hole gelmişse) bu processin həzir durumda getirilməmis continue denir

How to Create New Process?

Fork & exec

Fork: Yeni bir process yaratır. Bu yeni process konan processin bir kopyasıdır, fakat farklı bir idem yapar.

Exec: Herhangi bir programı program modundan process moduna geçiririz.

Fork ile oluşturulan bir process'in içinde exec çalıştırılabilir. (Tersi de mümkün)

Örnek

```
void main()
```

```
{
```

```
    int child;
```

```
    if ((child = fork()) == -1)
```

```
        return (-1);
```

```
    else
```

```
        if (child == 0)
```

```
{
```

```
            printf ("This is the child executing.\n");
```

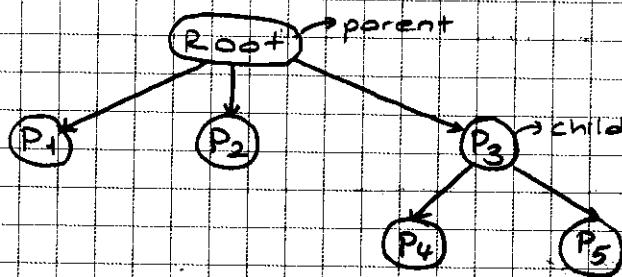
```
            exit (1);
```

```
}
```

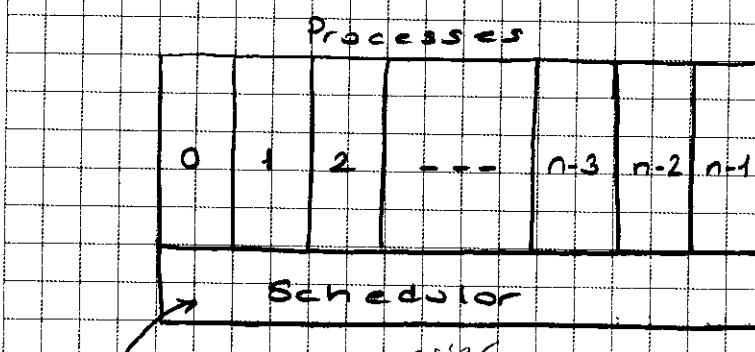
```
        printf ("This is the parent\n");
```

```
}
```

Parent - Child Relationship



Proseslerin Koşma Durumları



Bütün gelen ^{processler} interruptlar, ve hafiflemeyip burada
hallediyor, ist toraşa gitmüyor.

Tipik process tablosu

Process Management	Memory Management	File Management
<ul style="list-style-type: none"> Registersler Program Counter Stack Pointer Öncelik Scheduling parametreleri Process ID Parent processle ilişkili bilgi Processin boyutları prosesin aktif olarken CPU'yu kullanması 	<ul style="list-style-type: none"> Pointer to text segment Pointer to data segment Pointer to stack segment 	<ul style="list-style-type: none"> working directory User ID Group ID File directory

Bir interrupt geldiğinde

- 1- Hardware program counter'ı saklar.
- 2- Hardware interrupt vektöründen yeni bir program counter ekle. (Eskiyi saklar)
- 3- Bir procedure registerlerin türüklərini saklar.
- 4- Yeni stack oluşturur. Servis programı bakiş okuma işləməni yapar. Schedule deyince girer ve, hangi programı çalışdırın, belirler. O program tətqiflənir və başlıdır.

Threads

Genelde user-level threadlerden bahsedilecek.

Processes and threads

- A process is kernel-level entity

Prosesin yapısına onca system callı var. Dolayısıyla process'e kernel-level entity denir.

- A thread (or light weight process)

Sisteme getirdiği yük hafif olan bir processdir.

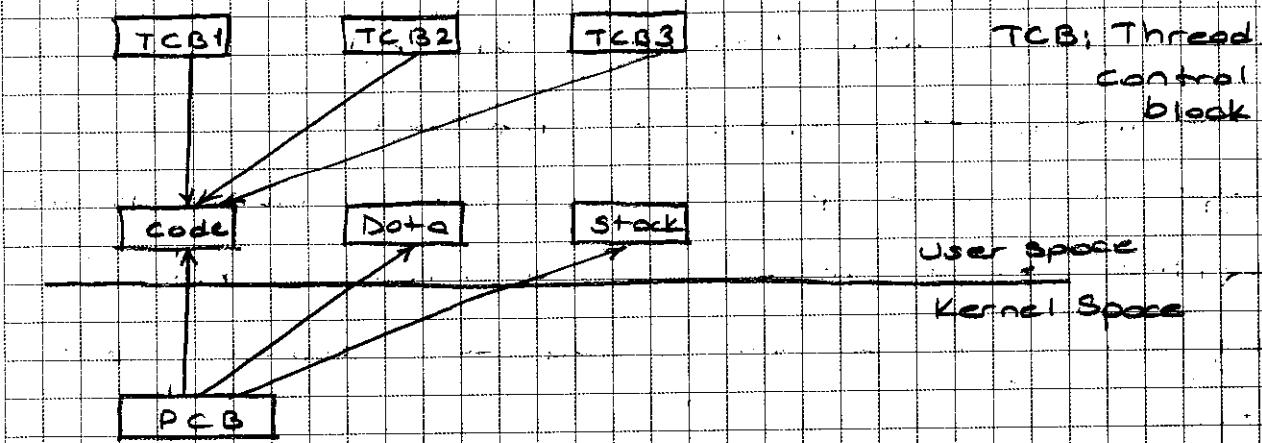
- A thread is a user-level entity

Thread'in bəstəsi - yəqinlər user-level'de bulunur.

Thread'a user level library'ler ilə wəqfiir. Dələyişli-

İla threadler sistemde işbirlikçi.

Process and Thread Data Structures



PCB code, data, stack'ta bir degrazılık yaptırı-

da TCB'lerin hepsi buna gider.

Characteristic of Threads

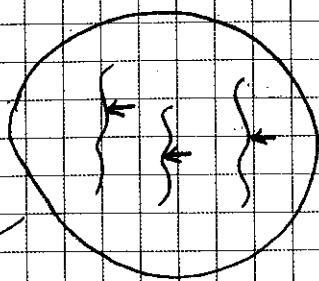
- The TCB (thread control block)

TCB'ının içinde

- Program counter
- Register set
- Stack space

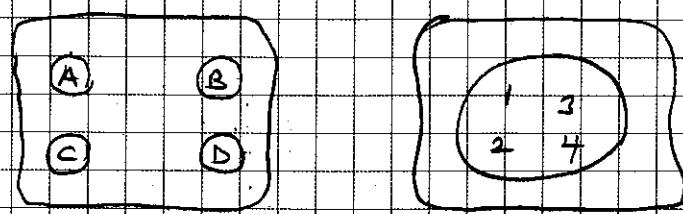
Her thread'in program

counter, usulü wordür



PCB TCB'ye nüfuz etmemistir

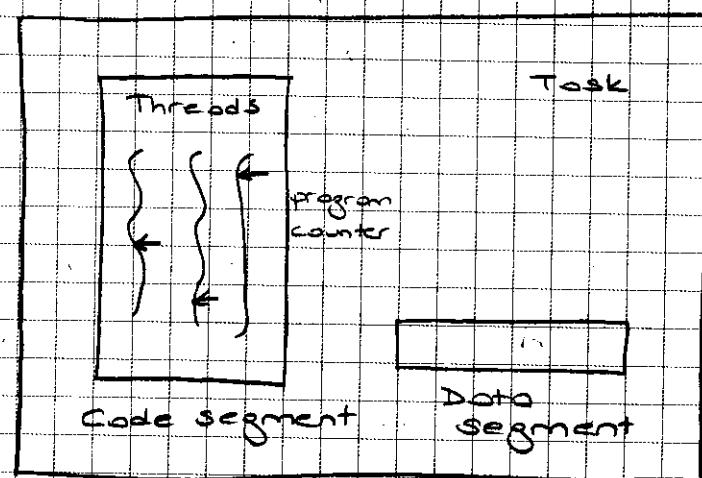
- Geleneksel olarak bir process bir tari içine eden tek bir thread olarak görevlendirilebilir.
- Process'in içindeki threadler aynı bellek üzerinde otururlar. Bu user-level space'dır.
- Eğer bir thread process değilkenlerinden birini de kullanırsa bu threadler o değilidğini görür.
- Eğer bir thread bir "daya" sahip olursa diğer threadler de bu dayayı okuyabilir.
- Threadlerde system-call'lar sağlanamadığı için threadler daha hızlıdır.
- Threadlerin sisteme getirdiği yük azdır.
- Thread'in yaratılmasıyla kernel'in yapısı etkilenecektir. Dolayısıyla kernel-level'e kaynaklar tüketilmeyecek ve thread yaratıp kullanmak daha ucuz olacaktır.



Bu ifadesi 4 threadlu bir process yaratıp kullanmakense, 4 threadli bir process yaratıp kullanmak daha iyidir.

• System threads in working on惚惚惚惚

Threads of The Task

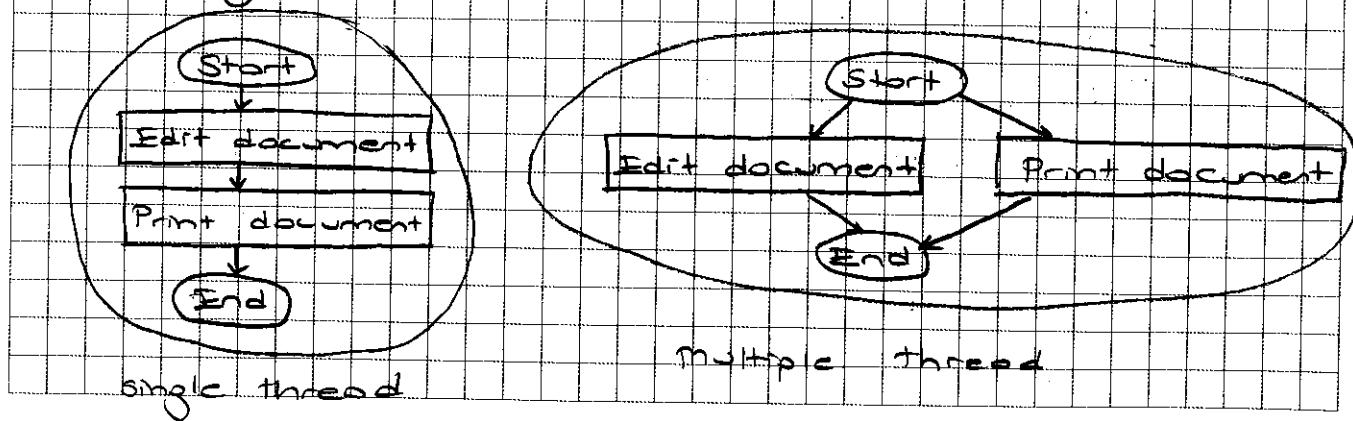


Threads syn code ve data segmenti paylaşır
yollar

Burada her threadin kendi yerini buluyoruz.

Single Versus Multiple Threads of Execution

(Tek tredde körmeyle çok threadde körmeye kor alıptırıyoruz.)



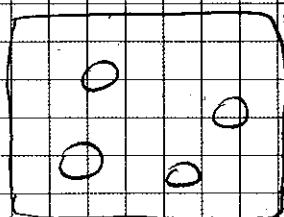
Multiple thread yapsınlar yazma tarifi için söyle
beraber yazıldıkları gibi yine de yazınca yazılan say
bosluk.

Multiple thread: Program bapladı, birinci thread
yarattı, ikinci thread bulduyu - miktada process sentan-
dırılıyor.

Some Benefits of writing Multi-threaded Program

- Performance artışı sağlar.
- Bir alt-takım sisteme paralellik sağlıyor ve her biri
bir thread paralel olsa kaynakların her birini
paylaşabilir.
- Increased application throughput ↑
- Increased application responsiveness ↑
- Enhanced process-to-process communication ↑

Parallelism:

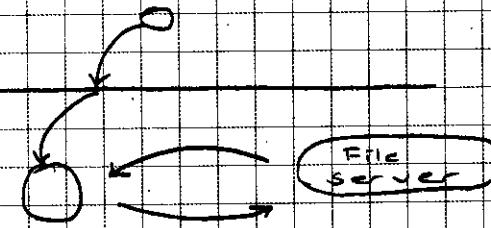


Bu sisteme de Δ islemci olsun. Bu durumda fork() threadler fork() processler öterinde Δ zamanları kaser. Burada kullanici tarafindan iszel bir effort saglanmasina gerek yok. Yani sistem buyu otomatik yapar.

Sadece CPU degil tom processin写itlerde atomatik olarak kullanilar.

Throughput: (CPU'nun etkin kullanimi)

Geleneksel tek threadli sistemi deneysel, bu process isletim sisteminde bir servis saglayici istediginda thread bekleyecektir. Guncel servis bitmesi gerek.



Kernel file serverden bir is istedir. FS'nm is bitirmesi beklenecaktir. Dolayisyla diger gelen threadler kerneli bekleyecektir. Yani kernel FS'nm cevap vermesini bekleyecektir ve baska talem yapmayacaktir.

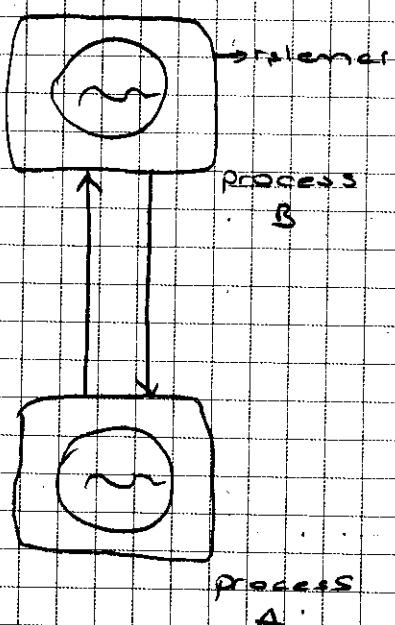
Multithreadingde ise birden fazla thread aynı anda çalışabiliyor. Bir thread takip ettiğimizde yalnız da varız eder. Bu da throughput'u arttırır.

Responsiveness: (Hızlı tepki verme)

Bir process'in bir parçasının bloklanması diğer parçaların da bloklanmasına neden olmamalıdır.

Bu tür application'lar birbirinden bağımsız threads kullanır. User level threads bloklanır, kernel level olanlar bloklanmaz.

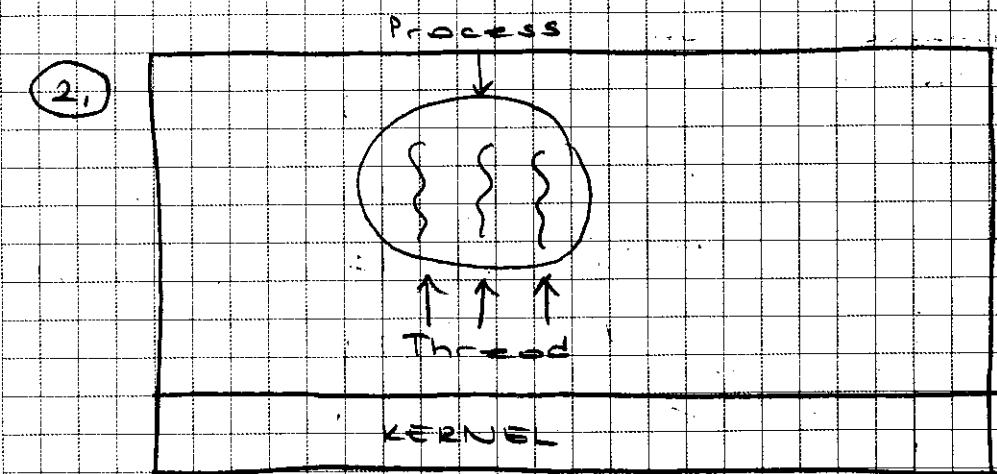
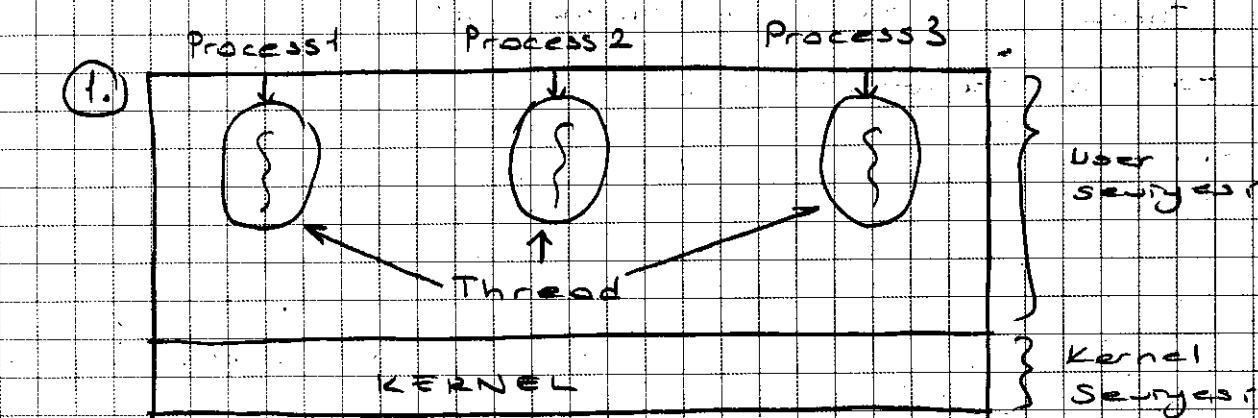
Communication:



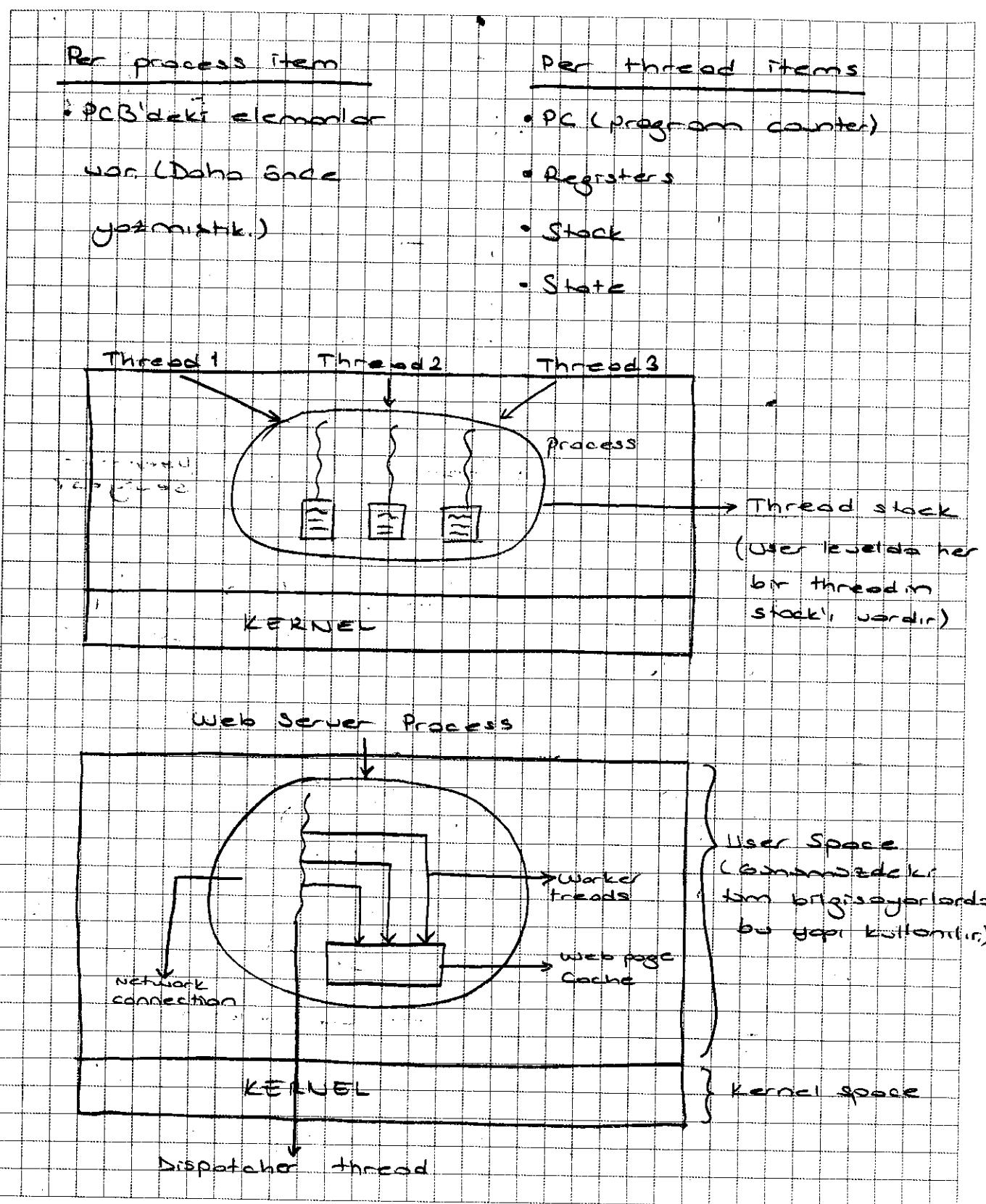
A processinden B processinden x degerti alip hesaplamaya başlayarak mesaj gecerek x'yi B'den alır. Bu timsahı sızır. Hesaplaması 9msa gecse 1 tane 10msa'ya ihtiyac vardır. 10 hesaplaması 100msa 102.4 m. Gerek. Hesaplaması 90msa sızır. Sonra sıradak 10 gecikme ile hesaplamalar yapıılır. (Tek threadde hesaplamalar)

Gök threading kullanarak bu görevler arasında
yalanızı. Genişme (9 tane ram 10mcc genişme var)
sayısı ile sayısından fazla ve thread sayı-
sını ortak olarak genişme sayısını artılabilm.

Ek bilgiler

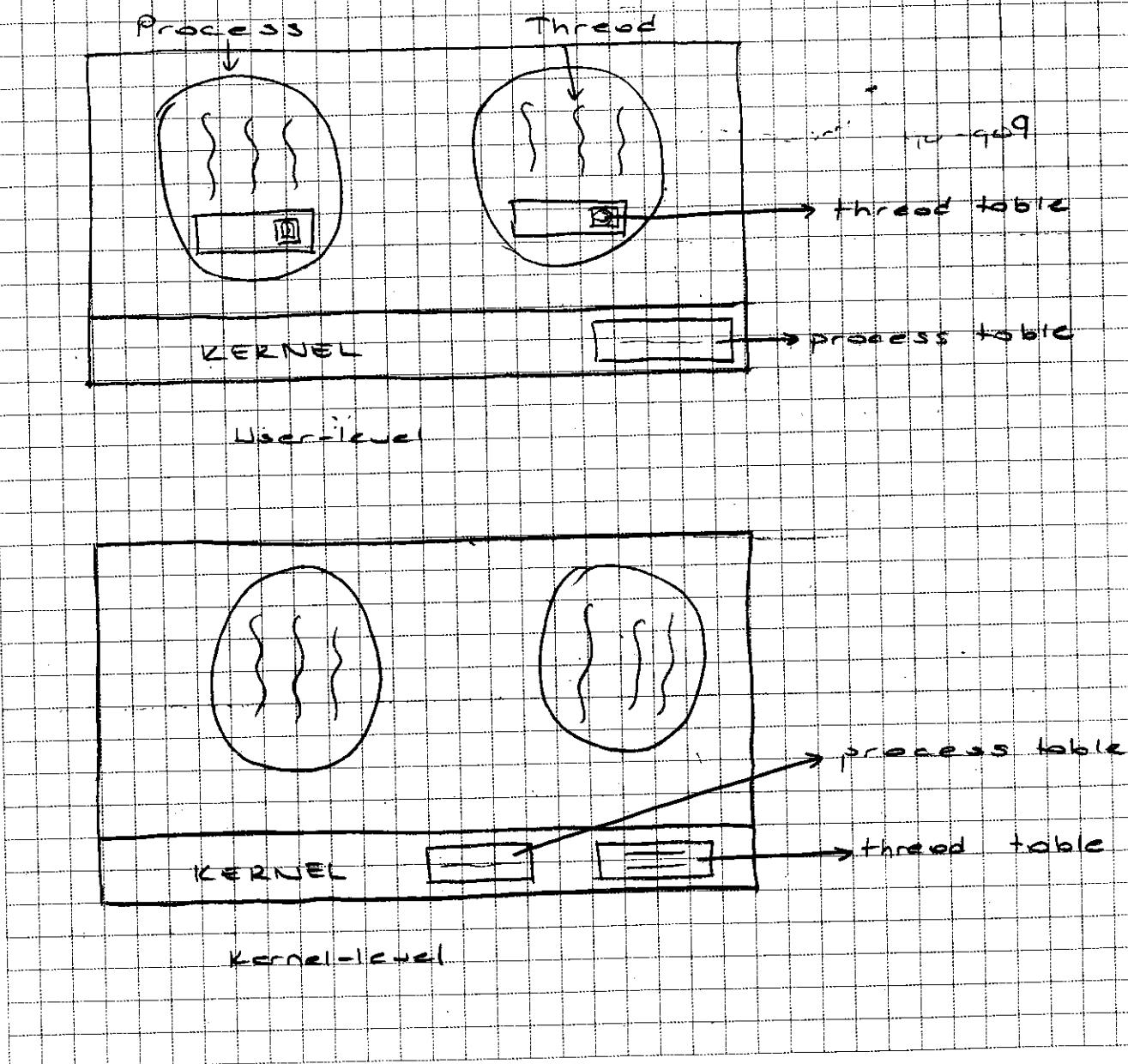


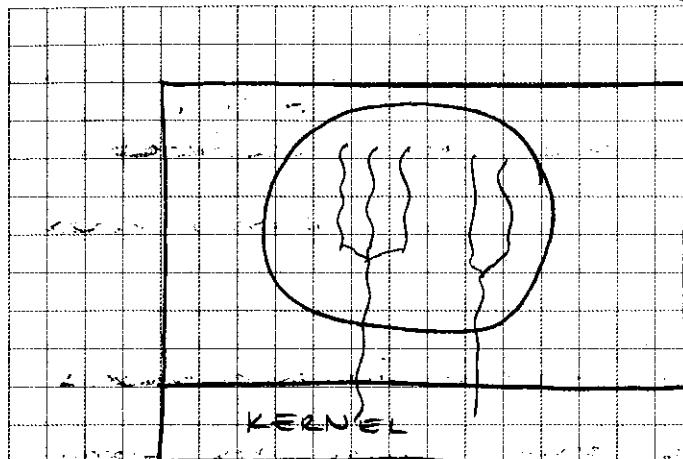
1. sisteme 1.ye göre daha se malzef getirin.
Günlük 2. sistemde 1. kullanım



Herbir yeni istek i̇m bir worker threads yorulur. Bu istek s̄erken gelir di̇c̄r i̇stekler de kerebilebilir.

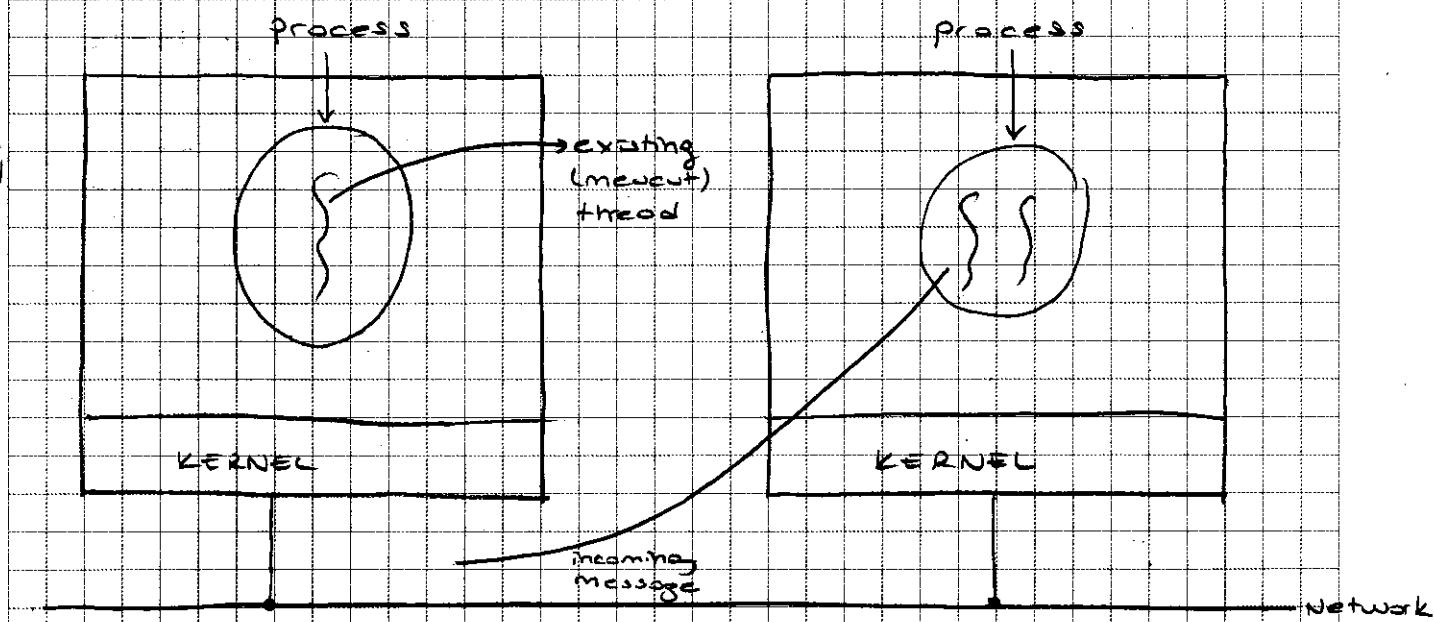
User-level Kernel-level Threads:





Hibernat yapı

Pop-up threads:



Networktan bir istek gelir ve yeni bir thread yapılmıyor. Olusturulan bu thread b^r gelen mesajla ilgileniyor. Yeni mesaj ne yapanın isteyince bu thread onu yapıyor.

18. 10. 2007

Process Scheduling (Tarifeleme)

Scheduling: Process veya processlerin secerke, işlenmesi veya işlemeler üzerinde kontrolmayacheduling denir.

Bu içinde process $\xrightarrow{\text{ready}} \text{running} \xrightarrow{\text{finishing}}$ geçer. Buju yapan OS processin scheduler denir. Scheduling Tkr seviyesi ayrılr.

1. Programın bulunduğu duruma göre tarifeleme yapılır. Process.

- İşlemde kullanılabilir durumda olabilir.
- Parçaları veya türünü ona bellete olabilir.
- Swap bellete olabilir.
- Henüz başlatılmamış olabilir.

2. Tarifeleme doneşme göre

- Long term scheduling:

Burada processin kullanılabilecek durumda olmasına göre hazırlık yapılıp olmasına karar verilir.

- medium term scheduling:

Burada processin ona belte olup olmamasına karar verilir.

- Short term scheduling:

Hangi processin kasıtlarıdır (CPU'ya分配される) kural verilir.

- I/O scheduling:

Birden fazla processin I/O cihazına erişimi stratejisi, Burada hangi processin I/O cihazına erişileceğine kural verilir.

Scheduling Criteria (Tartılıklama Kriterleri)

1. Fairness: Her bir processin hak ettiği CPU zamanını almasıdır.
2. Efficiency: CPU'yu % 100 kullanmak hedeflenir.
3. Response times: Interactive kullanıcıların response time'in minimize edilmesi hedeflenir.
4. Turn around: Batch uygulamaların tam turn-around zamanının minimize edilmesi hedeflenir, A2 zamanında daha fazla process başlatılabilir.
5. Throughput: Birim zamanda en çok işi bitirmek hedeflenir.

Deadline: "En sonuna kadar bu işi bitir" denildiğinde o zamana kadar bu işi bitirmek.

Performance Criteria (Kullanıcı Beklentileri)

Criteria

Aim

1. Response time (20ms gibi) Disk response time isteniyor. Kullanıcı en kısa zamanla kendisyle iletişim kurmayı, çok kullanıcyı sağlama.
2. Turnaround time: sistem gerekilmesi ile ilişkilip bittiğinde orasındakilerin kısır olması isteniyor.
3. Deadlines Korulanan deadline sayısını maksimize etmektedir.

System Oriented Performance Criteria (Sistemin Gerekçeleri)

1. Throughput: Burada maksimum sayıda işlem yapma isteniyor. Hedefleniyor.
2. Processor Utilisation: İşlemciyi maksimum (%100) ölçüde mesai tutmak hedeflenir (yöndərək tale)
3. Over head OS: İşlem sisteminin getirdiği yükü minimize etmek isteniyor.

* throughput ile interaktiflik ters düşebilir. Throughput interaktifliği tehlikeye sokabılır.

System Oriented Other Criteria

1. Fairness: System threadlerin eşitlik gibi davranıyor. Eğer böyle davranmasa bazı threadler kalır (örnekli bir kişiye kalmak = starvation)

2. Balancing Resources: İşletim sistemi ihtiyac duyduğumuz surer bir şekilde kaynakları meşgul tutmaya çalışıyor. Birbirin kaynakları maksimum düzeyde kullanmak isteniyor.

3. Efficiency Priorities: Sistem bazı processlere (yuksek öncelikli) daha yüksek öncelik veriyor ve bunların sistem zamanını daha fazla kullanmasını sağlıyor.

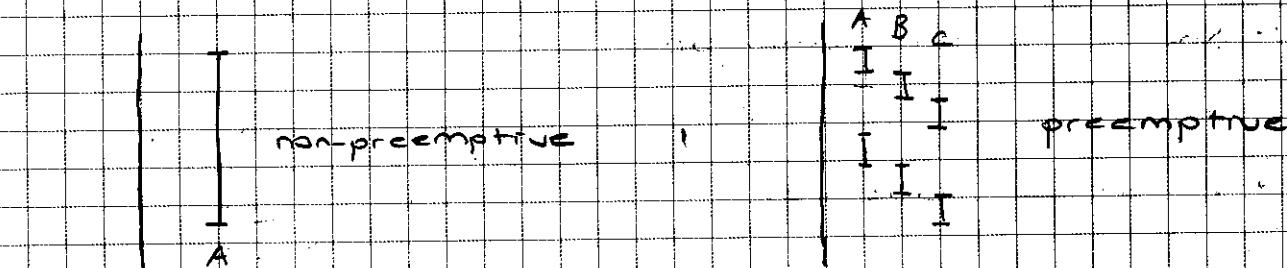
Important Factors (Tarifeleyici bu faktörlerle bakın!)

- I/O boundedness of a process (I/O ile ilgili bir process hazırlık kriteri gelir. Bir hazırlık kriteri olan takip eden process)
- CPU boundedness of a process (I/O birenden daha önceliklidir.)
- Is process interactive or batch?
- Process priority (starvation olmaması şeklinde soruluyor.)
- Page fault frequency (sayfa RAM'de değişse process durdurulur.)
- Preemption frequency (Prosesse verilen süre bitmeden önceki)
- Execution time received
- Execution time required to complete (Processin bitme süresi de kolaylaştırılır. Bütünerek dolaşır.)

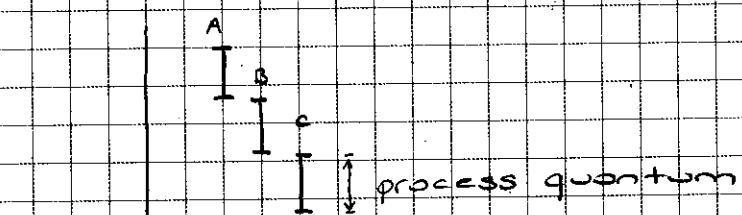
Types of Scheduling

non-preemptive: Bu durumda scheduling algoritması, CPU'yu process'in elinden almıyor. Burada process tamamen nüccaya kadar konuyor.

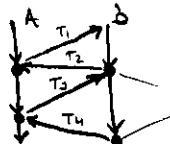
preemptive: Burada scheduling process'lerin istedigiz zaman kesebilirler ve başka process'lerin de CPU'yu kullanmasını sağlıyor.



the interrupt clock, process'lerin kosmosunu belli bir arada, saattir. Hangi process'in ne kadar süre zarde kozası (quantum) belli olur. OS belli bir



Bu durum da process'lerin makul bir zaman vermektedir. Aksı halde CPU zamanı tek bir process tarafından sürekli olarak kullanılamaz; Bu transfer fonksiyonu ile de sağlanabilir.



Transfer fonksiyonlarını programın time gerestrink design

Scheduling Algorithms

- FCFS (First come first serve)
- Round Robin
- Virtual Round Robin
- Priority
- Priority Classes
- Shortest Job first
- Shortest remaining time
- Highest Response Ratio next
- Feedback Queues

FCFS (First come First Serve)

Implementation: Her bir process hisse olduğunda hizmet etmeye başlıyor. O andanki process bitirildiğinde hizmet kuyruğundaki en eski (yeni) process başlar.



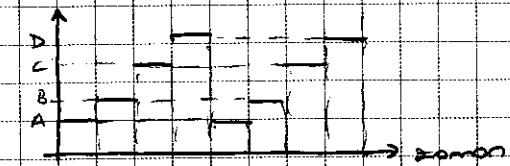
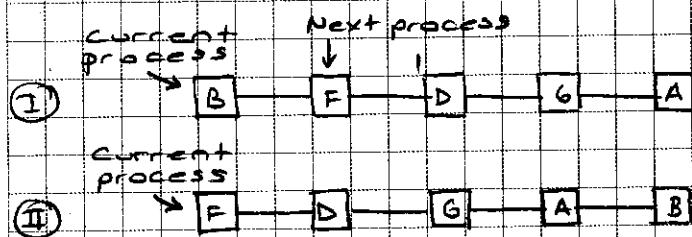
Characteristics:

- Gerçeklenmesi çok basit,
- non-preemptive
- Kiso ve I/O bağımlı processlere önem vermez.

Round Robin (RR)

Implementation: Prosesler bir FIFO yz jerler ve titirilir.

Burada her birin c sabit bir CPU zamanı verilm.



Tik durunda current process koyup quantum'a gider.
ni (konusa zamanı) bitiginde kesiip koyruğuna sa-
huna atılır ve sonraki process (next process)
kesilmeye başlanır. Yani bir process kuyulursa
sa o da kuyruğun sahuna atılır.

Characteristics:

- Preemptive (zaman paylaşımlı ortamında iyidir).
- Uygulamak basit.
- I/O yoğunluğlu processlere分配 vermesi.
- Quantum Size

Some Options:

- Large or small (çok büyük veya çok küçük olmasi)
- Fixed or variable (sabit veya değişken olmasi)

Quantum size çok büyükse FCFS'ye dönünt.

Eğer quantum size çok küçükse context switching

- Some for everyone or different (Bununla öncelikleri paylaşılabilir)

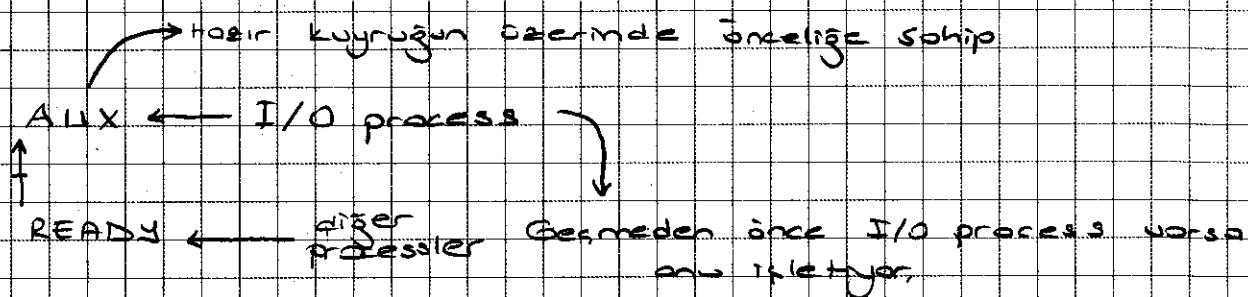
(CPU'nun işleyeceği birinci nr) olur.

* İş bir quantum, typical interactive commandan birazda büyük olacak.

Virtual Round Robin (VRR)

Round Robin algoritmasının CPU yoğunluğu processlerin modify edilmesi halidir.

Implementation:



AUX ve READY olmak üzere 2 ayrı kuyruk var. AUX dənəlidir.

I/O-processler sadece kalan commanda kəsər.

Characteristics:

- Performans qalınlığı göstermişdir ki Round Robin'den birazda daha iddialı.
- I/O işlem yapan processlər oncelik veriyor.

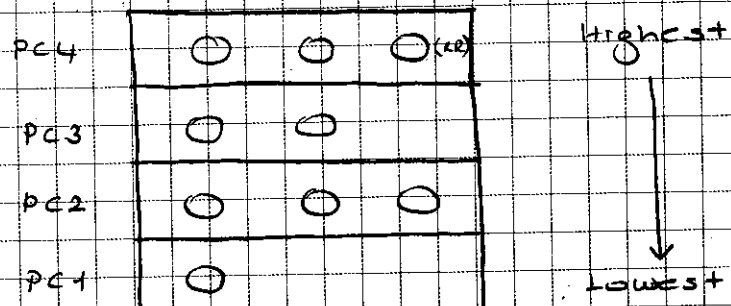
Priority

Implementation: Her bir prosesse bir öncelik verilir ve tarifeleyici sıralı en yüksek önceliğe sahip prosesi seçer.

Characteristics:

- Yükselik öncelikli processler sıralı kollar. Sonra sıralık base tarifeleyicilerin önceliğe sıralı olarak önceliklidir ki önceliği düşük olanlar da koşulur.
- Sistem processlerine (I/O processleri) yükselt öncelik verilir.

Priority classes



Implementation: Tarifeleyici en yüksek önceliğe sahip priority classes bulunan processeri koymaya başlıyor. (R2'iye göre) Sonra burada processlerin koşulması, tamamlanıkların sonra öncelik sırasına göre alt classları getiriliyor.

Characteristics:

- Eğer öncelikler zaman zaman ayarlanmasa da, sağdaki processler karışır. (Önceliğin değişmesi high class'taki process'in alt class'a geçmemesidir.)

Shortest-Job-First (SJF)

(Shortest-Process-Next (SPN))

Implementation: En kısa kalan zammalı process öncelikli processstir. Bir process'in kısa olup olmadığı, gecmişteki dönenlerine göre belirlenir.

Characteristics:

- non-preemptive
- FIFO kuyruğuna göre ortalamaya beklenen zamanı azaltıyor. Ortalamaya sürekli minimum turnaround zamanını sağlıyor.
- Kullanıcıların kötü kullanımına neden olur.
- Bir process'in ne kadar kasasının bilmesi gereklidir.
- Batch processler için uygun, zaman paylaşımı processler için uygun değil.

8	4	4	4
A	B	C	D

$$\text{Turnaround time} = \frac{40 + 3b + 2c + d}{4}$$

Karma zamanının karmaşıklığı, sırası, -
d de en büyük karmaşık karmaşık süreci oluşturmalıdır.
Karma sırası (Turnaround'a göre)

4	4	4	8
B	C	D	A

↓
en kısa
en büyük
en orta

en küçük
en büyük
en orta

- En kısa zamanda en çok iş bittiği nedeniyle.
- Her yeni processe göre yeniden sıralama yapılır.

Shortest Remaining Time (SRT)

SPN'in preemptive versiyonudur.

Implementation: Hangi processin en kısa sürede kapatılacağı belirler ve o tarifelere göre, Dövizistyle karan bir sure daha kısa süreli bir process geldiğinde karmaşıklık durdurulup kisa süreli process kapatılır.

Characteristics:

- Sistemde SJF'den daha fazla çok优先级更低的被选中。Kosma sırası takmin edilir, RR'de olduğu gibi interrupt gereklidir.
- Aynı zamanda processin kosma zamanının boydelenmesi gereklidir.

Highest Response Ratio Next (HRRN)

Implementation: Sıkılık etrafınıyı kaldırırmak için geliştirildi. Bu yöntemde bir iş CPU'yu ele geçirince bitinceye kadar devam ediyor. Yani burada işin önceliği var.

$$\text{priority} = \text{time waiting} + \text{service time} / \text{service time}$$

Herhangi bir işin işin üzerinde beklemesi engelleniyor. Öncelik seviyesi geldiğinde o iş de başlıyor.

Characteristics:

- Non-preemptive
- Kısa işler diğerlerine göre önceliği ele geçiriyor.

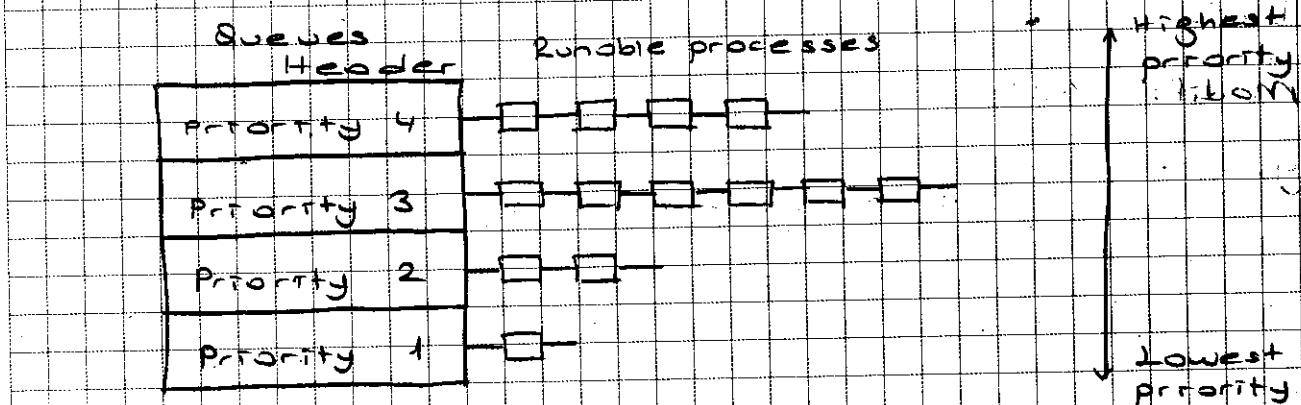
Yani işin yapılmasına sırasında ne olursa olsun işin işlemi de kesilmemesi öncelik taşıyor.

- Hala geleceğinin takmin etmenis gereklidir.

Feedback Queues

(Multi-Level Feedback Queues)

Implementation: Burada bir çok hizır kuyruklu
ağı var. Yani bir process hizırken bu kuy-
ruklardan en üsttekiyi yolluyor. Burada FIFO
mantığı geçerli.



preemptive = Bir process kırılarak herhangi bir anda kesip başka bir process katılır.

non-preemptive = CPU etc. gestoplenmesi, kalan process bitene kadar çalışmaz.

I/O process: I/O processi geldiğinde kozon process kesişir, I/O processi kapatılıktan sonra snekri process syni seneyece dener.

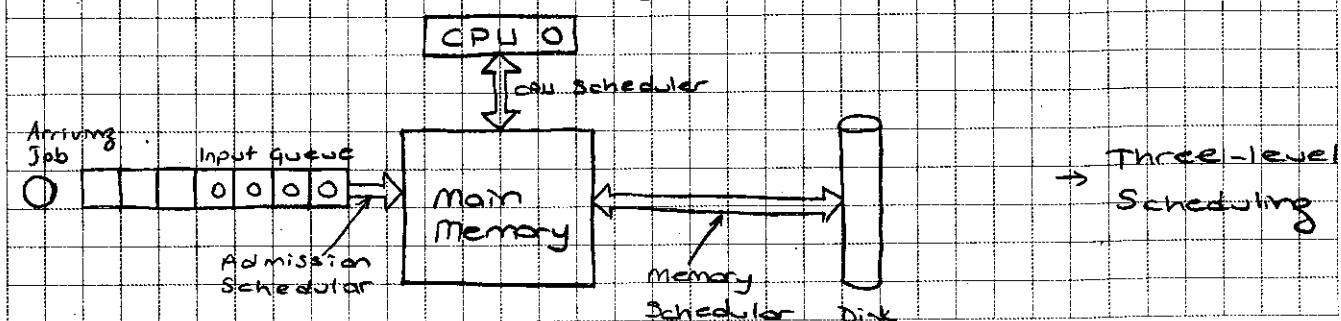
CPU bənd process. Bu tür proseslərin qurum
bitirildikdən sonra işləməyi tətbiq etdir.

Dispatching: Bir processin CPU'yu alıp kullanması
 için birinci ist serviselerdeki processlerin koşulları
 termenmişdikten sonra ona o process kabul edilir.
 Ancak bu process kabullenmiş: yükselt 'serviselerdeki bir
 process' gelmişse bu process'in koşulları karşı-
 lırek: yükselt servisdeki process kabul etmeye
 başlar.

Modifications: Bir process I/O boundlu olursa
 yukarıya doğru gitebilirler. Yani: I/O processler
 daha önceliklidir.

A Scheduling mechanism Should

- Scheduling algoritması, kısa zaman aralını tutuyor,
- Scheduling algoritması, I/O processlerin yanında
 çalışır, yani onların çalışma zamanından ge-
 lenti yapar.
- Prosesin doğasını bilmesi: gerçeker, Buna göre
 processe öncelik verir.

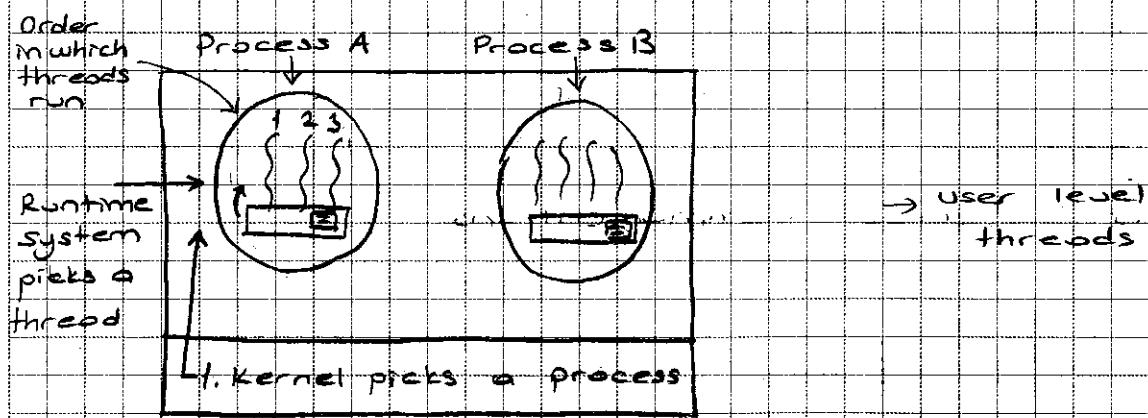


Bu, processler swap planında atılır. Page fault, faylası processlerin diskte atılır. Bu, memory scheduler yapsı.

Aynı algoritmalar 4 konudan olur.

- 1- Process swapın yapıldıktan sonra ne kadar süre geçti, process diskte ya da bellekte ne kadar bekledi.
- 2- Process hale getirile oktf olarak ne kadar çıktı.
- 3- Process ne kadar büyük. Büyük processleri öncelikli.
- 4- Process ne kadar önemli ne kadar öncelikli.

Thread Scheduling

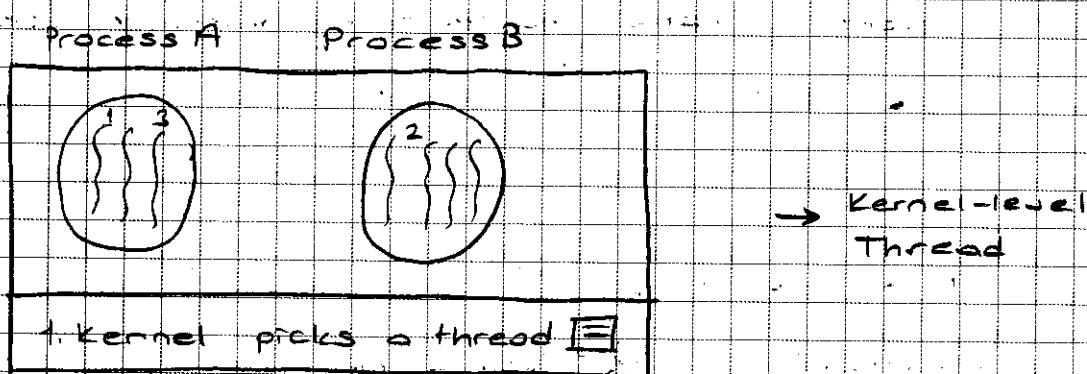


possible: A1, A2, A3, B1, B2, B3

not possible: A1, B1, A2, B2, A3, B3

Her bir processin quantum səməni, 50 msec.

Her bir process quantumunu 5'er msec'ye bölüyor,
kerneldeki scheduler sədəcə processləri tarifələr.
Sisteme dənənək yüksək getirir. Ama thread block-
ləndiğində təm process bloklanır.



Possible: A₁, A₂, A₃, B₁, B₂, B₃

Also possible: A₁, B₁, A₂, B₂, A₃, B₃

Kernel kendisi direkt olaraq threadləri seçəbilir. Sis-
teme olaq yüksək getirir.

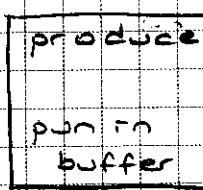
Inter Process Communications

Spooler Directory

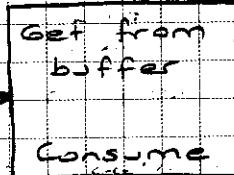
4	a b c	
5	- prog c	out = 4
process A	6 prog n	
process B	7 ****	in = 7
	8	
	9	

Process A fyle yedi. Process A kesiði. Process B yedi.

producer process



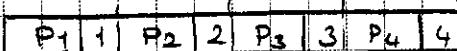
consumer process



Üretici datayı üretip buffer'a gönderiyor. Tüketicisi buffer'dan datayı alıp tüketiyor.

Process in time...

producer



↓

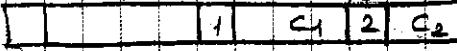
↓

↓

↓

3 instead of 2

Consumer



Başlama süresince gäre rastgele sonular üretiliyor.
Ancak bir sonun üretimi diğerin

P₁ erken çıktı ve 2 yerine 3 geldi.

A Race Conditions

Sürelerin başlama durumlarına ve sürelere göre değişik sonular üretilebilir. Bu nedenle durumu denir.

Critical Section

Critical Section: Bu kısımda process paylaşılmış değişkenleri modifiye eder, Paylaşımlı buffers içindeki process'in erişim hakkı olmaz.

Mutual Exclusion: Bir method sadece bir process'in (satırda) bir anda critical sectionda olmasına garanti ediyor (saglıyor).

Why Process Need to communicate?

Synchronize their executions: Prosesler kendi kasma zamanlarını同步 etmek için döndürler.

Exchange data and information: Prosesler data ve bilgi alışverişi içinde bulunmak için döndürler. Yani if bir işi yapmak gereklidirler,

Rules to Form Critical Section

1. No two processes

Bir anda sadece bir process critical sections'ı girebilir (Mutual exclusion)

2. No assumptions

Herhangi bir fiziksel yapıyı yoktanızıza, CPU'nun hızıyla, proseslerin kasası hızıyla ilgili hiçbir

forsetme yapmamız. Yapılıca processlerin hepsi bir anda critical sectionda olabilir.

3. A process outside:

Critical section'in dışındaki bir process (critical sectionda değil) tamı bittiğinde ve (ikinci) diğer process'ler bloklamayacaktır. Yani olsun critical sectiona girmelerini engellemeyecek.

4. No process should wait forever

Habbi process critical sectiona girmeden önce süren kader bekleyecektir. Yani bir process critical sectiona girmek istiyorsa mutlaka girecektir.

Mutual Exclusion Problem

Starvation

Also known as indefinite postponement.
(Belirsiz)

Definition → Diğer processlerin janına bir process kosması surekli olarak engellenir. Bunun nedeni kots scheduling algoritmasıdır.

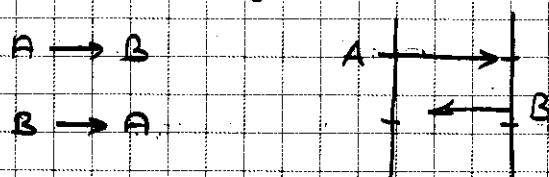
Förum tem bir aging (yastırma) algoritmasını kullanıyor.

Cause

Solution

Another Problem: Deadlock Two (or More)

İki veya daha fazla process olsa gerekli olmamış
çok dayanıklılık. Çünkü olsa olmazsa bir olayın
olmasını bekliyorlar.



A birsey yapıyor ve B'nin okumasını bekliyor. B de
birsey yapıyor ve A'nın okumasını bekliyor. Bu olay
sonusunda kada da devam ediyor.

How to implement mutual exclusion

Three possibilities

Application:

Program, kendisi programın içinde uygulanabilecek
bir metod geliştiriyor (yapıyor).

Hardware:

Birimin içi gereklit olan cmri sağlıyor.

OS:

OS programıyla biri servisler sağlıyor ve program
i, istediği 'mutual exclusion', gerçekleştiriyor.

Aşağıdaki temel olan konularla, işbirliğiyle ok
 critical sections gibi var ve silinen.
 enter-critical-section and
 exit-critical-section

* Bir OS bir adet tarafında yapar.

$$\text{Quantum zaman} = \frac{\text{Antreasyon zamanı}}{\text{Toplam process sayısı}}$$

Application Mutual Exclusion

Programcı tarafından gerçekleştirilen işbirliği doğrular
 etkin bir mutual exclusion gerçekleştirmesi olmalıdır.
 zorluk.

busy wait → Genelde meşgul beklemeye var.

Bir tane lock bayragı var. Bayragın 0 ise herhangi
 bir process critical sectionsına girebilir. Sırasına
 bayragı 1 yapın Diğer processler bayragı baka-
 rak bayragın 1 ise critical sectionsa giremeyecekler,
 0 ise girebilirler. Burada sıretli olarak bayragın
 kontrol ediliyor.

Hardware Mutual Exclusion

Test and Set Instruction:

$x := r$ and $r := 1$

x local değişken r global değişken başlangıç olarak sıfırda setleniyor ($r = 0$).

repeat ($\neg \text{test}(\text{set}(x))$) until $x = 0$

<critical section>

$r := 0$

TSL (Test Set Lock)

enter_region

TSL Register, Lock

CMP Register, #0 → karsileştirmeye yarılır.

JNE enter_region

→ 0 ise regiona girilebilse tekrar başa dön

RETI (return)

leave_region

Movc Lock, #0

RETI (return)

enter_region

<critical_section>

leave_region

lock = 0

lock variables

process A

bs:

if (lock == 0)

{

enter - cs;

lock = 1;

<cs>

}

else

{

goto bs

}

lock = 0;

:

:

↓

Strict Alternation A

while (TRUE)

{

 while (turn != 0); /* wait */

 critical_region();

 turn = 1;

process B

bs:

if (lock == 0)

{

enter - cs;

lock = 1;

<cs>

}

else

{

goto bs

}

lock = 0;

noncritical-region () ;

3

(a)

while (TRUE)

{

 while (turn != i); /* wait */

 critical-region ();

 turn = i;

 noncritical-region ();

}

(b)

critical-region'un dışında
iken değerin
critical-region'a
girmesini engellemeli.

Peterson's Solution

```
#define FALSE 0
```

```
#define TRUE 1
```

```
#define N 2
```

```
int turn;
```

```
int interested[N];
```

```
void enter-region (int process)
```

{

```
    int other;
```

```
    other = 1 - process;
```

```

interested [process] = TRUE;
turn = process;
while (turn == process && interested [other] == TRUE)
{
    void leave-region (int process)
    {
        interested [process] = FALSE;
    }
}

```

Hardware Mutual Exclusion

Exchange Instruction

swap (x, r)

x is a local variable

r is a global variable (initially $r=1$) \rightarrow belongs to $r=1$

$x \neq 0$

repeat exchange (r, x) until $x=1$,

<cs>

exchange (r, x);

in cs $\Rightarrow r \neq 0$ and $x \neq 1$;

$r=1$ ise critcal sections girebiliryormus. Aksi halde
giremiyor mus. repeat until'de sürekli bekliyor mus.

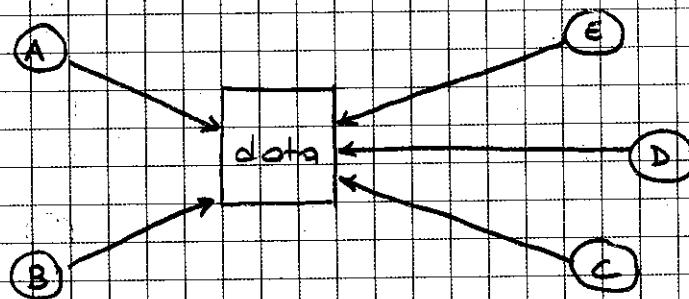
Hardware m. E. Characteristics

Advantages

- Bir veya birden fazla process tarafından kullanılır.
- Basit, doğalıyla verify edilmesi kolay
- Multiple (birden çok) critical section'ı destekler.

Disadvantages

- Busy waiting vs used.
- Starvation (processların çalışmaması, veya sure beklemesi)



A daha önce dataya erişti. Sonra E erişir. E erişip işlemelerini bitirdikten sonra A'dan başka bir processin dataya erişmesi beklenirken A tekrar dataya erişir. Yani bu data erişimleri denleyerek bir yelp yak.

• Deadlock

Another Hardware M.E.

Disabling Interrupts

Interruptların disable edilmesiyle diğer process'ın
istem CPU'yu ele geçirmesi mümkün değil. Böylece
bir process sürekli kozacık. Belki de interruptları
rin enable edilmesiyle kozacık. Tek process
sistemi kilitleyebilm. Bu da sistem performansını
degradebilir.

Mutual Exclusion Through OS

- Sleep / Wake-up
- Semaphores
- Monitors
- Message Passing

Producer Consumer Problem

(Sleep and Wake-up)

Bu yöntemde N varlıklı bufferli üretici tüketici
problemini çözmeye çalışız.



```

#define N 100

int count=0;

void producer (void)
{
    int item;

    while (TRUE)
    {
        item = produce_item ();
        if (count == N) ←
            sleep ();

        insert_item (item);
        count = count + 1;
        if (count == 1)
            wakeup (consumer);
    }
}

void consumer (void);
{
    int item;

    while (TRUE)
    {
        if (count == 0) →
            sleep ();
    }
}

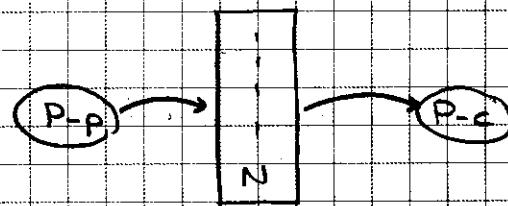
```

Birka tədər iñetidi.
 Producer işletisi yox
 count 1'e eft olunca
 wakes (consumer)
 komutunu göndəryor
 Fakat consumer koldığı
 yerdən devam edəcək.
 İm yoxsa moduna
 (sleep) gəşyər. Dəla-
 yisyla count bir dəfə
 1'e eft olmayaq ve
 təketim hərə olmayaq.

```

item = remove_item();
count = count - 1;
if (count == N-1)
    wakeup (producer);
consume_item(item);
}
}

```



Sleep() surek bloklanmasidir, surek CPU' u almasini isteklimesse. OS surek tarifeliyon Aksar duruma girmesi wakeup ile saglanir. (ready duruma).

Sleep and wakeup' taki problem sinyal sleepten once gonderilmisse kaybetiyor. Sleepten sonra gonderilmesi islemeler normal devam ediyor.

Semaphores

Gonderilen OSler semaphorus, monitors veya message passing'i destekliyor.

Semaphore is a non-negative integer ve two more visible operationsdan olusuyor ve initialize edilmeli.

Semaphore Operations

wait(s)

if $s > 0$ then $s! = s - 1$

else block this process

signal(s)

if there is a blocked process on this semaphore
then wake up

else $s! = s + 1$

Programlarda interruption bozulmaması için, semaphore enable eden, sena enable eden işlem sistemidir. Yani programlar bâlganmcalar.

More on Semaphores

1- Initialize etmek gereklidir.

2- İki tip semaphore var: binary semaphore.

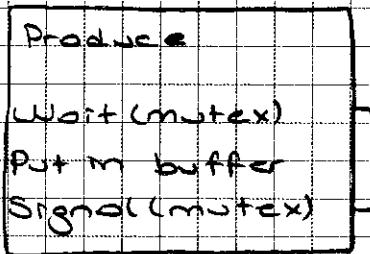
İçinde semaphore degerlerini 1 ya da 0 alabilir,

Counting türü semaphorlerde semaphore degerleri non-negative alabilir,

counting - non negative

Producer - Consumer Problem by Semaphores

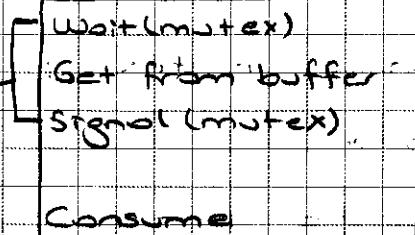
Process (P-p)
producer



mutex = 1

cs

(P-c)



Başlangıçta mutex = 1 olmalı. P-p aktif olsun. P-c'yi.
yazdırın.

DP

P-A

think();
drawA();

P-B

think();
drawB();

P-C

think();
drawC();

Ekranda ABC formek istiyorsunuz.

Burada ifan ekrana söyledim.

P-A

think();
drawA();
signal(b);

P-B

wait(b);
think();
drawB();
signal(c);

P-C

wait(c);
think();
drawC();

Sonucta ekranda ABC yazılacak.

Burada başlangıçta $b=0, c=0$ olsun.

melidir.

Semaphore
değeri sıfır

~~Day~~ N buffer problems

```
#define N 100

typedef int semaphore;
semaphore mutex=1;
semaphore empty=N;
semaphore full=0;

void producer (void)
{
    int item;
    while (TRUE)
    {
        item = produce_item ();
        down (&empty);           // "Buffertabbar var nu? bokar.
        down (&mutex);
        insert_item (item);
        up (&mutex);
        up (&full);
    }
}

procedure consumer (void)
{
    int item;
}
```

```

while (TRUE)
{
    down (&full);
    down (&mutex);
    item = remove_item ();
    up (&mutex);
    up (&empty);
    consume_item (item);
}
}

```

Semaphore, statement seyresinde bir yeterleklik
 Buffer dolussa gonderici process, buffer bossa al-
 ci process blokunur. Bunun dipinda bloklenme sese-
 konusu degildir.

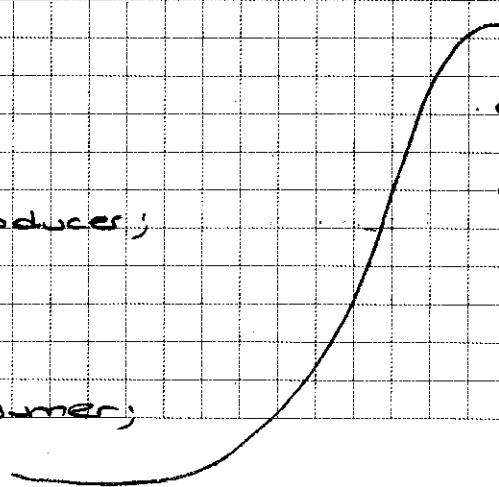
Monitors

monitor example

```

integer i;
condition c;
procedure producer();
|
end;
procedure consumer();
|

```



```

end;
end monitor;

```

82
Bir anda sadece 1 process aktif olabilir. Bir process
kesarken diğer processlerin hangisi monitora gire-
mez.

N bufferli条件制約問題の解法

monitor Producer Consumer

condition full, empty;

integer count;

procedure insert (item; integer);

begin

if count = N then wait (full); // process monitor
dianda bloklenir.
insert_item (item);
count := count + 1;

if count = 1 then signal (empty)

end;

function remove; integer;

begin

if count = 0 then wait (empty);

remove := remove_item;

count := count - 1;

if count = N-1 then signal (full)

end;

```
count := 0;  
end monitor;  
  
procedure producer;  
begin  
  while true do  
    begin  
      item = produce_item;  
      Producer_Consumer.insert(item);  
    end  
  end;  
  
procedure consumer;  
begin  
  while true do  
    begin  
      item = Producer_Consumer.remove;  
      consume_item(item);  
    end  
  end;
```

Message Passing

Mesaj geçme sentrasyonu ve bilgi dağıtmayı sağlıyor.

Message Operations

send (destination, & message) → Hedef banksa bir process

receive (source, & message) → Mesajın nereden geldiğii

Producer Consumer Problem Using Message

```
#define N 100
```

```
producer()
```

```
{
```

```
int item;
```

```
message m;
```

```
while (TRUE)
```

```
{
```

```
produce-item (&item);
```

```
receive (consumer, &m); ← A // Buffer doluydu producer  
burada blokeonur.
```

```
build-message (&m, item);
```

```
send (consumer, &m); → B
```

```
}
```

```
}
```

consumer ()

}

int item;

message m;

(for (i=0; i < N; i++) // Bufferde ne kadar boş
alan varsa o kadar boş
send (producer, &m); // mesaj gönderir.

while (TRUE)

{

③ → receive (producer, &m); // Bufferde mesaj yoksa kon-
mer burada bloklanır.
extract_item (&m, &item); // alınan mesaj silinir.

④ ← send (producer, &m); // mesajı silmesi ile buf-
ferde bir alan boşalır. ve
consume_item (item); bunun sonucunda pro-
ducer bir mesaj gönderir.

}

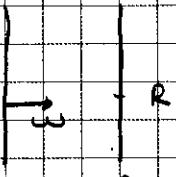
}

Eğer konakları, yeter send ve receive blokları.

ÖDEV: Eger bu buffer almışsaydı, o zaman N bufferli
bir yapı nasıl oluşturabilirdik?

write_ch (P-B, m)

read_ch (P-A, m);

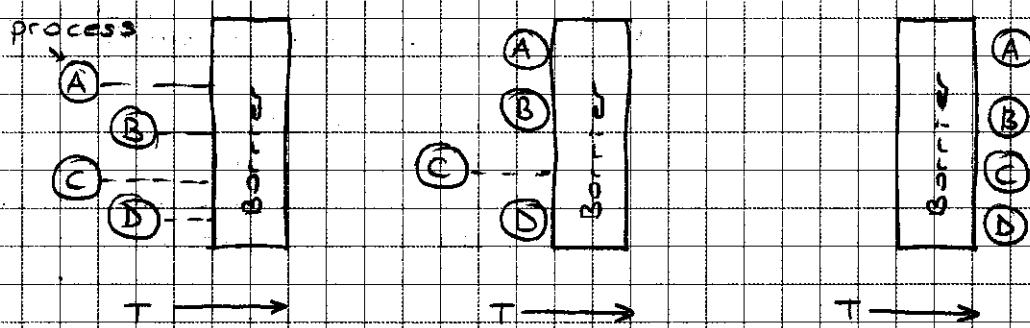


↓ Direct data transmission
buffer kullanılmış.

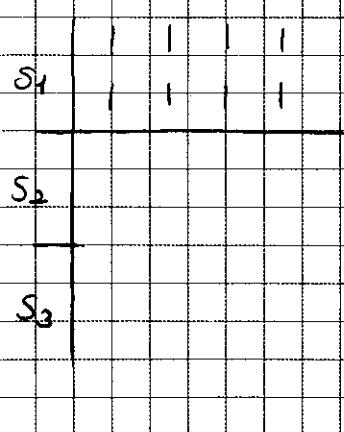
Eğer B okuma yapmasa A wide bloklar, B R'ye geldiginde A wide degilse yani bilgi yapmasa B R'ye bloklar.

* Mesaj germede istediginiz kader send yapinis. Hem okunin hem de yapisinin hali olsugu uretici tuketici.

Barriars

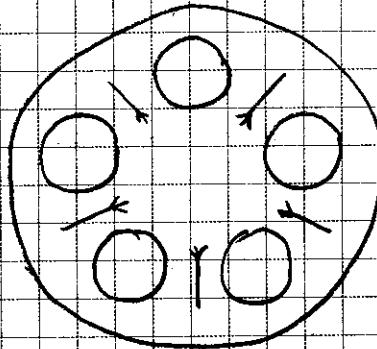


Barrier buton processlerin birlikte takildigi (bloklandig) bir yer. Processlerin hepsi birden barriers gelimeden serbest bırakilmazlar.



Sı丞de kodlar işteken sonusor (datolar) di丞er
streslerde kullanılacaksa, bu adet processlerin hepsi
sınırı sı丞de dolmuştur. Sağlamollsın, bunun
için bu naktaya bir barrier konulur. Bu tür process-
ler kozmaden yani birde datolar olusturulma-
dan di丞er streslerle gecilmesi, sadece eger bir
process sı丞de alka konuyorsa ve diğer bir pro-
cess sı丞de processin ürettiği datayı kullanacak
data üretmediği ram process bloklanacaktır. On-
lemek ram barrier kullanılır.

The Dining Philosophers Problem



```
#define N 5
void philosopher (int i)
{
    while (TRUE)
    {
    }
```

```

think();
take_fork(i);
take_fork((i+1)%N);
eat();
put_fork(i);
put_fork((i+1)%N);
}

}

Semaphore S[N];
void philosopher(int i)
{
    while (TRUE)
    {
        think();
        take_forks(i); // Götelləri elə gəsin
        eat();
        put_forks(i); // Götelləri bırak
    }
}
void take_forks(int i)
{
}

```

```

down(& mutex);
state[i] = THINKING;
test(LEFT); // Sağndaki ve solundaki processlere
            // birip durumlarını kontrol
test(RIGHT); // ediyor
up(& mutex),
}
void test(int i)
{
    if (state[i] == HUNGRY && state[LEFT] == EATING
        && state[RIGHT] == EATING)
    {
        state[i] = EATING;
        up(& s[i]);
    }
}

```

Siyamkuyrukların processleri bloklanıyor. Bloklanan processleri yiyen processler uyarıyor.

The Readers and Writers Problem

```

typedef int Semaphore; // Bir database birden fazla
                      // kisi tarafından okunabilmeli
semaphore mutex=1;   // ama tek kisi tarafından
semaphore db=1;

```

int rc=0; // Okuyucu sayisi.

void reader (void)

{

while (TRUE)

{

down (& mutex);

rc=rc+1;

if (rc==1) down (& db);

up (& mutex);

rc=rc-data.base();

down (& mutex);

rc=rc-1;

if (rc==0) up (& db);

up (& mutex);

use_data.read();

}

}

void writer (void)

{

while (TRUE)

{

```

    think-up-data();
    down(&db);
    write-database();
    up(&db);
}
}

```

The Sleeping Barber Problem

```

#define CHAIR 5

typedef mt semaphore;
semaphore customers = 0,
semaphore barber = 0,
semaphore mutex = 1,
mt waiting = 0;

void Barber (void)
{
    while (TRUE)
    {
        down(&customers);
        down(&mutex);
        waiting = waiting - 1;
        up(&barber);
        up(&mutex);
    }
}

```

```

    cut_hair();
}

{

void customer (word)
{
    down (& mutex); //critical section gr.
    if (waiting < CHAIRS)
    {
        waiting = waiting + 1;
        up (& customers); //misteri sayangin oritur,
        up (& mutex); //critical section don sik.
        down (& barbers); //Barberbar yakes bekle,
        get_haircut();
    }
}

```

else

```

{
    up (& mutex);
}

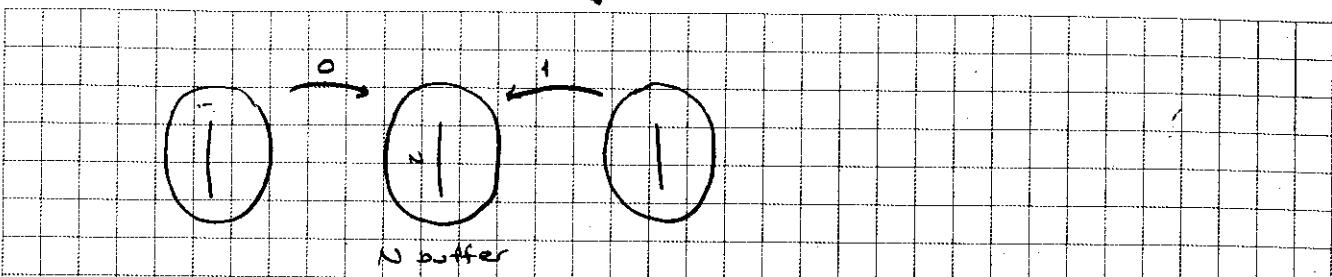
```

DESAIN CERABBI:

```

char read (0)();
char read (1)();

```



Kanal sayisi kadar guard var.

guards[0] = {0,1}

guards[1] = {0,1}

i = ALT (char-read, guards ...) // Bir index değerini
geri döndürür ve
indexe göre ilgili
kodla bükür ve
sunulan işlem yapılır.

i = 0; ise 0'inci kanal

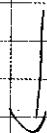
i = 1; ise 1'inci kanal

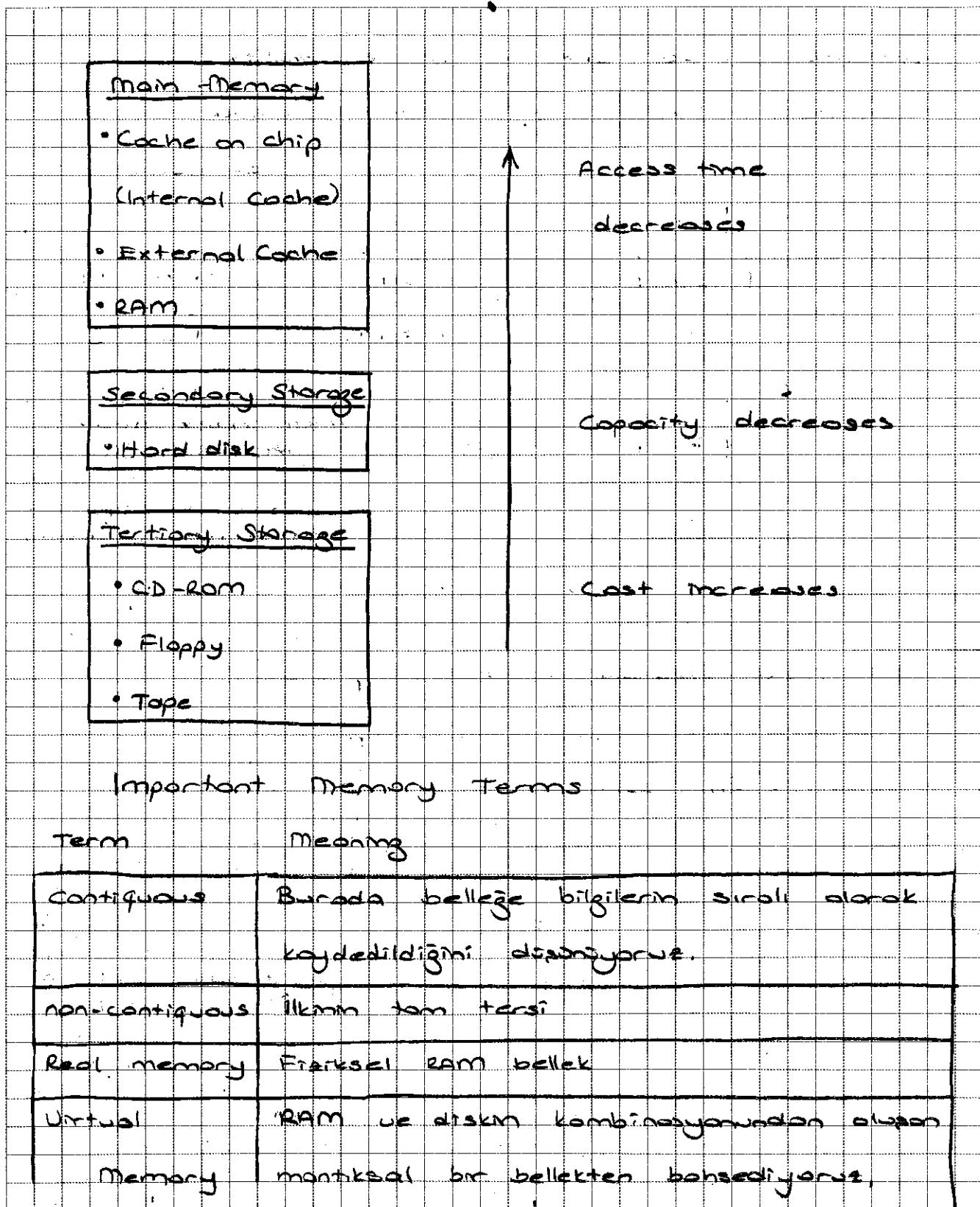
i = ALT[i] (char-read, guards ...)

Memory Management

- Storage Hierarchy
- Important Memory Terms
- Early memory management schemes
- Free & Allocated bellek alanlarının yönetimi

Storage Hierarchy



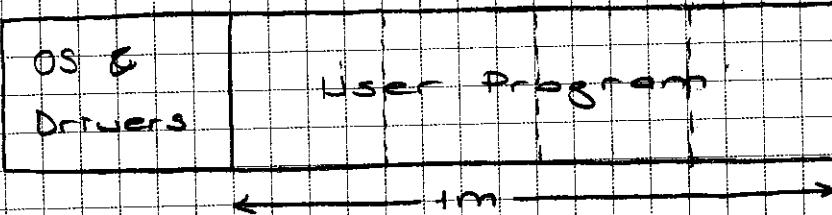


Important Memory Terms

Term	meaning
contiguous	Burada belleğe bilgilerin sıralı olarak kaydedildiğini düşünüyorsunuz.
non-contiguous	ilkmin tam tersi
Real memory	Fiziksel ram bellek
Virtual	RAM ve diskin kombinasyonundan oluşan
Memory	matematiksel bir bellekten bahsediyorsunuz,

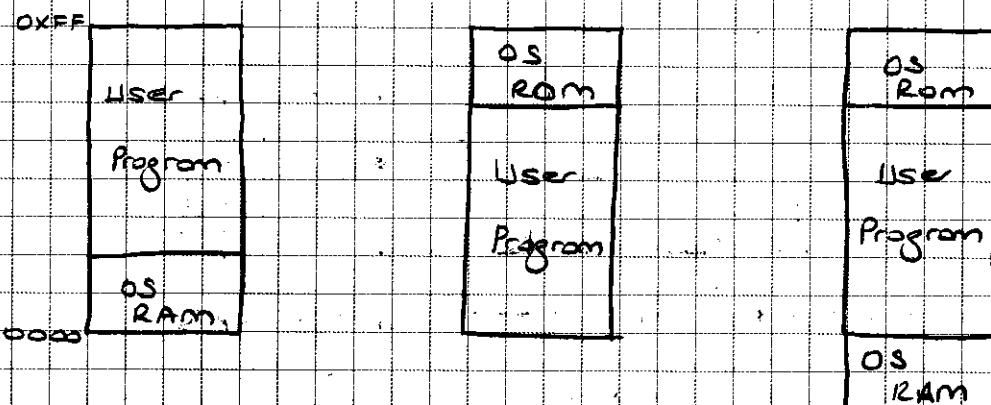
Logical	Software ile görülen bellek
Physical	Hardware ile görülen bellek
Block	Disk tarafından transfer edilebilen en küçük bilgi
Principles of Bellet locality	Bellek alanında her programlar belli lokal alanlara erişme eğilimlidir. Yani belli alanlara çok fazla erişir.
Fragmentation	Belleğin küçük parçalara bölünmesi

Contiguous Memory Allocation (Sıralı)

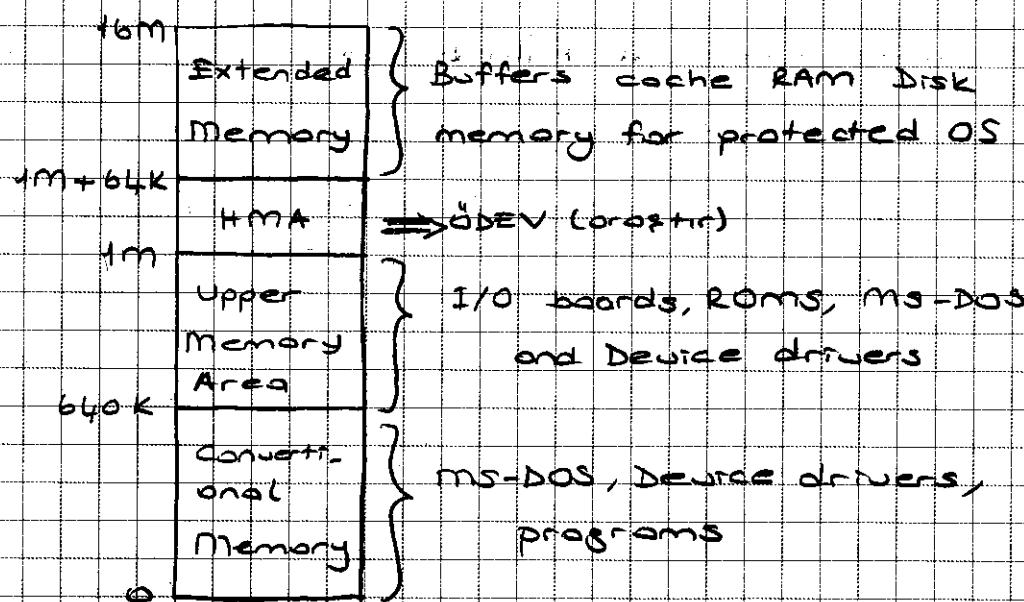


En büyük programın size, kalan bellek size sine göre sınırlanırılmıştır.
Yükleyeceğiniz program kalan bellekten fazla ise program, parça parça belleğe yüklenir. Bir personin kafası tamamlanmaya yoksas karan kisimlar bellekten alınıp diğer personler yüklenir. Bu tarz çalışma yöntemine swapping denir.

Geminde kullanılan sistemde bellek yönetimi
ş. şekildaydı.



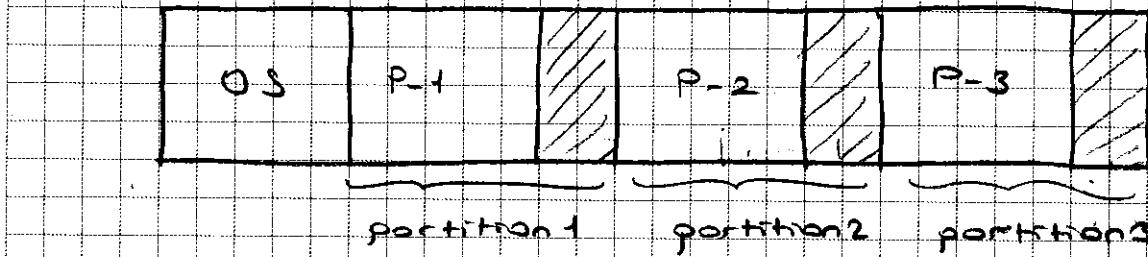
DOS Memory Layout



Eğer multi user sistem oluşturuyorsak, kırılganın
birbirine bit de olası. ögrenmemeleri için RAM por
solara sıyrılır ve bu plakalar protected yapılır.

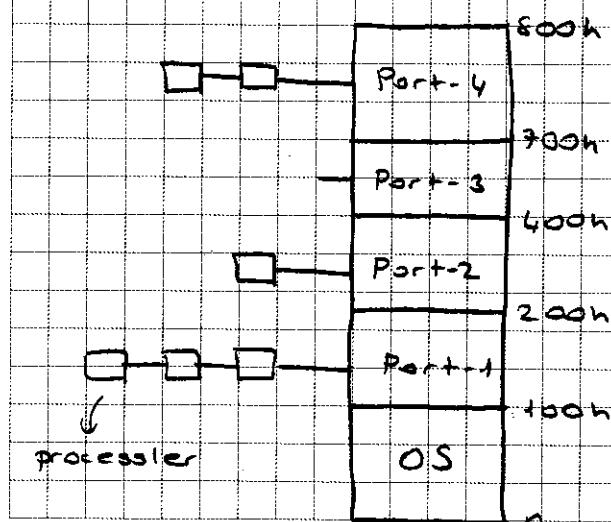
Multi Programming with Fixed Partition :-

MFT (multi programming with a fixed number of task)



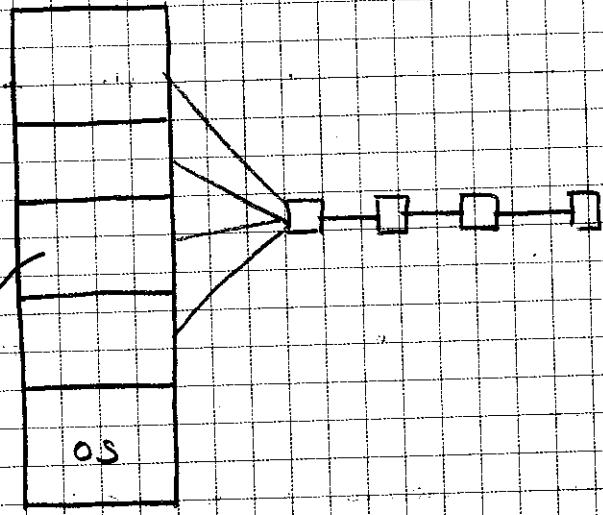
IBM 360 tarafından kullanılmış.

Partitionların sayısı ve size operatör tarafından deha adakları istatistiklerle belli olur. Belirtilen program 1 (Port 1'de) kendi içinde ayrı ayrı olandan deha özellikleri kendi komutosu, Belirtilen (diğer partitionlarda) yer almaması rağmen komutosu.



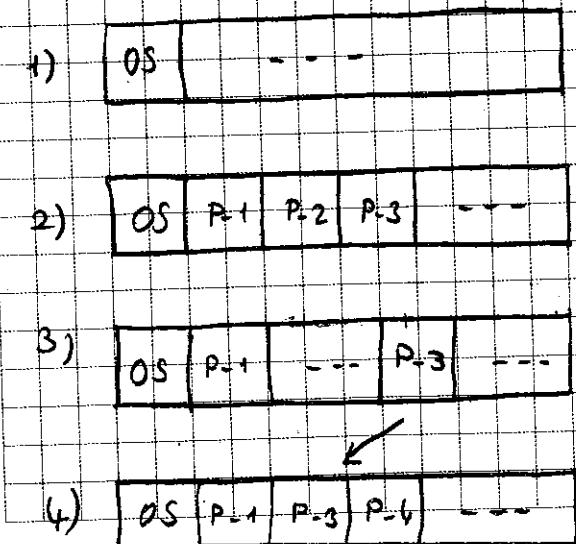
Prosesler, kuryuklarda tutulur.

Buonda da her bir bellek planları, kullanimıza oldu.



Bu alanda her bir process bellek planının tomanını
kullanimmasında ragmen bellek planı daşı gizlisiyse
ve her planlarda kuryuktaki prosesler konusuya.

Multi Programming with Dynamic Partitions



3'ten 4'e geciste process-3 yerinden yerlestirilmesi.

Eger bu yapilmassaydi, orada bosluklar kalanacakti.
(fragmentasyon)

Fakat relocation'in de maliyeti çok ve zorlu
oluyor.

Problems with Dynamic Partition

1. Proseslerin yüklenmesi ve silinmesi zamanında
fragmentasyon oluyor.

2. Bu problem yerinden yerlestirme (relocation)
ile gideriliyor.

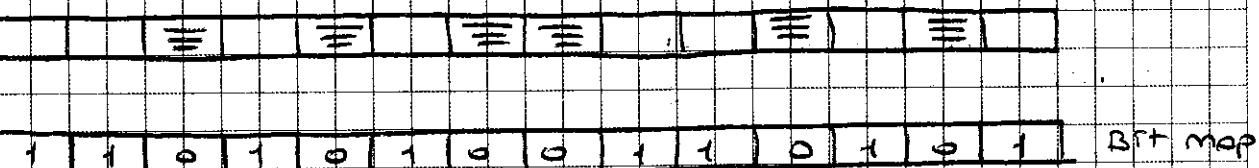
3. Relocation hizi oldukça zorundadır. Bu seyde relo-
cation'un sisteme getirdiği yük artmaktadır.

4. Free bellek alanlarının yönetilmesi güçtür.

Bu bellek alanlarının tanisi yerlestirme algoritması
ile belirlenecektir.

How to Keep Track of Unused Memory

- Bit maps

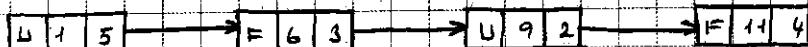


0 → kullanılmamış

1 → kullanılmış

Yani bir bellek genişliği olduğunda zaman kullanılmış
olarak sayılır.

• linked lists



6'dan başlıyor, 9'dan başlıyor

3 tane devam ediyor 2 tane alan da

U → kullanılmış (use) F → boş olmalar (fail)

Bu nedenle çok fazla kullanılmış. Beltek kapasitesi ne
kadar büyükse o kadar fazla boş kapasitesi olur.
Yani beltekte daha fazla yer tutar.

Placement Algorithm

First fit: Bir önceki process bellegin boşunden itibaren
ilk uygun sağladığını alır ve yerleştirir.

Next fit: Process son kalınan noktadan boşluğucak

first fit algoritması, uygunlarını yerleştirir.

Best fit: process en uygun boşluğu yerleştirir. Bu
yöntemde fragmentasyon minimuma indirmek hedefdir.

Performans Issues

Best fit: genelikle en os basitigi birakmasina nüfusen performans alerek en katladidir.
First fit basittir. Fakat hizlidir. Performans decision dan en iyisidir.

Next fit, first fitten borsa kotsu.

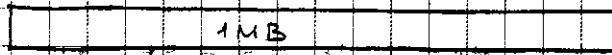
~~1.VİZE
SONU~~

Another Allocation Schemes

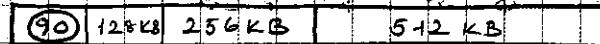
Buddy System

Burada bellek 2'nni üstleri seklinde tahsis edilir.

Mitally



90KB



90KB \rightarrow 128 - 90KB borsa gider

300KB



90KB sunlandiginda ilk yordakki 512KB lik

alan borsa gider.

512KB

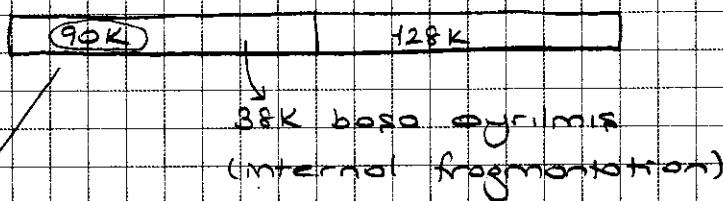
(300KB)

Buddy, mimkin olan en kisa komanda en uygun bellek tahsisini yapar. Kullanan bellek miktarı best ter katsalar.

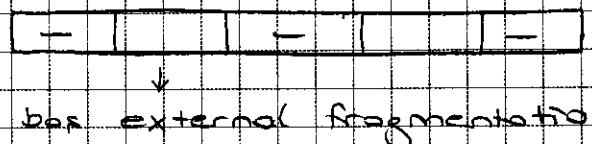
Bu metod deallocation'ı hızla yapar. Manager 2^k lik alan serbest bırakırsa manager 2^k ye bağlı boşlukları kontrol edip o alanları boş bırakır.

* Buradaki fragmentation processin içinde sadece internal fragmentation denir.

Eğer fragmentation processin dışında ise bu external fragmentationdır. Mesela $128K'$ yi $90K$ 'ya ayıran -şek 38K internal fragmentationdir.



Bu alan bir processe bitmeye kadar başka tahsis edilemez.



Bu alan yeni processlere tahsis edilebilir.

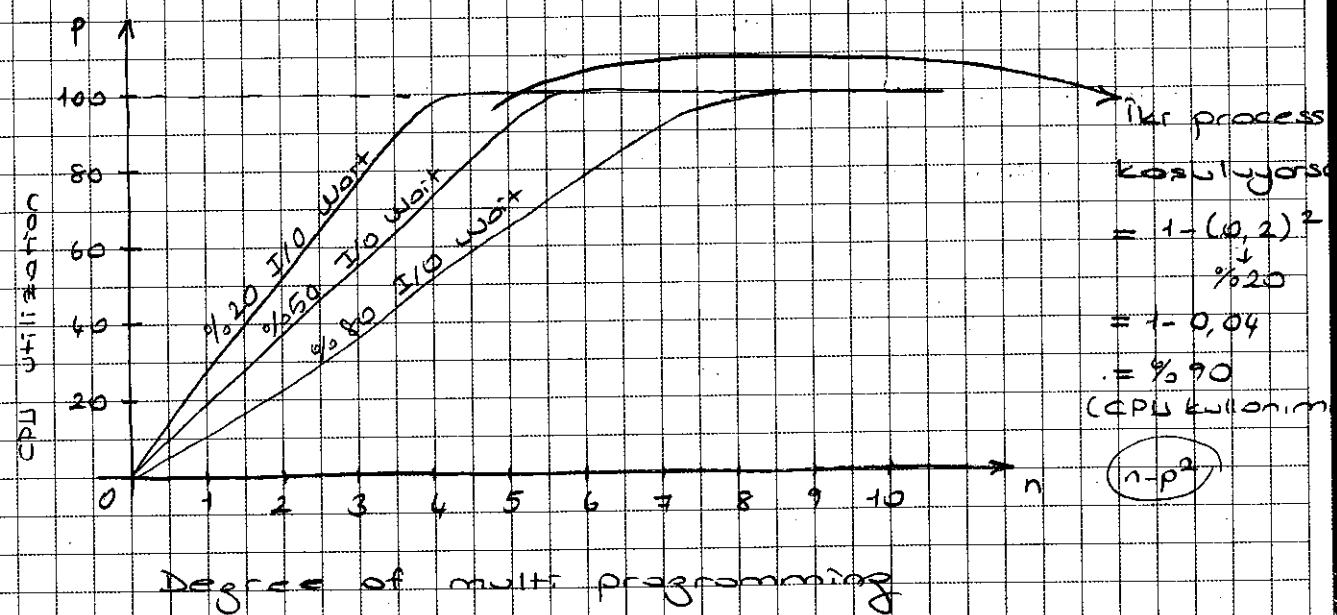
Problems with Memory Management

Techniques used for

- Unused memory (fragmentation'dan dolayı, kullanılmayan bellekler = internal veya external olarak)

- Bir program parçaları his kullanılmaz. Belirgin ölçümlü bir kısmına program parçaları yüklenmesine rağmen bu parçalar his kullanılmayabilir. Simdiye kadar gösterilen yöntemler bu soruyu çözmemiştir.
- Process size'ı fiziksel bellek size'ı ile sınırlı mı olacak? Sınırlı olmamalı.

~~BU~~ problemlerin tek çözümü virtual memory dir. Bir den fazla process'in sistemde konusmasının ne yararı olabilir?



n = multi programming derecesi (process sayısı)
 p = input output bekleye yüzdesi

q4
 CPU - utilization = $t - p$ (formüldeki hanesi olur.)

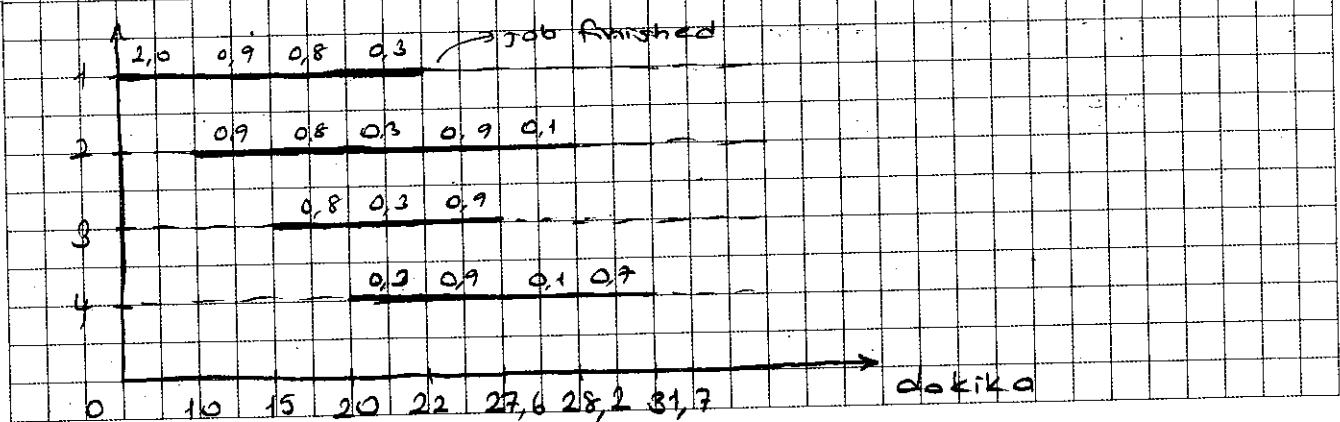
proses sayısını artırdığınca, multi programming derecesini artırıldığında CPU utilization optimizabledir. Ama hiçbir zaman %100 olmaz.

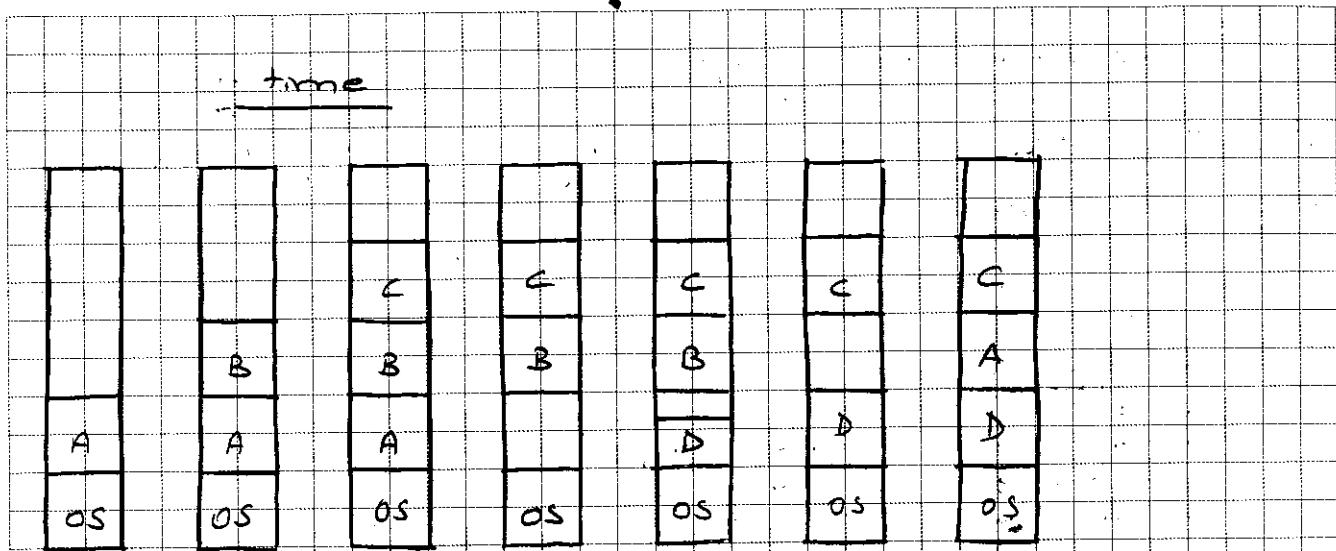
Job	Arrival time	CPU minutes needed
1	10:00	4
2	10:10	3
3	10:15	2
4	10:20	2

skiller:

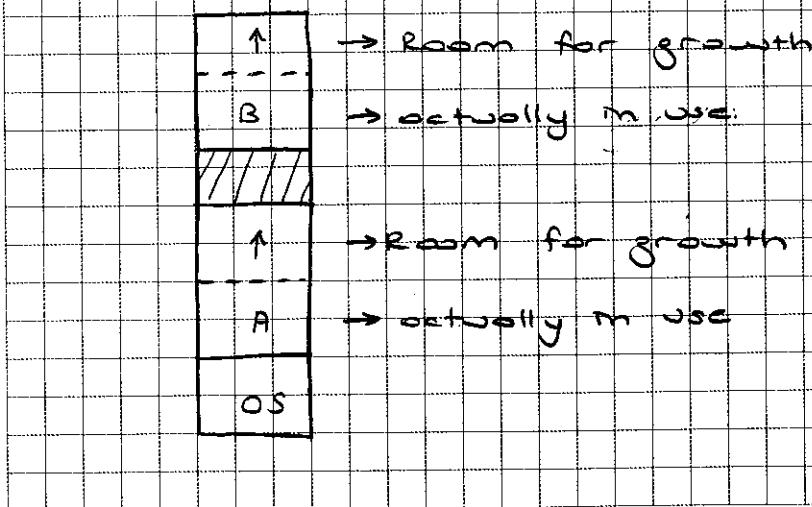
CPU kullanımının
ile ilişili tablo

#number of processes	1	2	3	4
CPU idle (bos)	0,80	0,64	0,51	0,41
CPU busy	0,20	0,36	0,49	0,59
CPU/process	0,20	0,18	0,16	0,15

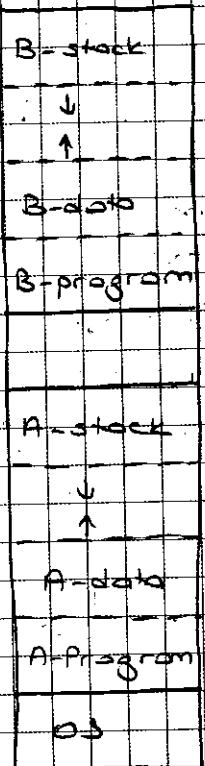




D'ye dijende A swap eddimiz, D kontrularmaz. Sonra B swap eddimiz ve A tekrar ysklememiz.
A'nın swap array'sa otimus, tek puanlı bir islem.
A çok büyük bir program (1MB vs) Dolayısıyla A'nın sık sık swap array'sa otimus. ve sonra tekrar eğrilişsi, sistemi yavaşlatır. Bu problemi en iyi virtual memory çözür.



Prosesör konstrukksı genislemeye alımlarını da kullanır. Genislemeye alımları da tamamen kullanıldı. Bu anda aşağıdaki gibi bir durum ortaya çıktı.



Eğer data boyayırsa stack boyası boyası, basılığın tamamı data kullanısın derse. Eğer ikisi de boyayırsa program sırada caldııp sonra tekrar program ikinci belirli alanları stoptır.

Virtual Memory

Problems with Memory Management Techniques so far

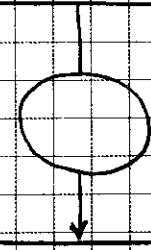
- Fragmentation'dan dolayı kullanılmayan bellek miktarı.
- Programlar büyük parçalarının yerine ki bu parçalar run time sırasında hiç kullanılmıyor

- Program size: Kortendeur program bellek siteini kapilit.

Bu problemlere assembler onyozogiz.

Virtual Memory

Virtual memory of process on disk



Map (translate) virtual adres to real
Virtual adreslerin gerçek adresle döner formde istenilen yapan.

- Virtual (kullanımsaldır)
- Virtual memory disk üzerinde gerçek oturum oluşturular
- Virtual memory size: fiziksel memory (Ram) sizeinden büyük
- Birçok processler VM'lerin bir şekilde adresler
- Bir process kendiye sahip bütünyle bir VM'yi adres olurken ve bu adresler fiziksel adres dönerlerdir.

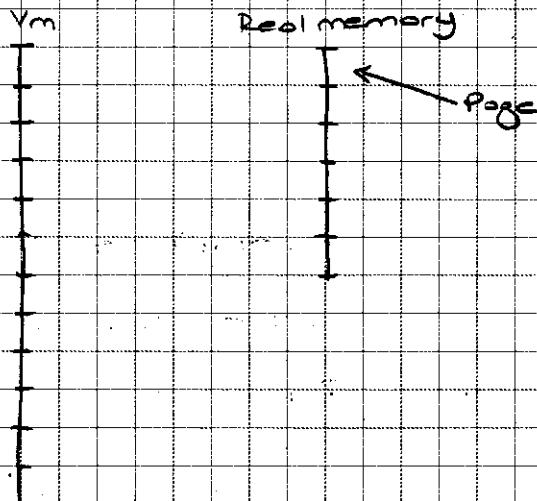
Did we solve the problems?

1. Kullanılmayan bellek alanı yok.
2. Bir program kullanılmayan parçaları içerebilir. Bu problemi çözduk. Bir programın parçaları istek üzerine belleğe aktarıldığı için sorun ortadan kalktı.



3. Proses'in size'ın fiziksel bellek size ile sınırlı değil.

(Purc) Paging



- 1) Vm ve real memory parçalarına lessit uzanır.

İşbu) boşluklar.

- 2) programlar pagellere bölünür. Dolayısıyla bir processi

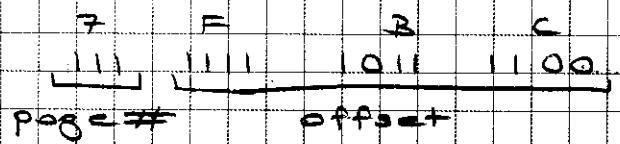
ekle oldigmizda hem VM uc hem de real memory iki bilisim sifir oluyor.

Page #	offset within page
--------	--------------------

Virtual memoryden gerekli adres kullaniyor (elde ediliyor).

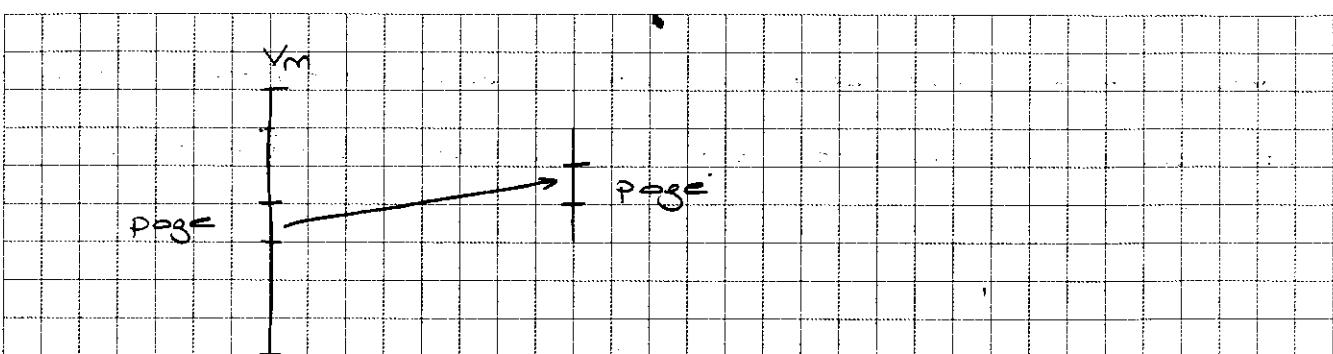
Interpretation of an address

- 16 bit adreslemem onlam.
- 64 bytes beltek adresimiz var. (0000-FFFF)
- Bir sayfa 4K icse (12 bits)
- The memory has 16 pages (4 bits)

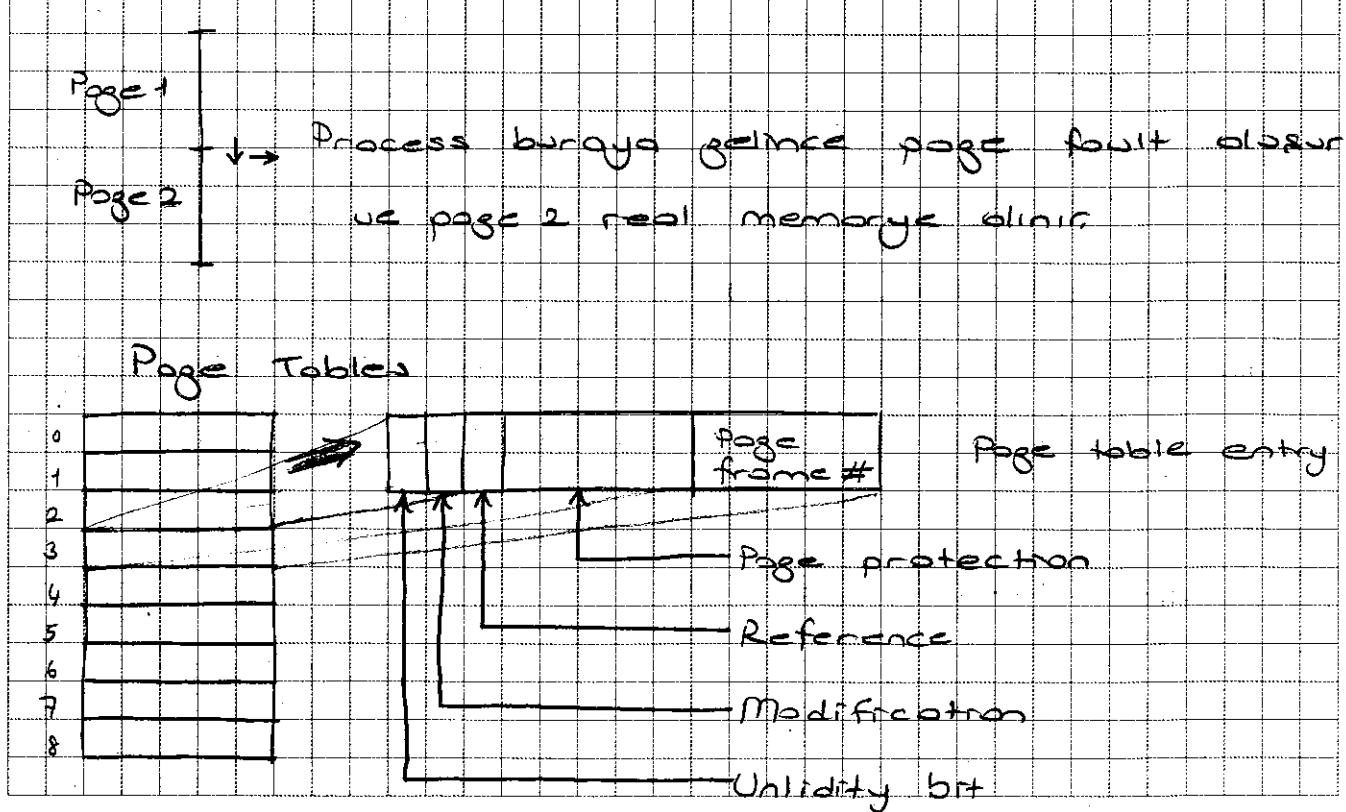


Paging (continuous)

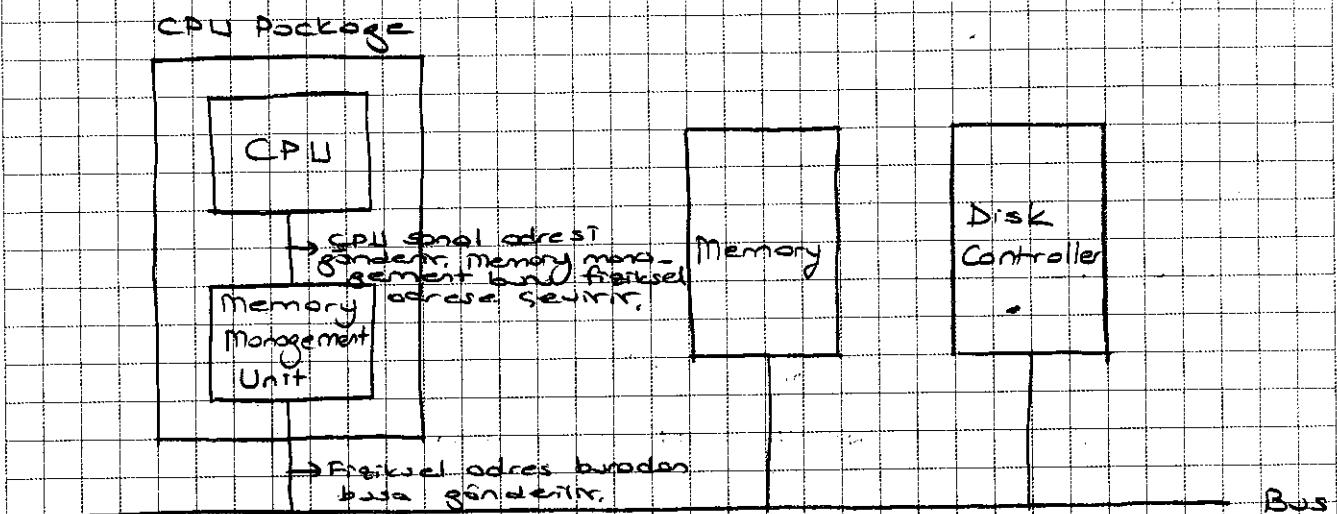
- Bir process sayfasi virtual memory'den Real memory'e transfer edildiginde page numbersi, urhaldan reala donusturulmeli. Program kosarken bu donusur yapilmali.



- Bu döndürmeli yazılım software ve hardware tarafından yapılmır.
 - Bir processin kafasına basıldığı zamanın sadece istenilen parçalar istek üzerine real memorye yüklenir.
 - Sayfalar sadece istek üzerine belleğe yüklenir.



Page frame numbersı birebir programının fiziksel bellekte nereye yerleştirildiğini öğreniyoruz.



Uzadığımız programlarda sənəd adresləri vardır.

Page Table Entry Fields

Validity

Eğer bu bit sıfırmissa sayfa bellekte demektir.

Reference

Eğer bu bit sıfırmissa bu sayfaya erişilməz demektir. (Okuma ya da yazma olabilir)

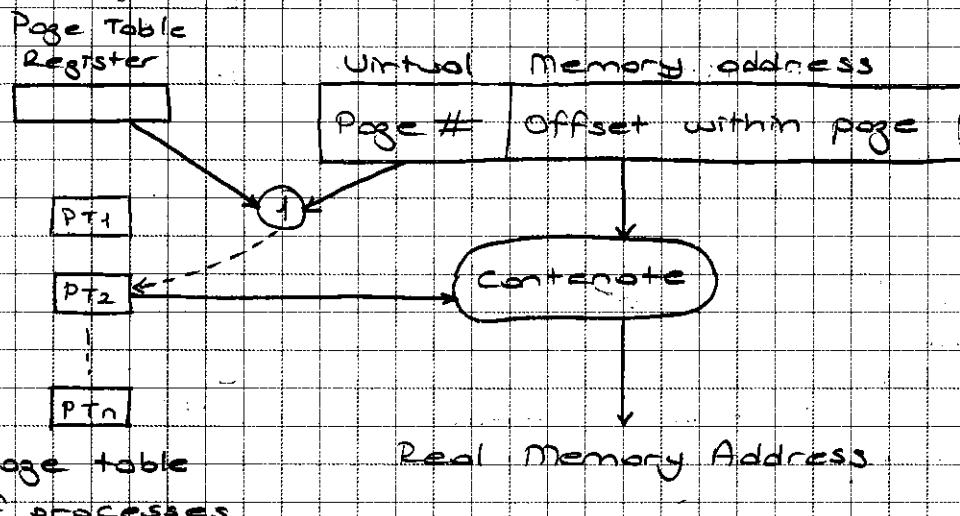
Modified

Sayfanın modified edildiği (değiştirildiği yani üzemde istenməz yarıldığı) belirtir. Eger bellek doluyaşa ve bellekten bir sayfanın atılması isteniyorsa oturacak

sayfa modified edilmesi sayfaların seçilir. Çünkü modify edilmiş sayfalar otomatik olarak harddisk ye-
zilmek zorundadır. Bu sebeple olsın
Protection-bit

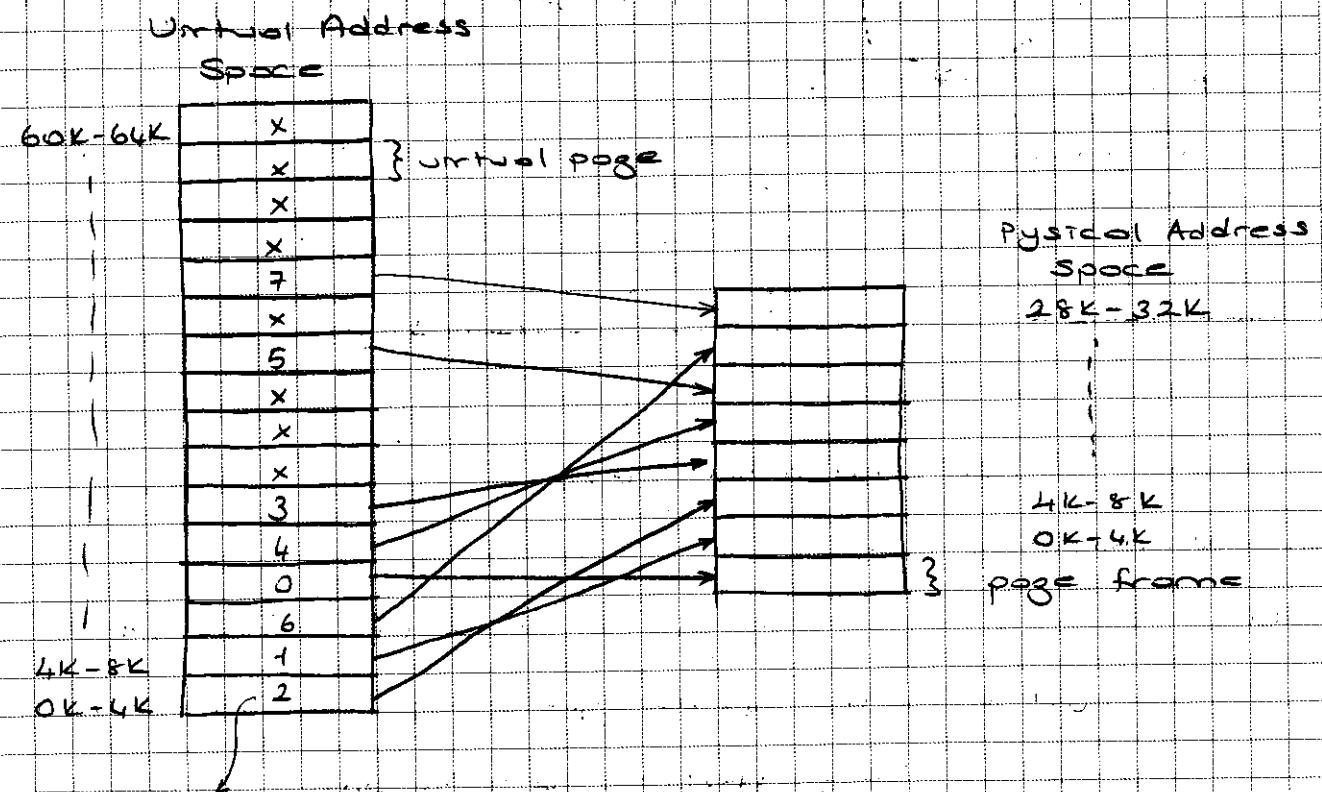
Erisim hakları belirlenir yazma ya da okuma rıtm.

Address Mapping in Paging

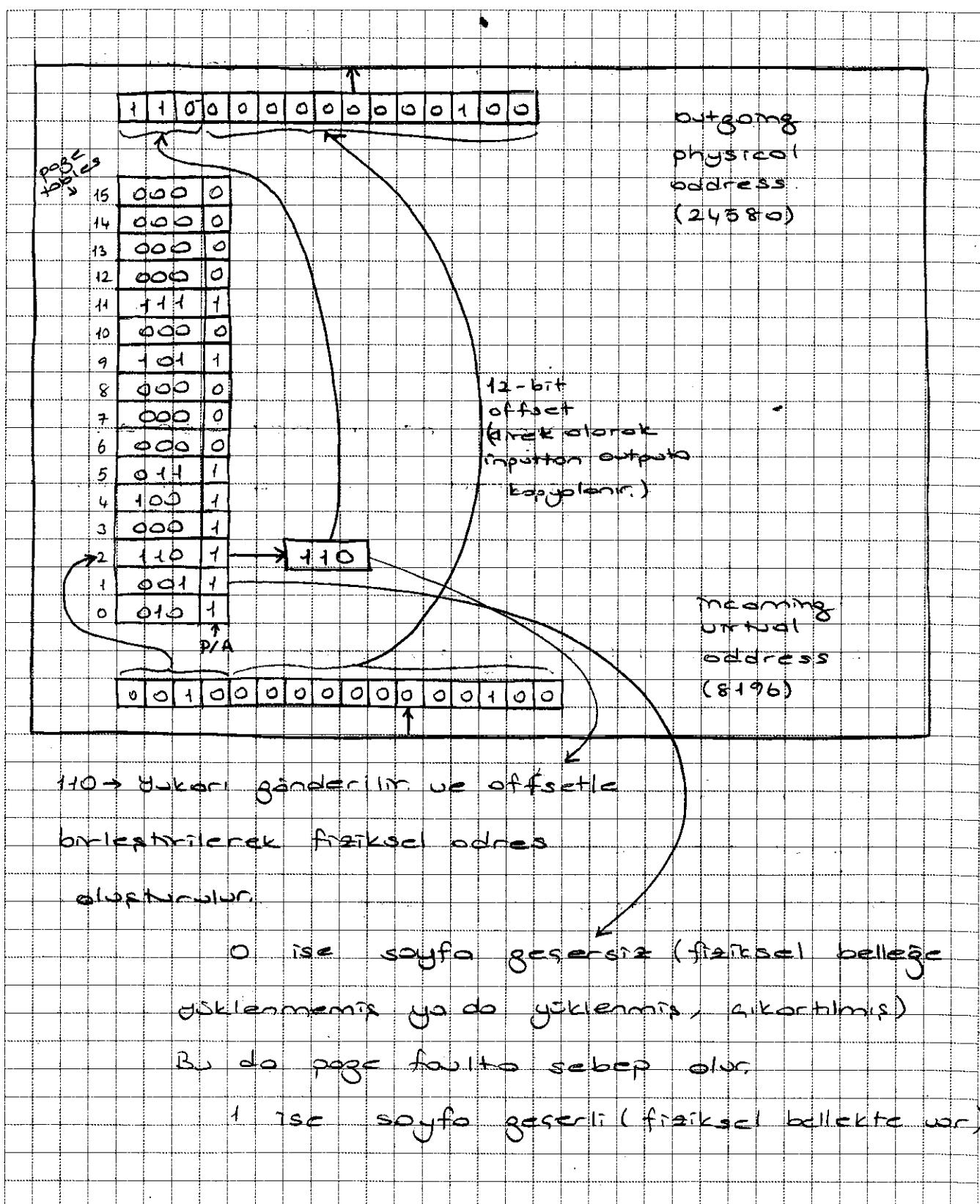


Programın icrası sırasında her bir sayfanın referanslanması yapıldığında page table entry'sinde kontrol edilir. Sayfa geçerli mi, değil mi diye?
Eğer geçerlilik biti setlenmişse (PT_2) → ile offset numarası birleştirilir ve gerçekte bellek adresi oluşturular.

Eğer sayfa belirkti degilse page fault olur. İşletim sistemi devreye girer ve sayfanı alıp belirte getirir. Eğer belirtik durumda bu durumda belli sayfaların diske geri yazılması geçer. Bu sayfanın algoritma page replacement algorithmasıdır. Bu algoritma RAM'daki kullanılmayan ya da en az kullanılan sayfaları diske geri gönderir ve yeni sayfanı RAM'a yükler.



Sonuç belirteks bu soygunun fiziksel
belirteker nerede olduğunu belirtir



110 → Yukarı gönderilir ve offsetle birleştirilerek fiziksel adres elde edilir.

0 ise soyfa geçerli (fiziksel belleğe yüklenmemiştir ya da yüklenmiş,nikortılmış)

1 ise soyfa geçerli (fiziksel bellekte var)

Bu da page fault sebebi olur.

1 ise soyfa geçerli (fiziksel bellekte var)

- Birçok processler virtual memoryde yaradı konuları. Bu nedenle page fault olur.
- Segmentasyon ve pagging bir arada kullanılır.

Segmentasyon

- Page size 1024 bit.
- Program mantıksal olarak alt parçalarla bâlweis halinde. Bu nedenle subprogramlar olmak üzere her birisine ayrı bir segment numarası verilebilir.
- Adres uzayı:

segment #	offset with segment
-----------	---------------------

- Segment map table

Nelerden oluşur:

→ Segment number

→ Fiziksel boyutlu segment adresinden kullanıcıya

üzerinden

→ Segment genişliği

Problems with segmentation

- Dynamic partition ile ilgili tüm problemler gider.
lar.
- Fragmentation
- Relocation (Compaction ve fragmentation'ı aradan kaldırmak için gereklid.)

Segmentation with Paging

- Segmentation sona bellekte yeri geçer bellekte ise paging kullanılır.
- Segmentlerin her birisi pagelerden oluşur.
Buna göre adres uygulamada 3 bileşenden oluşur.

Segment #	Page #	Offset within page
-----------	--------	--------------------

Bu durumda fiziksel bellekte o segmentin sadece ihtiyac duyulan programları (pageleri) tutulur.
Boylece böylelikle fragmentation enlenir. Fakat internal fragmentation olabilir.