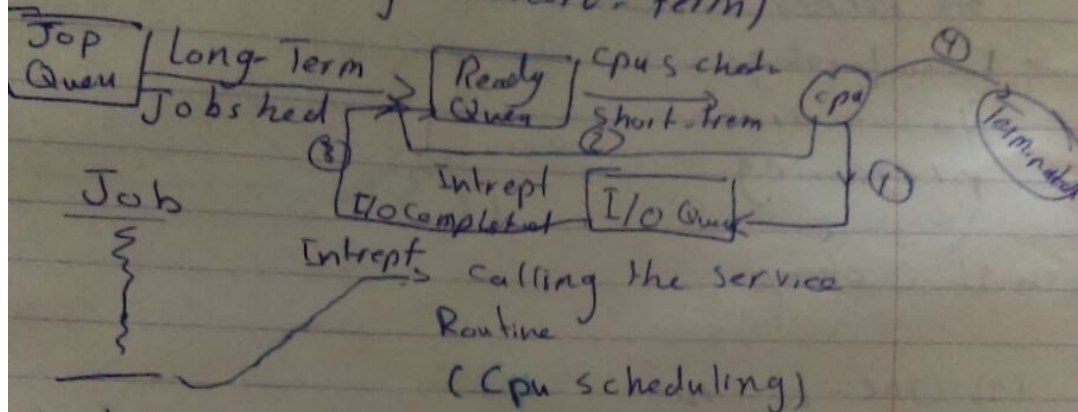


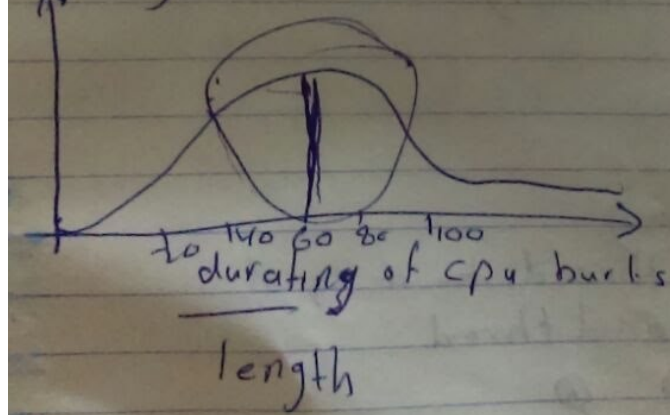
- Concurrency allowed
- Cost Reasonable
- Reliability, Ok

Chapter 5 29, 30, 31

Cpu Scheduling (Short-term)



- Our objective is to maximize the cpu utilization frequency



* What are the cases the CPU scheduling will be invoked?

Cases (1) and (4) are called non-preemptive (مستقر)

Case (2) and (3) are called Preemptive (متحرك)

* CPU Scheduling Criteria :-

(1) CPU utilization

(2) Throughput

max

(3) Turnaround time

min

(4) Waiting time

* Throughput: # of jobs that finished execution per unit of time (averaged)
Turnaround time: - it the time from submitting the job till it finished execution

② it the time wait the process spend in the ready queue

* Cases (1) & (4) are called
non-preemptive (non-preemptive)

* Cases (2) & (3) are called
preemptive (preemptive)

* CPU scheduling criteria:

(1) CPU utilization max

(2) Throughput: # of jobs that (max)
finish execution per unit of time (average).

(3) Turn around time: it's the time from
submitting job till its finish execution.
(min)

(4) Waiting time: it's the time process
spends in the ready queue (min)

(5) Response time, the time from submitting
the job until the first response come
from the computer (psychology) (minimum).

* New Lecture:
CPU Scheduling:

1) FCFS: First come First serve.

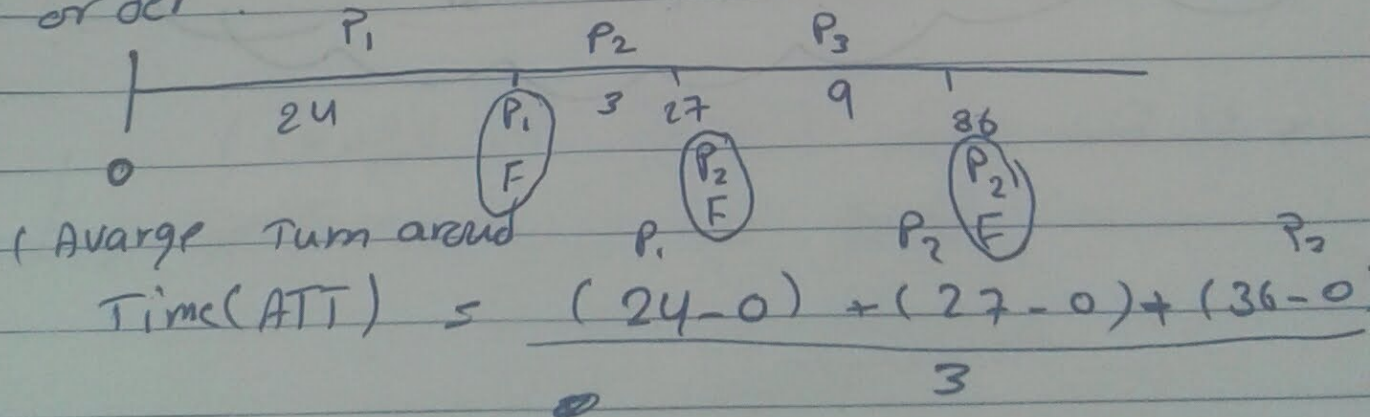
* the CPU is give to the process which arrived first

Ex: Assume the ready queue look like:

<u>Process</u>	<u>Burst time</u>
P ₁	24
P ₂	3
P ₃	3

Compute the average turn around time and average waiting time.

* The process arrived in the same order



$$= \frac{87}{3} = 29 \text{ unit}$$

* Average waiting time (AWT)

$$= \frac{(24-0-24) + (27-0-3) + (36-0-9)}{3}$$
$$= \frac{51}{3} = 17 \text{ unit}$$

Note: Waiting time = Turnaround time - CPU burst

Problem in FCFs :

is called ^{مكب}convoy effect:

↑ كيفة ترتيب العمل

إذا اختلف ترتيب العمل كما هو في الجدول أدناه

	$\bar{J}_1$	$\bar{J}_2$	$\bar{J}_3$
	3	9	24
0			

$$ATT = \frac{3+9+36}{3} = \frac{48}{3} = 16 \text{ unit}$$

$$AWT = \frac{0+3+12}{3} = \frac{15}{3} = 5 \text{ units}$$

* Performance depends on arrival time of process (convoy effect)

* لا يجيد النظام الداخلي في الأداء
→ Performance is 'not good'.

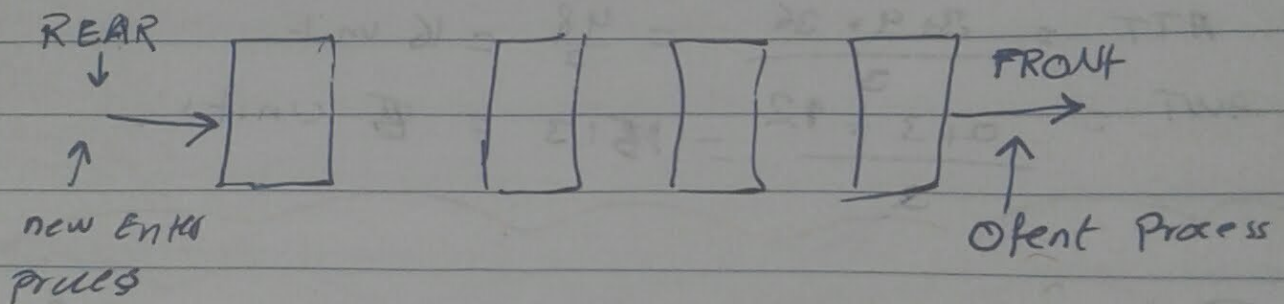
FCFS:

Implementation:

1) Keep a counter of the time ~~the~~ when process enters memory.

Counter is generally located in process controller block (PCB).

(2) Queue of jobs in the Ready Queue



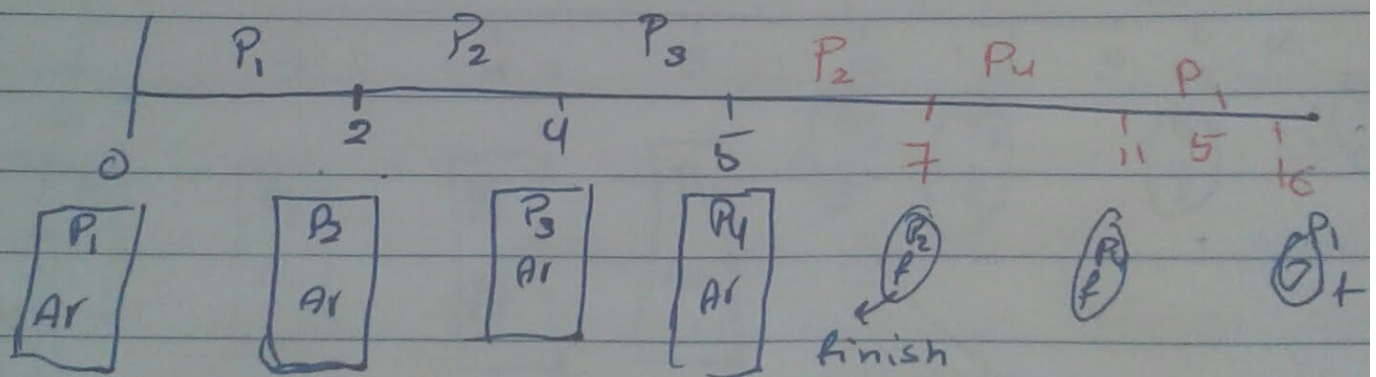
[8] Short Job first :

the CPU is given to the Process with smallest CPU burst time.

Ex: Assume the ready queue:

Process	arrival time	cpu - burst
P_1	0	5
P_2	2	2
P_3	4	0
P_4	5	4

Note: minimum burst in the Queue.



* there are two versions of the algorithm:
(a) Preemptive;
if new process arrives with CPU burst

Short Job first (SJF)

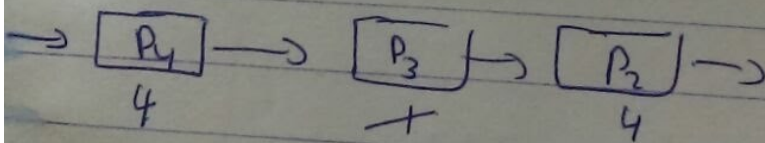
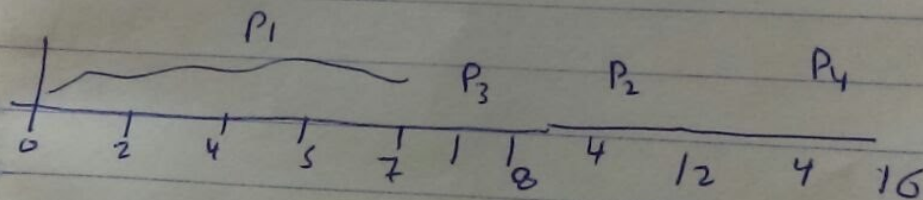
Two versions

(1) preemotive

(2) Non -

if new process arrives with cpu burst less than the remaining time of the running cpu pairs, the cpu continues executing the running cpu pairst and then switching to another cpu burst

Proces	Arrival Time	Cpu Time
P ₁	0	7
P ₂	2	4
P ₃	4	1
P ₄	5	4



SJF gives the optimum solution, that is gives the minimum waiting time

Problems :- Starvation

solution : A going : As time progresses
give the process which was waiting for
a long time some priority

Major Problem :- how the OS
decides or knows in advance the
length of the next CPU burst time

answer : there's no way to decide
exactly the length of the next
CPU burst

Solution : Estimation

α : Assume

T_n = actual length of n^{th} CPU burst

y_n = Estimated length of n^{th} burst

$$0 \leq w \leq 1$$

Given the formula :

$$y_{n+1} = w * T_n + (1-w) * y_n$$

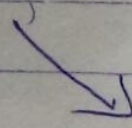
If $w = 0$ - $y_{n+1} = y_n$ not practical

if $w = 1$

$$y_{n+1} = T_n \text{ a little}$$

bit better, but still no practical

$$w = \frac{1}{2}$$

$$y_{n+1} = w * T_n + (1-w) * y_n$$


$$x_{n+1} = w * T_n + (1-w) * [w * T_{n-1} + (1-w) * [w * T_{n-2} + (1-w) * [w * T_{n-3} + (1-w) * [w * T_{n-4} + \dots]]]]$$

$$= w * T_n + w(1-w) T_{n-1} + w(1-w)^2 T_{n-2} + w(1-w)^3 T_{n-3} + w(1-w)^4 T_{n-4} + \dots$$

$$\text{if } w = \frac{1}{2}$$

$$x_{n+1} = \frac{T_n}{2} + \frac{T_{n-1}}{2^2} + \frac{T_{n-2}}{2^3} + \frac{T_{n-3}}{2^4} + \frac{T_{n-4}}{2^5} + \dots$$

It is enough to estimate the next cpu burst length using this formula by only calculating the first few terms

13) Priority Scheduling

each job is arranged a priority number generally, low number means low priority

The cpu is given to the process with high priority number

Process	Arrival time	Priority	cpu burst
P ₁	0	5	7
P ₂	2	3	4
P ₃	4	10	1
P ₄	5	1	4

There are two versions of this algorithm

1. Preemptive: If a new process arrives with priority higher than the running process, then CPU switches to the new process

(2) Non-preemptive: —————, the CPU continues with the running until it finishes its burst, then switches to the new process

Problems :- Starvation

Solution :- Aging

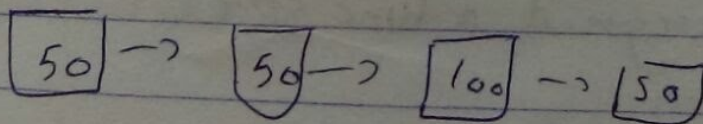
With time increase (increment) the priority number for the waiting process

(4) Round-Robin (RR)

each process is assigned a slice of time (quantum)

The CPU executes this quantum Q and then to the another process

- The processes are served using FCFS

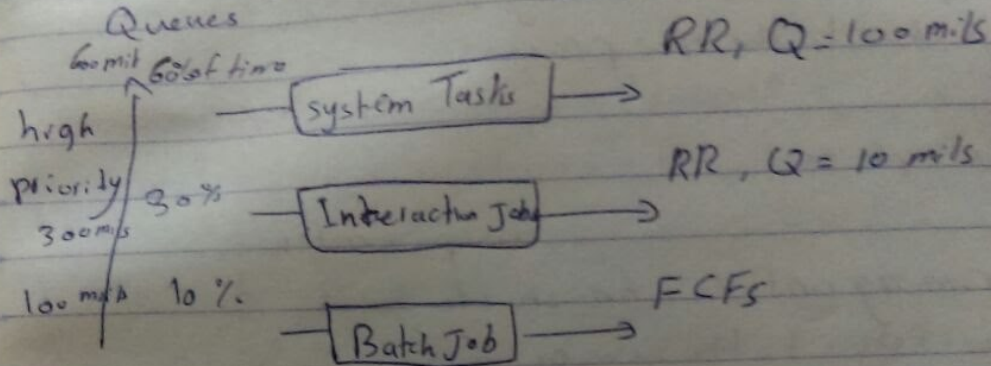


Fair to users

Reasonable :- the around and waiting time

(5) Multi-level Queues

The Ready Queue is divided into several Queues



There should be a scheduling algorithm for each Queue

- Scheduling among the queues :-

- 1) Fixed priority : serve all process in "system task" then all process Interactive jobs - after all the process "Batch jobs"

There are two versions :-

- Preemptive
- Non preemptive

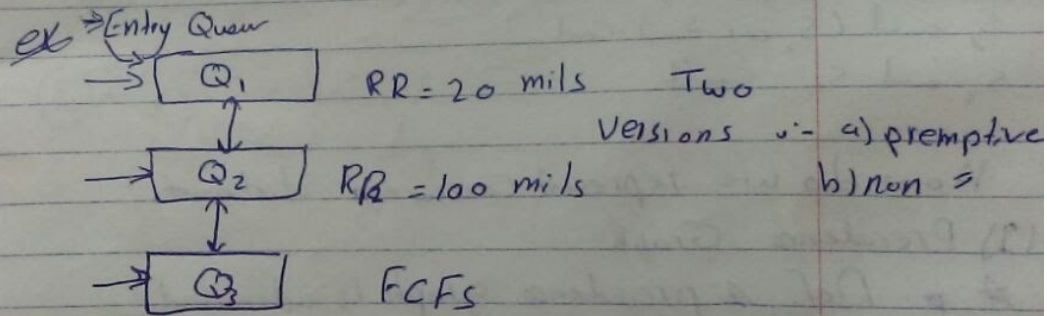
(2) Problem - Starvation

Time slice

Each queue is assigned a time slice to run its process

[6] multi-level Feedback Queues.

- The Ready queue is divided into several queues.
- Each queue has its own scheduling algorithm
- The process can move up (upgrade) between queues
- = = = low (demand) = =
- There should be one queue among the queues called the Entry Queue, where the processes enter the first time



Algorithm Evaluation

with one is the best algorithm?

Deterministic Model similar what we did in the example. poor performance

1) Simulation Better than Deterministic model

Implementation

~~Handwritten scribbles~~

chapter 6 Concurrency and Synchronization

ex Given the following statements:-

Start

$S_1: a = x + y$ (cpu-1) or gadget
 $S_2: b = z + 1$ (cpu-2) or gadget
 $S_3: c = a - b$
 $S_4: w = c + 1$

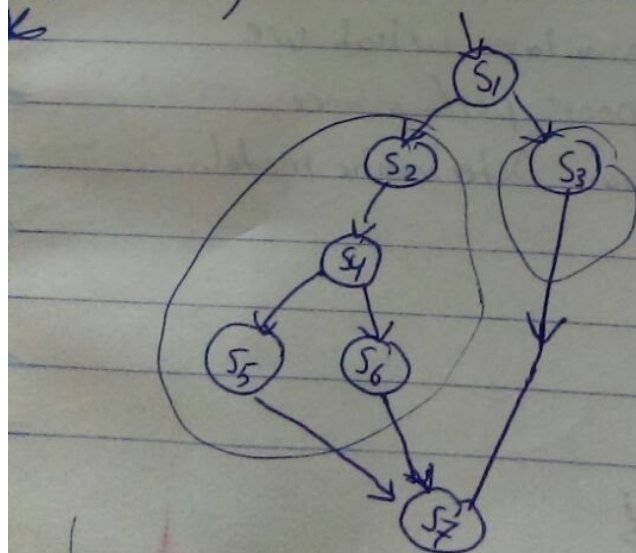
S_1 and S_2 can be executed concurrently
 S_1 and (S_3 or S_4) can't = =
 S_2 and (S_3 or S_4) can't = =
 S_3 and S_4 can't = =

How do we represent Concurrent Statements?

1) Precedence Graph

A Def: A precedence graph is a directed graph whose nodes represent statements

An edge from (S_i) to (S_j) means that S_j can only be executed after S_i .



S_3 and $(S_2 \text{ or } S_4 \text{ or } S_5 \text{ or } S_6)$ can be executed concurrently

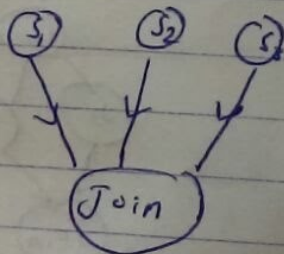
S_5 and S_6 can be executed concurrently
3 different CPUs

[1] produce Graphs

[2] Fork and Join structures

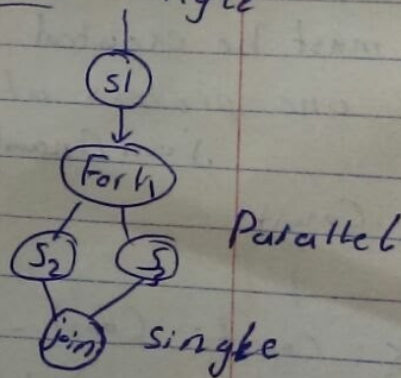
Fork and join are system calls

(S_2) and (S_3) are concurrent
as $\rightarrow$ Fork

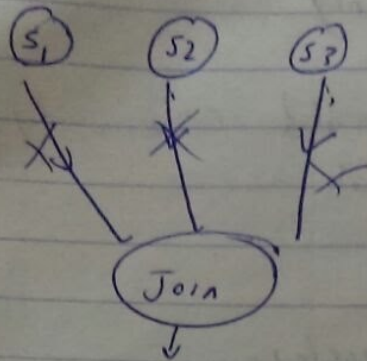


Count = 3

```
function join;
{ Count = Count - 1;
  if (Count == 0) then
    quit;
}
```



Count
10



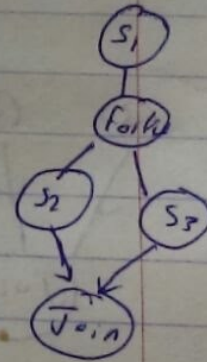
The Result depends on executed of "join"
for this to work perfect, the "join" instruction
must be executed "atomically", that is
one process at time

Join Concurrently process - جزئی آکثر

Count
}

$S_2: \text{Count} = \text{Count} - 1;$

$S_3: \text{Count} = \text{Count} - 1;$



Count = 3;

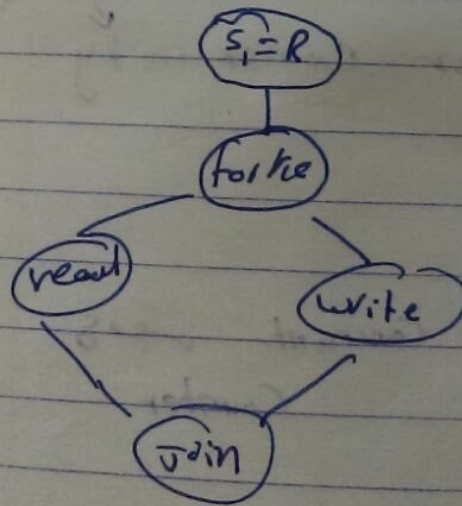
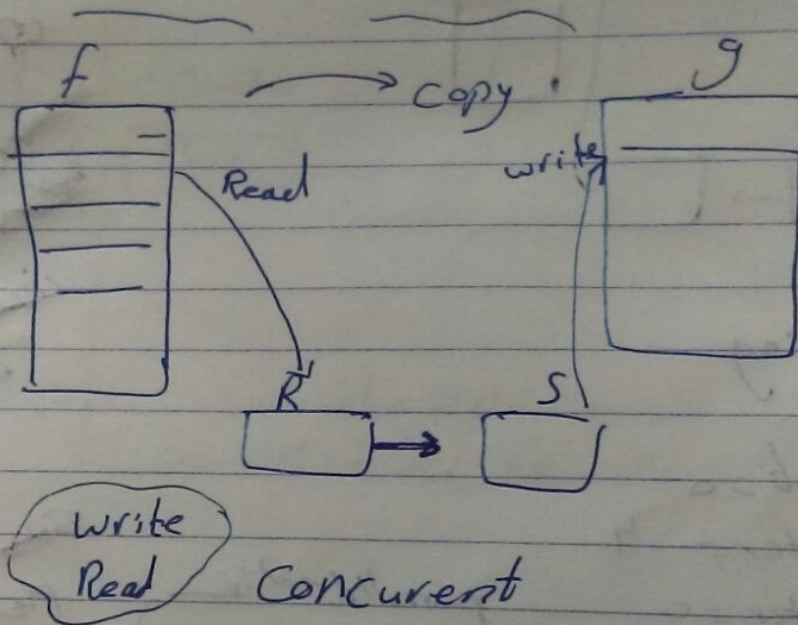
S1;

Fork L1

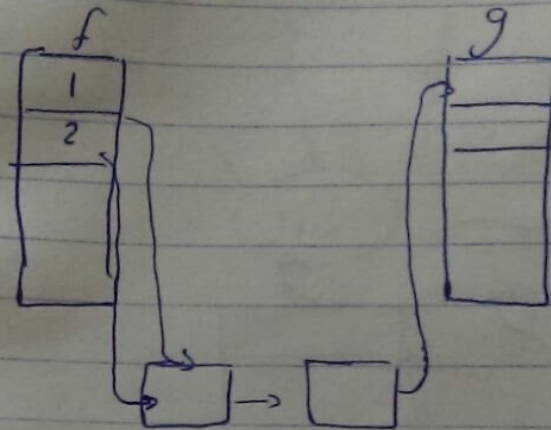
S2;

S4;

Fork L2



(3) Parbegin and parent statement (structure)

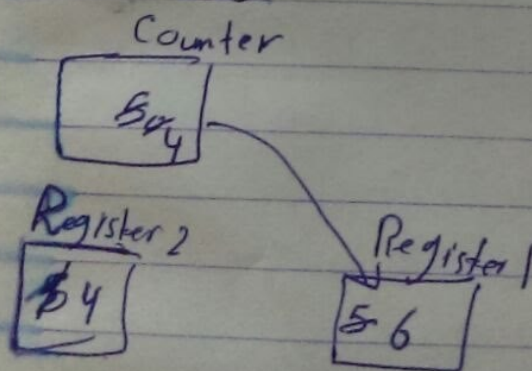


Bussy waiting

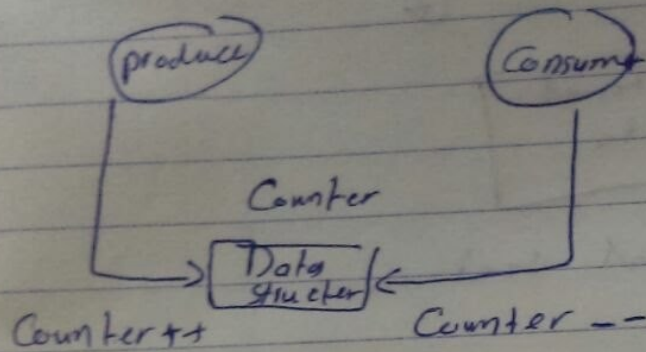
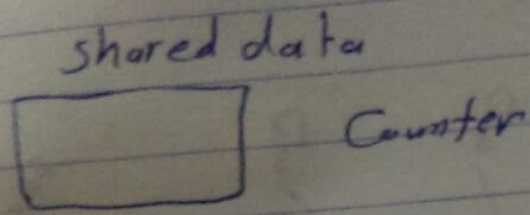
int Count = 0
 Full: Counter = 4
 Empty: Counter = 0

parallel ~~line~~

producer and consumer concurrency

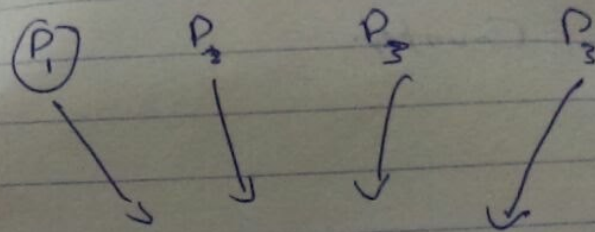


Concurrent ~~line~~
 Counter



بالعربي

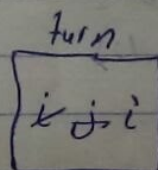
Progress: في يوم من الأيام كنت في
إذا الإجمال سبب فاضلة دروس



Structured Data

critical section

entry section
Reminder section :-



P_i

P_j

Reminder Section

Reminder section

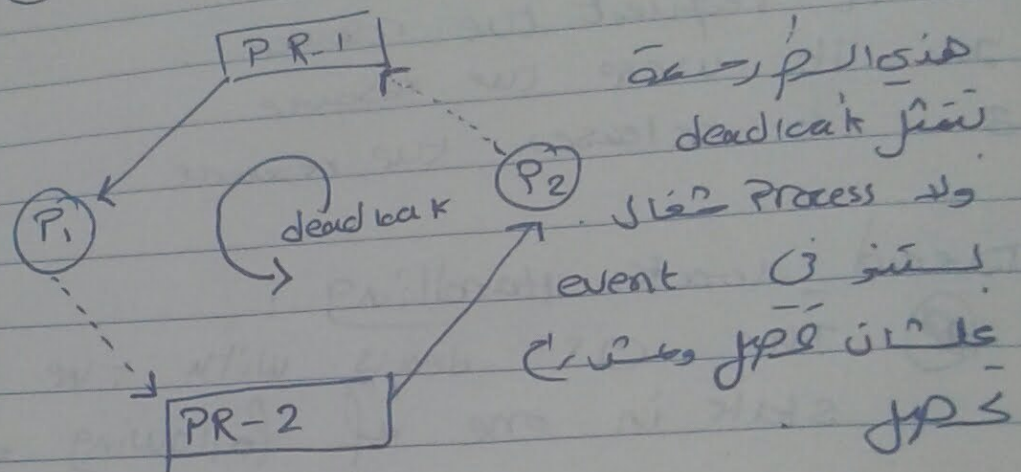
دوری دوری

Reding Bakery

Ch. 7

* New Lecture:
Deadlock

○ Resource Allocation Graph



* System model:

there are n resource types

$R_0, R_1, R_2, \dots, R_{n-1}$

tape CPU Printer

- For every resource type R_i , there are w_i instances of this type

$R_2 = \text{Printer}$

$w_2 = 5$

- $R_0, R_1, R_2, \dots$

* each Process uses the resource in the following order

1. it request the resource.
2. it uses the resource.
3. it releases the resource.

Dead lock Handling :

① The OS deals with the deadlock state in one of following two ways

1) it allows the deadlock to happen and then it recovers from it

(2) it prevent or avoids the OS to enter a deadlock state.

* In both cases, this requires ^{such} extreme measures such as, preempt[←] some of the resources from the process

* The OS must decide
(1) which resource must be preempted
(2) what will be happen after that.

* Condition for a ~~dead lock~~ ^{to occur} to occur:
4 condition must hold ^{simultaneously} for dead lock to occur:
وإذا لم يفسد في أي من هذه الحالات
dead lock ^{يحدث}

(1) mutual Exclusion: the Resource type must be used exclusively, that is, non-sharable.

(2) Hold and wait
There exists a process holding some resource type & waiting for another process to release its resource, at the same time this process is waiting for the first process to release its resource.

(3) No preemption

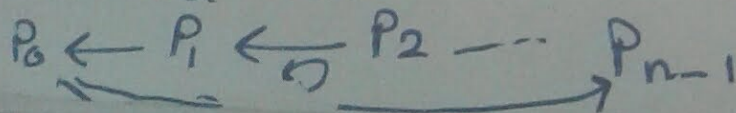
(4) Circular wait (cycle)

Assuming we have n processes, $P_0, P_1, \dots, P_{n-1}$

P_0 is waiting for process P_1 to release its resource.

P_1 is waiting for process P_2 to release its resource.

P_i is waiting for process P_{i+1} to release its resource. cycle



* Resource Allocation Graph:

Graph

$$G = (V, E)$$

V : set of vertices

E : set of edges

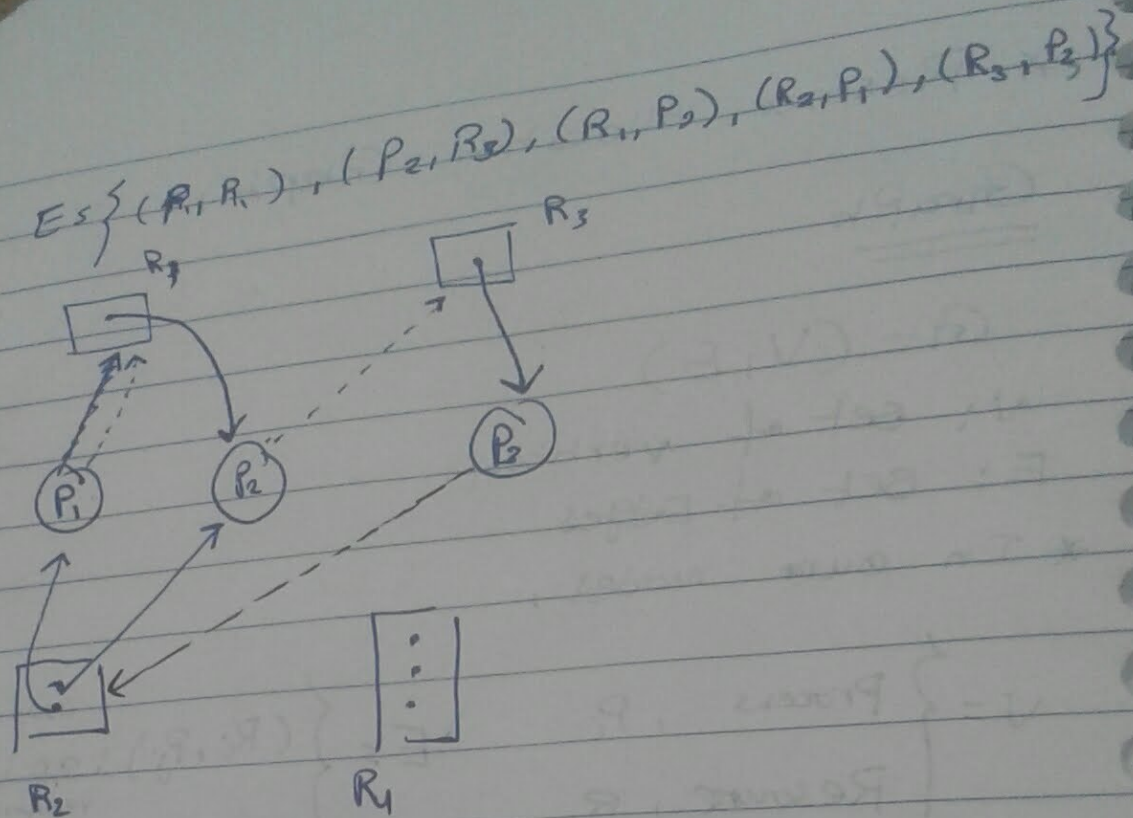
* In our model,

$$V = \begin{cases} \text{Process, } P_i \\ \text{Resource, } R_j \end{cases}$$

$$E = \begin{cases} (R_i, P_j): \text{an instance of resource is assigned to process } P_j \\ (P_i, R_j): \text{Process } P_i \text{ is requesting an instance of resource } R_j \end{cases}$$

$$\text{EX: } P = \{P_1, P_2, P_3\}$$

$$R = \{R_1, R_2^2, R_3^1, R_4^3\}$$



Is there a dead lock?
 deadlock $\Rightarrow$ cycle $\Rightarrow$ $\times$

No cycle $\Rightarrow$ No dead lock

* Assume process P_3 requested from the OS an instance of R_2 ?

there is two cycle: $P_1 R_1 P_2 R_3 P_3 R_2 P_1$
 $P_2 R_3 P_3 R_2$

There will be dead lock

New Lecture:

Deadlock Prevention:

The idea is to make sure at least one of the 4 necessary condition must not hold.

(1) Mutual Exclusion

it always hold for non-sharable devices, that is the OS can't break this condition.

(2) Hold-and-wait : back to draw.

- let process demands all its resources at the beginning

- All the resource are granted to the process at one time.

problem: Starvation.

★ Dead lock prevention :-

The idea is to make sure at least one of the 4 necessary conditions must not hold.

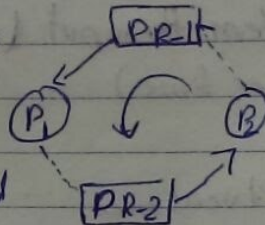
(1) Mutual Exclusion :-

It always hold for non sharable device.
 the OS can't break this condition

(2) hold-and-wait

- Let process demands all its resources at the beginning

- All the resources are granted to the process at one time
Problem: Starvation



Process resource not going

(3) No preemption

When a process request another instances of the resource which is not available, let the process release its resource which holds

Problem
Starvation

4) Circuit wait: Impose a linear order of the resource types depending on the order of use.

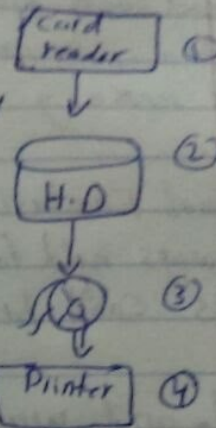
Conclusion :-

Dead lock: prevention tries to make sure that one of the 4 necessary condition do not hold in order to prevent a dead lock from happening.

a side affect (disadvantage) of

Dead lock prevention - low device utilization

- reduce system throughput (number of process finish execution per time)



Dead lock Avoidance :-

Def: a system is in a safe state if the sequence of processes $\langle P_0, P_1, P_2, \dots, P_{n-1} \rangle$ such that :- P_0 can take the available resource, runs and finish execution.

P_1 can take the available resource + the resources released by P_0 , run and finish execution

$P_2 = \dots = \dots = \dots = \dots = \dots$
 P_0, P_1

$P_i = \dots = \dots = \dots = \dots = \dots$

$P_{n-1} = \dots = \dots = \dots = \dots = \dots$
 $P_0, P_1, \dots, P_{i-1}$
 $P_0, P_1, \dots, P_{n-2}$

- if there is a safe sequence, then there is no dead lock

ex: A system with 12 tape drive, 3 processor
at a certain moment, the system, looks like:-

Process	Max needs	Allocated	Current needs
P_0	10	5	5
P_1	4	2	2
P_2	9	2	7

Is the system safe?

Available: $12 - 9 = 3$

Yes, there a safe sequence

$\langle P_1, P_0, P_2 \rangle$ No dead lock

- Assume at same time, P_2 requested one tape unit, and the OS made a mistake and gave it to the P_2

Is the system safe $\langle P_1, ?$

20

No safe sequence

Dead lock

Implementation
Banker's Algorithm :-
We should take the following points in
consideration :-

- ① For simplist purposes we will consider only one resource type, but this can be generalized easily to more than one resource type.
- ② Each process must declare in advance the maximum instance (units) of the resource it needs.
- ③ When a process requests a resource, it may have to wait.
- ④ When a process gets all its resource, it must release (return) them in a finite time.

Data structure used in Algorithm :-

1. Array MAX:

$MAX[i] = k$, process P_i , requests at most k instances of the resource

2. Array Allocation

$Allocation[i] = m$, the process P_i is currently holding m instance of the resource

3 Arrays NEEDS:-

$NEEDS[i] = n - p_i$ currently need
(requires) to n instance of the resource

4 Array Available = R (The system
has currently R instance of the resource)

Algorithm:-

1- let w = available Define an array
 $k[i] = 1$, for all i

2- Find on i such that:
if (no such i exists), Go to step (4)

3- $w = w + Allocation[i]$
 $k[i] = 0$

Go To step (2)

4- if ($k[i] = 0$ for all i) then system
is safe,
else
system unsafe

Process	max	Allocation	NEEDS
P_0	10	5	5
P_1	4	2	2
P_2	9	2	7

w
 $\boxed{3510}$ 12

$\langle P_1, P_0, P_2 \rangle$

$\text{Max } [i, j] = k_i$ means P_i requests
 at most k_i instance of resource R_j

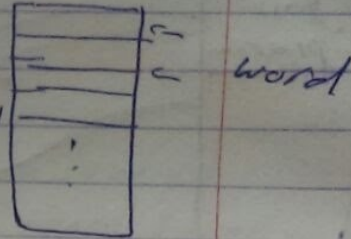
R_1	R_2	R_3

Chapter 8

memory management (Ordinary mem)

* memory is an array of word each is addressable

In chapter 8, we will consider that all program must be brought into memory for execution



word = 4 byte

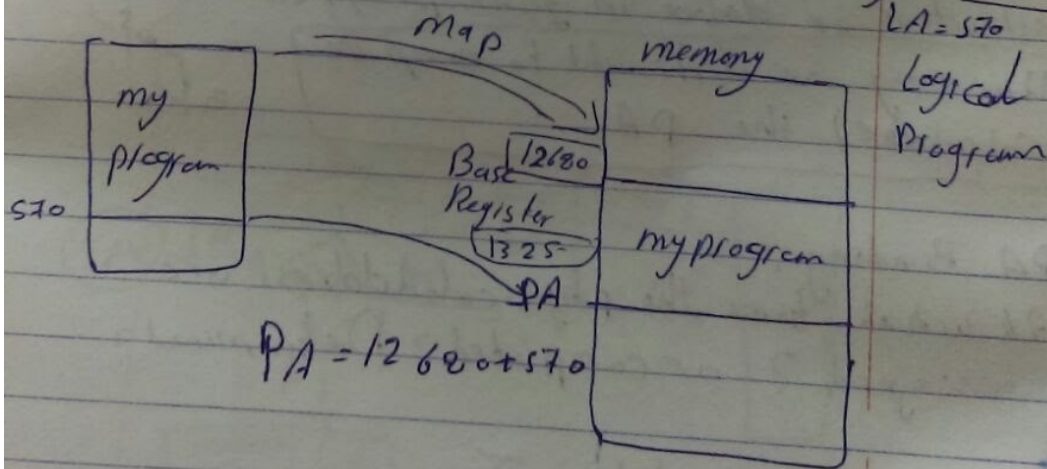
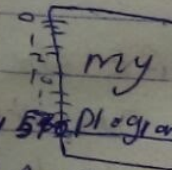
* Logical Vs Physical Address :-

Logical address (LA) : The address

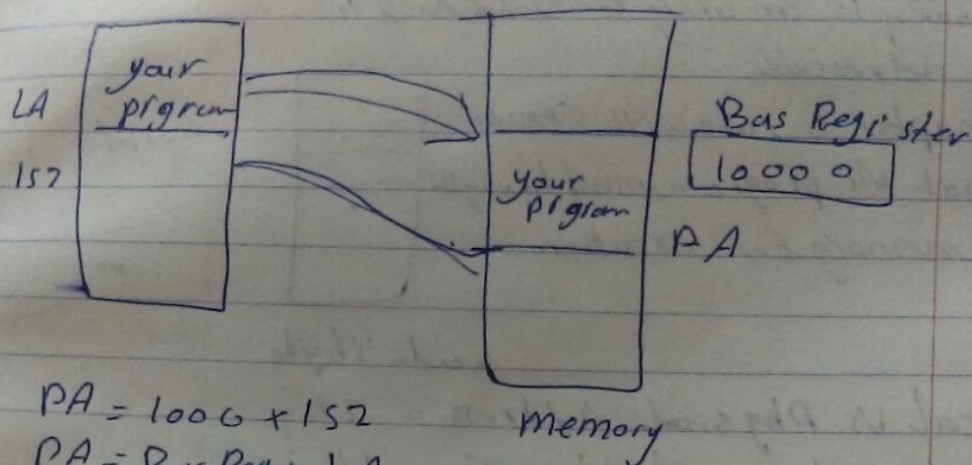
seen in your program and generated by

the CPU. Physical Address (PA) : It

is the actual address seen in memory



Logical Address Vs physical Address



At the end, The cpu needs to obtain (compute) the PA in order to fetch the instruction or data in your program

How easy or difficult to compute (calculate) the PA

of
address

PA Binding :-

At what time the physical Address are assigned ? or computed ? Determined ?

(1) At Compilation Time

(2) At Loading Time

The disadvantage. The programme can't be moved during execution

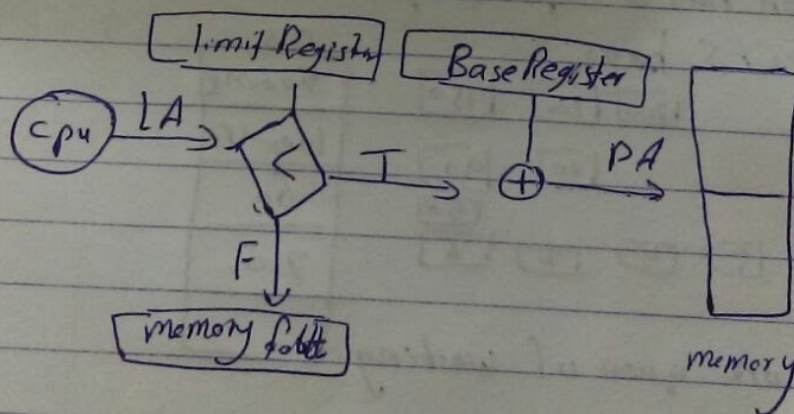
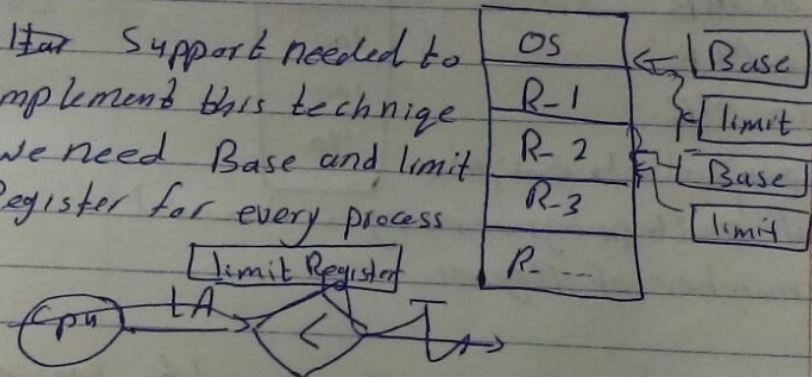
(3) At execution Time

The PA is determined or computed at execution. The program can be moved during execution.

[1] Multiple partitions

- Memory is divided into partitions (regions)
- Each region can hold only one process (programme)

- For support needed to implement this technique we need Base and limit Register for every process



Two versions available:-

A) Fixed Regions: In IBM-OS: MFT → Task
 Multiprogramming with ← 6 fixed number of

- The number of regions is fixed, and the size of the regions are fixed.

ex 1 4 Region:-

1 × 500 MB

2 × 100 MB

1 × 40 MB

OS
500
100
100
40

درجات متعددة ومتغيرة

- Degree of multiprogramming is bounded by the number of Regions

Job scheduling: how process are assigned to regions?

(1) each region has a separate queue which serve on FCFS basis

OS
500 MB
100 MB
60
20

200	200	450
70	90	50
5	15	7
6		

(2) only one queue of waiting jobs

- 4/25/2020
- (5)
- (a) FCFS with or without skip
 - (b) Best fit only
 - (c) Best available fit

Problems :- Major Problem

Internal fragmentation: its the unused free space in the assigned region

- IBM-360-OS

EBF Variable (Dynamic) Region

Fixed Regions: Internal Fragmentation

[B] Dynamic (Variable) Regions

IBM, MVT

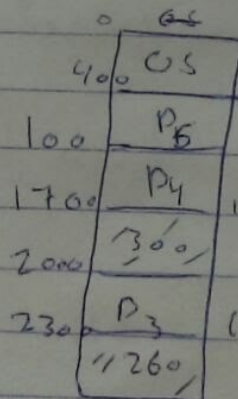
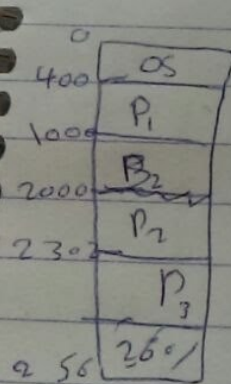
Multiplying with Variable number of Task

ex Assume memory 2260 KB

OS = 400 KB

Given the following Queue of waiting jobs

Process	Memory M	Time Required
P ₁	600K	10
P ₂	1000K	5
P ₃	300K	20
P ₄	700K	8
P ₅	800K	15



After a while memory will be constructed

- 1) Allocate Regions for Process
- 2) Holes external diagram

At

T₅₀

T = 25

Job scheduling : How the OS select
a hold to first a process

- 1- First fit
- 2- Best fit
- 3- Worst fit

first fit +
Best fit have

Better performance

Problem: external fragmentation (holes)

Solution: Combaction defragmentation

The OS Reallocate the jobs every
certian amount at time event

use like

[2] Non Contiguous Memory Mangment
(Paging)

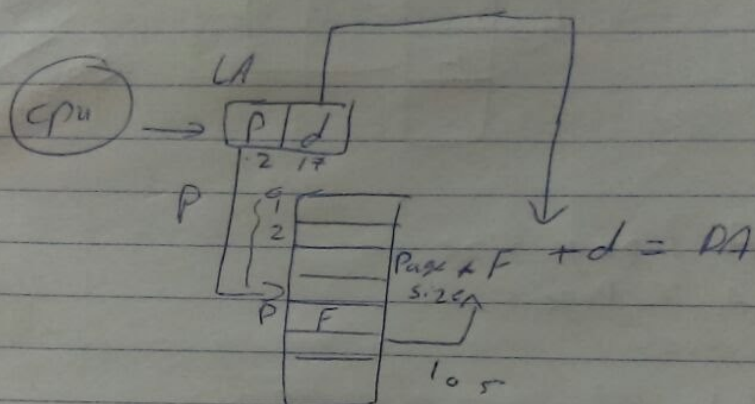
Base
Regs.

- Divid the logical program into equal
size partitions calle "pages"
- Divid memory into partion same size called "frame"

Page	Logical Program	Page #	Frame #	Address	Content
0	A	0	100		
1	B	1	108	101	B
2	C	2	101	102	
3	D	3	105	103	E
4	E	4	106	104	
5		5	103	105	C
6		6		106	D
7		7		107	
8		8		108	A
9		9		109	
10		10		110	

support needed to implement this Algorithm
We need "Page Table"

$P = \text{Page \#}$
 $d = \text{Page offset}$



Page Table

$$P = LA / \text{Page size}$$

$$d = LA \% \text{Page size}$$

Assume: Page size = 100

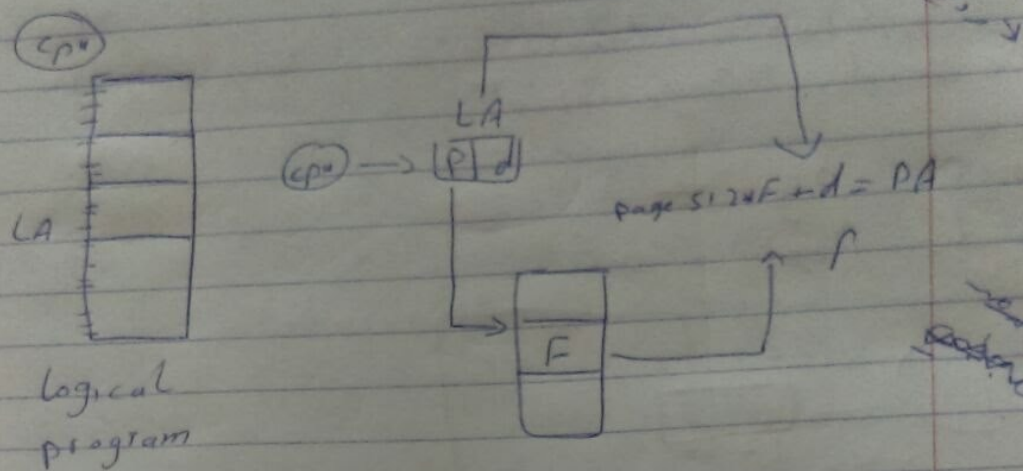
Given LA = 217

$$P = 217 / 100 = 2$$

$$d = 217 \% 100 = 17$$

$$PA = 100 * 105 + 17 = 10517$$

$$\begin{array}{r} 0 \\ \hline 100 \\ \hline 100 \end{array}$$



$$P = LA / \text{page size}$$

$$d = LA \% \text{Page size}$$

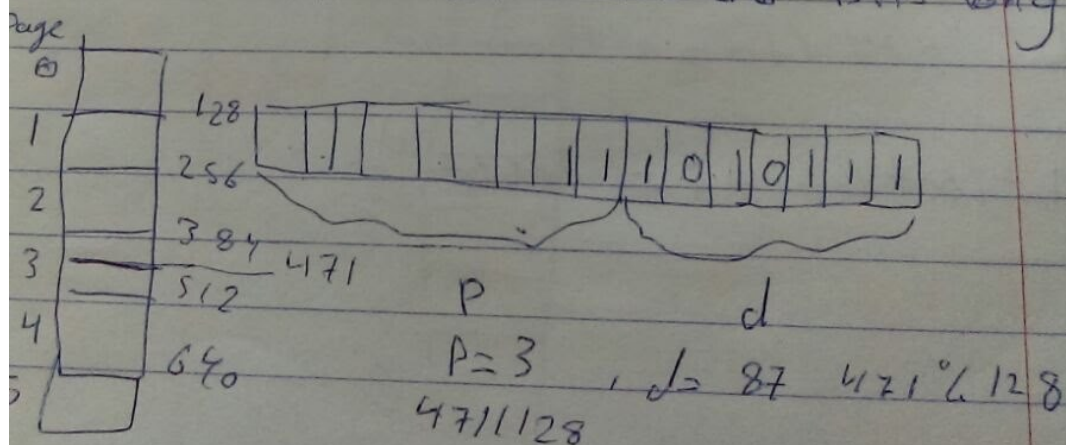
The OS take advantage of page size
The page is always 2^n bytes

$$2^{10} \leq \text{page size} \leq 2^{13}$$

$$1024 \leq \text{page size} \leq 8192$$

Assume Page size = $128 = 2^7$

LA = 471 Assume LA = 24 Bits long

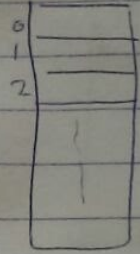


The OS take the n low bits of the logical address = d
 The remaining Bits = P

ex Dec-10-OS: LA = 20 Bits
 page size = ~~2048~~ 512 = 2^9

This mean, the max program can be executed in this system is:-

11 Bits	9 Bits
P	d

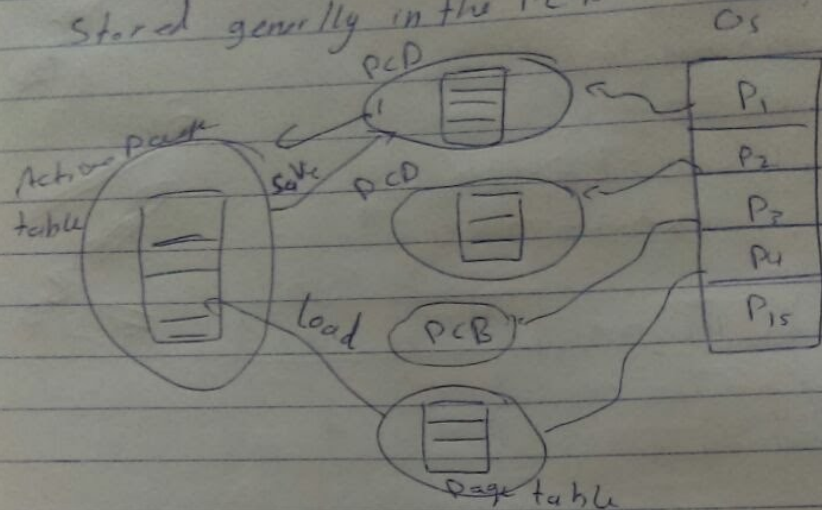

 $\Rightarrow$ max program contains 2^{11} pages
 $2^{11} \times 2^9 = 2^{20}$ bytes

IBM-370 OS:
 LA = 24 Bits
 page size = 2048 = 2^{11}

13 Bits	11 Bits
P	d

max program = $2^{13} \times 2^{11} = 2^{24}$ bytes

- Each process has a page table which is stored generally in the PCB



Note: In every instruction, the CPU access Page table to get the frame #, f

- Active page table: Contains the page table of the current running process
- The active page table is flushed with every context switch
- The CPU is accessing the active page Table. Therefore, the active should be stored in a high speed storage medium
- Where to store the page Table?

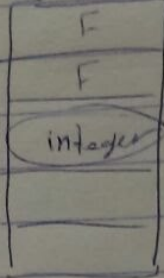
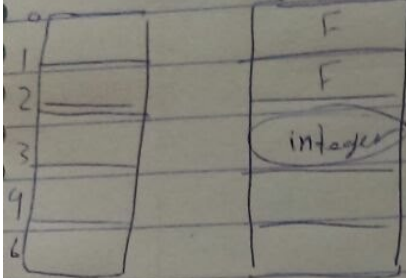
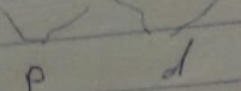
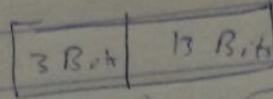
[1] Registers

The OS dedicated a number of registers to store the active page table

Ex 1m, PDP = 11, OS, LA = 16 Bits
 Page size = $2^{19} = 2^{13}$

This means the max
 program contains

$2^3 = 8$ pages

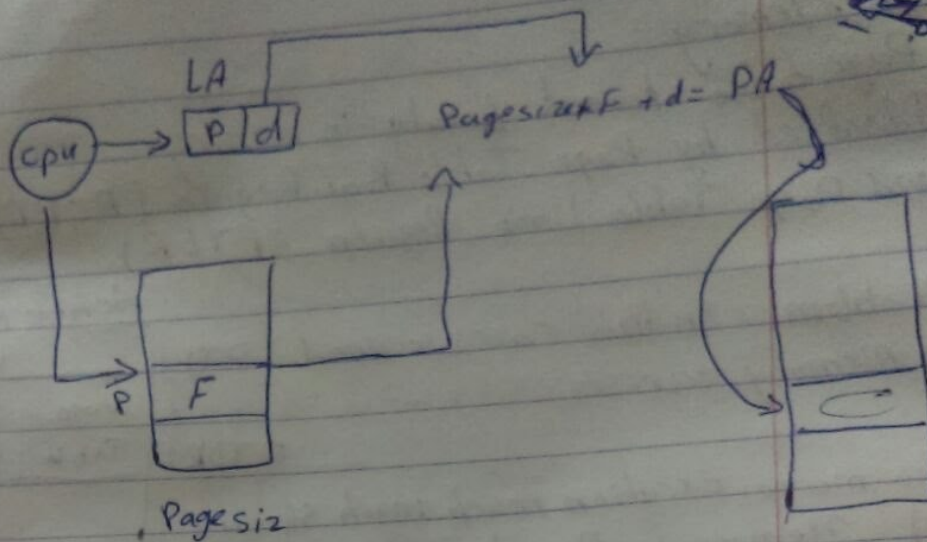


∴ Page table must contain
 Assume page table entry
 8 entries = 4 bytes

$2^3 = 8$

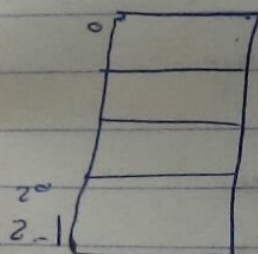
Page table

Page table size
 = $8 \times 4 = 32$ bytes



ex $LA = 32 \text{ Bits}$
 $\text{Page size} = \frac{LA}{2} = 2^{12}$

20 Bit	12 Bit
P	d



$\Rightarrow \text{Page Table Size} = 2^{20} \times 4 = 4 \text{ MB}$

Logical
Program

(2) Memory

- Page table stored totally in memory
- Identified by Page Table Base Register (PTBR) and Page Table Limit Register (PTLR)

- Problem: In this case we need two memory accesses



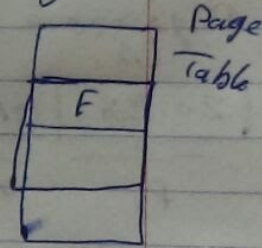
- This makes execution much much slower

(3) Memory + Register

- The main page Table is stored in memory

- A few Register called "Associative" Register are assigned to the active Pages tables, called some time

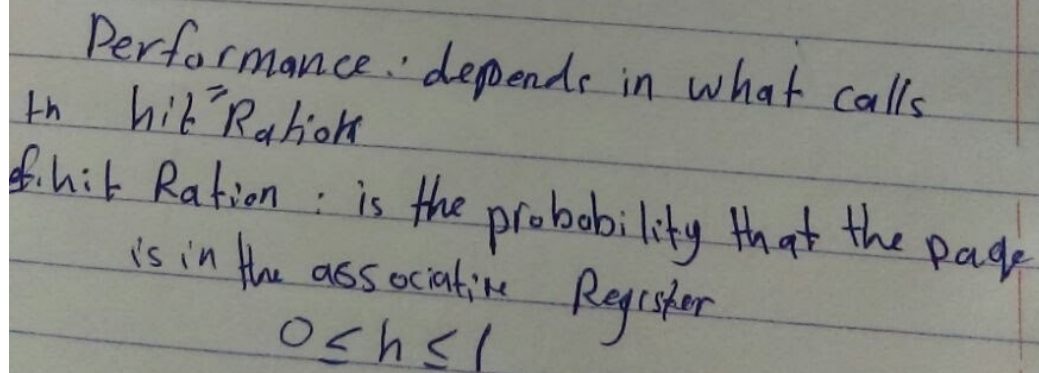
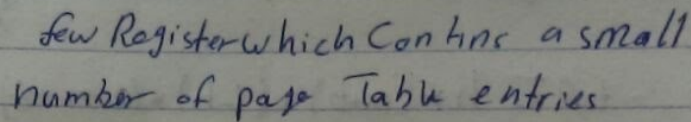
Translation Look-aside Buffer (TLB) which contain the active Pages during execution



- Initially, the CPU checks (examines) if the page in which execution will take place in is the associative Register

If so, continue execution

else if gets the page and insert it in the associative and continue execution.



$h=0$ every page execution, the page is not in associative Register

$h=1$, all executions, the page in the associative Register

ex Assume :- $EAT = h \times (m+t) + (1-h) \times (2m+t)$

memory access $m = 100$ nsecs

Search (lookup) time in the associative Register $t = 2$ nsecs

Hit Ratio, $h = 0.95$

Effective Access Time (EAT) =

$$0.95 \times (100 + 2) + 0.05 \times (2 \times 100 + 2) \\ = 107$$

Memory Protection :-

- a Bit is added to the page label