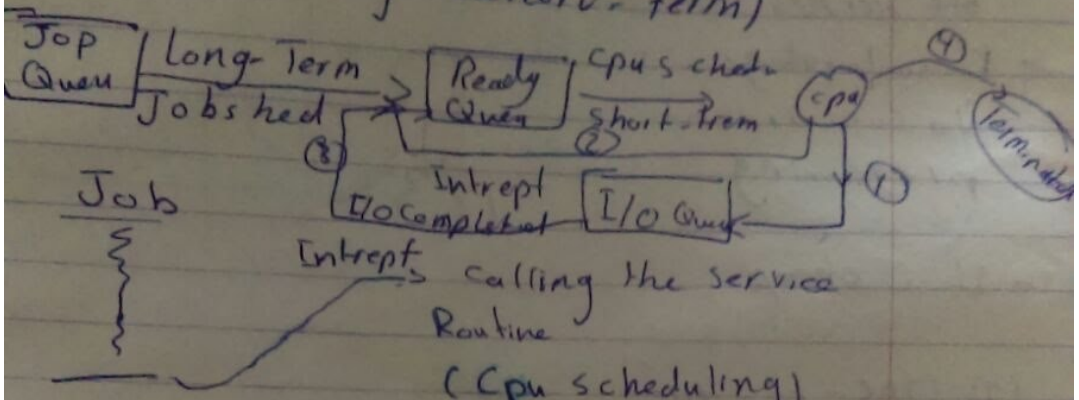
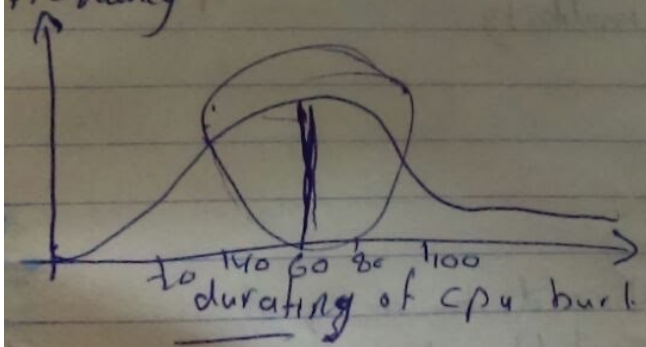
- Conarancy allowed
- Cost Reasonable
- Rebibility, Ok

Chapter 5

Cpu Scheduling (Short-term)



- Our objective is to maximize the cpu utilization



* what are the Cases the cpu scheduling will be invoked?

Cases (1) and (4) are called non-preempti
(مبسط دو افلاد)

. Case (2) and (3) are Called preeptive (قوة بالقوة

* Cpu scheduling Criteria :-
(1) Cpu utilization
(2) Throughput max
(3) Turre arround tim min
(4) waiting time

* Throughput :- # of Jobs that finished exhution per unit of times (averaged)
Turren time :- it the time from submeting the Job till it finished exutiont
① it the time wait the process Spend in the ready Quen

* cases: (1) & (4) are called non-Preemptive (غير مسبق)

* cases (2) & (3) are called Preemptive (مسبق)

* CPU scheduling criteria:

(1) CPU utilization max

(2) Throughput: # of jobs runt (max) finish execution Per unit of time (averges).

(3) Turn around time; its the time from submitting job still its finish execution. min

(4) waiting time: its the time process spends in the ready Queue (min)

5) Response time, the time from submiting the top until the first response come from the computer (Psychology) (minimum).

* New Lecture:

CPU Scheduling:

1) FCFS: First came First save.

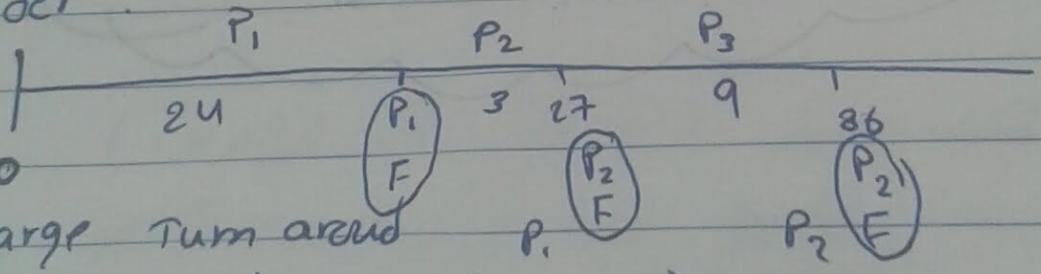
* the CPU is give to the process which arrived first

Ex: Assume the ready queue look like:

Process	Burst time
P_1	24
P_2	3
P_3	3

Compute the avarge turn around ~~time~~ time and avarge waiting time.

* The Process around in the same order.



(Avarge Turn around Time(ATT) = $\frac{(24-0)+(27-0)+(36-0}{3}$

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

* Avarge waiting time (AWT)

$= \frac{(24-0-24) + (27-0-3) + (30-0-9)}{3}$

$= \frac{51}{3} = 17$ unit

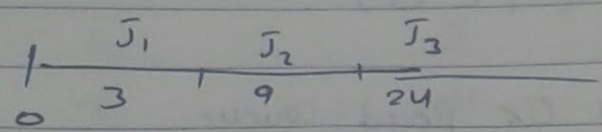
Note: Waiting time = Turnaround time - CPU burst

Problem in FCFS :

is called convoy (مركب) effect :

↑ كيفية ترتيب المركب

اذا اختلف ترتيب وصول [illegible] رح يتغير Time



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

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

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

* لا يمكن اعتماد الالخوارزم على الوصول)

↳ Performance is not good.

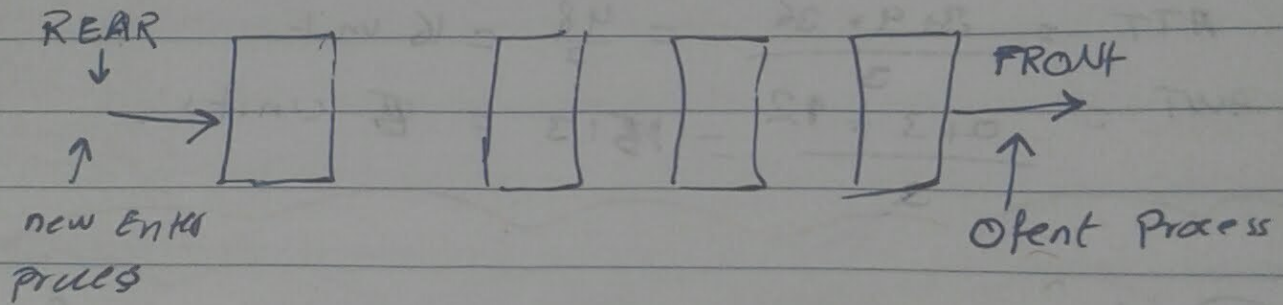
FCFS:

Implemntion:

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



*

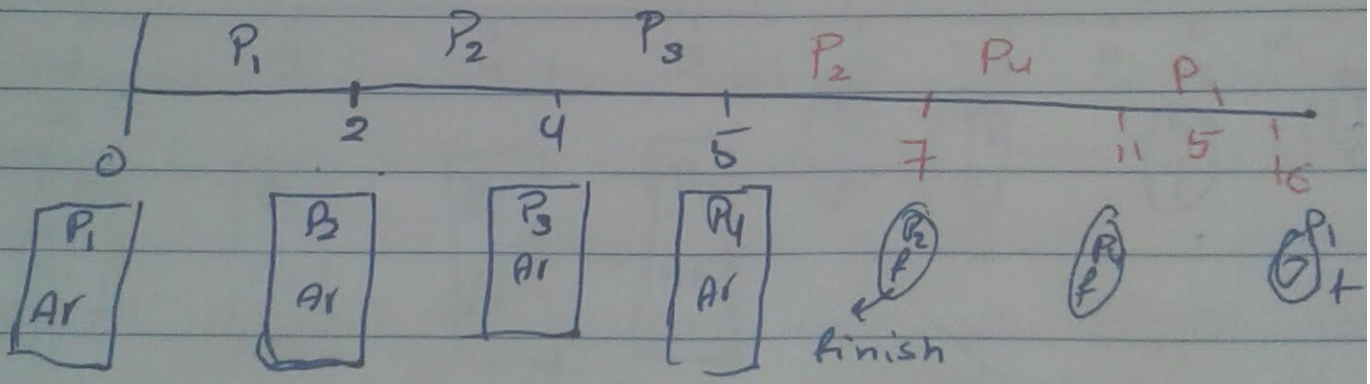
[2] Short Job first :

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	~~7~~ 5
P_2	2	~~4~~ 2
P_3	4	~~1~~ 0
P_4	5	4

Note: minumum burst in the Queue.



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

Short Job first (SJF)

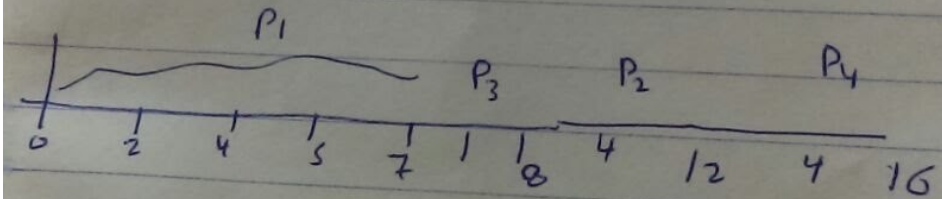
Two versions

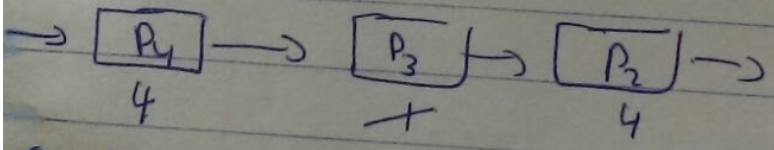
(1) preemptive

(2) Non =

if new process arrives with cpu burst less than the remaining time of the running cpu pairs the cpu continous executing the runing cpu pairst and then switcheting to another cpu burst

Proces	Arrival Time	Cpu Time
P_1	0	7 0
P_2	2	4
P_3	4	1 0
P_4	5	4





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

Problems :- Starvation

solution : Aging : As time progresses give the process wich 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: theres no way to decide exactly the length of the next cpu burst

Solution : Estimation

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

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

if $w = 1$ $y_{n+1} = T_n$ a little bit better. but still no practical

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

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

$$Y_{n+1} = w * T_n + (1-w) * [w * T_{n-1} + (1-w) * [w * T_{n-2} +$$

$$(1-w) * [w * T_{n-3} + (1-w) * [w * T_{n-4} \ldots +$$

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

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

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

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

(3) Priority Scheduling

each job in 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 alogrith

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

(2) Nonpremptive: = = =
= = = = =, the cpu continues with the runing until it finish its burst, then switches to the new processs

Problems :- Starvation
solution :- Aging
with time incease (increment) the priority number for the waiting process

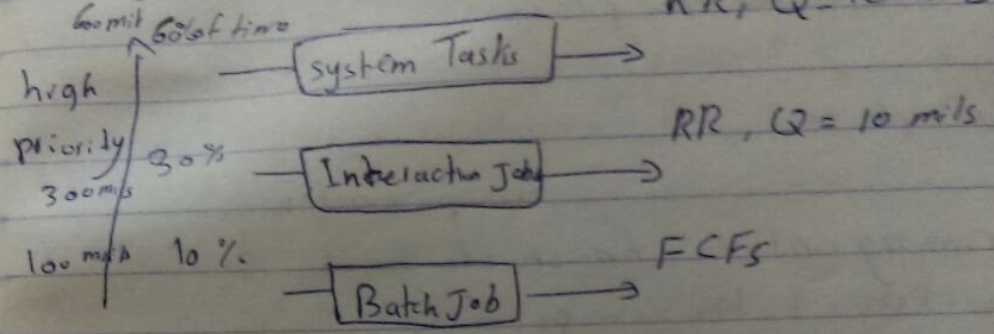
(4) Round - Robin (RR)
each process is assigned a slice of time (quntam)
- The cpu excutes this quntam Q and then to the another process
- The processes are served using FCFs

50 → 50 → 100 → 50

Fair to users
Reasonable :- the a around and waiting time

(5) Multi-level Queues

The Ready Queue is divided into several Queues



There should be a sheduling algorithm for each Queue

- Scheduling among the queues :-

(1) fixed priority : serve all process in "system tasks" then all process Interactive Jobs after all the process "Batch jobs"

There are two versies :-

- Preemptive
- Non preemptives

(2) Problem :- starvation

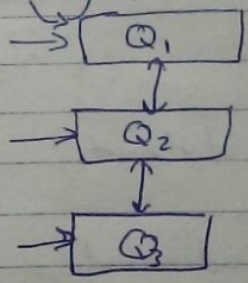
Time Slice

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
- " " " " down (demote) " "
- There should be one queue among the queues called the Entry Queue, where the processes enter the first time

ex ⇒ Entry Queue



RR = 20 mils

RR = 100 mils

FCFS

Two versions :- a) preemptive
b) non =

Algorithm Evaluation
with one is the best algorithm?
1) Deterministic Model .similar what we did in the example. poor performance
2) Simulation Better than Deterministic model
3) Implementation

chapter 6 Concurrency and Synchronizat

ex Given the following statments :-

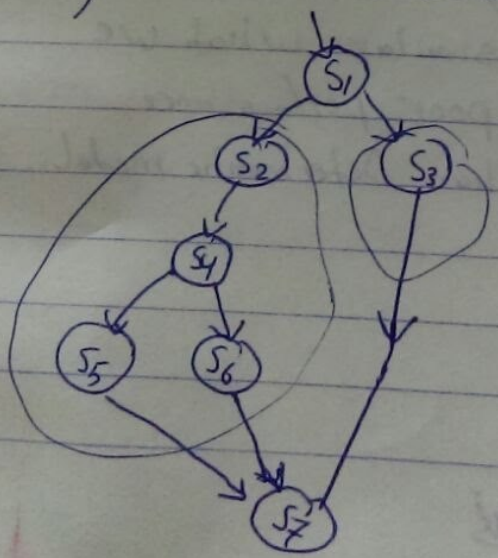
S_1: a = x+y (cpu-1) or gadget
S_2: b = z+1 (cpu-2) or gadget
S_3 c = a - b
S_4 : w = c+1

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

How do we represent Concurrent statments?

1) Precedence Graph

Def: A precedence graph is a directed graph whose nodes represent statments
An edge from (S_i) to (S_j) means that S_j can only executed after S_i.



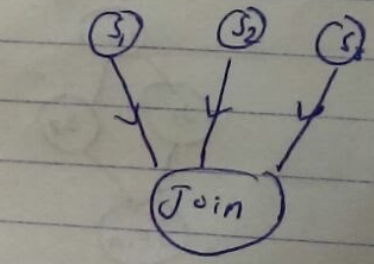
S_3 and (S_2 or S_4 or S_4 or S_6) can be executed concurrently

- S_5 and S_6 can be executed concurrently 3 different cpus

[1] Precedence Graphs
[2] Fork and Join structers

Fork and Join are system calls single

S_2 and S_3 are concurrent after fork

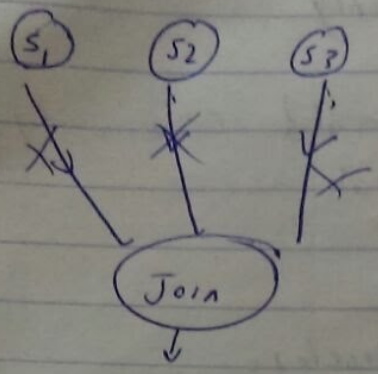


Parallel

single

Count = 3
function join;
{ Count = Count - 1;
if (Count != 0) then
quit. Computation;
}

Count
2 1 0

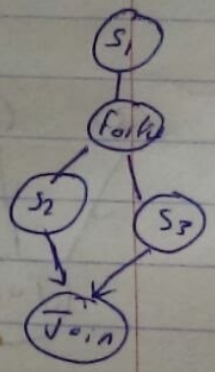


The Result depends on executed of "join"
for this to work perfect, the "join" instruction must be executed "atomically", that is one process at time

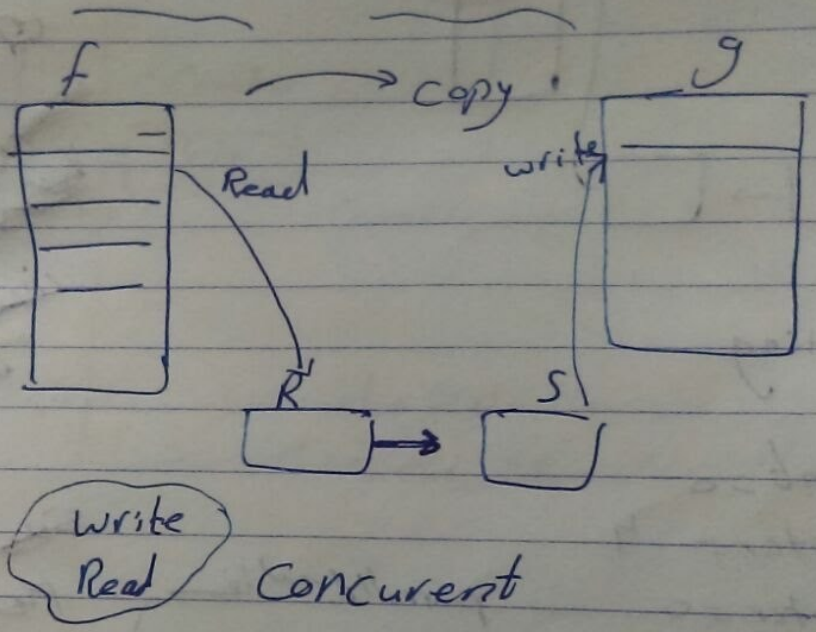
join Concurrently تعني process أكثر من

Count
}

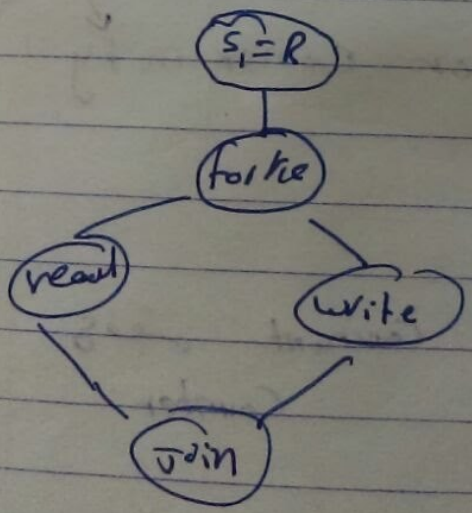
S_2: Count = Count - 1;
S_3 = Count = Count - 1;



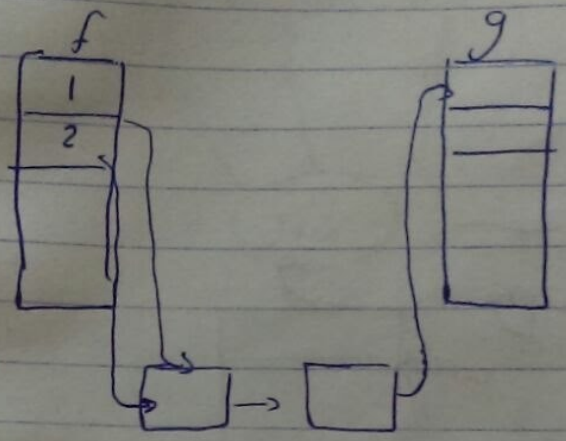
Count = 3;
S1 ;
Fork L1
S2;
S4;
Fork L2



Write
Read
Concurent



(3) Parbegin and parend statment (structer)



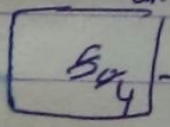
Bussy waiting

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

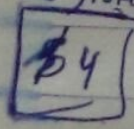
parrallel ~ ie

producer and Consumer concruntly

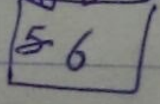
Counter



Register 2

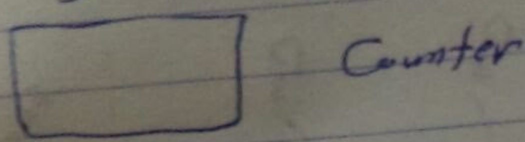


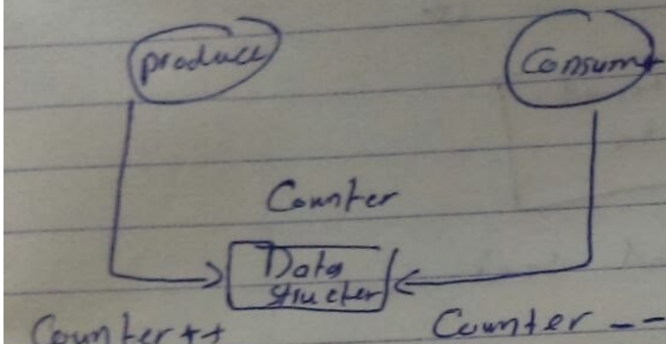
Register 1



Concrunt ie
Counter

shared data

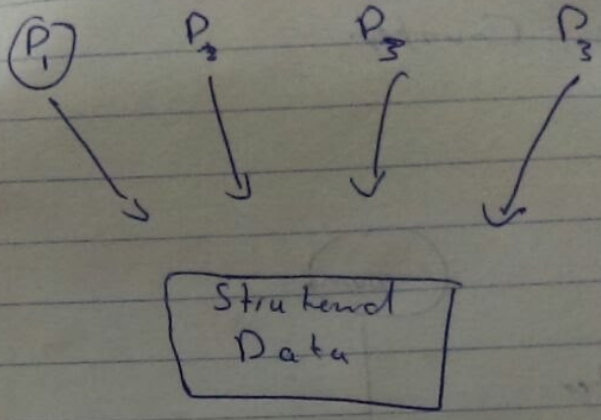




بالعربي

Progress: في بروسس بدها تدخل تشتغل

"إذا الكريتكال سيكشن فاضية دا بروسس



critical section

entry section :- →
Reminder section :-

turn

i j i

Pi Pj

reminder Section

Remian section

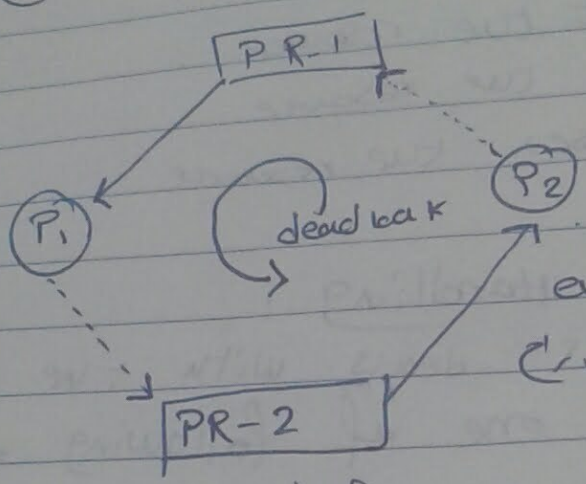
دوري دورك

Reding Bakery

Ch. 7

* New Lecture:
Deadlock

○ Resource Allocation Graph



هذا الرسم يمثل deadlock ولا Process شغال بستنى event عشان تخلص وتشارك بالمورد.

* System model:

- There are n resource types

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

Tape CPU Printer

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

R_2 = Printer

$w_2 = 5$

- R_0 , R_1 , R_2 , ... (2, 1, 5)

* each Process usees the resource in the following order

1- it request the resource .
2- it usese the resource .
3- it releases the resource .

Dead_lock Handling :

(1) The OS deals with the deadlock state in one of following ~~of~~ 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 extreme measures such as, preempt يسلب some of the recources from the process

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

* condition for a deadlock to occure (simultansly):
4 condition must hold for deadlock to ocur: يجب تحقق الاربع شروط لكي يحدث deadlock

(1) mutual Exclusion: the Resource type must be used exclusivly, 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 relese its recurce

(3) No Preemption.

(4) Circular wait (cycle)
Assuming we have n proces, $P_0, P_1, \ldots, P_{n-1}$
P_0 is waiting for Proces P_1 to release its resource.
P_1 is waiting for Process P_2 to to relase its resource.
:
:
i is waiting for Proce P_{i+1} to release its resource. cycle

$P_0 \leftarrow P_1 \leftarrow P_2 \cdots \rightarrow P_{n-1}$

* Resource Allocation Graph :

Graph

$G = (V, E)$

V : set of vertices

E : set of Edges

* In our model,

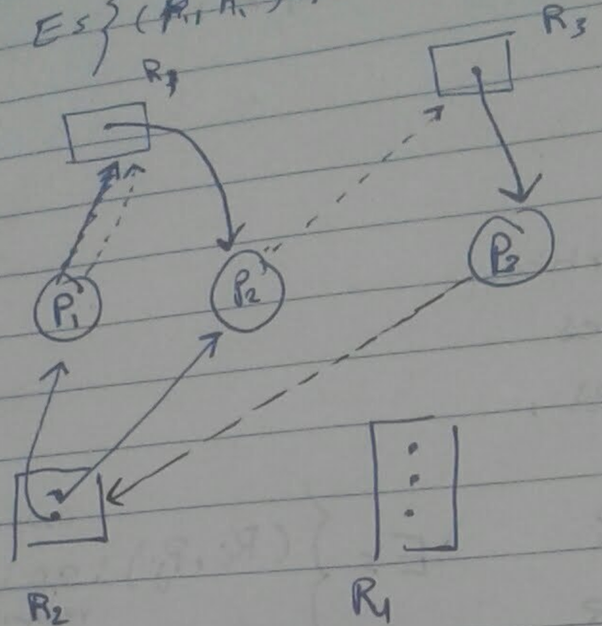
$V = \begin{cases} \text{Process} , P_i \\ \text{Resource} , R_j \end{cases}$

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

Ex: $P = \{P_1, P_2, P_3\}$

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

E = {(P_1, R_1), (P_2, R_3), (R_1, P_2), (R_2, P_1), (R_3, P_3)}



Is there a dead lock?

* طالمافيش cycle تكون في deadlock

No cycle ⟹ 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 begining
- All the resource are granted to the process at one time.

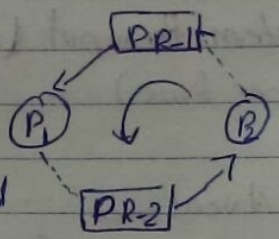
problem: Starvation..

OS

* 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 devise
the Os can't break this condition

(2) hold-and-wait
- Let process demands all its resourses at the begining



- All the resources are granted to the process at one time
Problem : Starvation
Process resource أخذ كل شيء

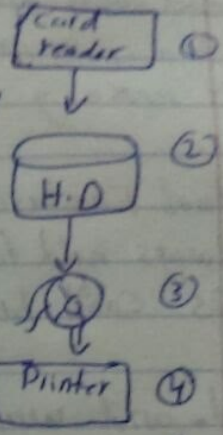
(3) No preemption
when a process request another instances of the resource which is not available, let the prossess relase its reasource wich holds

Problem
Starvatio

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

Conclusion :-

Dead lock: prevention trys to make sure that one of the 4 nessary condition do not hold in order to prevent a dead lock form happening.



a side affect (disadvange) of Deadlock prevention - low device utilization
- reduce system throughput (number of process finish execution peer time)

Dead lock Avoidance :-

Def: a system is in a safe state if the sequence of processes $< P_0, P_1, P_2, \dots, P_{n-1} >$ such that :
P_0 can take the available resource, runs and finish execution.
P_1 can take the avalible resource + the resources released by P_0, run and finish execution

P_2 = = = = = = =
= P_0, P_1 :

P_i = = = =
= - - -

P_{n-1} = = P_0, P_1, 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 tapedrive, 3 processer 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

$< P_1, P_0, P_2 >$ No dead lock

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

Is the system safe $< P_1, ?$

2 0

No safe sequence

Dead lock

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

① for sumblist purbases we will consider only one reasorse type, but this can be generalized easly to more them one resours type.

② each proses must declare in advance the maximum instense (units) of the resoure) it need

③ When a process request a resourse, it may have to wait

④ When a proces gets all it resourse, it must reles (return) them in afinite time

Data structer used in Algorithm :-

1- Array MAX.
MAX[i] = k, process P_i, requests at must 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 Pi currently need (requries) to n instan of the resource
4 Array Available = W/ 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 an 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

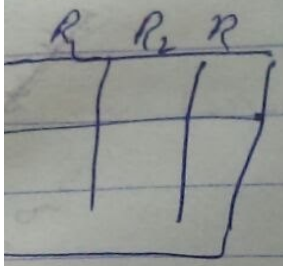
Process	max	Allocation	NEEDS
P_0	10	5	5
P_1	4	2	2
P_2	9	2	7

W
3 5 10 12

<P_1, P_0, P_2>

i	K
0	1
1	1
2	1

Max [i, j] = k, means P_i request at most k instance of resource R_j

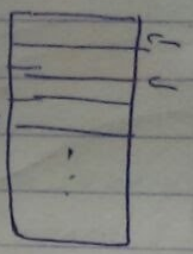


Chapter 8

memory Management (Ordinary mm)

* memory is an array of word each is addressable

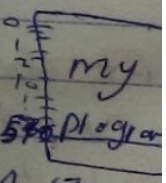
- In chapter 8, we will Consider that all program, must brought into memory for excution



word = 4 byte

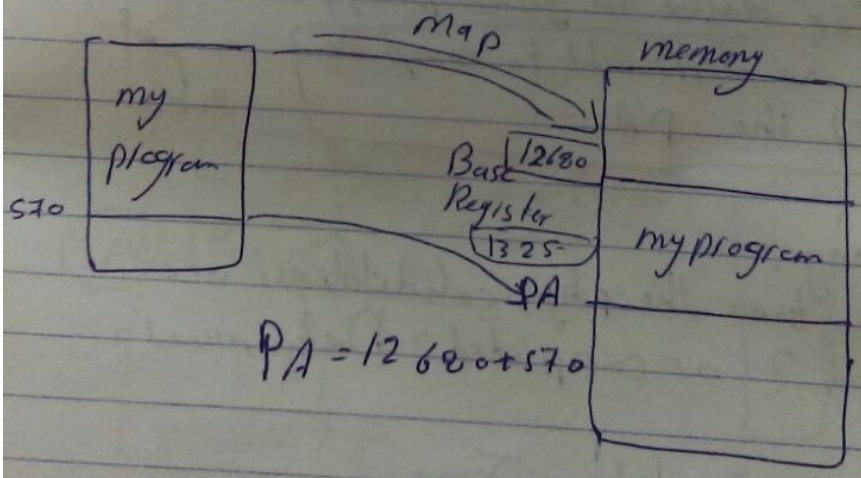
* Logical Vs Physical Address :- 2 by

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



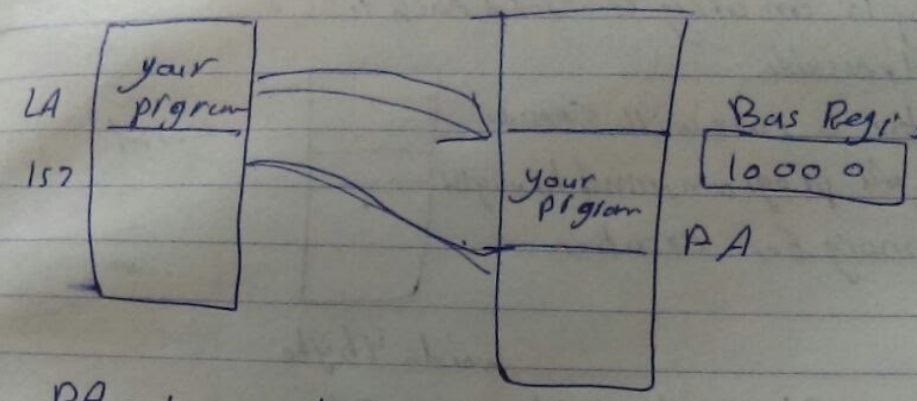
LA = 570

Logical Program



PA = 12680 + 570

Logical Address Vs physical Address



PA = 1000 + 152
PA = Bas Reg + LA

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

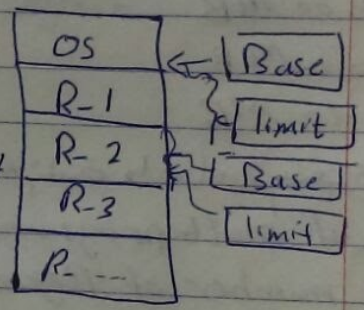
PA Binding :-
At what time the physical Address are assigned ? or computed ? Determined ?

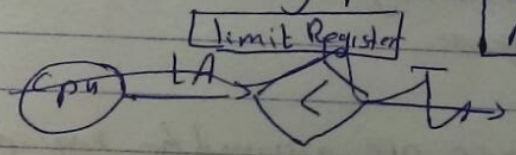
(1) At Compilation Time
(2) At loading Time
The disadvantage. The programe can't be moved during execution

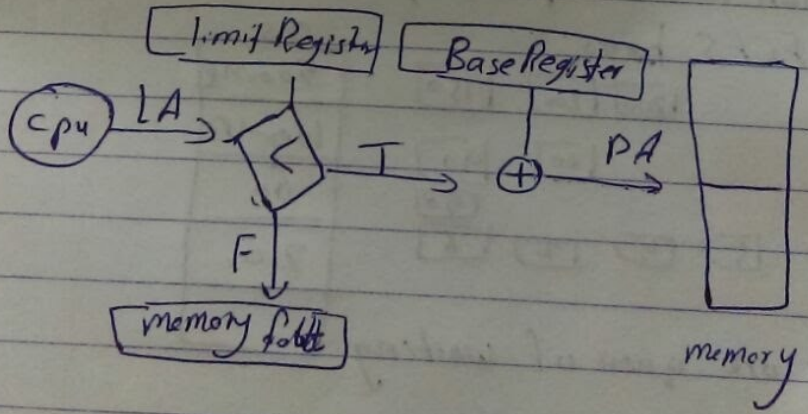
(3) At execution Time
The PA is determened or computed at execution. The program can be moved during execution.

[1] Multiple partions
- Memory is divided into partions (regins)
- Each regen can hold only one process (programe)

- Hw Support needed to implement this technige we need Base and limit Register for every process







Two versions available:-

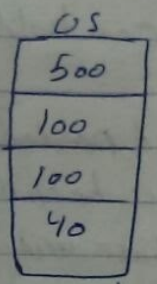
A) fixed Regions: In IBM-OS: MFT → Task Multiprograming with ← a fixed number of

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

ex: 4 Region:-

1*500 MB
2*100 MB
1*40 MB

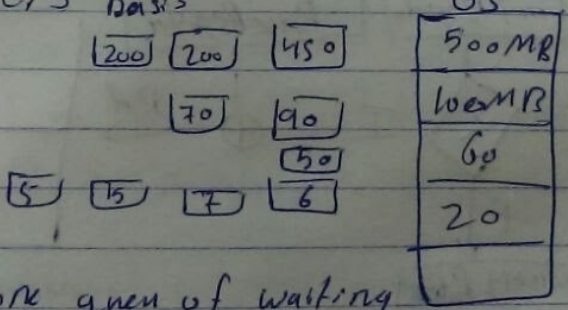
عدد ثابت و احجام متغيرة



- Degree of multiprogramming is bounded by the number of Regions

Job sheduling: how procas are assigned to regions?

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



(2) only one queue of waiting Jobs

4/ 5

(a) FCFS with or without Skip
(b) Best fit only
(c) Best available Fit

Problems :- Major Problem
Internal fragmentation: its the unused fre
space in the assigned region
- IBM-360-OS
[B] Variable (Dynamic) Rgion

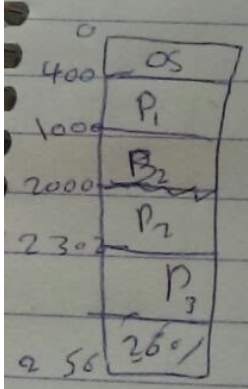
Fixed Regos: Internal fragmentation

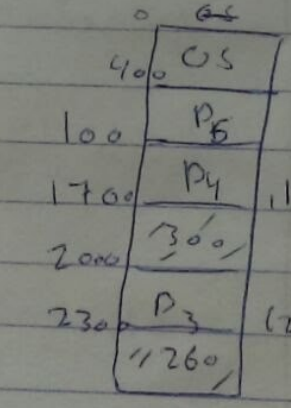
[B] Dynamic (Variable) Regions:
IBM, MVT

Multiplying with Variable number of Tasks

Ex- Assume memory 2260 KB
OS - 400 K
Given the following Queue of waiting jobs

Process	Memory N	Time Required
P_1	600K	10
P_2	1000K	5
P_3	300K	20
P_4	700K	8
P_5	500K	15





After a while memory will be constructed

(1) Allocated Regions for Process

(2) Holes external fragm

Alt

T =

T50

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

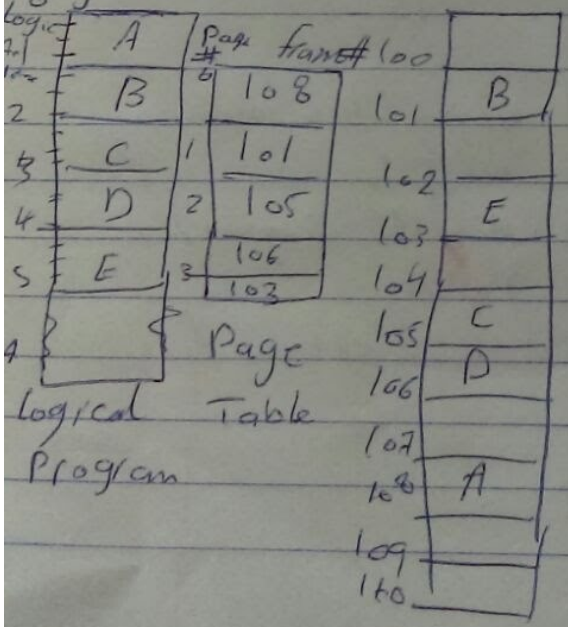
1. first fit
2. Best fit
3. worst fit

> 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
we be

[2] Non Contingnous Memory Manyment (Poging)

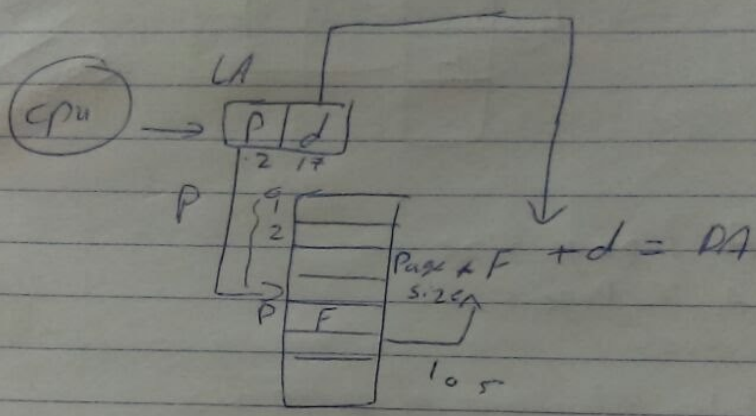
- Divid the logical program into equal site partions calle "pages"
- Divid memory into partion same size called "f



Base limit Regs.

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

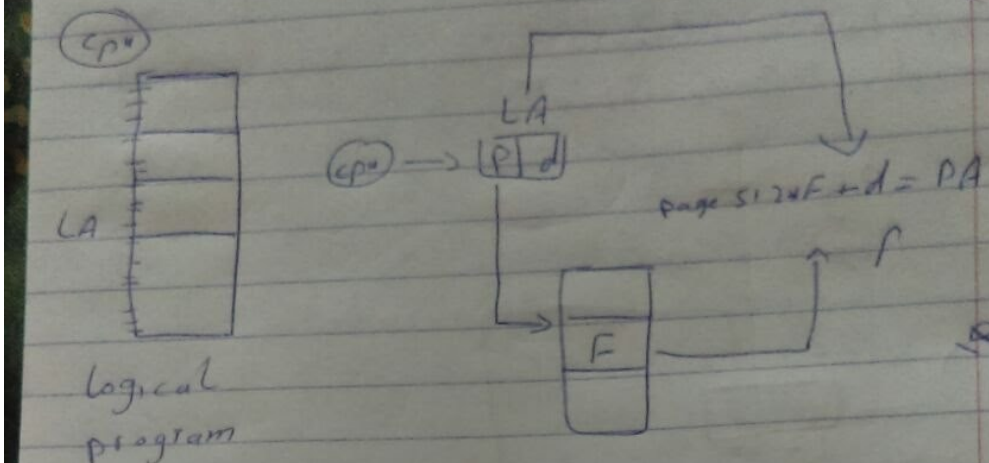
P = Page #
d = Page offset



Page Table

P = LA / Page size
d = LA % Page size
Assume: Page size = 100
Given LA = 217
P = 217 / 100 = 2
d = 217 % 100 = 17

PA = 100 * 105 + 17 = 10517



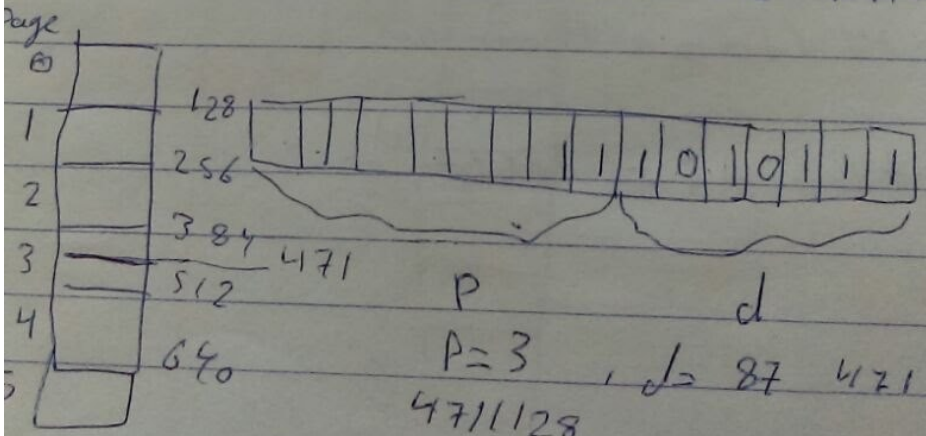
P = LA / page size
d = LA % Page size

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

to
$2 \leq$ page size $\leq 2^{13}$
1024 — 8192

Assume Page size = 128 = 2^7

LA = 471 Assume LA = 16 Bits long



The OS take the n Low bits of the logical addres = d

The remaing Bits = P

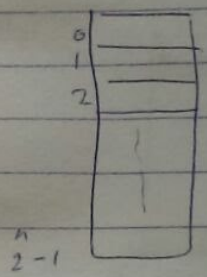
ex Dec-10-OS:- LA = 20 Bits

page Size = ~~2048~~ 512 = 2^9

20 Bits

11 Bits	9 Bits
P	d

This Mean, the max program Can be executed in this system is :-



$\Rightarrow$ max program Contains 2^{11} pages

$2^{11} * 2^9 = 2^{20}$ bytes

IBM = 370 OS:

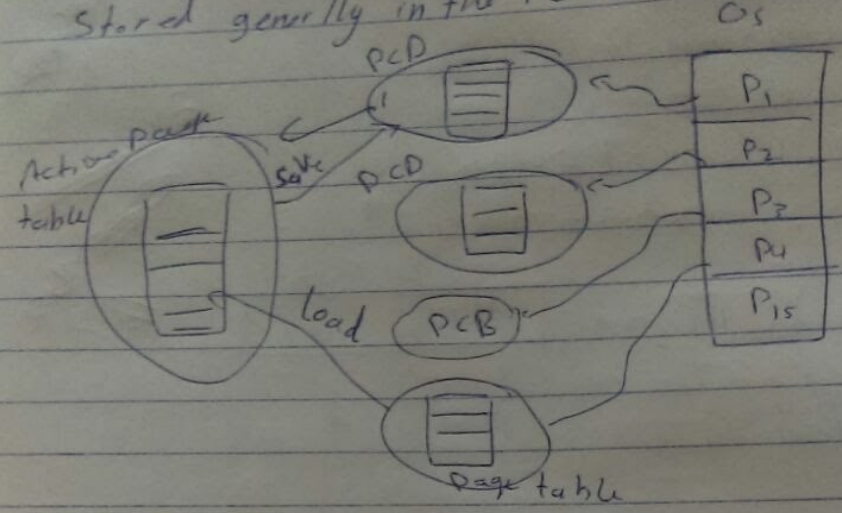
LA = 24 Bits

page Size = 2048 = 2^{11}

13 Bit	11 Bit
P	d

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

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



Note: In every instuction, the cpu accesd 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 accesing the active page Table Therefore, the active should be stored in a high speed storage medium
- Where to store the page Table?

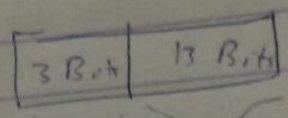
[1] Registers

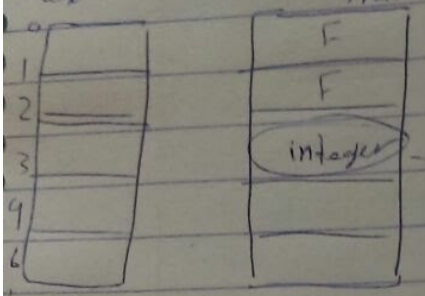
The OS dedicated a number of regis to store the active page table

Ex: In PDP-11, OS, LA = 16 Bits

Page size = 2^{13}

This mean the max program Contains 2^3 = 8 pages



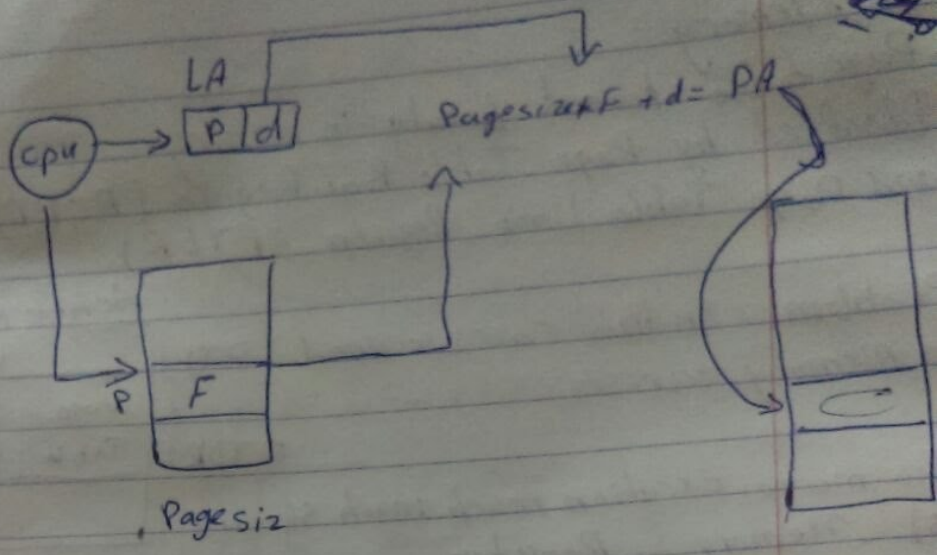


∴ Page table must contain

Assume page table entry = 4 bytes

Page table size

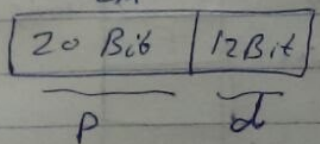
= 8 * 4 = 32 bytes

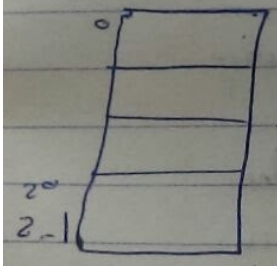


Page siz

ex LA = 32 Bits

Page size = 4096 = 2^{12}

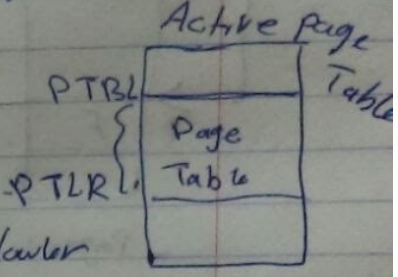




logical Program

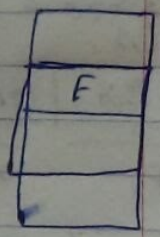
$\Rightarrow$ Page Table size = $2^{20} * 4 = 4$ MB

(2) Memory
- Page table stored totaly in memory
- Identified by Page Table Bar Register (PTRR) and Page Table Limit Register (PTLR)

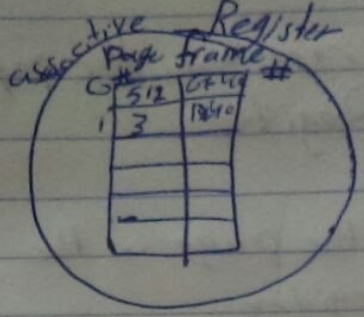


- Problem : In this case we need two memory accesses
- This makes excution much much slowler

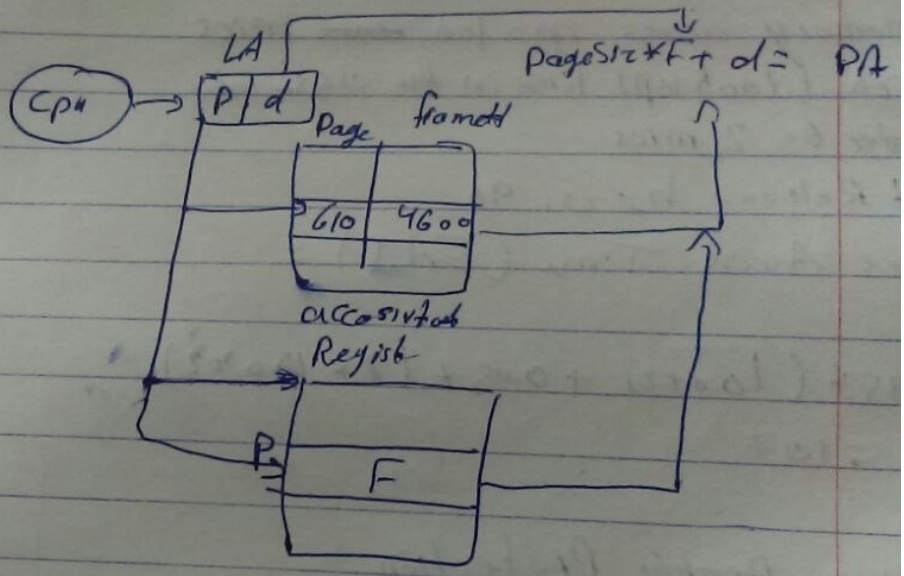
(3) Memory + Register
- The main page Table is stored in memory
- A few Register called "ASSocitive" "Register" are assigned to the active Pages tables, called some time Translation Look-aside Buffer (TLB) which contain the active Pages durring execution



- Initally, the cpu checks (examines) if the page in which excution will take place in is the assocition Register
If so, Continue excution
else if gets the page and insert it in the associtive and Continous excution.



few Register which Contins a small number of page Table entries



Page Table memory

Performance: depends in what calls th hit Ration

of hit Ration: is the probability that the page is in the associative Register

$0 \leq h \leq 1$

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*(m+t) + (1-h)*(2m+t)

memory access m = 100 ~~Mics~~ mics

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

Hit Ration. h = 0.95

Effective Access Time (EAT) =

0.95 * (100+2) + 0.05 * (2*100+2)

= 107

Memory ~~Probt~~ Protection :-

- a Bit is added to the page table