OS

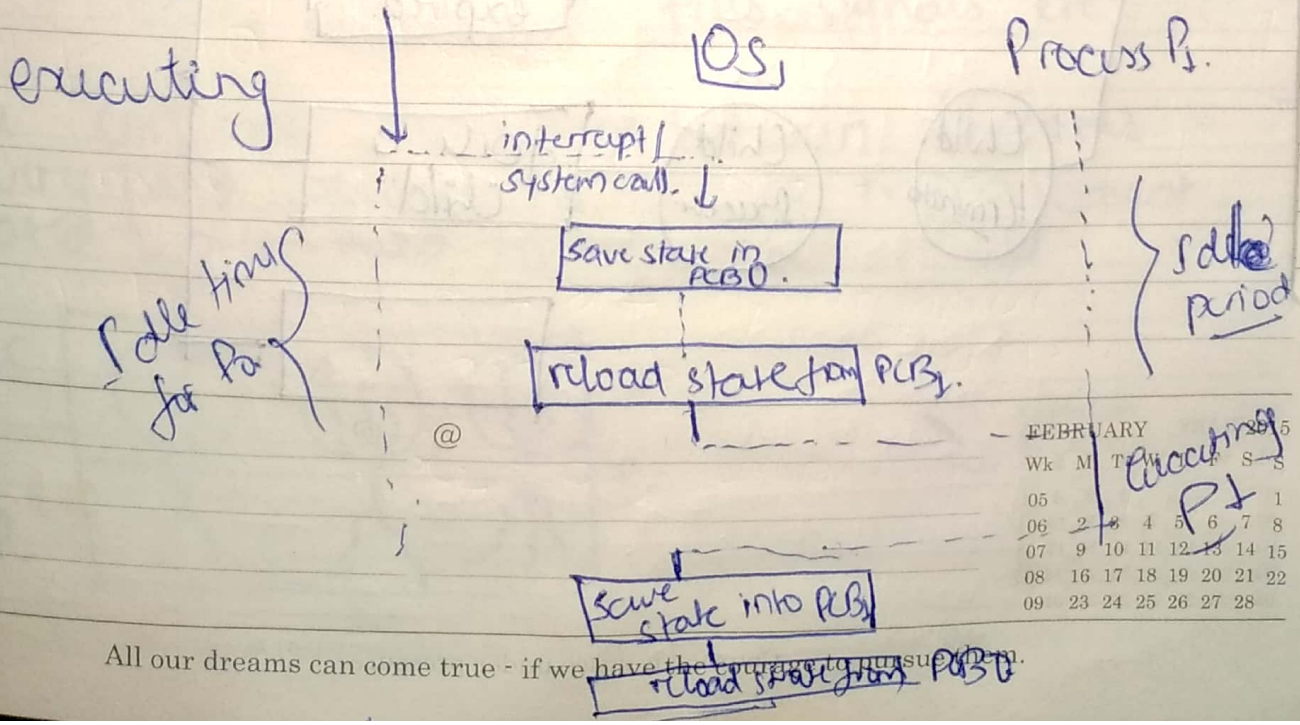
Process

A process in UNIX is created by the fork system call. Every process except process 0 is created when another process executes the fork system call. Process 0 is a special process that is created when the system boots. After forking another process (Process 1) process 0 becomes the swapper process.

Process 1 known as "init" is the ~~another~~ ancestor of every other process in the system.

```
pid = fork();
```

Context Switching

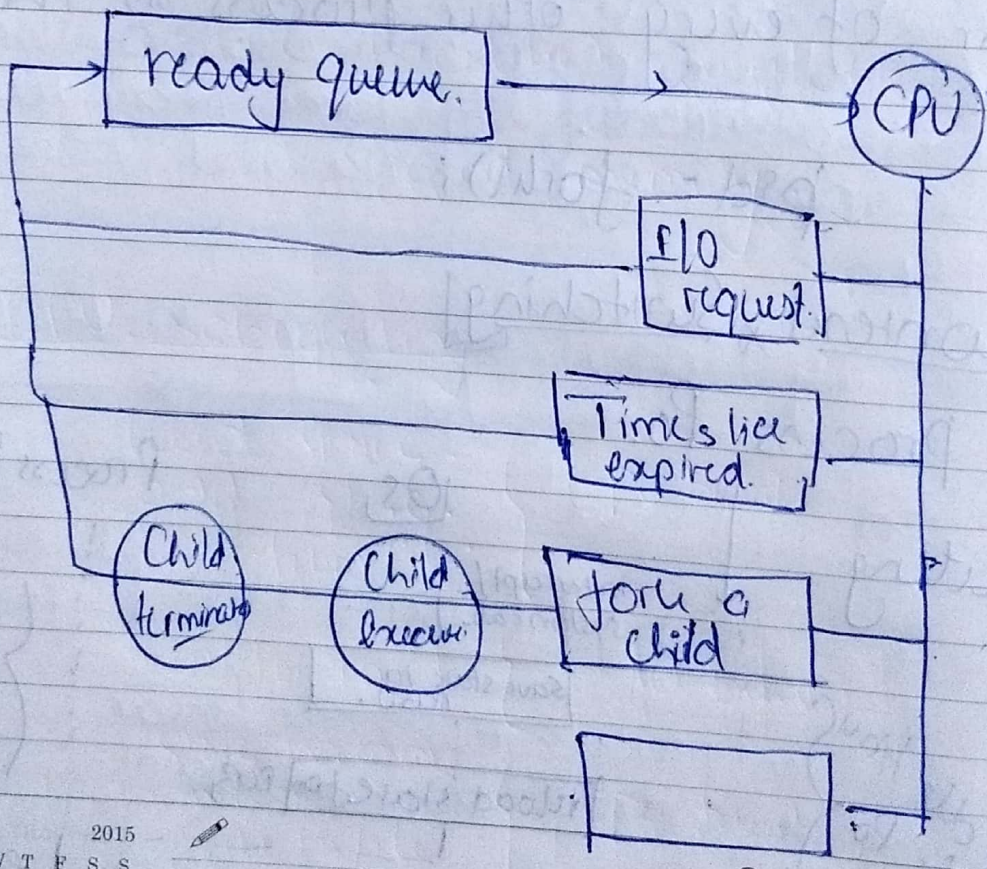


→ More context switching leads to higher idle time for processor

→ As an operating system designer your task should be to minimize this idle time.

Process Scheduling

Objective of multiprogramming is to maximize CPU utilisation.



Scheduler → Short term scheduler (CPU scheduler).
→ long term scheduler - (Job scheduler).

Controls the degree of multiprogramming

Process

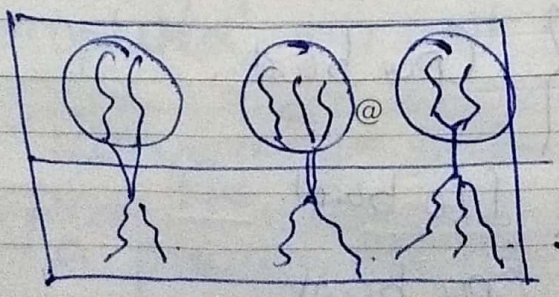
It is defined by the resources it uses and by the location at which it executes.

Threads.

A thread is a basic unit of CPU utilisation, and consists of a program counter, a register set and a stack space.

It shares with its peer threads (from the same process) its code section, data section & operating system resources, like open files, signals etc.

The context switching between threads is very fast because no memory management related work need to be done.



FEBRUARY 2015

Wk	M	T	W	T	F	S	S
05							1
06	2	3	4	5	6	7	8
07	9	10	11	12	13	14	15
08	16	17	18	19	20	21	22
09	23	24	25	26	27	28	

1st Assignment

→ Term paper on kernel and threads in different OS.

→

Inter x Process x Communication

- Information sharing
- Computation speed up
- Madularity
- Convenience.

IPC facilities available.

(i) shared memory.

CPU Scheduling

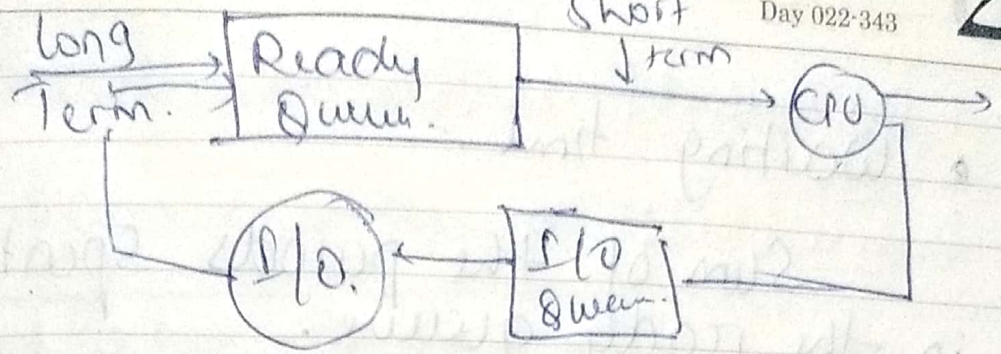
load store
add store
read from file
} CPU burst.

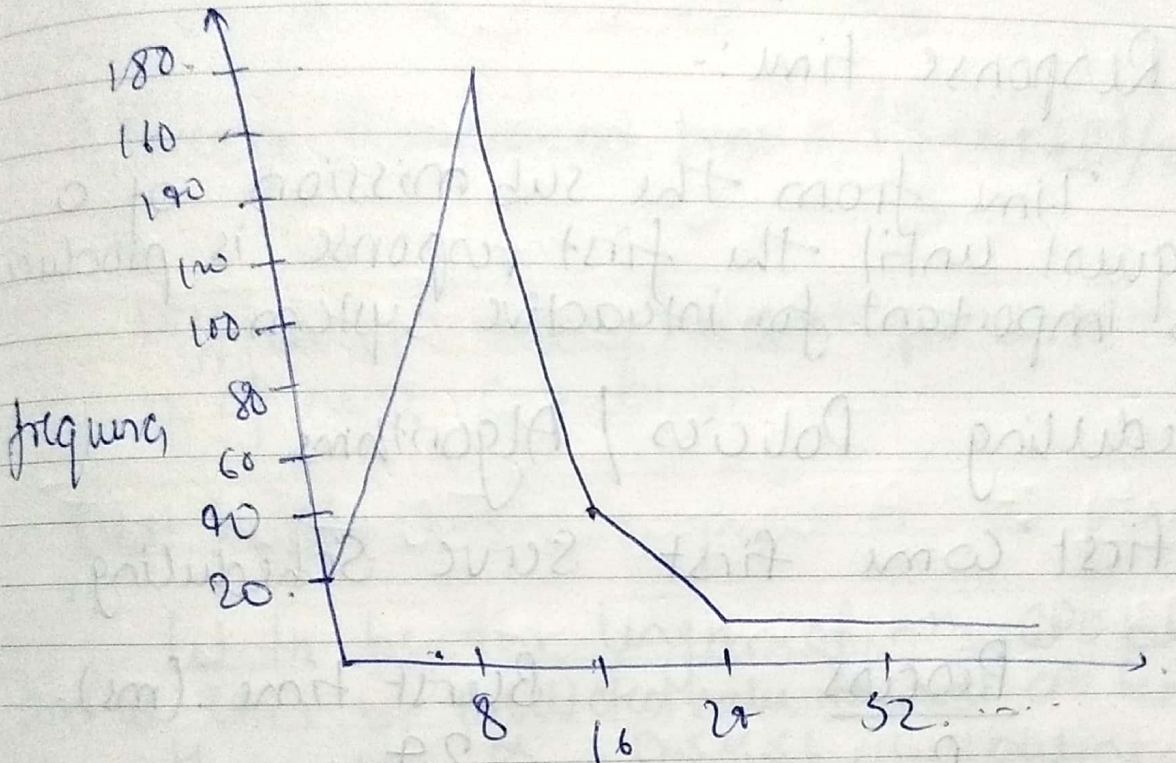
wait for IO
} I/O burst.

store increment index
write to file
} CPU burst.

wait for I/O
} I/O burst.

load store
add store
} CPU burst.





Scheduling criteria.

- CPU utilization :- to keep the CPU busy as much as possible.
- Throughput : The no. of processes completed per unit time.
- Turnaround time - the interval from the time of submission of a process to the time of completion.

- Waiting time :-

Sum of the periods spent waiting in the ready queue.

- Response time :-

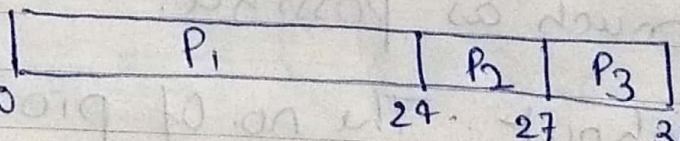
Time from the submission of a request until the first response is produced. Is important for interactive system.

* Scheduling Policies / Algorithms.

1. first come first serve Scheduling.

Process	Burst time (ms).
P_1	24
P_2	3
P_3	3.

Gantt chart.

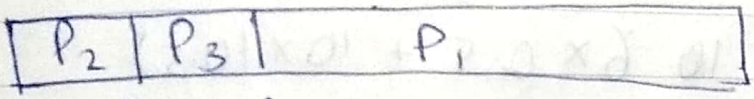


Performance is represented by average turnaround time :

$$= \frac{24+27+30}{3} = \frac{81}{3} = 27.$$

2. Shortest Job first scheduling.

Same example as 1:



Average turnaround time = (3+6+30)/3
= 13.

Difficulty: We need to predict the length of next cpu burst.

<u>One Single way</u>.

Let t_n be the length of n^{th} CPU burst, and let τ_{n+1} be predicted value for next CPU burst. then, for α, $0 \le \alpha \le 1$, we define

$$\tau_{n+1} = \alpha t_n + (1-\alpha)\tau_n$$

// τ_n is the guess of the CPU burst at n^{th} slot.

if $\alpha = 0$, then $\tau_{n+1} = \tau_n$ i.e. the recent history has no effect on prediction.

if $\alpha = 1$, then $\tau_{n+1} = t_n$. i.e only the most recent CPU burst information, history has no effect.

if $\alpha = 0.5 | 0.7 | 0.3$ for mix of both the history and current history

Actual CPU burst(ti): 6ms, 4ms, 6ms, 4, 13
$\alpha = 0.5$

Guess(zi): 10ms, 8ms, 6ms, 6, 5,

$z_{i+1} = 10\ 6 \times 0.5 + 10 \times 10.5)$
$= 3 + 5 = 8.$

$z_{i+2} = 4 \times 0.5 + 8 \times 0.5 ms$
$= 6$

$z_{i+3} = 6 \times 0.5 + 4 \times 0.5 = 5.$

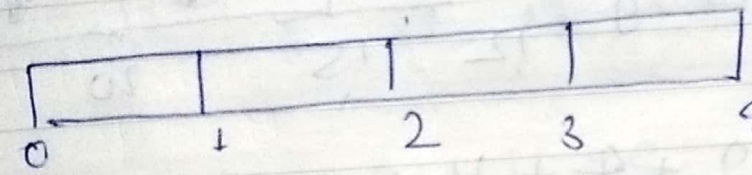
Priority :- A computer 7094 at MIT in 1973 they found that at task of 1967 was still unexecuted.

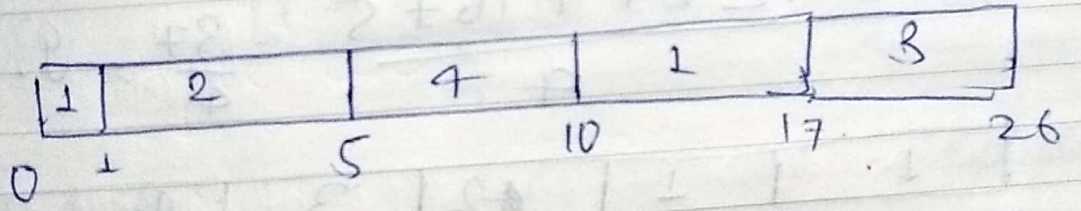
Aging :- To gradually increase the priority of jobs that wait in a system for long time.

Preemptive Algorithms.

SJF (Shortest Job first scheduling).

Job.	Arrival time	Burst time.
1	0	8
2	1.	4
3	2.	9.
4	3.	5.





Average turn around time:

$$= \frac{(17-0)+(5-1)+(26-2)+(10-3)}{4}$$

$$= \frac{17+4+24+7}{4} = \frac{52}{4} = 13.$$

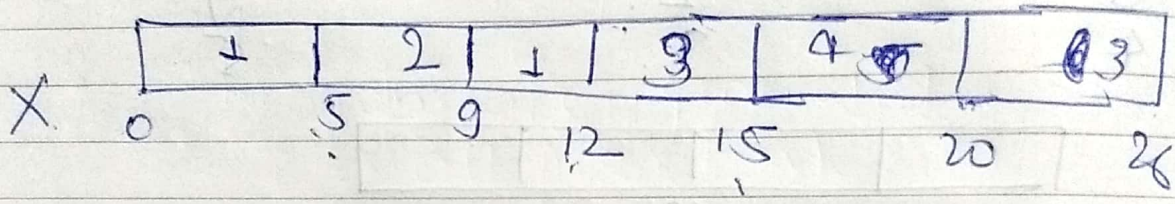
Non preemption.

$$\frac{(8-0)+11+14+24}{4} = \frac{57}{4} = 14.25$$

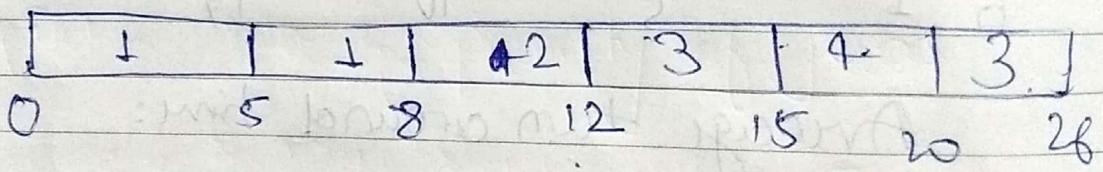
@

Job.	Arrival time.	BT	Priority
1	0	8	3rd
2	5	4	3rd
3	10	9	1st
4	15	5	2nd.

Preemptive :-

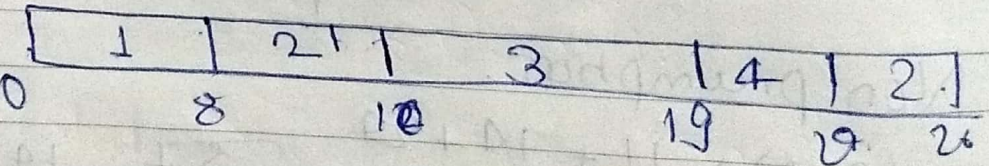


$$\frac{12+4+16+5}{4} = \frac{37}{4} = 9.25$$



$$\frac{8+7+16+5}{4} = \frac{36}{4} = 9.$$

ATT =



$$\frac{8+16+4+9+9}{3} = \frac{47}{3} = 16.$$

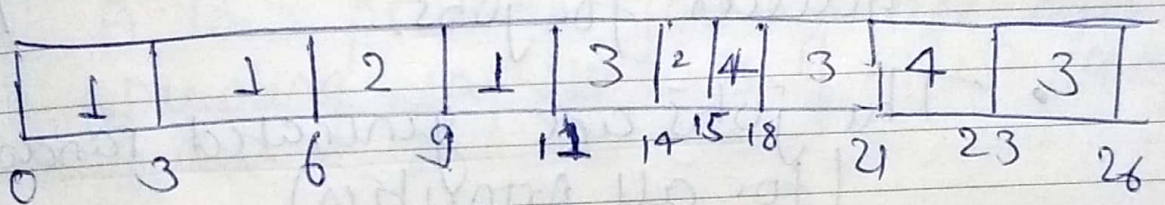
@ 11.75

JANUARY 2015

Wk	M	T	W	T	F	S	S
01				1	2	3	4
02	5	6	7	8	9	10	11
03	12	13	14	15	16	17	18
04	19	20	21	22	23	24	25
05	26	27	28	29	30	31	

Round Robin (There should be a time quantum for every process).

Job	Arrival time	Burst time
1	0	8
2	5	4
3	10	9
4	15	5



$$ATT = \frac{11 + 4 + 16 + 8}{4} = \frac{39}{4} = 9.75$$

Priority

1st → System process/Job.
2nd → Interactive Process/Job.
3rd → Batch processing.

→ Ready Queue →

Assignment-2 :-

Determine the length of the "Ready Queue" used in CPU scheduling of your computer system.

- Assume there are three(3) priority queues for jobs.
- The jobs are generated randomly (for all priorities).
- Define job sizes (say, 1, 2, 3, 4). CPU unit of time

Input: No. of priority of jobs
Ready Queue size :- 20/30

Output: 1. Total number of jobs generated after simulation time of 300 time units.

2. Total jobs served by CPU
3. No. of Jobs remaining in ready queue @

Different modes:

Mode 1

Whenever a higher priority job arrives in ready Queue low priority job in CPU has to be prempted. and high priority job is to be allocated.

(b). A low priority job will age after spending, say 10 units of time in ready Queue.

Mode 2

(a). Here ready Queue will act as first come first serve queue.

(b) The job transfer from the priority Queues to ready queues will as per following probability.

1st priority is 0.5
2nd priority is 0.3
3rd priority is 0.2.

@

Concurrency: Mutual exclusion and Synchronisation.

Management of processes and threads.

Multiprogramming :- The management of multiple processes within a ~~multiple~~ uni-processor system.

Multiprocessing :- The management of multiple processes within a multi processor system.

Distributed processing :- The management of multiple processes executing on multiple distributed computer system.

* PARAM is based on cluster computing

Concurrency is the fundamental aspect of all three abovementioned method.

Design issues :-

- Communication among processes
- Sharing of and competing for resources
- Synchronisation of the activities of multiple processes.
- Allocation of processor time to processes

@

Principle of Concurrency :-

In a single-processor multiprogramming system, processes are interleaved in time to yield the appearance of simultaneous execution.

Eg.: If two processes use same global variable and both perform read & write on the variable 'a'.

To manage the allocation of resources in a system optimality.

PROCESS SYNCHRONIZATION

The critical section problem (CS Problem).

- Consider a system consisting of n processes
  $\{P_0, P_1, P_2, \ldots P_{n-1}\}$
- Each process has a segment of code, called a critical section, in which the process may be changing common variables, update a table, writing a file and so on.
- When one process is executing in its critical section, no other process is to be allowed to execute in its critical section.

General structure of a typical process p_i.

repeat

Entry section

Critical section

Exit section

remainder section.

until false;

Two process Solution (P_i & P_j)

Algo 1.

// a common integer variable called "turn" initialized to either i or j.

Structure of process p_i

repeat

while turn ≠ i do no op;

It requires strict alternation.

Critical section.

turn = j;

remainder section

until false;

Issue: If process j has wide gap between two critical section than process i then i will have to wait for j^th critical section to arrive.

i	j
⇒ csi1	⇒ csj1
⇒ csi2	⇒
⇒ csi3	⇒ csj2.

Algorithm 2:- var flag : array[0 .. 1] of boolean

Structure of process i

repeat.

```
flag[i] = true
while flag[j] do no.op
```

Critical section

```
flag[i] = false;
```

Remainder section

until false;

Algorithm 3: Structure of process P_i

```
repeat
    flag[i] = true
    turn = j
    while (flag[j] and turn == j) do no op;

    Critical section;

    flag[i] = false.

    Remainder section

Until false;
```

05 February
THURSDAY
Day 036-329

2015
Week-06

08
09
10
11
12
01
02
03
04
05
06
07

FEBRUARY 2015

Wk	M	T	W	T	F	S	S
05							1
06	2	3	4	5	6	7	8
07	9	10	11	12	13	14	15
08	16	17	18	19	20	21	22
09	23	24	25	26	27	28	

@

Success is never wondering wh

Semaphore:

Usage: we can use semaphore to deal with the n-process critical section problem

"mutex" initialize to 1

Structure of process P_i is organized as

repeat
wait(mutex);
Critical section.
P_x
Signal(mutex)
remainder section.
until false;

→ Counting semaphore implementation with binary semaphore.

Let S be a counting semaphore. To implement it in terms of binary semaphores we need the following data structure;-

Var S1: binary semaphore.
S2: Binary semaphore.
S3: binary semaphore.
C: integer;

Initially $S_1 = S_3 = 1$, $S_2 = 0$, and the value of integer c is set to the initial value of the counting semaphore S (No. of processes you want to allow simultaneously in CS).

Entry section → wait operation:-

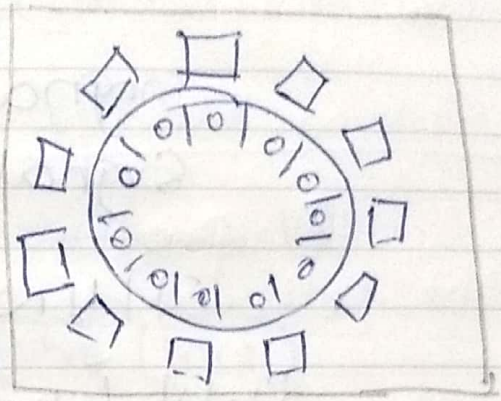
```
wait(S3);
wait(S1);
c = c-1;

if(c<0)
    then begin
    signal(S1).
    wait(S2).
    end. end
else. signal(S3)
signal(S3).
```

Exit section → signal operation.

```
wait(S1);
c = c+1.
if c<=0 then signal(S2).
signal(S1);
```

Classical Problem of Synchronization:

The Dining Philosophers problem.



Consider a no. of philosophers who spend their lives thinking and eating.

- When a philosopher thinks he/she does not interact with his colleagues.
- from time to time a philosopher gets hungry and tries to pick up the two chopsticks that are in his both sides of the plate.
- When one finishes eating, he puts down both of its chopsticks.

One Simple Solution: we represent each chopstick by a semaphore.

var chopstick : array [0..n-1] of semaphore.
all elements of chopsticks are initialized to 1.

Structure of philosopher i

repeat
wait (chopstick [i]);

wait (Chopstick [i+1 mod n]).

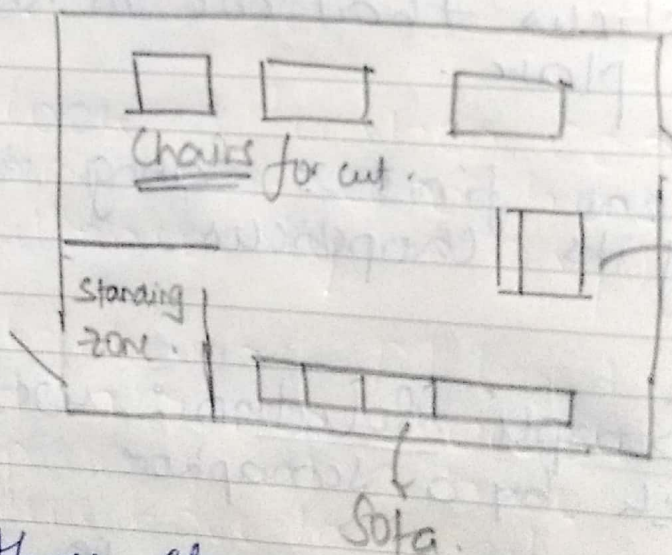
EAT

signal (Chopstick [i+1 mod n]).
signal (Chopstick [i]).

THINK.

Until false.

BARBER × SHOP × PROBLEM



→ Shop has three chairs and three barbers

→ A waiting area that can accommodate four customers on a Sofa.

→ Has a standing zone for additional customers.

→ Total no. of customers in the shop can be 20.

```
Semaphore max_capacity = 20;
Semaphore sofa = 4;
Semaphore barber_chair = 3;
Semaphore coord = 3

void customer().
{
wait (max_capacity)
wait(
enter_shop();
wait (sofa);
sit_on_sofa();
wait (barber_chair);
get_up_from_sofa();
signal (sofa);
sit_in_barber_chair();
signal (cust_ready);
wait (finished);
leave_barber_chair();
signal (leave_b_chair);
pay();
signal (payment);
wait (receipt).
exit_shop();
signal (max_capacity);
}
```

```
Void barber().
{
while (true).
{
wait (cust_ready);
wait (coord);
cut_hair();
signal (coord);
signal (finished);
wait (leave_b_chair);
signal (barber_chair)
}.
}
```

```
Void Cashier().
{
while (true).
{
wait (payment);
wait (coord);
accept_pay();
signal (coord);
signal (receipt);
}
```

DEADLOCKS

Several processes may compete for a finite no. of resources.

System Model

A process may utilise a resource in the following sequence.

1. Request.
2. Use
3. Release.

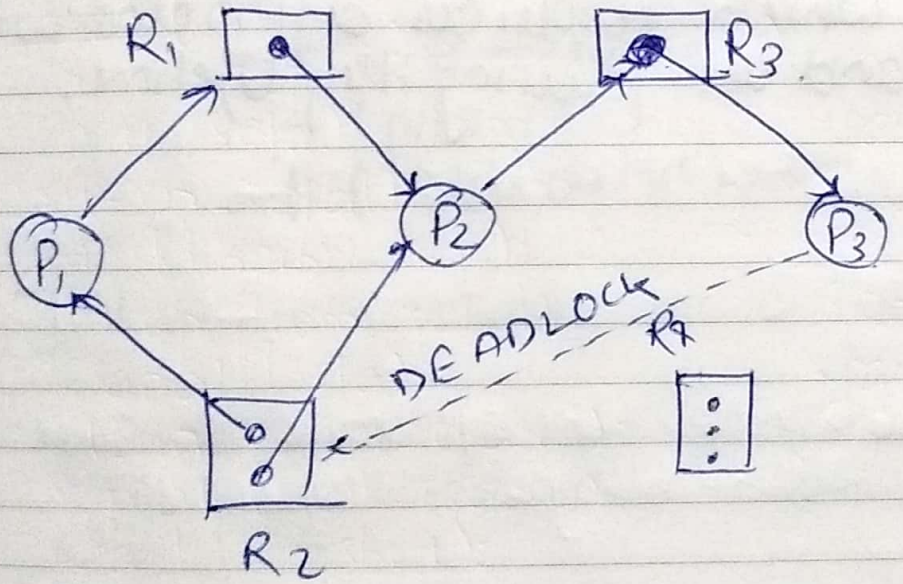
A deadlock situation can arise if all the following four conditions hold ~~system~~ simultaneously in a system;-

1. Mutual Exclusion:- At least one resource must be held in a non-sharable mode; that is, only one process at a time can use the resource. Others have to wait.

2. Hold and Wait :- A process must be there which is holding at least one resource and waiting to acquire additional resources that are currently being held by other processes.

3. No Preemption:- Resources cannot be pre-empted
4. Circular wait:- There must exist a set of waiting processes $\{P_0, P_1 \cdots P_n\}$ such that P_0 is waiting for a resource held by P_1, P_1 is waiting for P_2 ---- P_{n-1} for P_n and P_n for P_0.

Resource Allocation Graph.



$P = \{P_1, P_2, P_3\}$ $R = \{R_1, R_2, R_3, R_4\}$.

$E_2 \{ P_1 \rightarrow R_1, P_2 \rightarrow R_3$

$\}$

13 February
FRIDAY
Day 044-321

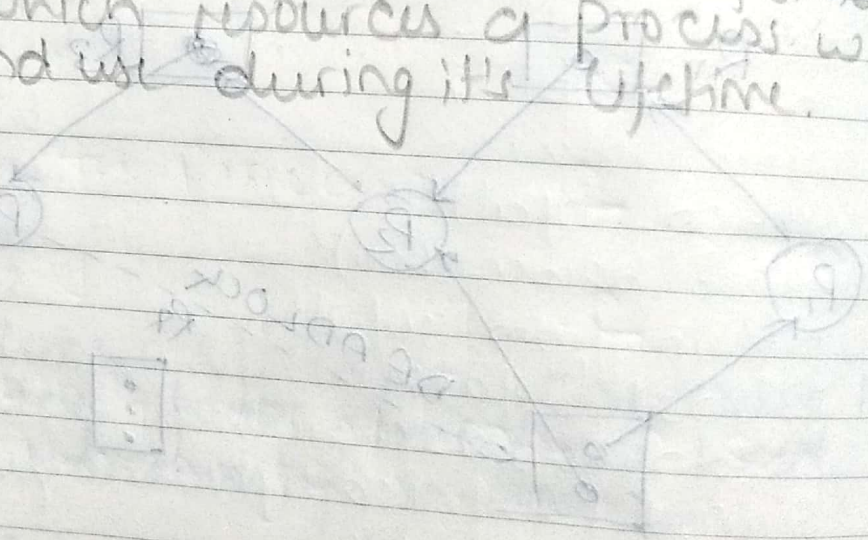
2015
Week-07

Methods of Handling Deadlocks.

Deadlock Prevention: by ensuring that at least one of the recovery conditions cannot hold

Deadlock Avoidance

Requires that the operating system be given in advance, additional information concerning which resources a process will request and use during its lifetime.



FEBRUARY 2015

Wk	M	T	W	T	F	S	S
05							1
06	2	3	4	5	6	7	8
07	9	10	11	12	13	14	15
08	16	17	18	19	20	21	22
09	23	24	25	26	27	28	

Deadlock Prevention

Circular wait :

Order the resource types and each process requests in an increasing order.

Let $R = \{R_1, R_2 \ldots R_m\}$ be the set of resource types we assign to resource type a unique integer number.

$f : R \rightarrow N$

$f(\text{printer}) = 1$.

$f(\text{Disk drive}) = 5$.

The

@

Deadlock Avoidance :

Safe state :- A state is safe if the system can allocate resource to each process (upto it's maximum) in some order and still avoid deadlock. A system is safe if there exists a safe sequence.

Ex :- We consider a system with 12 magnetic tapes drives and three processes

	Max. Need.	Current Need.
P_0	10	5
P_1	4	2
P_2	9	2/3 → safe state → deadlock

At time t_0, whether the system is in safe state or not. If safe, in what sequence?

February
MONDAY
Day 047-318

16

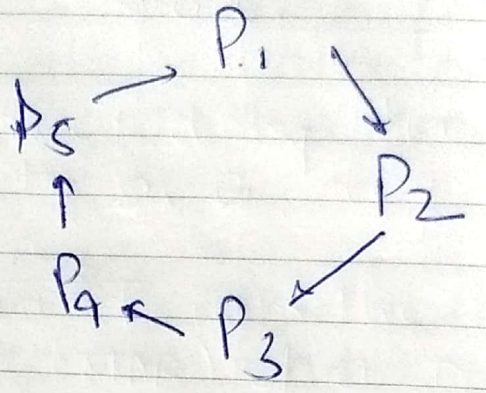
MARCH 2015

Wk	M	T	W	T	F	S	S
09	30	31					1
10	2	3	4	5	6	7	8
11	9	10	11	12	13	14	15
12	16	17	18	19	20	21	22
13	23	24	25	26	27	28	29

Today is the tomorrow we worried about yesterday.

Recovery from Deadlock.

Process Termination



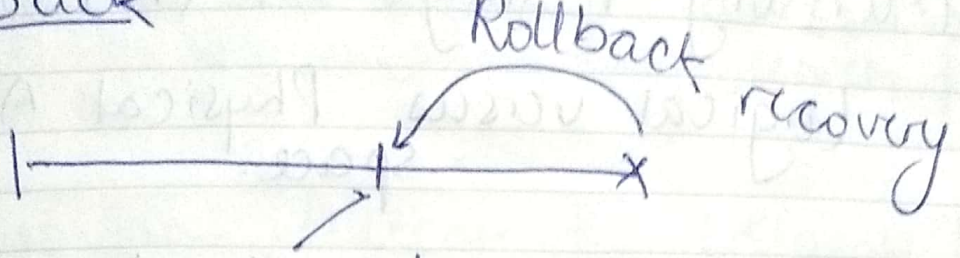
- Abort one process at a time

To choose a process:

1. What is the priority of the process?
2. How long the process has completed and how long is required?
3. How many and what types of resources the process has used?
4. How many more resource a process needs to complete its execution?
5. Whether the process is interactive or batch?

@

Rollback



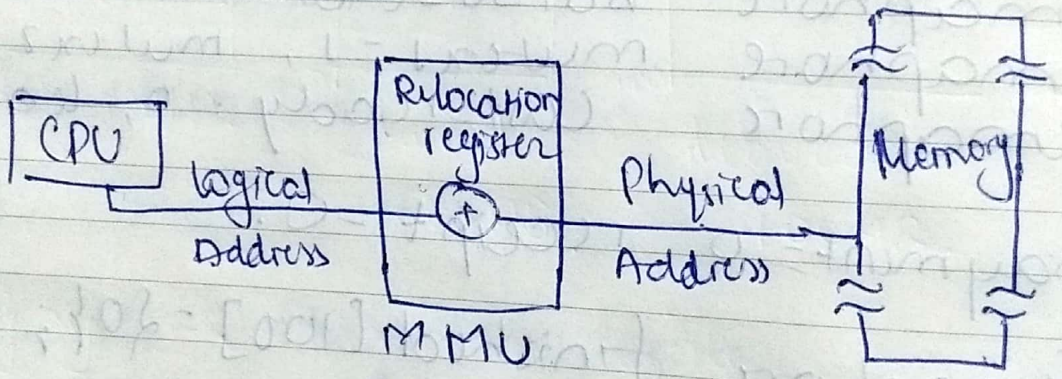
Barbershop Problem (Revisit)

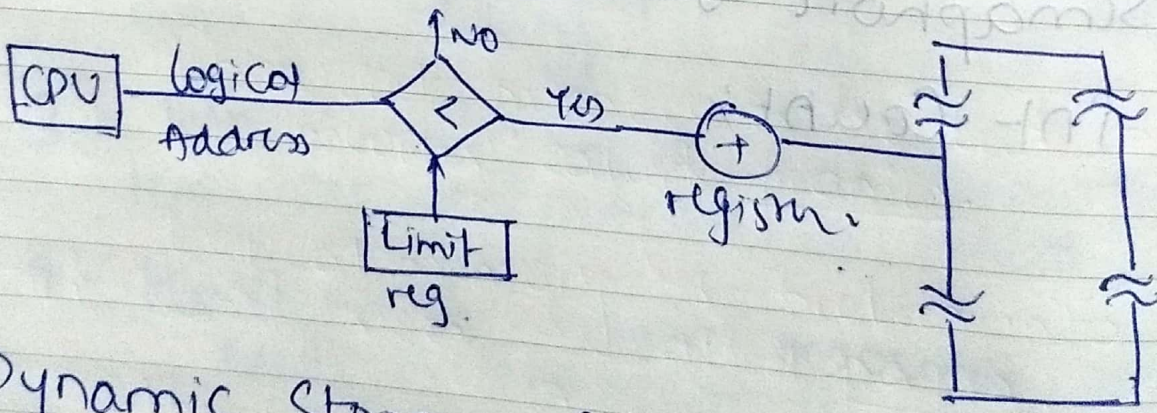
```
Semaphore max_capacity = 20;
Semaphore sofa = 4;
Semaphore barber_chair = 3, word = 3;
Semaphore mutex1 = 1, mutex2 = 1;
Semaphore cust_ready = 0, leave_b.chair
                       = 0,
payment = 0, receipt = 0.
Semaphore finished[100] = {0};
int Count;
```

* Memory Management (RAM).

Logical versus Physical Address space.

- An address generated by the CPU is commonly referred to as a logical address
- An address loaded into the "Memory Address Register" (MAR) of a memory is commonly referred to as a physical address.





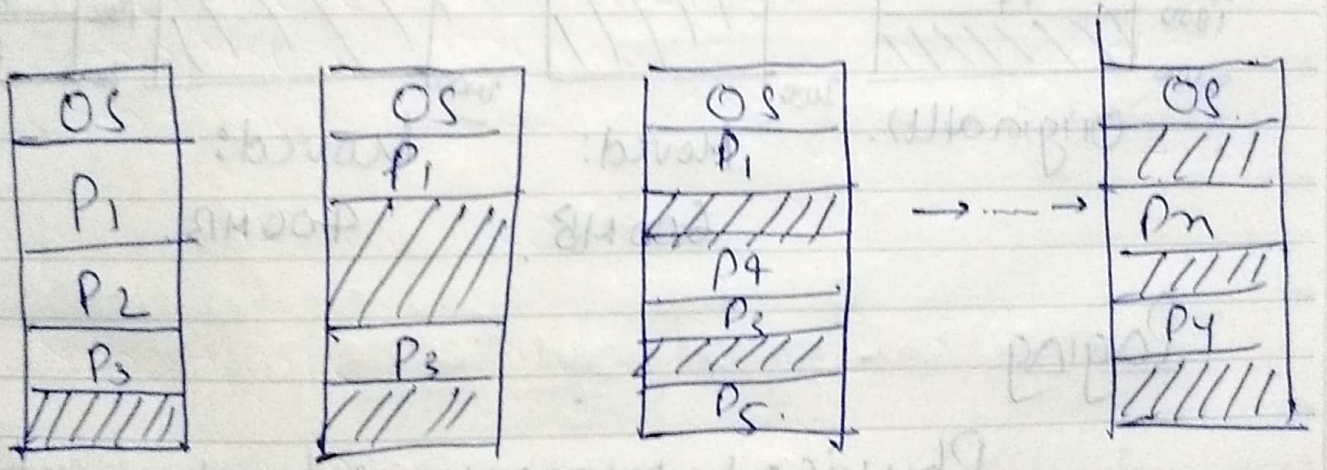
Dynamic Storage Scheme

- first - fit :- Allocate the first hole that is big enough and stop searching as soon as a free hole is found.

- Best-fit: Allocate the smallest hole that is big enough ~~such~~ search for the entire list.

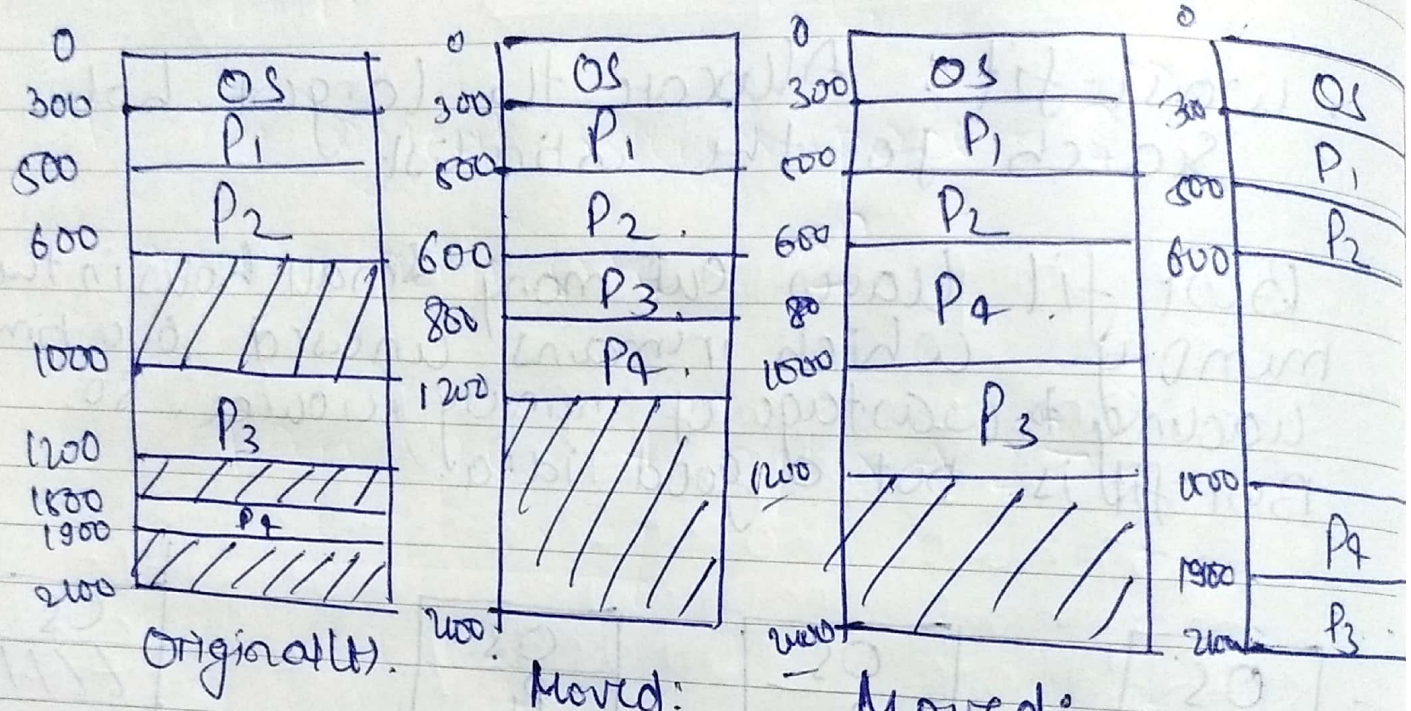
- Worst-fit: Allocate the largest hole. Search for the entire list.

Best fit leaves out many small holes in the memory which remains unused over time leading to wastage of memory resource. So, Best fit is not a good idea.



External fragmentation
Internal fragmentation.

Compaction: Compaction is possible only if relocation is dynamic (not static).



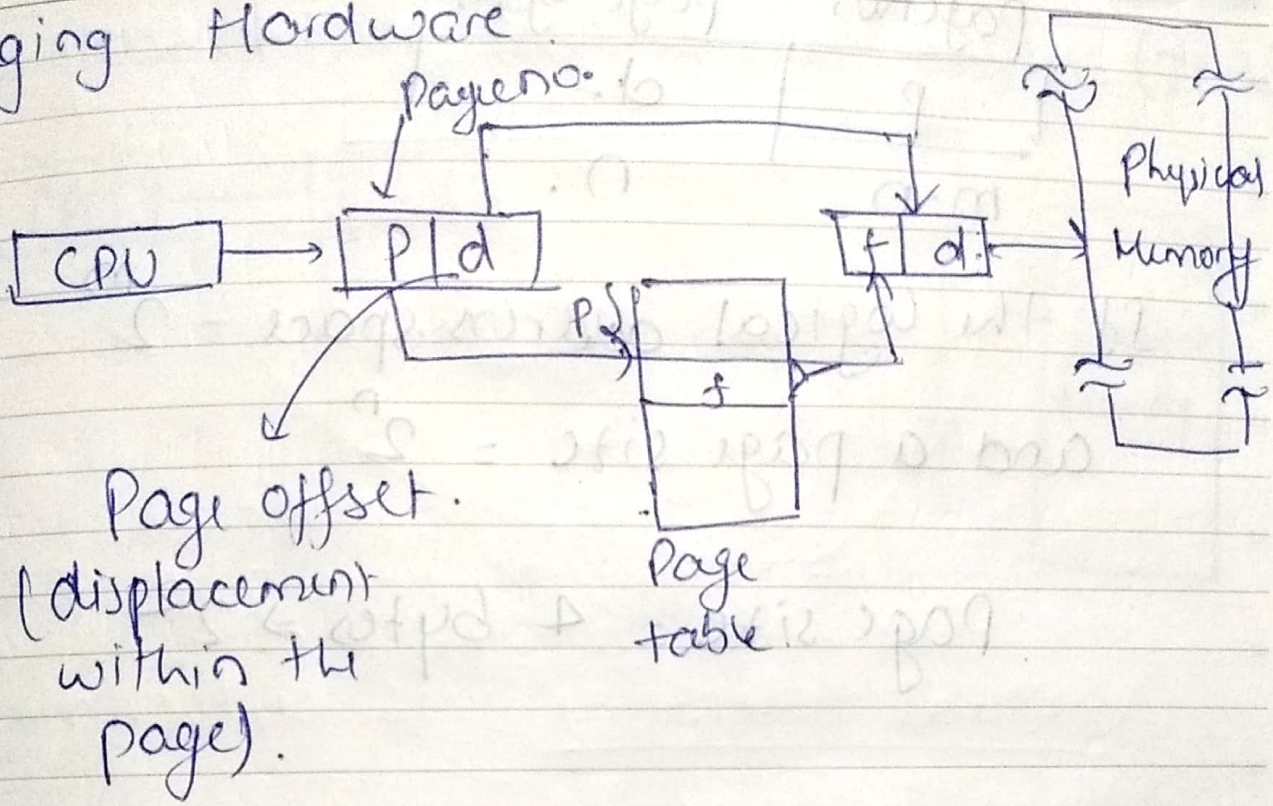
Paging

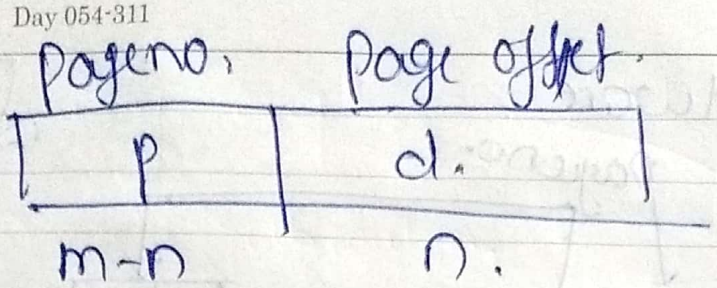
Physical memory is broken into fixed size blocks called frames.

Logical memory is broken into blocks of the same size called pages.

The secondary storage (hard disk) is also divided into fixed size blocks.

Paging Hardware





If the logical address space = 2^m

and a page size = 2^n

Page size = 4 bytes $\Rightarrow 2^2$

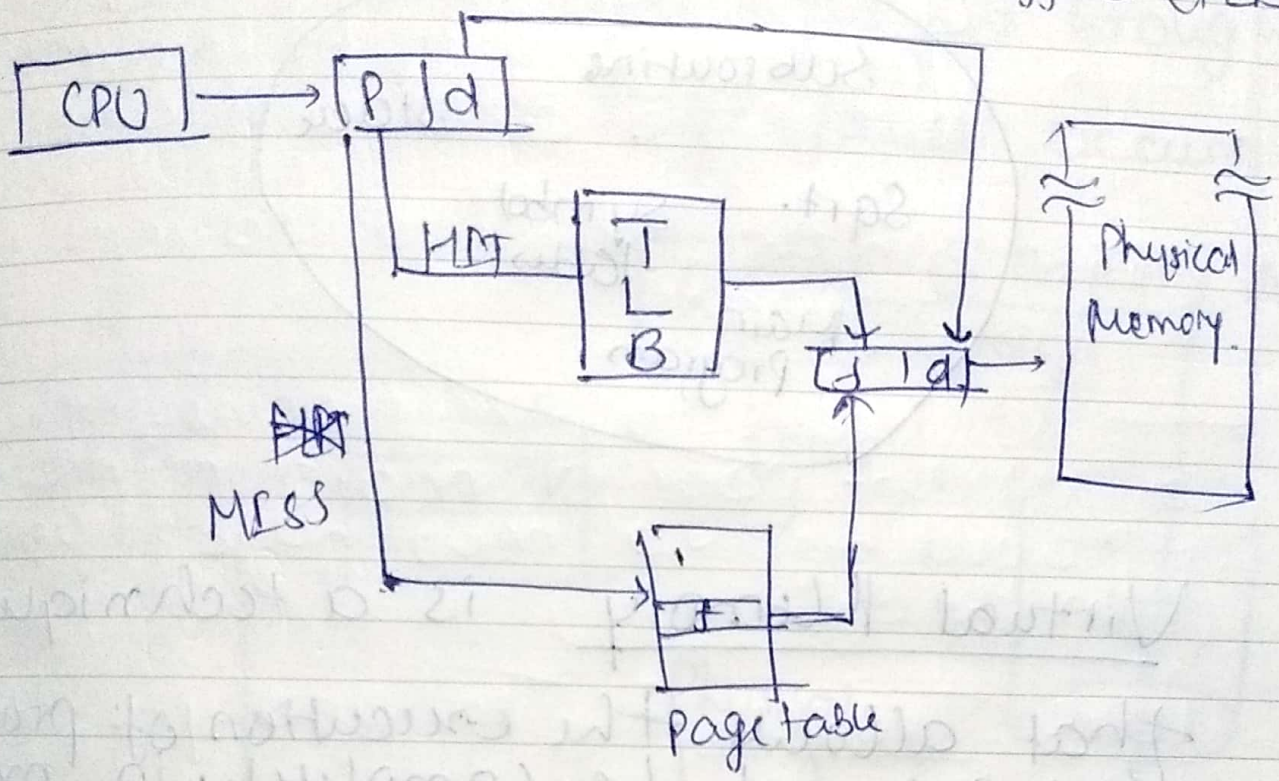
Structure of a page table

Each OS has its own method of storing page tables. It is usually stored in the memory.

- Most of the OS allocate a page table for each process.
- A pointer to the page table is stored in the process control block (PCB). @

A small, special, fast, lookup hardware cache, called associative registe

or translation look aside buffer (TLB).



Let the TLB hit ratio = 80%.

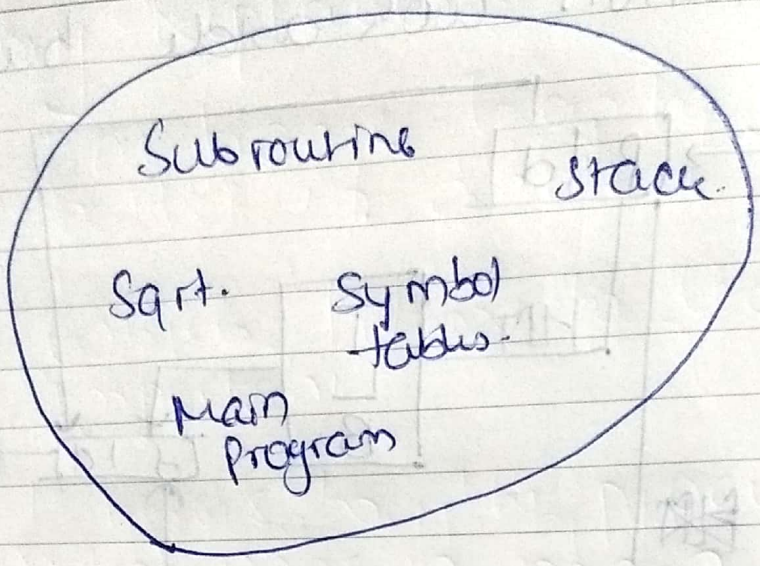
20 nsec is rqd. to search the associative register (TLB).

100 nsec is rqd to access memory.
Then 120 nsec is required when the page no. is in TLB.

However if it fails. (miss) then it will take 20+100+100 nsec (220 nsec).

To find the @effective memory access time by probability

Eff. mem. access time = 0.8×120 + 0.2×220 = 140 nsec



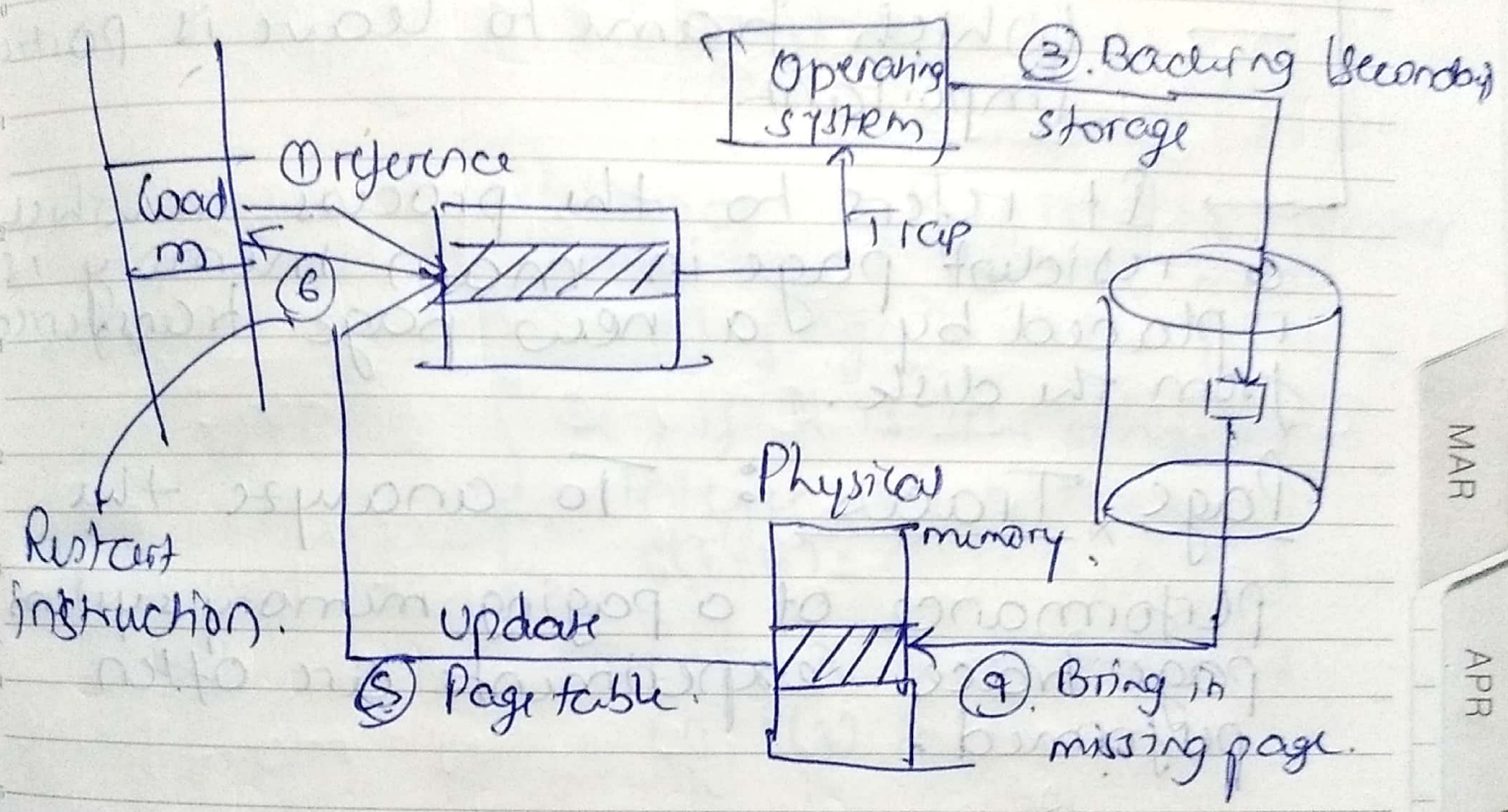
Virtual Memory is a technique that allows the execution of processes that may not be completely in memory.

- The programs can be larger than the physical memory.
- The physical memory is partially filled up with other processes.
- It abstracts main memory into an extremely large, uniform array of storage.
- Commonly implemented by demand paging.

@

What happens when the process tries to use a page that was not brought into memory?

→ Page fault occurs.



@

MARCH 2015

Wk	M	T	W	T	F	S	S
09	30	31					1
10	2	3	4	5	6	7	8
11	9	10	11	12	13	14	15
12	16	17	18	19	20	21	22
13	23	24	25	26	27	28	29

Memory Replacement Policy

→ One frame has to leave memory to let in another frame.

→ Which frame to leave is particularly important.

→ It refers to the process in which a resident page in main memory is replaced by a new page transferred from the disk.

Page x Traces: To analyze the performance of a paging memory system, page trace experiments are often performed.

A page trace is a sequence of page frame numbers (PFNs) generated during the execution of a given program/process.

$P(n) = r(1)\ r(2) \ldots r(n)$, $r(t)$ is the PFN request at time t.

forward distance $f_t(x)$: for page x @ time t is the number of time slots from time t to the first referenced of page x in the future.

$f_t(x) = \begin{cases} k, & \text{if } k \text{ is the smallest integer such that } r(t+k) = r(t) = x \text{ in } P(x). \\ \infty, & \text{if } x \text{ does not reappear in } P(x) \text{ beyond time } t. \end{cases}$

Backward Distance $b_t(x)$ is the number of time slots from time t to the most recent reference of page x in the past.

$b_t(x) = \begin{cases} k, & \text{if } k \text{ is the smallest integer such that } r(t-k) = r(t) = x \text{ in } P(x). \\ \infty, & \text{if } x \text{ never appeared in } P(x) \text{ in the past.} \end{cases}$

Let $R(t)$ be the resident set of all pages residing in main memory at time t. Let $q(t)$ be the page to be replaced from $R(t)$ when a page fault occurs at time t.

Page replacement policies.

1. Least recently used (LRU):

This policy replaces the page in r(t) which has the longest backward distance.

$$g(t) = y \text{ iff } b_t(y) = \max_{x \in R(t)} \{ b_t(x) \}$$

2. Optimal (OPT) algorithm:

This policy replaces the page in r(t) with the longest forward distance.

$$g(t) = y \text{ iff } f_t(y) = \max_{x \in R(t)} \{ f_t(x) \}$$

3. First in first Out (FIFO):-

This policy replaces the page in R(t) which has been in memory for the longest time.

4. Least frequently used (LFU):-

This policy replaces the page in R(t) which has been least referenced in the past.

8. Random Replacement:

This is a trivial

Page trace Experiment:

Consider a paged virtual memory system with two-level hierarchy.

Main memory (M_1) and Disk memory (M_2)

Assume a page size of four words. The no. of page frames in M_1 is 3 labeled a, b & c.

The no. of pages in M_2 is 10, identified by 0, 1, 2, ... 9. The ith page in M_2 consists of word addresses 4i to 4i+3 for all i = 0, 1, 2, ..., 9.

A certain Programa generates the following sequence of word addresses

Word trace: 0, 1, 2, 3, 9, 5, 6, 7, 8, 16, 17, 9, 10, 11, 12, 28, 29, 30

Page trace:

Word trace: 8, 9, 10, 9, 5, 12, 9, 5

Page trace:

Initially all Pfs are empty

	Pf		
LRU	a		3/11
	b		
	c		
	faults.		
OPT	a		4/11
	b		
	c		
	faults		
FIFO.	a		2/11.
	b		
	c		
	faults.		

A page fault causes the following sequ
to occur:-

1. Trap to the operating system
2. Save the user registers and proces
state
3. Determine that the interrupt was a
page fault.
4. Check that the page reference was
legal and determine the location
of the page on the disk.
5. Issue a read from the disk to a
free frame.
6. While waiting, allocate the CPU to
some other user.
7. Interrupt from the disk (I/O) compl
8. Save the registers and process state
for the other user (if step 6 execute
9. Determine that the interrupt was from
the disk.
10. Correct the page table and other
tables to show that the desired
page is now in memory.

11. Wait for the CPU to be allocated to this process again.

12. Restore the user registers, process state, and new page table, then resume the interrupted instruction.

Performance of Demand Paging

Let p be the probability of a page fault ($0 \leq p \leq 1$) we expect 'p' to be close to zero.

i.e. Only few page faults occur.

Effective access time = $(1-p) \times ma + p \times$ page fault.

Where ma ⇒ memory access time.
≅ 10 to 100 or more nano seconds.

We may consider an average page fault service time of 25 ms and memory access time of 100 ns.

Effective Access time = $(1-p) \times 100 + p \times 25 \times 10^6$ ns.

@ $= 100 + 24,999,900 \times p$ nsec.

$\cong 25,000,000\ p$ nsec.

We can see that the effective access time is directly proportional to the page fault rate.

If one access out of 1000 causes a page fault (i.e. $p = \frac{1}{1000}$) the effective access time is 25,000 ns ≈ 25 μsec

The computer will be slowed down by a factor of 250 ($\because \frac{25000}{100}$) because of demand paging.

If we want less than 10% degradation we need:

$$110 > 100 + 25 \times 10^6 p$$

$$10 > 25 \times 10^6 p$$

$$\frac{10}{25 \times 10^6} > p$$

$$4 \times 10^{-7} > p$$

∴ less than 1 memory access out of 2,500,000 to page fault is permitted for a reasonable slowdown.