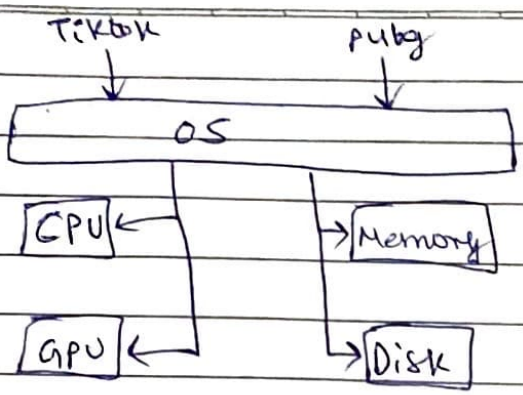
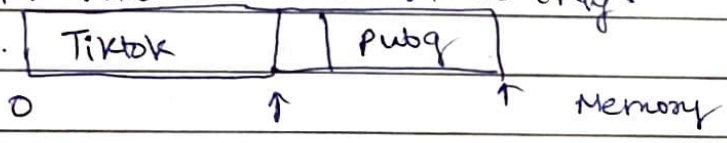
Operating System

Date
Page No.



① Resource Management (R.M.)
② Abstractions (App. doesn't need to contain R.M. code)
③ Isolation & security
④ Access to the hardware through OS only.



OS providing Isolation & security

- An OS is a piece of software that manages all resources of a computer, hardware and software. It allows a user to execute his/her programs without worrying about the underlying complexity of the system. because OS acts as Resource Manager.

- O.S. Goals ⇒
1) Maximum C.P.U. utilization ⇒
2) No process starvation
3) High priority execution (eg scanning external device using Antivirus)

- Types of OS ⇒
① Single process O.S. ⇒ At a time, only one process will execute

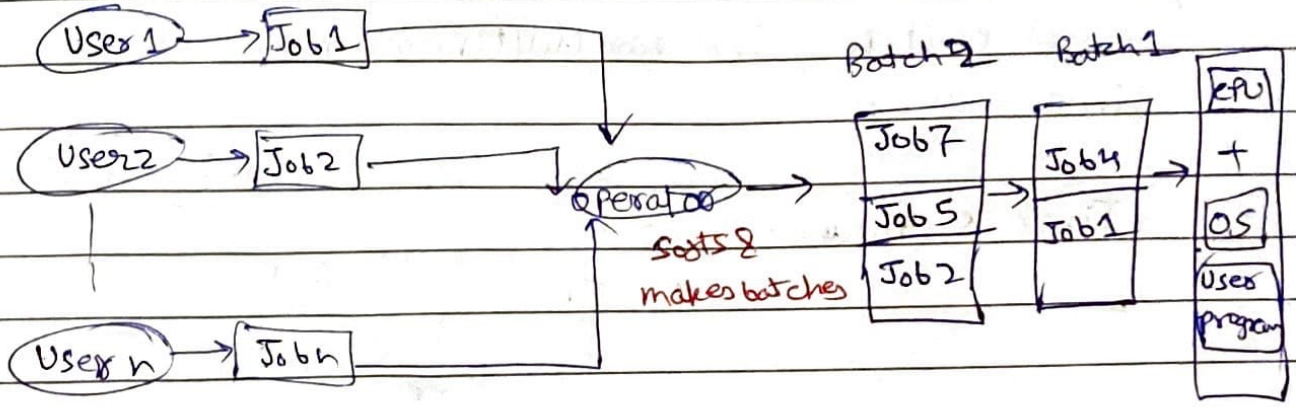
Maximum CPU utilization ✗
No process starvation ✗
High priority execution ✗

eg MSD OS

(2) Batch processing OS →

* Used punch cards (contained digital info)

Date
Page No.



* Jobs in a single batch are executed one by one.

Max CPU util ✗
Process starvation ✗
High priority execution ✗

eg ATLAS

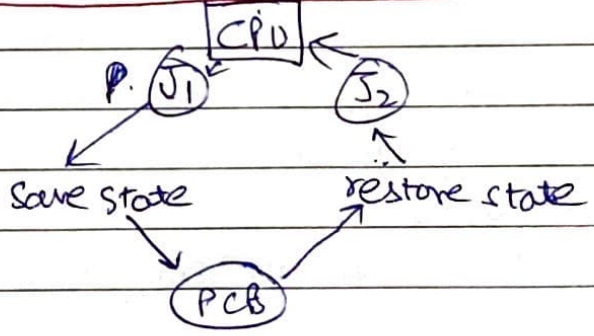
(3) Multiprogramming OS →

* Single CPU
* Ready queue → | J_1 | J_2 | J_3 | J_4 |

(wait state)

* J_1 goes for some I/O & OS will schedule J_2 to execute till J_1 completes I/O. This is called Context switching

* When J_1 goes to wait state, O.S. will save its context in P.C.B. (Process Control Block)



eg THE

④ Multi Tasking OS ⇒

① Single CPU

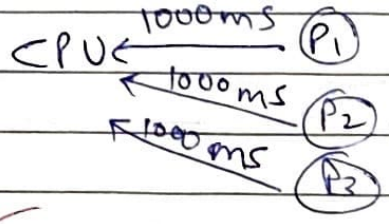
Date _____
Page No. _____

② Context switching like ~~Multi~~ Multiprogramming OS

③ Time sharing

eg CTSS

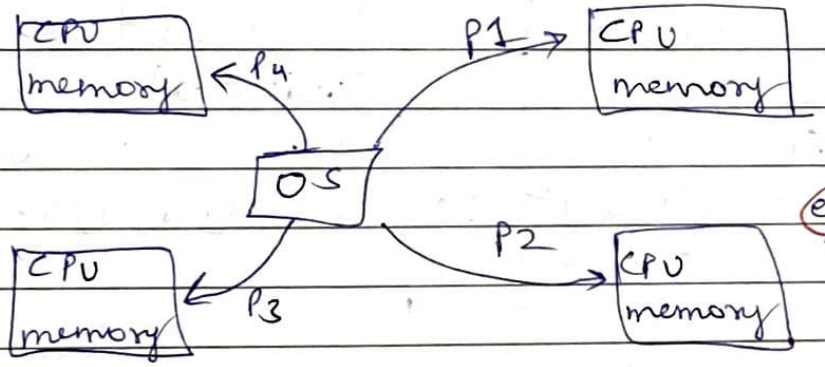
④ Time Quantum.



⑤ CPU utilization ✓
No process starvation ✓
Priority Execution ✓

⑤ Multi processing OS →
Similar to Multitasking, but no. of CPU ≥ 1 eg windows

⑥ Distributed OS ⇒



* Loosely coupled
* Inter connected

eg. Online Compilers

⑦ Real Time Operating System ⇒
* No chance of error
* Industrial applications eg Air Traffic control

LECTURE 3 ⇒

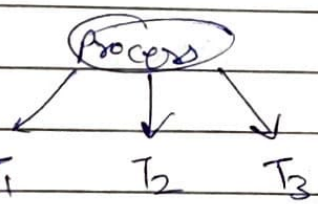
Date
Page No.

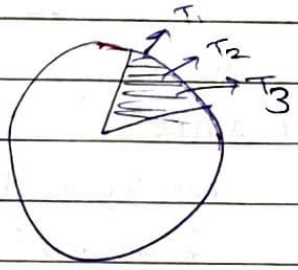
(.exe) → double click → (process)

Program under execution is called process

* Thread → Light weight process
→ independently executable

eg JPG to PNG converter, Multiple tabs in browser, text editor





Multi Tasking	Multi Threading
① More than one process. concept.	① More than 1 threads.
② Makes use of time quantum.	② Parallel execution of threads.
③ Process Isolation & memory protection.	③ No isolation & memory protection.
④ Processes are scheduled.	④ Threads scheduled.
⑤ No. of CPU ≥ 1	⑤ No. of CPU ≥ 1.

Q) How does browser handles multiple tabs?

A) When browser gets opened, two processes are created
① Browser process
② Renderer process (Responsible for rendering HTML)
For each tab that gets opened, a new rendered process is created.

* It's advantage is that if a browser tab gets hanged or become unresponsive, only that renderer process is affected.

* Since threads can share the same address space and global space, opening tabs as new threads may lead to security risks, therefore it makes sense to create them as new process for process isolation.

Date
Page No.

Per Process items	per Thread items
Address space	Program counter
Global variables	Registers
open files	Stack
Child processes	State
Pending Alarms	
signals and signal handlers	
Accounting Information	

* No. of threads should depend on number of cores.

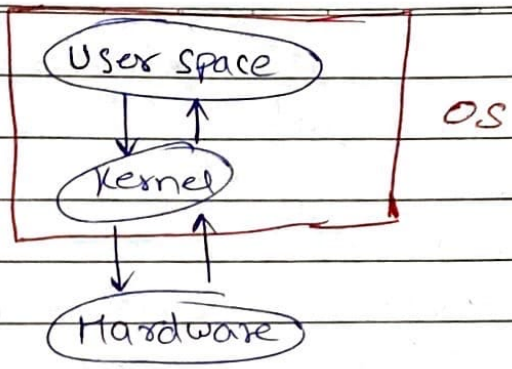
* Thread scheduling ⇒
Threads are scheduled based on their priority. Even though threads are executing within the runtime, all threads are assigned processor time slices by the operating system.

Thread context switching	Process context switching
① OS saves state of thread and switches to another thread of same process.	① OS saves current state of process and switches to another process by restoring its state.
② Doesn't include memory switching. (But program counter, registers & stack are included)	② Includes switching of memory address space.
) fast switching.	③ Slow switching
CPU's cache state is preserved	④ CPU's cache state is flushed.

LECTURE-4

Date
Page No.

⊖ Components of OS ⇒



⊖ Kernel ⇒

① Does process management.
- Process creation, termination
- Process & thread scheduling
- Process synchronize
- Process communication

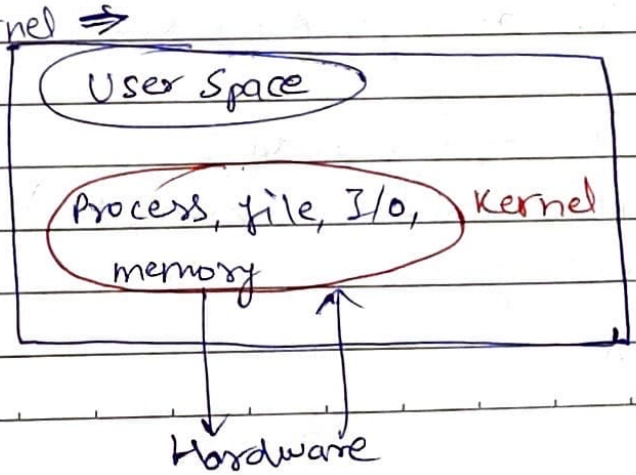
② Does memory management
- Allocate / deallocate
- Free space management

③ File Management
- Create / Delete
- Dir. management

④ I/O management.
- Management & controlling of all I/O devices.
- Spooling
- Buffering
- Caching

⊖ Types of kernel ⇒

① Monolithic kernel ⇒



* Software Interrupt switches from user mode to kernel mode & vice versa

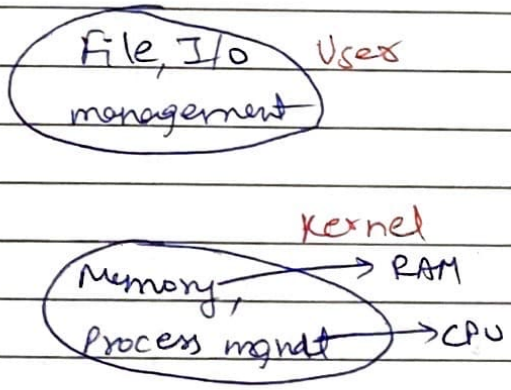
Date
Page No.

* Advantage – faster communication b/w different components of kernel

* Disadvantage – Kernel Bulky
less reliable since all components are in the kernel space. one goes down, all goes down

* eg – Linux, Unix, MS DOS

(2) Micro Kernel ⇒



* Advantage → Less bulky kernel
→ More reliable

* Disadvantage → Performance degrade
→ overhead UM ⟷ KM

* eg – L4 Linux, Symbian OS (Nokia phones)

Q) How does UM & KM communicate?

A) * Using IPC (Inter process communicator).
(1) Using shared memory
(2) Message passing

③ Hybrid Kernel ⇒

* Combined approach

*

Date _____

Page No. _____

File Mgmt. — Usermode

Process, memory, I/O mgmt — kernel space

* eg MacOS, Windows 7

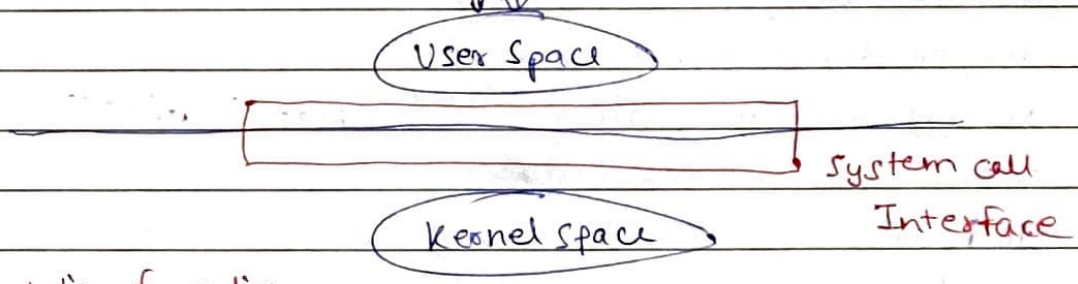
LECTURE 5

System calls ⇒

AIM ⇒ Create folder named "movie"

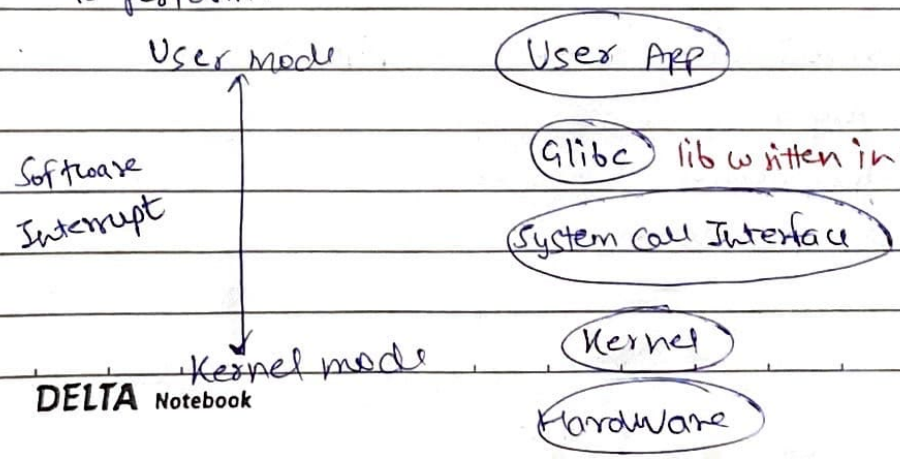
GUI → New Folder

CLI → mkdir movie



Implementation of mkdir (written in C) is present in Kernel space

* A system call is a mechanism using which a user program can request a service from the kernel for which it does not have the permission to perform.



Types of System calls ⇒

Date _____
Page No. _____

(1) Process control

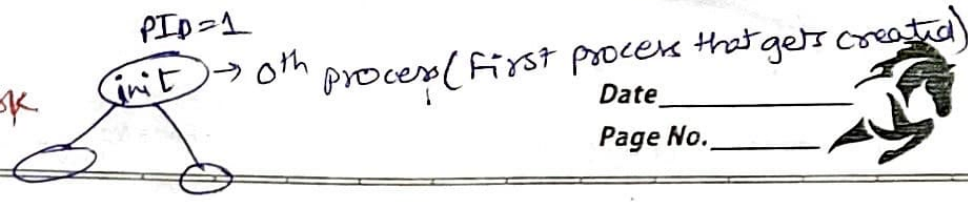
a) end, abort
b) load, execute
c) create process, terminate process fork(), exit()
d) get process attributes, set process attributes.
e) wait for time
f) wait event, signal event
g) allocate and free memory

(2) File Mgmt ⇒
a) create, delete file open()
b) open, close open() close()
c) read, write, reposition read() write()
d) get file attributes, set file attributes
chmod(), chown()

(3) Device management
a) request device, release device
b) read, write, reposition read() write()
c) get & set device attributes
d) logically attach or detach devices

(4) Information maintenance getpid() alarm() sleep()
a) get or set time, get or set data
b) get system data, set system data
c) get or set process, file or device attributes

(5) Communication Mgmt
a) Create, delete communication connection pipe()
b) send or receive messages
c) Transfer status information
d) Attach/detach remote devices

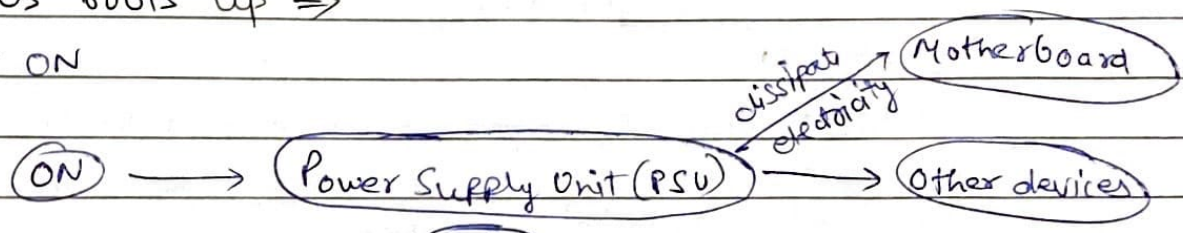


Date _____
Page No. _____

LECTURE 6 ⇒

✠ How OS boots up ⇒

① Power ON



Basic Input Output system

② CPU loads BIOS/UEFI ⇒

Unified Extensible Firmware Interface

a) CPU initializes
b) Goes to a chip BIOS

③ BIOS/UEFI run tests & init hardware

a) Loads some settings from a memory area — Backed by CMOS battery
All BIOS related settings are stored here

b) BIOS program loads with settings
POST ⇒ Power on self Test (Tests ~~other~~ components like RAM etc)

④ BIOS/UEFI hands off to Boot device (Boot Loader) ⇒

- Disk (HDD or SSD)
- CD
- USB device

Boot Loader is present in ⇒

a) MBR (Master Boot Record) (0th sector of device)
b) EFI (Extensible Firmware Interface)
- A new partition in disk contains bootloader

⑤ Bootloader loads the OS

Windows → Boot manager
MacOS ⇒ boot.efi
Linux ⇒ GRUB

LECTURE 7

Date
Page No.

32 bit vs 64 bit OS ⇒

* Register | 1 | 2 | 3 | 4 | — 4 bytes register

00000000

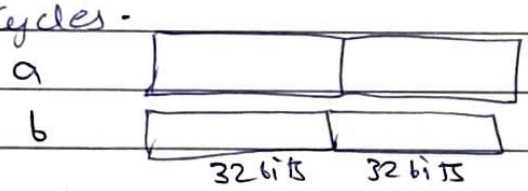
2^{32} unique addresses
↳ 4GB

A 32 bit OS can only support only 4GB of RAM.

* | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | — 8 bytes register

2^{64} → 17179869184 GB

For a 32bit CPU to add 2 numbers of 64 bits, it would take 2 CPU cycles.

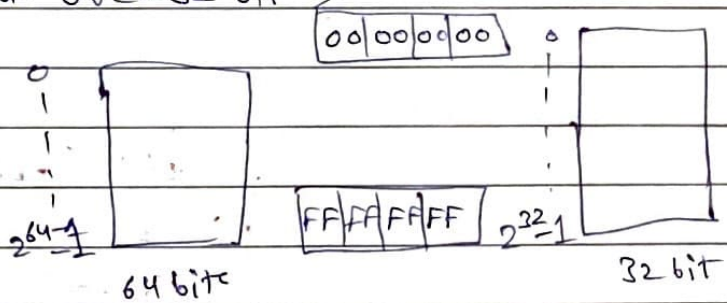


a + b = (Last 32 bits) + (First 32 bits)

Two CPU cycles

* Advantages of 64 bit over 32 bit ⇒

① Addressable space



② Resource usage ⇒

Since we can't have RAM more than 4GB, resources are used according to a 4GB RAM.

In 64 bit OS, RAM size can be very big (as seen above), resource usage can also be increased.

③ Performance

64 bit >> 32 bit

Date
Page No.

* A 32 bit processor in a single CPU cycle. can handle 32 bits only.

④ Compatibility →

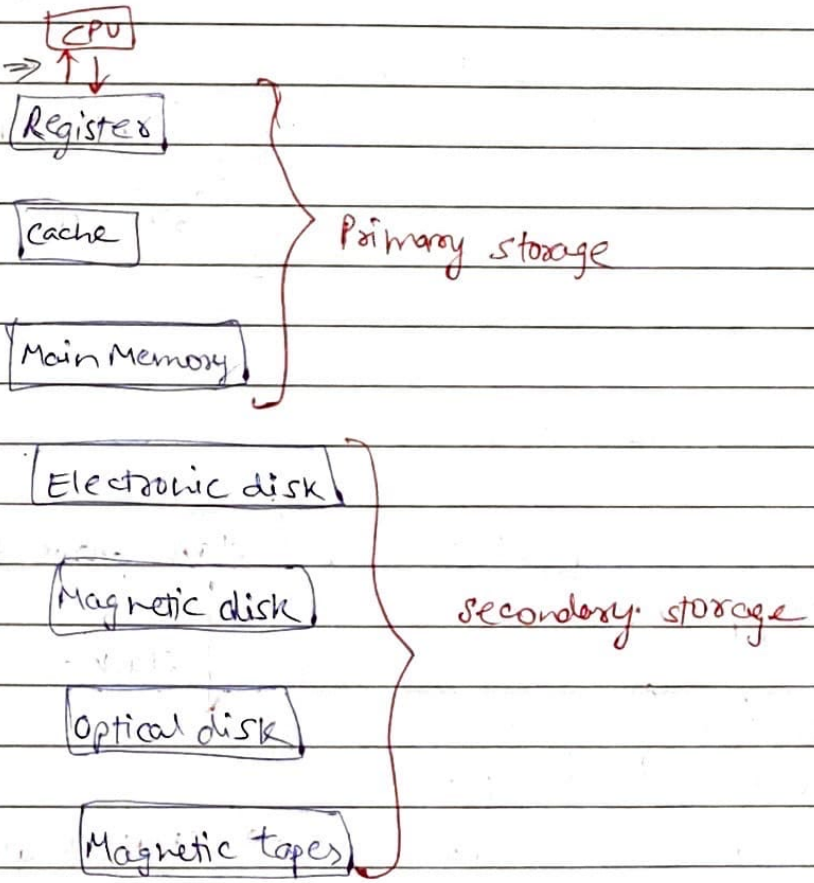
64 bit CPU can run both 32-bit and 64 bit OS. 32 bit CPU can run only 32 bit OS.

⑤ Better Graphics performance →

8 bytes graphics calculations make graphics-intensive apps run faster.

LECTURE 8

* Types of storages ⇒



LECTURE 9

* How OS creates a process ⇒

① Load the program & static data to memory

↓

used for initialization

(2) Allocate Runtime Stack

- Part of memory used for local variable, function argument & return value

Date _____
Page No. _____

(3) Allocate heap (For runtime) ⇒

- part of memory used for dynamic allocation.

(4) I/O tasks ⇒

Unix – Input → handle
Output → output handle
error → Error handler

eg fprintf(stderr, "")
error handler / File descriptor

(5) OS hands off control to main()

```
main(){

return 0;
}
```

Parent → P1 → return 0

Parent process expects 0 as safe value return

⊖ Process Architecture ⇒

stack grows downwards.

stack overflow (set base case)

heap grows upwards

insufficient/out of memory (deallocate/free memory)

data → Static & global data

text → Executable code

⊖ Attributes of process ⇒

Every process' entry is done in process table.

1	P1
2	P2
3	P3

→ PCB (Process control Block)

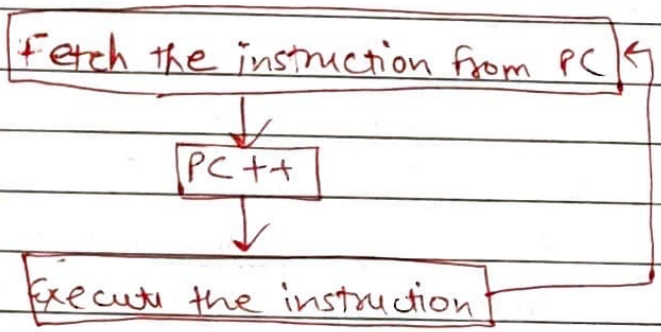
* For context switching, process table stores all data related to a process in its PCB.

Date
Page No.

*

PCB →

Process ID	→ OS allocates process Id
Program counter	→ Tells keeps track of instructions
process state	→ Current state of process
priority	→
register	→ stack pointer, base pointer, control registers [saves the registers]
open file List	→ open file descriptors
open devices List	→ open devices



LECTURE 10

* Process State ⇒

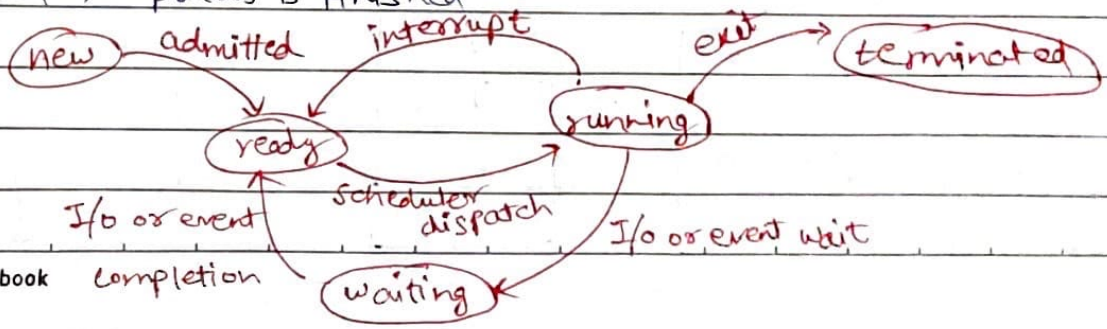
① New State ⇒ When program is ~~now~~ becoming a process.

② Ready ⇒ when process is in memory.
- Present in ready queue waiting for CPU

③ Running ⇒ When process has been allocated CPU

④ Waiting ⇒ waiting for I/O completion

⑤ Terminated ⇒ process is finished



Different kinds of Queues ⇒

Date
Page No.

① Job Queue

- All processes in new state goes into Job queue.
- Job Scheduler moves process from Job Queue → ready queue
- Job scheduler is called Long Term scheduler

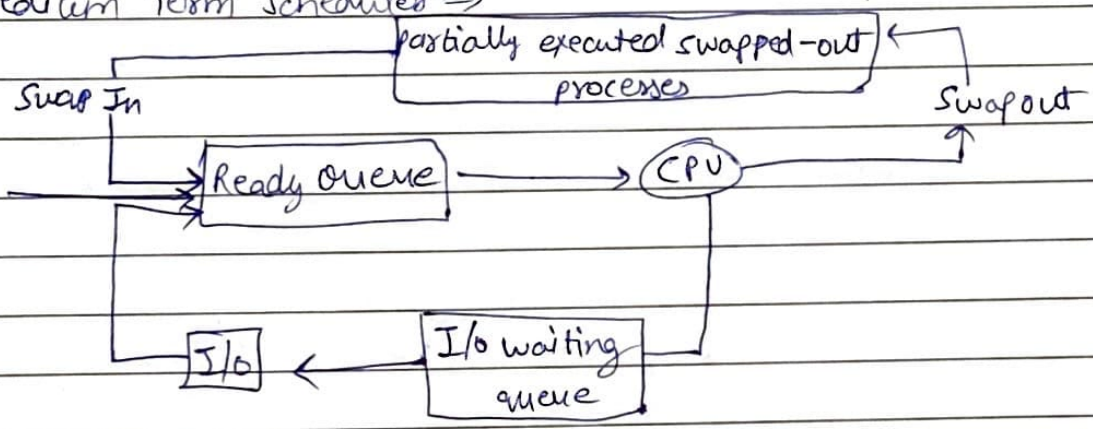
② Ready Queue ⇒

- CPU scheduler dispatches jobs from ready queue to running state
- Short Term scheduler.

- Degree of multiprogramming → No. of processes that can be present in a ready queue at a given time. Degree of multiprogramming is governed by Job scheduler / LTS
  - eg. P1 --- P5, degree of mult = 5

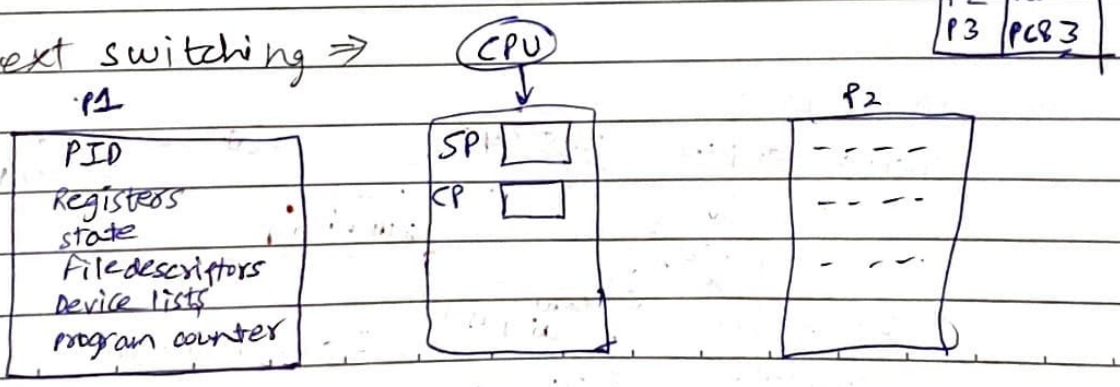
LECTURE 11

③ Medium Term scheduler ⇒



- Medium Term Scheduler is responsible for swapping in/out processes when ready queue is full.

④ Context switching ⇒

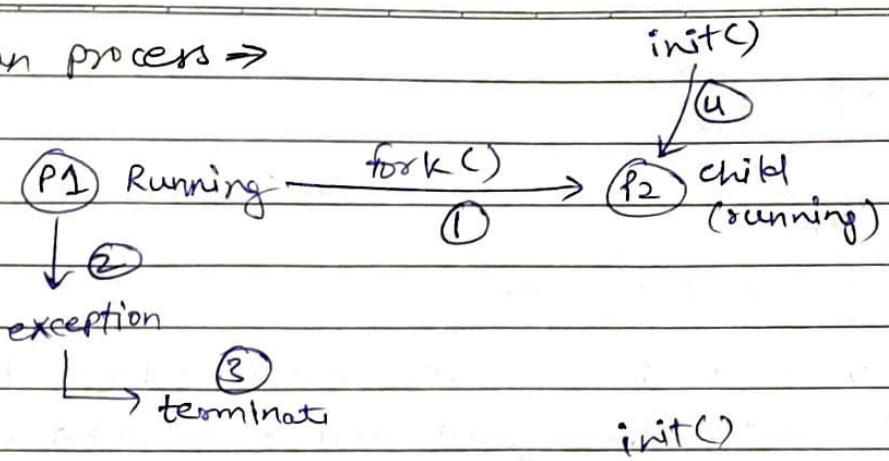


P1	PCB1
P2	PCB2
P3	PCB3

* Kernel is responsible for context switching.
* Pure overhead

Date
Page No.

- Orphan process →



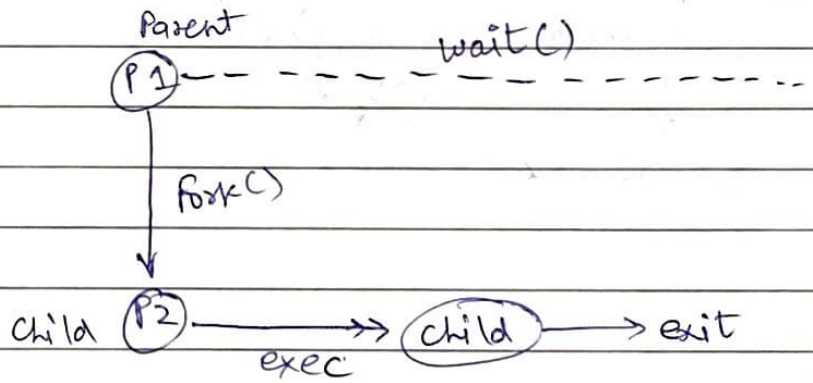
After becoming an orphan process, ~~P2~~ ~~Parent~~ will become P2's parent.

echo $$ to get terminal's PID

How to create an orphan process

① Create bash file ⇒ sleep 200 & → Detached mode
② A new process will get created whose PPID will be 1.

- Zombie Process ⇒



Suppose wait() = 5 minutes, and child process gets terminated after 2 min., the child process will release all the resources but since parent process is still waiting for child process' response, it will keep on waiting till wait() During this time, child process P2 entry will persist in the process table.

How to create Zombie process

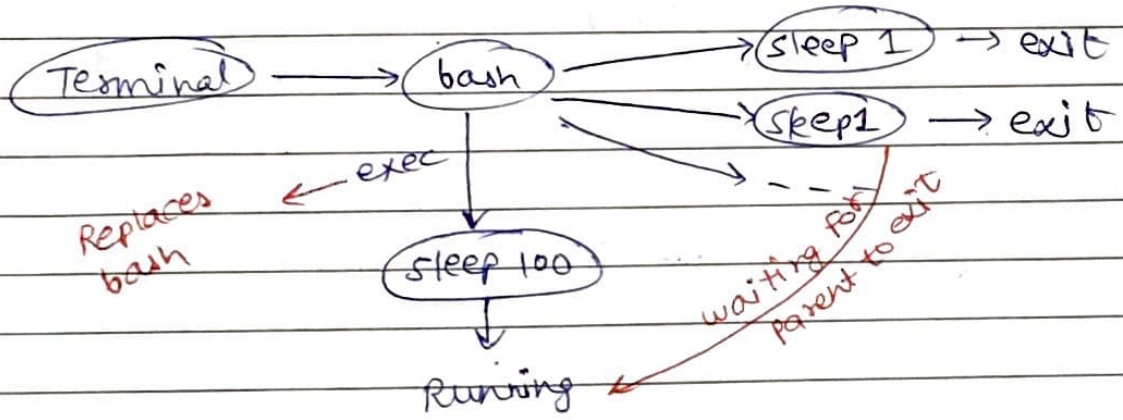
```
for i in{1..100}
do
    sleep 1&
done
exec sleep 100
```

→ Make sure to use exec

* Exec ⇒ Whenever we run any command in a Bash shell, a subshell is created by default; and a new child process is spawned (Forked) to execute the command.

When using exec, however the command following exec replaces the current shell.

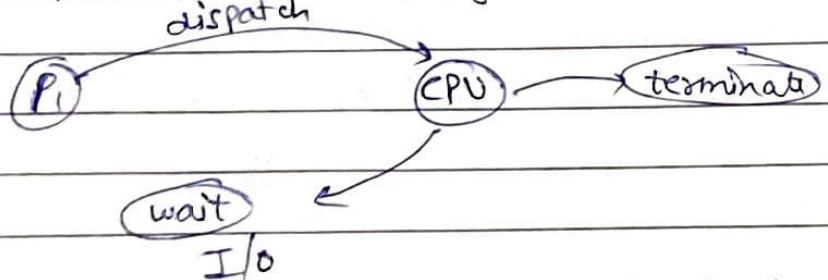
This means no subshell is created and the current process is replaced with this new command.



LECTURE 12

CPU scheduling ⇒

① Non-preemptive scheduling ⇒



* Once, CPU is allocated to P1, it will hold the CPU until it is terminated or it goes to do some I/O.

Date _____
Page No. _____

* No concept on Time Quantum.

② Preemptive scheduling ⇒

* Once time quantum finishes for a process, it is moved to the ready state.
* Releases CPU when terminated or go to I/O.

starvation ⇒ Non-Preemptive will cause more starvation
CPU ⇒ CPU utilization will be more in preemptive
overhead ⇒ More in preemptive (T.Q.)

* CPU scheduling Goals ⇒

① Max CPU utilization
② Minimum Turn Around Time
Time difference (Terminates — time at which process first arrives in ready Q)
③ Minimum wait time (waiting for CPU allocation)
④ Minimum Response time
Time difference (Time at which process gets CPU for the first time — time at which process first arrives in ready Q)
⑤ Max. Throughput (No. of processes completed per unit time)

* Arrival Time (AT) ⇒ Time when process is arrived at ready Q
* Burst Time (BT) ⇒ Time required by time process for its execution
* Turnaround Time (TAT) ⇒ Time taken from first time process enters ready state, till it terminates. (CT − AT)

* Wait Time ⇒ Time process spends waiting for CPU
(WT) (WT = TAT − BT)

Date
Page No.

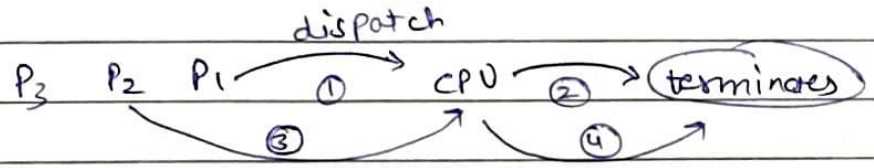
* Response Time ⇒ Time duration b/w process getting into ready Q and process getting CPU for the first time.

* completion Time ⇒ Time taken till process gets terminated
(CT)

TAT = CT − AT

⊖ CPU scheduling Algorithms ⇒

① First come first serve (FCFS) ⇒

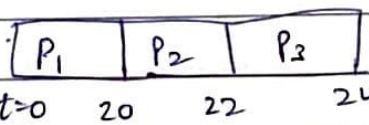


~~Convoy effect~~

~~Gantt~~	Pno	AT	BT	CT	TAT	WT
	1	0	20	20	(20−0) = 20	20−20 = 0
	2	1	2	22	(22−1) = 21	21−2 = 19
	3	2	2	24	(24−2) = 22	22−2 = 20

Avg. WT = 13

Gantt chart

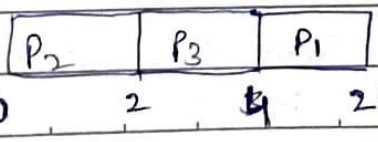


~~First~~ FCFS when process P1 comes ~~at~~ last

Pno	AT	BT	CT	TAT	WT
P2	0	2	2	2	0
P3	1	2	4	3	1
P1	2	20	24	22	2

Avg. WT = 1

Gantt chart



Convoy Effect ⇒

* If one proces has longer BT. It will have major effect on average WT of diff processes, called Convoy Effect.

Date _____
Page No. _____

* Convoy Effect is a situation where many processes, who need to use a resource for a short time, are blocked by one proces holding that resource for a long time.

* This causes poor resource management.

② Shortest Job First Algo ⇒ (SJF)

* Proces with shortest B.T. will get CPU

* It's next impossible to find exact Burst Time of a proces. B.T. is usually calculated using heuristics (known info).

* There SJF may schedule a wrong proces first.

Non-preemptive — from estimation

Pno.	AT	BT	CT	TAT	WT
P_1	0	8	8	8-0=8	8-8=0
P_2	1	4	12	12-1=11	11-4=7
P_3	2	9	26	26-2=24	24-9=15
P_4	3	5	17	17-3=14	14-5=9

Avg. = 7.75 unit

G.C.

P_1	P_2	P_4	P_3

t=0 8 12 17 26

Criteria: AT+BT

If P_1 BTs was 80, it would lead to convoy effect.

Preemptive

Pno	AT	BT	CT	TAT	WT
P_1	0	7	17	17-0=17	17-8=9
P_2	1	4	5	5-1=4	4-4=0
P_3	2	9	26	26-2=24	24-9=15
P_4	3	5	10	10-3=7	7-5=2

P_1	P_2	P_4	P_1	P_3

t=0 1 5 10 17 26

Avg. = 6.5 units

No convoy effect

Date
Page No.

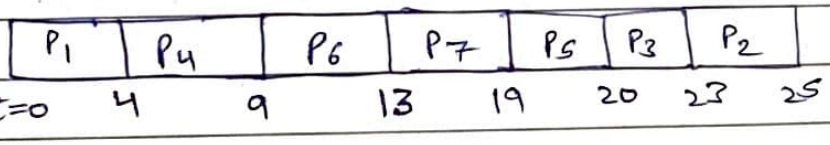
(3) Priority scheduling algo ⇒

* SJF is a special case of priority scheduling as process with minimum BT is given higher priority.

Non preemptive ⇒ priority

Pno (P)	AT	BT	CT	TAT	WT
(1) 2	0	4	4	4-0=4	4-4=0
2 4	1	2	25	25-1=24	24-2=22
(3) 6	2	3	23	23-2=21	21-3=18
(4) 10	3	5	9	9-3=6	6-5=1
(5) 8	4	1	20	20-4=16	16-1=15
(6) 12	5	4	13	13-5=8	8-4=4
(7) 9	6	6	19	19-6=13	13-6=7

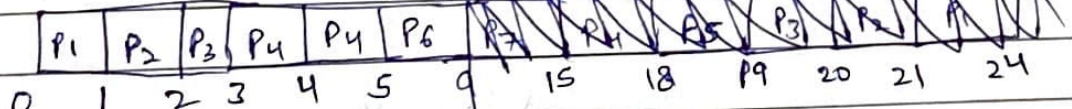
Avg. = 9.71



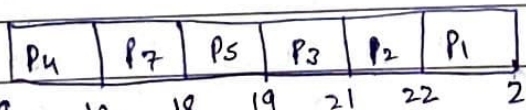
Preemptive ⇒

Pno	P	AT	BT	CT	TAT	WT
(1)	2	0	4 3	25	25	21
(2)	4	1	2 1	22	21	19
(3)	6	2	3 1	21	19	16
(4)	10	3	5 4 3	12	9	4
(5)	8	4	1	19	15	14
(6)	12	5	4	9	4	0
(7)	9	6	6	18	12	12

Avg wt t =



After 15, both P1 & P2 have arrived



Will result in convoy effect for low priority jobs

Biggest Disadvantage ⇒

For both ~~&~~ Preemptive & non preemptive,
indefinite waiting or extreme starvation

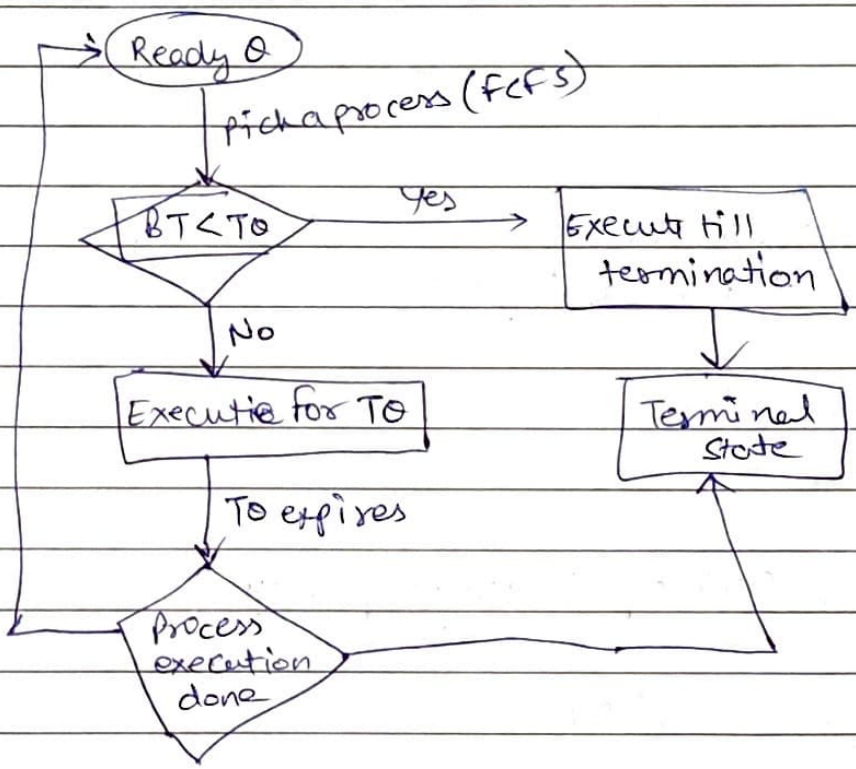
Date _____
Page No. _____

Real Life example ⇒ IBM 7094 at MIT
Job submitted in 1967
when checked in 1973, job was still in ready Q

Ageing ⇒
Gradually increasing priority of lower job

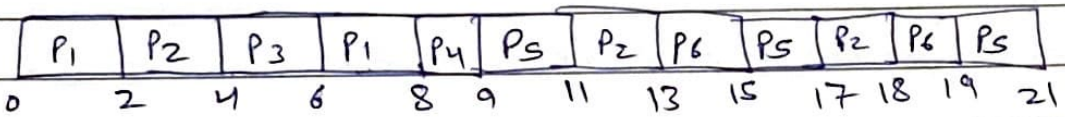
(4) Round Robin ⇒

- Most Popular
- FCFS (Preemptive) version
- Criteria AT + TQ
- ~~Does not~~ Designed for time-sharing
- Easy to implement

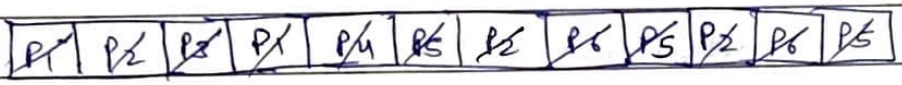


Pno	AT	BT
1	0	4 X 2 0
2	1	~~5 3~~ X 0
3	2	~~2~~ 0
4	3	~~1~~ 0
5	4	~~6 4 2~~ 0
6	6	~~3~~ X 0

Date _____
Page No. _____



TQ = 2



* More context switching, no convoy effect.

LECTURE 14

(1) Multi Level Queue Scheduling ⇒

(1) System process ⇒ created by OS
(2) Interactive process ⇒ user input required (foreground)
(3) Batch process ⇒ No Input (Background)

System process (SP) Q → Highest priority

Interactive process (IP) Q

Batch process (BP) Q → Lowest priority

* Each queue will have its own scheduling algo.

SP > IP > BP

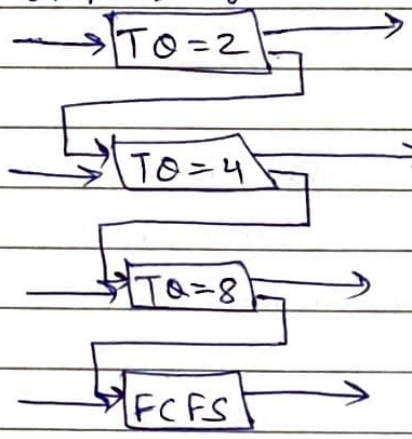
Date
Page No.

* Batch processes will have to wait for CPU, so Convoy effect is present.

(2) Multi Level ~~Queue~~ Feedback Queue scheduling algo =>

* Multiple sub Queue => Inter queue movement of processes is allowed.
* Separating process based on Burst Time.
  - BT high → Lower Queue
  - I/o bound & interactive process → Higher priority
* Ageing => Lower priority jobs priority will be increased.
* Flexible
* Configurable according to OS design

Sample Design



. If a job doesn't finish in TQ, it is moved to the lower queue.

* MLFQ design =>
  (1) No. of Queues
  (2) Scheduling algo in each Queue
  (3) Method to upgrade a process to higher Queue
      - Ageing
  (4) Method to downgrade/demote a process to lower Queue
  (5) Which queue will the process will go to first.

Date _____
Page No.

Comparison ⇒

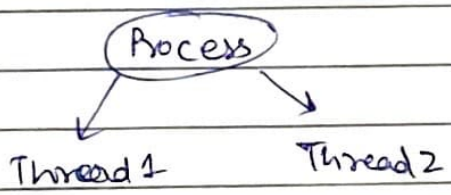
	FCFS	SJF	PSJF	Priority	P-Priority	RR	MLQ	MLFQ
Design	Simple	Complex	Complex	Complex	Complex	Simple	Complex	Complex
Preemption	No	No	Yes	No	Yes	Yes	Yes	Yes
Convoy eff.	Yes	Yes	No	Yes	Yes	No	Yes	Yes
Overhead	No	No	Yes	No	Yes	Yes	Yes	Yes

Ageing

LECTURE 15

Concurrency ⇒

* Multiple instruction at the same time



Thread ⇒

* Light weight process
* An independent path of execution in a process
* Used to achieve parallelism by dividing a process' subtask which are independent path of execution.

Q) How each thread get access to CPU. ⇒

T1. ⟶ TCB Thread Control Block

* Each thread has its own program counter.
* Thread control Block just like Process Control Block

~~Threads context switch~~

* Multi Threading in single CPU is not possible.

Benefits of MultiThreading ⇒

① Responsiveness ≡ Interactive

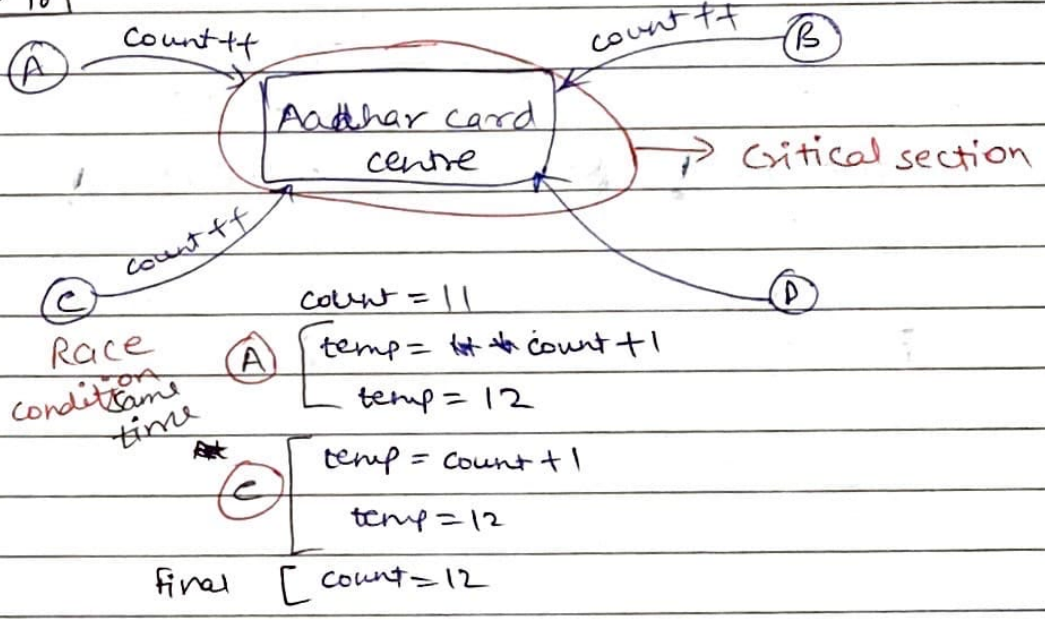
Date _____
Page No. _____

② Resource sharing ⇒ shared address space & resources

③ Economical ⇒ Context switching is more economical

④ Threads better utilization of Multicore CPU.

LECTURE 16



Critical section ⇒

- Refers to the segment of code where process/threads access shared resources, such as common variables and files, and perform write operations on them.
- ~~Single~~ processes/threads execute concurrently, any process can be interrupted mid execution.

Race condition ⇒

- Race condition occurs when two or more threads can access shared data and they try to change it at the same time.
- Because the thread scheduling algo can swap b/w threads at any time, you don't the order in which threads will attempt to access the shared data. Therefore, result of change in data is dependent on thread scheduling algo.

- Both threads racing to change the data.

Race condition / Critical section solution ⇒

Date
Page No.

① Atomic operation ⇒
- operation performed in the same CPU cycle

② Mutual exclusion ⇒
- Support of locks / mutex

③ Semaphores

Q) Can we use single flag to overcome Race condition?

A)

turn initially = 0

T_1	T_2
while(1){	while(1){
while(turn != 0);	while(turn != 1);
C.S.	C.S.
turn = 1	turn = 0
}	}

$T_1 \rightarrow T_2 \rightarrow T_1$ - - -
Order is fixed

Solution of C.S. should have 3 conditions ⇒
① Mutual Exclusion
② Progress (Means, the order shouldn't be fixed, if a thread is not in C.S., it shouldn't make other threads wait)
③ Bounded waiting ⇒ No thread should wait indefinitely.

With single flag, ② condition which is progress is not fulfilling as the order depends on the value of turn initially.

Improvement in above solution, Peterson's solution ⇒
Flag[2] → indicate if a thread is ready to enter the C.S.,
Flag[i] = true implies T_i is ready.
turn → indicates whose turn is to enter the C.S.

Date
Page No.

T_1	T_2
while(1){	while(1){
flag[0] = T	flag[1] = T
turn = 1	turn = 0
while(flag[1]==T && turn==1);	while(turn==0 && flag[0]==T);
C.S.	C.S.
flag[0] = F	flag[1] = F
}	}

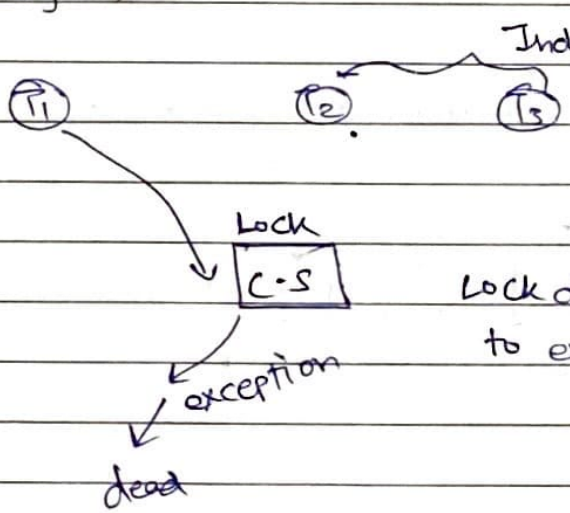
Mutual exclusion ✓

Progress ✓

Peterson's solution works only for 2 threads.

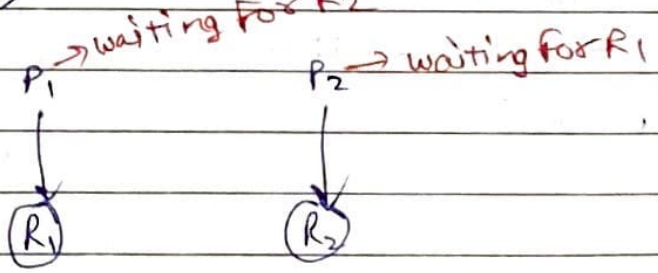
Locks disadvantage ⇒

① Contention ⇒



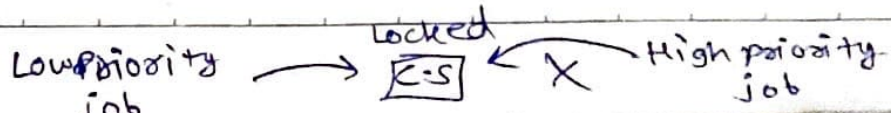
Lock doesn't gets released, due to exception.

② Dead Lock ⇒



③ Debugging issue

④ Starvation ⇒



LECTURE 17

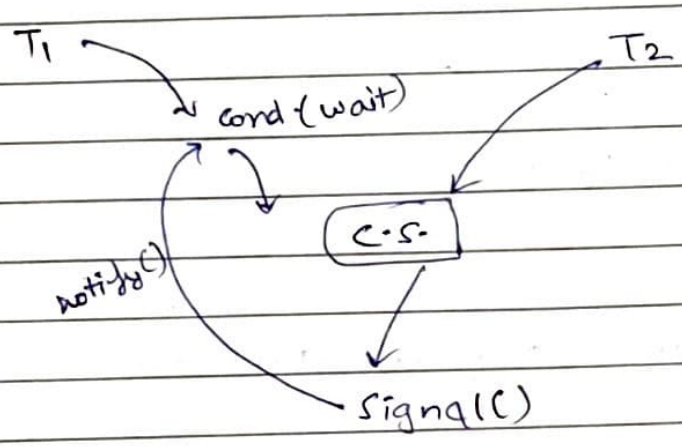
Date
Page No.

① single Flag ✗

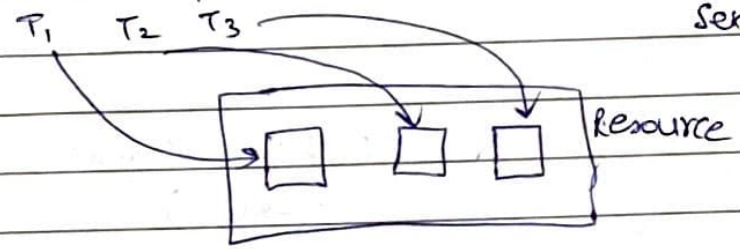
② Peterson's soln. ✗ 2 threads

③ Locks / Mutex "Good solution but not great (Busy waiting)
CPU cycles are wasted

④ Conditional variables ⇒
wait() signal()



⑤ semaphores ⇒
- when multiple resources are present.



Semaphore = ~~3 2 1~~ 0

```
wait(s){
    s->value--;
    if(s->value<0){
        add to s->blocklist
        Block()
    }
}
```

```
signal(s){
    s->value++
    if(s->value<=0){
        remove P from s->Block.
        wakeup(P)
    }
}
```

semaphore S(2);
T_1 → wait() → s→val = 1 → After CS → signal
T_2 → wait() → s→val = 0
T_2 → wait() → s→val = -1 → Block()

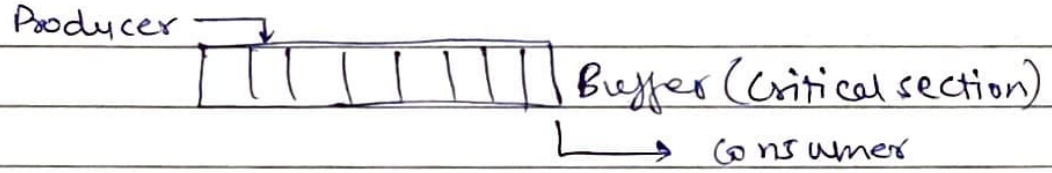
semaphore(1) ⇒ is just mutex.

Date
Page No.

- Semaphore is a synchronization method.
- An integer equal to the number of resources.
- Allows multiple program threads to access the finite instance of resources whereas mutex allows multiple threads to access a single shared resource one at a time.
- Binary semaphore value can be 0 or 1.
- Counting semaphore can range over an unrestricted domain.

LECTURE 18

0 Producer-Consumer Problem ⇒
1. Producer Thread
2. Consumer Thread



Problems ⇒
1) Sync b/w producer & consumer
2) Producer must not insert data when buffer is full.
3) Consumer must not pick/~~consume~~ remove data when buffer is empty.

Solution ⇒

Semaphores
1. m, mutex → Binary sema. used to acquire lock on buffer.
2. empty → a counting. sem., initial value is n
3. Full → tracks filled slots. ~~give~~ initial value is 0.

Producer
```
do {
  wait(empty); // wait until empty > 0
               then empty → value
  wait(mutex);
```

Consumer
```
do {
  wait(Full); // wait until full > 0,
              then full--
  wait(mutex);
```

Date
Page No.

// C.S., add data to buffer
signal (mutex);
signal (full); // increment full→value
} while(1)
	// remove data from Buffer
	signal (mutex);
	signal (empty); // increment empty
	} while(1)

LECTURE 19

- Reader/writer Problem ⇒
  1. Reader Thread → Read
  2. Writer Thread → write/update

- if > 1 readers are reading
  // no problem

- > 1 writes OR 1 writer && some other thread (R/w) parallely
  // race condition, inconsistent data

- Solution using Semaphores ⇒
  1. mutex → Binary semaphore
     → to ensure mutual exclusion, when read count (rc) is updated.
  2. wrt → Binary semaphore
     → common for both reader & writer
  3. read count (rc) → integer
     → Tracks how many readers are reading in C.S.

writer sol.

```
do{
   wait(wrt);
  // do write operation
   signal(wrt);
}while(true);
```

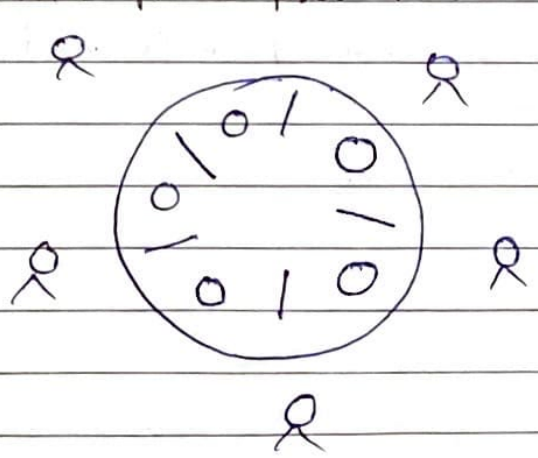
reader Soln ⇒

```
do {
    wait(mutex); // to mutex readcount variable
     rc++;
    if(rc==1)
      wait(wrt); // ensures no writer can enter if there is
                    even one reader
      signal(mutex);

    // C.S. : Reader is reading
    wait(mutex)
     rc--; // a reader leaves
    if(rc==0) // no reader left in CS
       signal(wrt); // writer can enter
    signal(mutex); // reader leaves
} while(1)
```

LECTURE 20

- Dining philosopher problem ⇒



- 5 philosophers
- 5 noodles
- 5 chopsticks

Semaphores ⇒

① each chopstick – semaphore (Binary sema) Date _____ Page No. _____

semaphores fork/chopstick[5] {1}

② wait() → fork[i] ⇒ ph[i] → acquire

③ release() → fork[i] → fork → release

Solution ⇒

```
do {
    wait (fork[i]);
    wait (fork[(i+1)%5]);      → Make this C.S.
    // eat
    signal (fork[i]);
    signal [fork[(i+1)%5]);
    // think
} while(1)
```

Above solution will lead to Deadlock.

- When each ph. tries to grab his right fork, he will be waiting for ever (Deadlock).

a) Allow atmost 4 phil. to be sitting simultaneously.

b) Allow a phil. to pick up his fork only if both forks are available and to do this, he must pick them in a critical section (atomically).

Make critical sec. in above sol.

c) Odd / even rule

Semaphores are not enough to provide a complete solution.

LECTURE 21

Date
Page No.

Deadlock =>

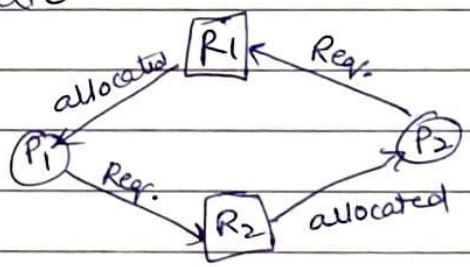
- Two or more processes are waiting on some resources availability, which will never be available as it is also busy with some other process. The process/ threads are said to be in Deadlock.

Resources => Memory space, CPU cycles, files, locks, sockets, IO devices etc.

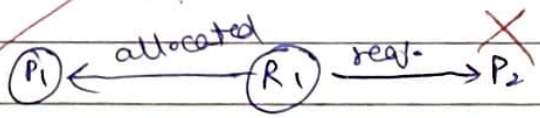
Q) How a process / thread utilize a Resource?
A) 1) Request
2) Use
3) Release

Necessary conditions for deadlock =>

1. Mutual Exclusion
2. Hold & wait



3. No preemption



4. Circular wait

All above conditions should occurs simultaneously.

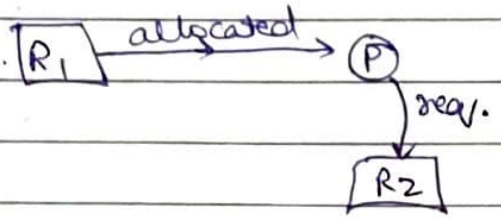
- Resource Allocation Graph ⇒

- vertex → ① Process vertex (P)

Date
Page No.

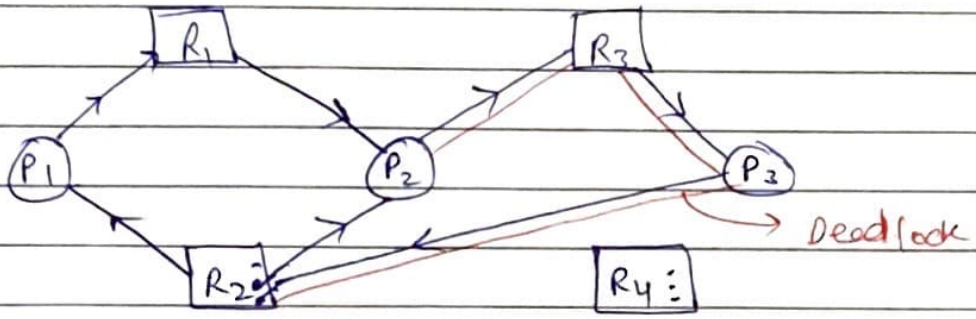
② Resource vertex [R]

- Edges → ① Assign
② Request



- Multiple instance of Resource ⇒

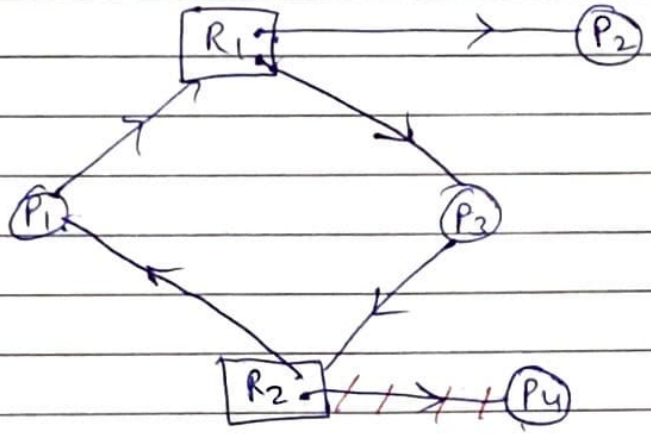
example



By definition of R-A-G. ⇒

① RAG no cycle → NO deadlock

② RAG cycle → may be DL.



- P_4 executes & releases R_2.
- R_2 gets assigned to P_3
- Even after cycle, no deadlock

Methods to handle DL ⇒

① Prevent or Avoid Deadlock ⇒

Date _____
Page No. _____

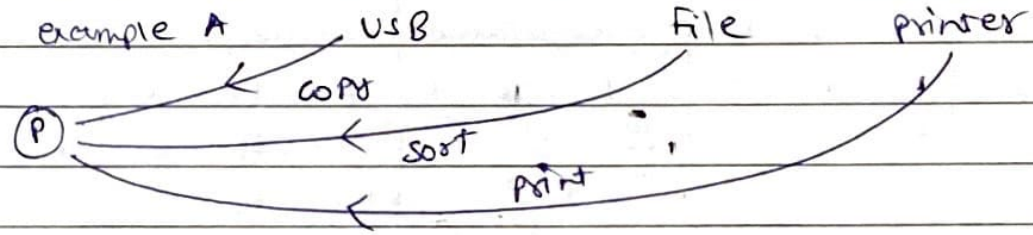
② Allow system to go in DL
↳ Detect → Recover

③ Ostrich algo
↳ Deadlock ignorance
– App. developer to handle everything

Deadlock prevention ⇒

① Mutual Exclusion.
– mutex only on non shareable resource
eg – read-only file → no mutex

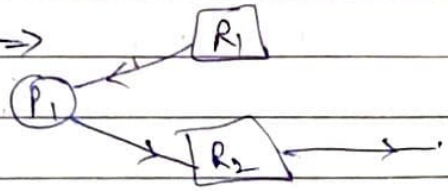
② Hold & wait
– To ensure H&W condition never occurs in the system, we must guarantee that whenever a process requests a resource, it doesn't hold any other resource.



all Allocated for the entire duration.

example B : allow a process to request resources only when it has none. It can request any additional resources after it must have released all the resources that is currently holding.

③ No preemption ⇒



• P_1 releases R_1.

• P_1 will request R_1 & R_2 together.

May lead to ~~lock~~ lock collision

Date
Page No.

④ Circular wait ⇒

LECTURE 22

⊖ Deadlock Avoidance ⇒

- Current state → ① no. of processes
  ② max need of R. of each process
  ③ Currently allocated amount of R. to each process
  ④ max. amount of each resource

- A state is safe if the system can allocate resources to each process (up to its max) in some order and still avoid DL.

- Safe state can be achieved only if a safe sequence exists

⊖ Banker's algorithm ⇒

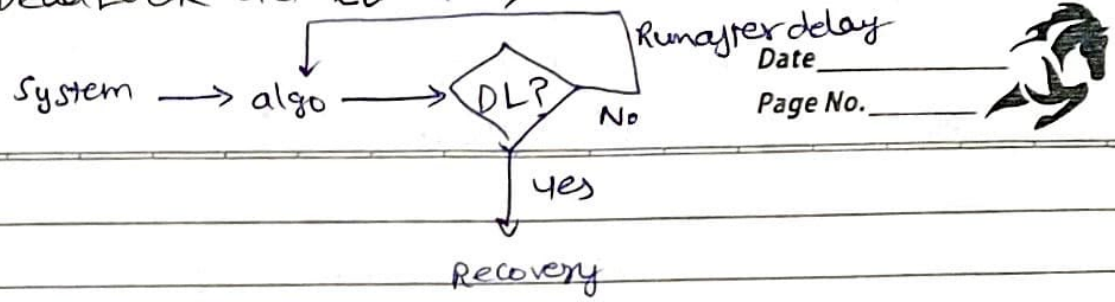
P no	Allocated			max need			available			Remaining need		
	A	B	C	A	B	C	A	B	C	A	B	C
P_1	0	1	0	7	5	3	10−7 = 3	5−2 = 3	7−5 = 2	~~7~~ 4	~~5~~ 3	~~3~~
P_2	2	0	0	3	2	2	allocated +2, +0, +0 → 5+2, 3+1, 2+1			~~1~~ 2	~~2~~ 2	
P_3	3	0	2	9	0	2	7	4	3	~~6~~ 0	~~0~~ 0	
P_4	2	1	1	4	2	2	7	4	5	~~2~~ 1	~~1~~ 1	
P_5	0	0	2	5	3	3	7	5	5	~~5~~ 3	~~3~~ 1	
Total already	7	2	5				10	5	7			

Total ⇒ A ⇒ 10
B ⇒ 5
C ⇒ 7

→ calculate safe sequence for safe state

$P_2 \rightarrow P_4 \rightarrow P_5 \rightarrow P_1 \rightarrow P_3$ → Safe sequence

- Deadlock detection ⇒



Date _____
Page No. _____

See/ Read rest of the session online

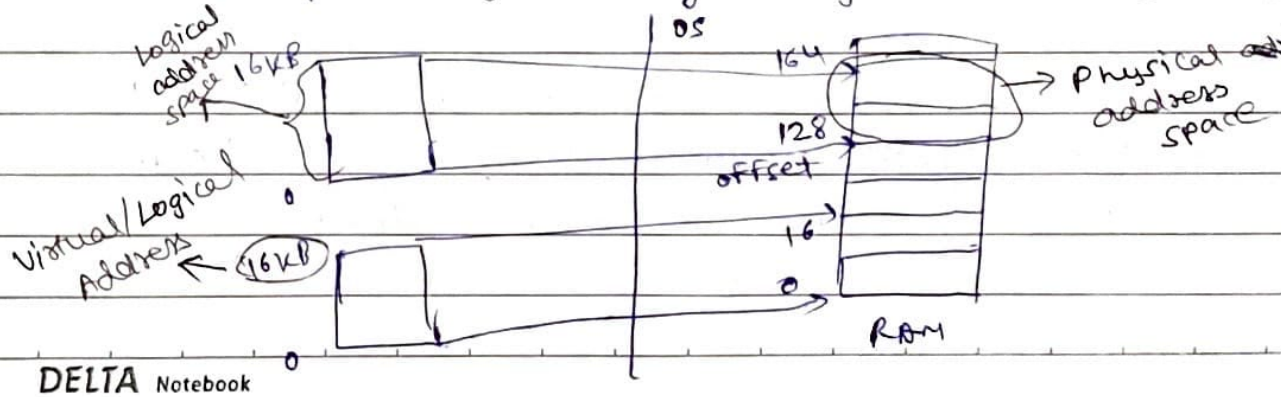
LECTURE 23

Code only

LECTURE 24

- Memory Management in OS
- Logical Address ⇒
  - An address generated by CPU
  - User has indirect access to physical address through logical address
  - Logical address doesn't exist physically. Hence, Virtual address.
  - Set of all logical addresses generated by any program is referred to as Logical Address space.
  - Range 0 – max

- Physical Address ⇒
  - An address loaded into the memory-address register of the physical memory.
  - User can never access physical address of the program.
  - It is computed by Memory Management Unit (MMU).



- The runtime mapping from virtual to physical address is done by a hardware device called Memory Management Unit (MMU).

Date _____
Page No. _____

Q) How OS manages isolation and protect? (Memory Mapping and Protection)

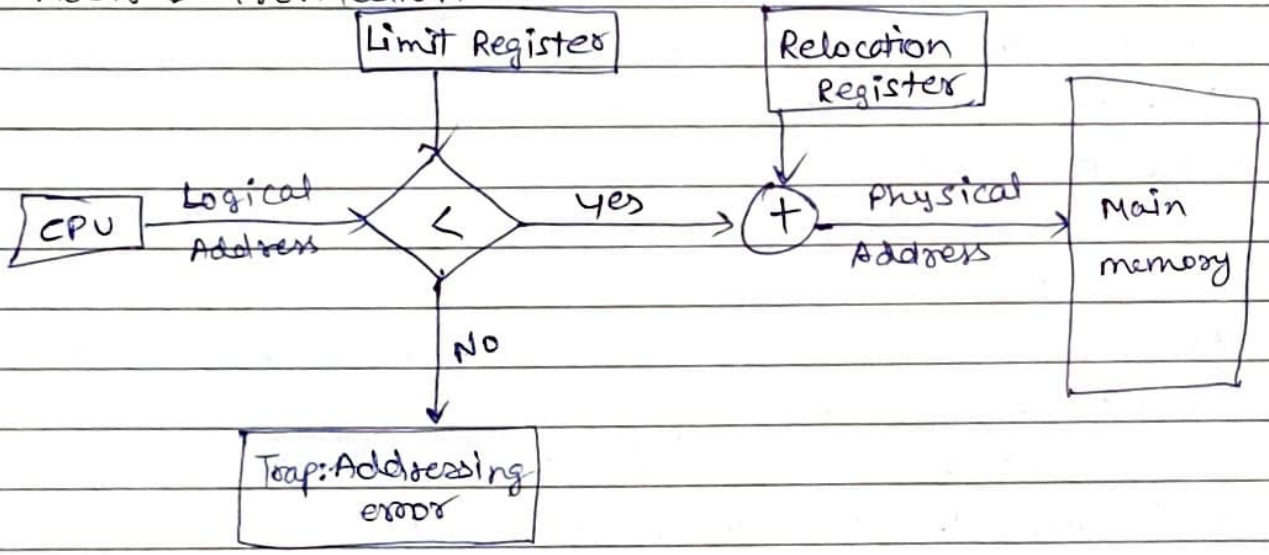
A) a) OS provides Virtual Address space

b) To separate memory space, we need the ability to determine the range of legal addresses that the process may access and to ensure that the process can access only these logical addresses

c) The relocation register contains value of smallest physical address (Base address [R]), the limit register contains the range of logical addresses.

d) Each logical address must be less than the limit register.

Address Translation ⇒



- When CPU scheduler selects a process for execution, the dispatcher loads the relocation and limit registers with the correct values as part of context switching.

Allocation Method on Physical Memory ⇒

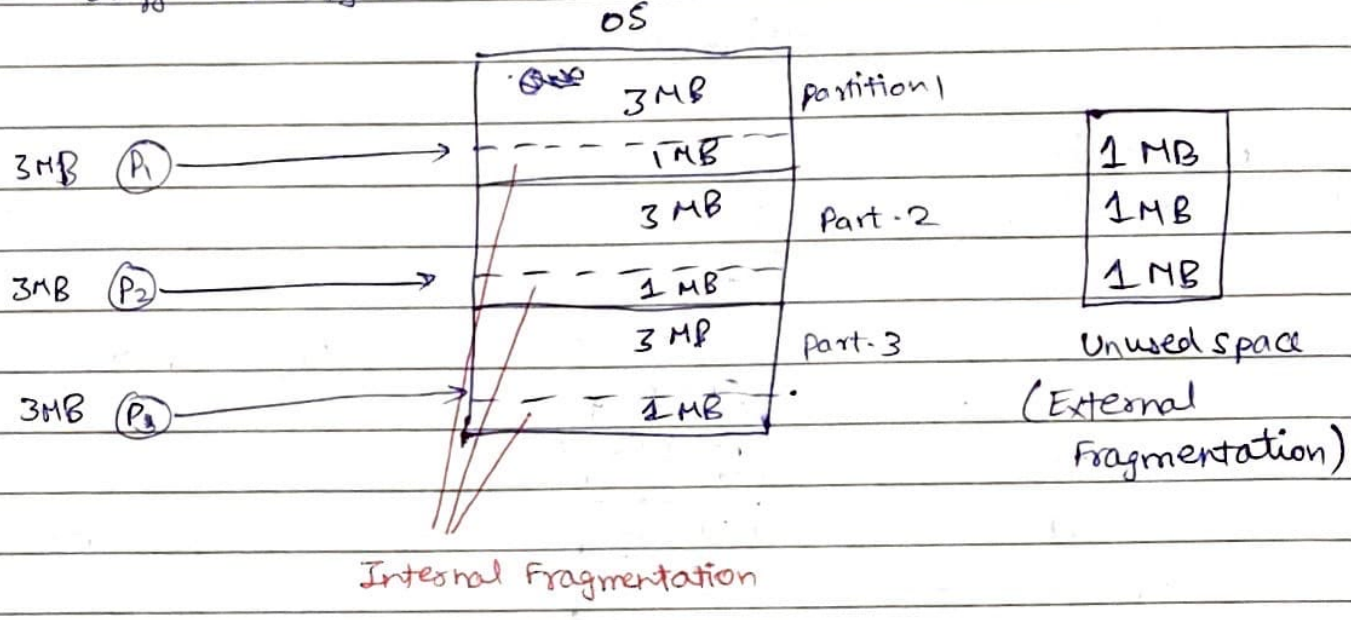
① Continuous Allocation ⇒

Date _____
Page No. _____

- Each process is contained in a single contiguous block.

Fixed partioning ⇒

i) Main memory is divided into partitions of equal or different sizes.



Internal fragmentation ⇒

If the size of the process is lesser than the total size of the partition then some size of the partition gets wasted and remain unused.

External Fragmentation ⇒

The total unused space of various partitions cannot be used to load the processes even though there is space available but not in the contiguous form.

Limitation on process size ⇒

- Max size of process should be <= size of max. size partition.
- Can't club two partitions for allocation to a process.

Low degree of multi programming ⇒

Because the size of partition can't be varied according to size of processes.

Dynamic partitioning ⇒

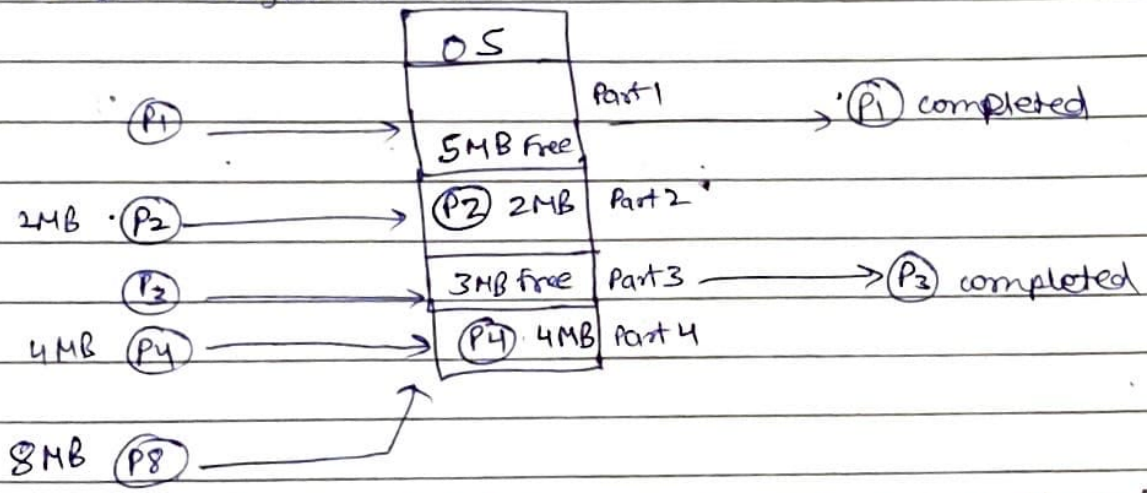
- Partition size is not declared initially. Date _____ Page No. _____
  It is declared at the time of process loading.

Advantages over fixed partitioning ⇒

1. No internal fragmentation
2. No limit on size of process
3. Better degree of multiprogramming

Limitations ⇒

1. External Fragmentation.



P8 can't be loaded into memory even though there is 8 MB space available but not contiguous.

LECTURE 25

Defragmentation / Compaction ⇒

- OS uses Linked List type data structure to keep trace of free space.
- Defragmentation means, more free space to the same side and allocated space to the other side.
- User doesn't need to know about defragmentation because OS takes care of it with help of Logical addresses.

Limitations →

Date
Page No.

- The efficiency of the system is decreased in case of compaction since all the free spaces will be transferred from several places to a single place.

Q) How free space is stored/represented in OS?

A) Free holes in the memory are represented by a free list (Linked List)

Q) How to satisfy a request of size n from a list of free holes?

A) Various algo which are implemented by OS in order to find out the holes in linked list and allocate them to the processes.

① First Fit
- Allocate the first hole that fits.
- Simple and easy
- Fast/Less time complexity

② Next Fit
- Enhancement of first fit but starts search always from last allocated hole.
- Same advantages of First fit.

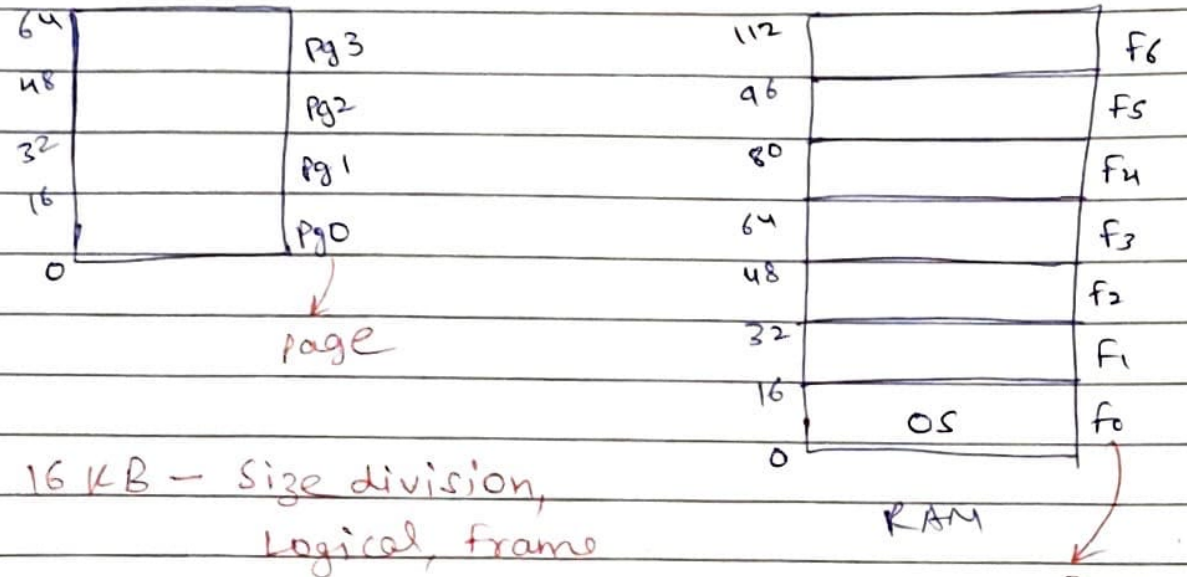
③ Best fit.
- Allocates smallest hole that is big enough.
- Slow
- Less Internal Fragmentation.
- High External Fragmentation

④ Worst Fit.
- Allocates the biggest hole that is big enough
- Slow
- High internal fragmentation.
- Less external fragmentation

LECTURE 26

Date
Page No.

Paging (Non contiguous) ⇒



16 KB – Size division, Logical, frame

Page → Logical address, fixed size division

page size = frame size

Depends on processor architecture.

- Paging is the process of division of process into fixed size page. and division of RAM into fixed size Frames.

Page Table → present inside memory, every process has its own page table

Logical page no.	Physical frame no.
Pg 0	frame 3
Pg 1	f 7
Pg 2	f 5
Pg 3	f 2

Logical address → Process → 64 bytes

$2^6 = 64$

To identify 64 bytes, we need 6 bits

example ⇒ 25th byte = 01 | 1001
page no (p) offset (d)

Date
Page No.

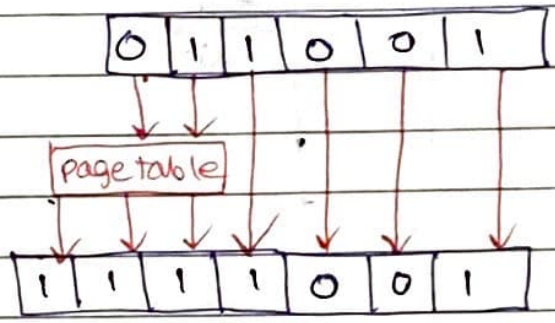
16 + 9

~~Logical~~

Similary, since frame size is also of 16 size.
If 25th byte is ~~at~~ in frame 8th (f7).
then the 25th byte should be at 112 + 9 = 121st byte

121 ≅ 111 | 1001 offset (d)
frame no. (f)

Address Translation ⇒



Page Table contains the base address of each page in the physical memory.

- Page Table is stored in main memory at the time of process creation and its base address is stored in Process control Block (PCB)

- A page table base register (PTBR) is present in the system. that points to the current page table. changing page tables requires change in only this register at time of context switching.

- Non-contiguous allocation of the pages of the process is allowed in the random free frames of the physical memory.

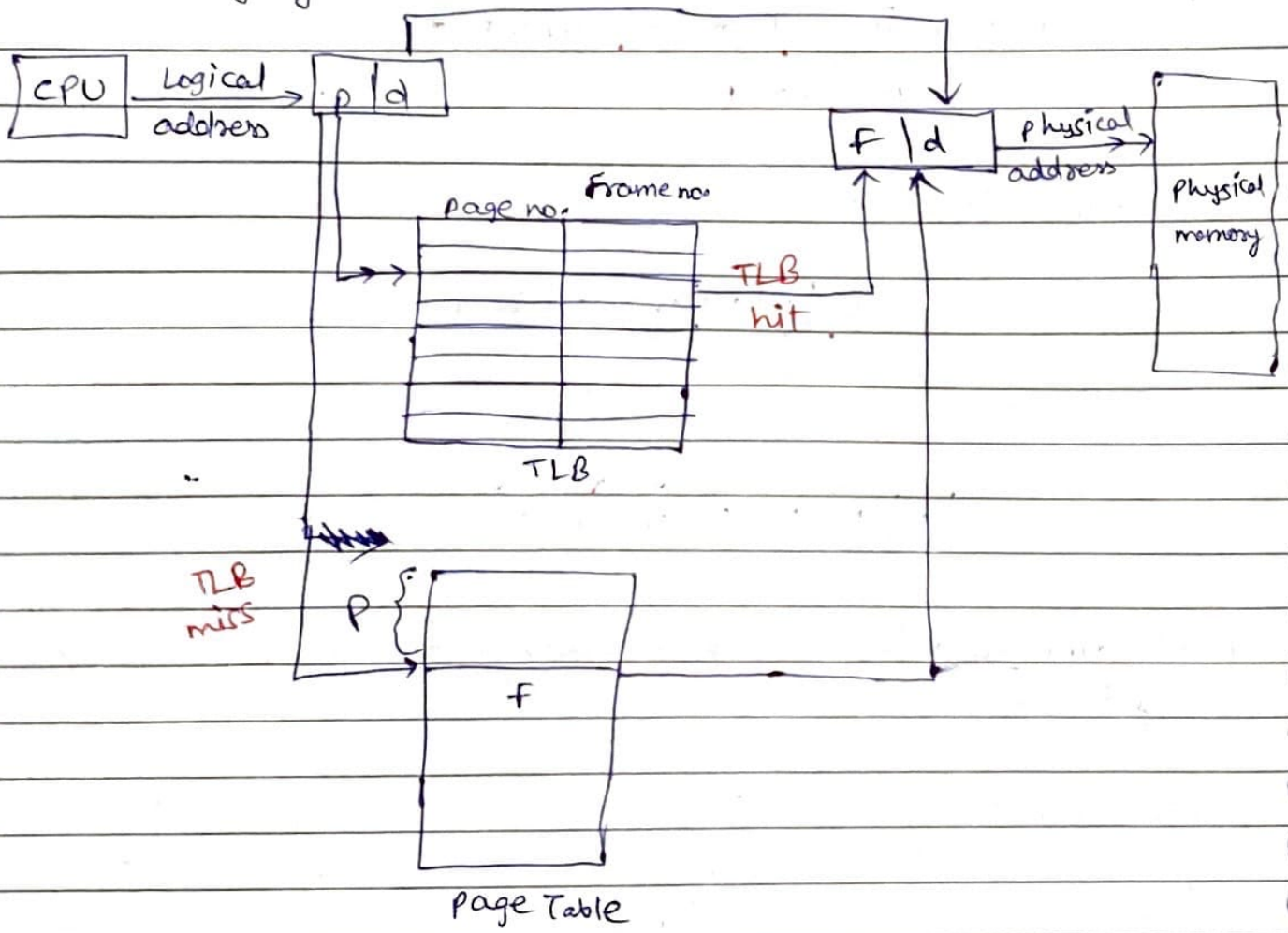
Q) Why paging is slow and how do we make it fast?

A) Translation Look-Aside Buffer (TLB)

Date _____
Page No. _____

- A hardware support to speed up paging process
- It's a hardware cache, high speed memory
- TBL has key & value.

Paging hardware with TLB



Q) What happens to TLB after context switching?

A) We can do →

① Flush TLB (Not Efficient)

② Unique identifier that will identify unique process.

Address Space Id

ASId	Pg no.	F. no.
0	10	100
1	10	45

- Address Space Identifiers (ASIDs) is stored in each entry of TLB. ASID uniquely identifies each process and is used to provide address space protection and allow to TLB to contain entries for several different processes.

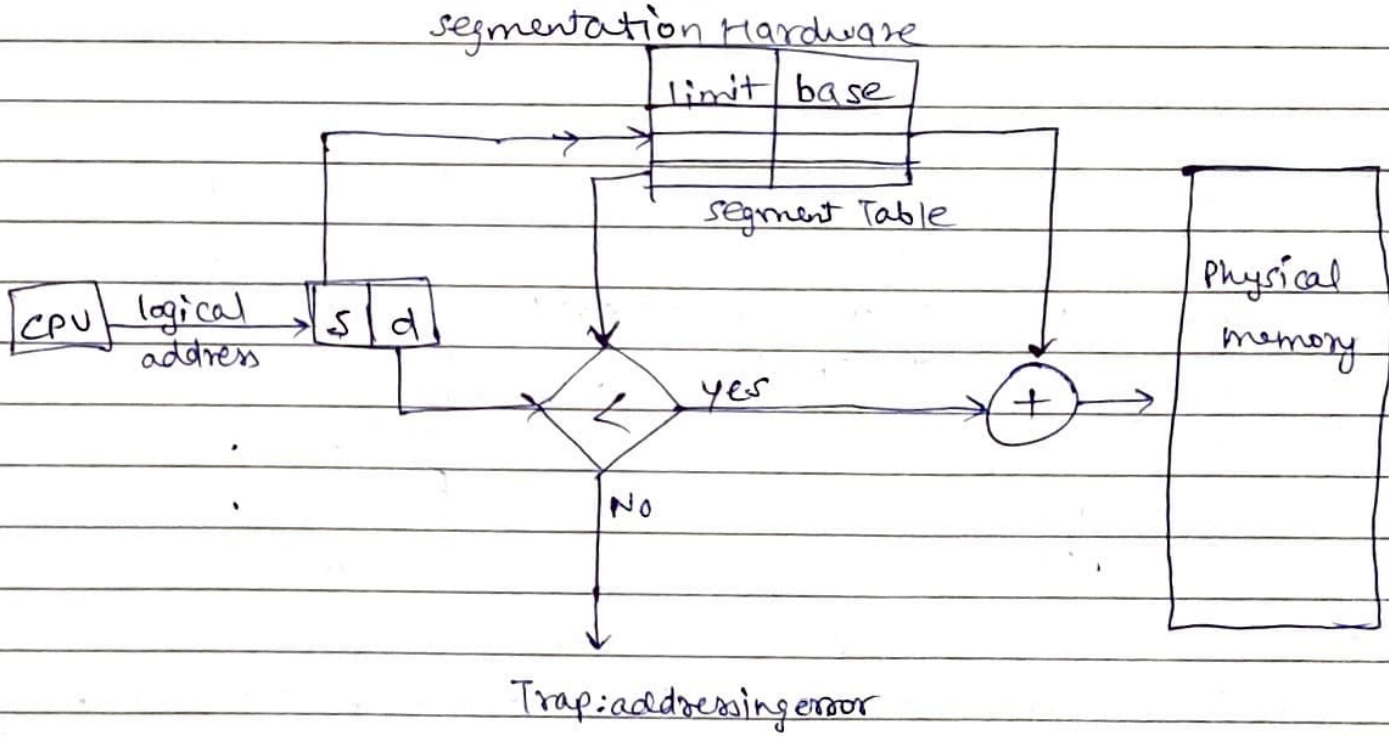
LECTURE 27

Segmentation ⇒

Since, paging can break a function into multiple pages, this can lead to some inconsistency or performance degrade. To solve this, segmentation does variable partitioning of logical address space.

- Compiler to user program's responsibility to break program into segments.

Q) MMU Translation?



Advantages ⇒

1. No internal fragmentation
2. One segment has a contiguous allocation, hence efficient working within segment.

③ The size of segment table is generally less than the size of page table.

Date
Page No.

④ Results in a more efficient system because the compiler keeps the same types of functions in one segment.

⊖ Disadvantages ⇒

① External fragmentation

② Different size of segment is not good at time of swapping.
Most common segments are ranked higher & the rest are swapped-in from swap memory.
swap memory

- Modern system architecture provides both segmentation & paging implemented in some hybrid approach.

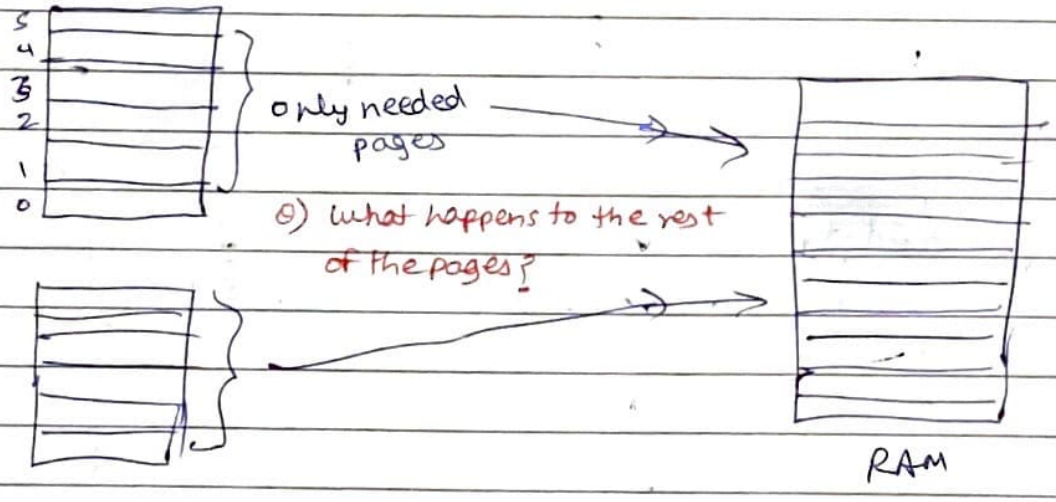
LECTURE 28

⊖ Virtual Memory Management ⇒

① Physical Memory (

② Logical Memory

③ Virtual Memory ⇒



Q) what happens to the rest of the pages?

A) Rest of the pages are stored in reserved space of HDD.
swap space

RAM + Swap space = Virtual memory

- Virtual memory is a technique that allows the execution of processes that are not completely in the memory.
- Creates an illusion of having a big main memory.
- Instructions have to be executed in physical memory (RAM)

Demand paging ⇒
- In demand paging, the pages of process which are least used, get stored in the secondary memory.
- There are various page replacement algorithms.
- Pager is responsible for swapping-in/out pages.

Q) How Demand paging works?
A) ① When a process is to be swapped-in, the pager guesses which pages will be used.

② Instead of swapping in whole process, the pager brings only those pages into memory.

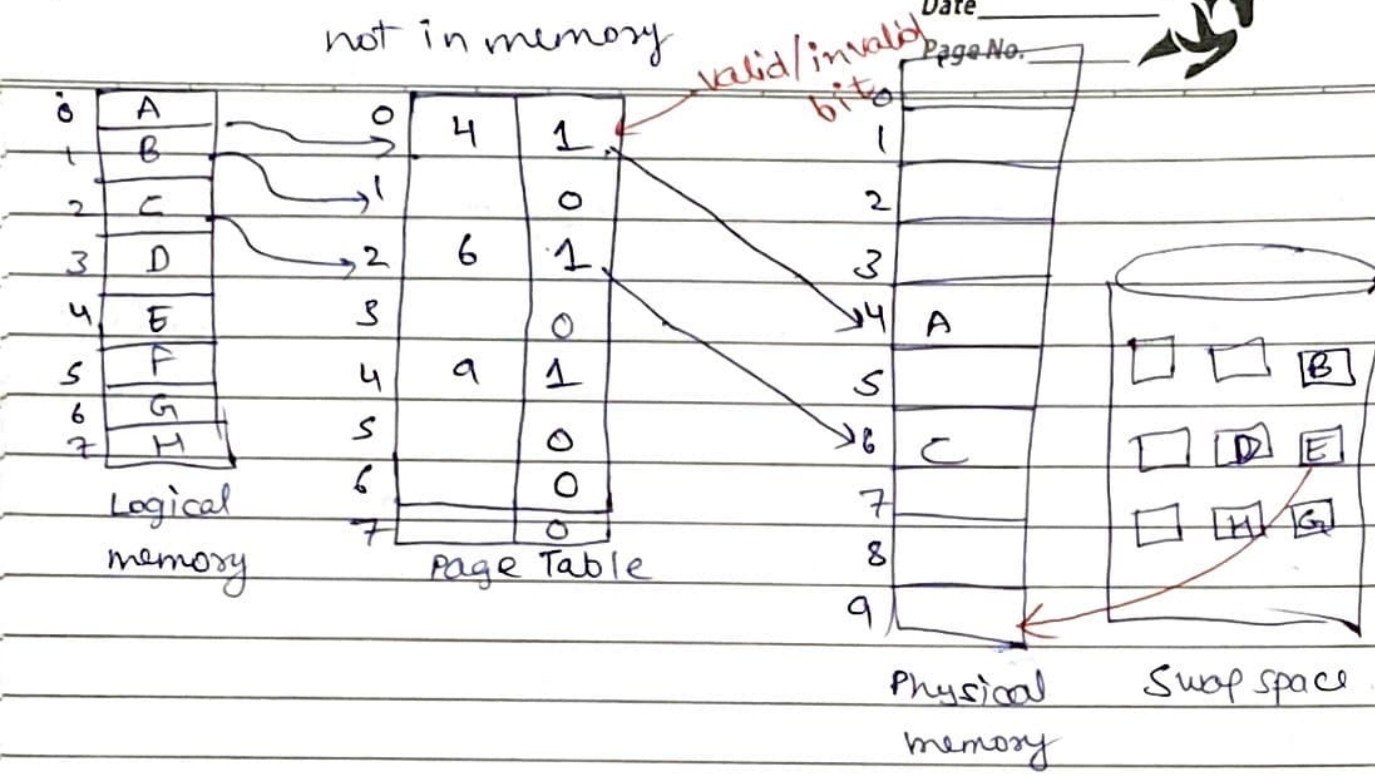
③ Above way, it decreases the swap time and the amount of physical memory needed.

④ Valid-Invalid bit scheme in the page table is used to distinguish b/w pages that are in memory and that are on disk.

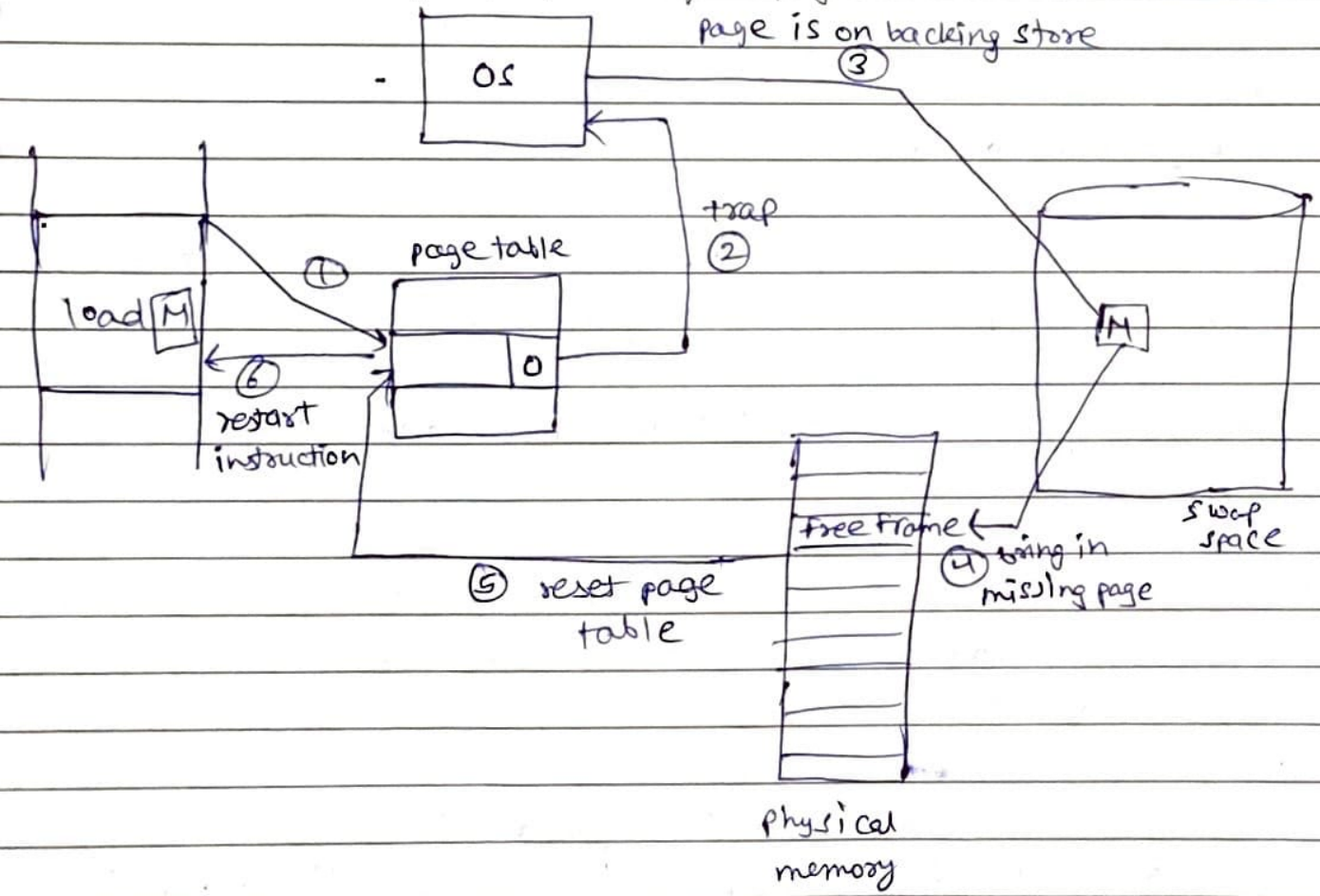
1 → valid bit → both legal and in memory
0 → invalid bit → page either not valid (not in the LAS of the process) or is valid but is currently on disk

Logical address space

Page table when some pages are not in memory



Steps in handling a page Fault



Date
Page No.

⊖ Pure Demand paging ⇒

- In extreme case, we can start executing a process with no pages in memory.
- Never bring a page into memory until it is required.
- Obviously, this will lead to a lot of overheads. So, to solve this, we use locality of reference to bring out reasonable performance from Demand paging.

Dis?

⊖ Advantages of virtual memory ⇒

① The system can become slower as swapping takes time.

② Thrashing may occur.

LECTURE 29

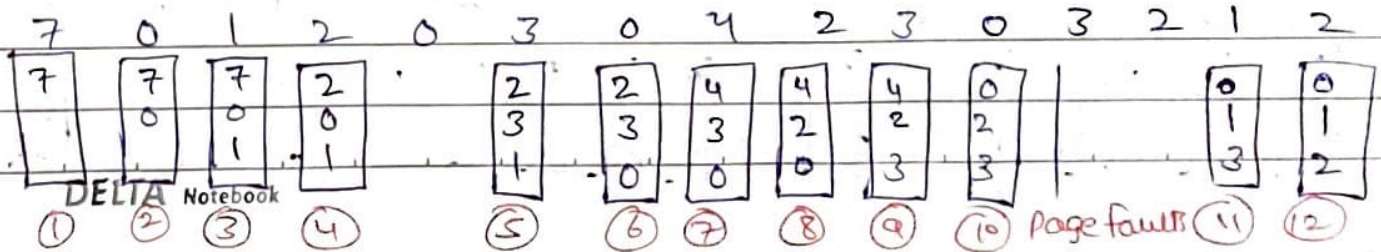
⊖ Page replacement algos ⇒

⊖ Page faults ⇒
swapping in a page because page doesn't exist in RAM.

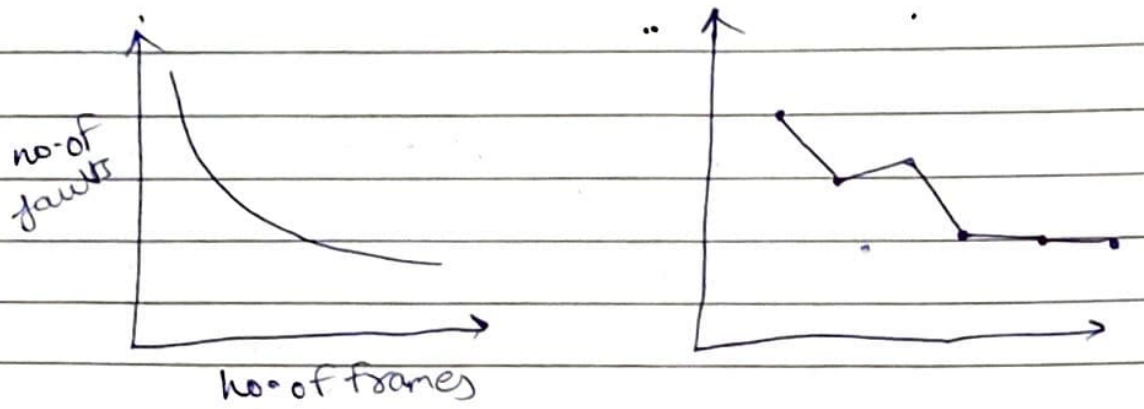
⊖ Page fault service Time

① First In First Out (FIFO) ⇒

- Allocate frame to the page as it comes into the memory by replacing the oldest page.
- Easy to implement.
- Performance not good.



- Page which is in heavy use shouldn't be replaced.
- Belady's anomaly is present ⇒



example ⇒ 1 2 3 4 1 2 5 1 2 3 4 5

2 → 12 (P.F.)

3 → 9

4 → 10

5 → 5

6 → 5

(2) Optimal Page Replacement (OPR) ⇒

(1) Best

(2) Pg-faults is minimum

(3) Almost impossible to implement.

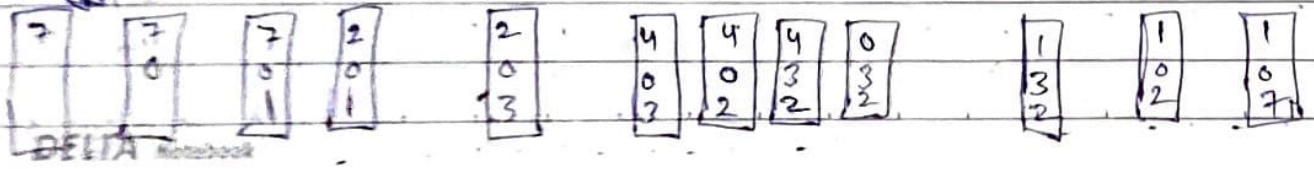
- Q. Find a page that is never referenced in future. If such a page exists, replace this page with new page

If no such page exists, find a page that is referenced farthest in future. Replace.

(3) Least Recently Used (LRU) ⇒

(1) Use recent past as an approximation of the near future then we can replace the page that has not been used for the longest period.

7 0 1 2 0 3 0 4 2 3 0 3 2 1 2 0 1 7



Can be implemented in 2 ways ⇒

Date _____
Page No. _____

① Counters

- Associate time field with each page table entry.
- Replace the page with smallest time value.

7 0 1 2 0 3 0 4 2 3

Universal counter = 0

0 → 5
1 → 6
2 → 4
3 → 6
4 →
7 →

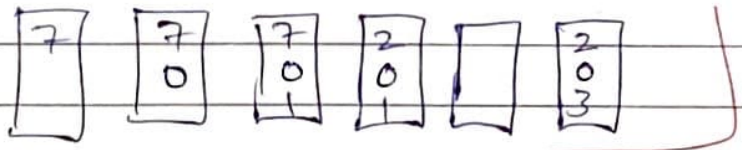
Replace the page with least counter value

② Stack ⇒ (Actual Implementation is done using Hashmap & queue → Doubly Linked list)

- Keep a stack of page number.
- Whenever page is referenced, it is removed from the stack & put on top.
- By this, most recently used is always on the top, & least recently used is always on bottom.

- As entries might be removed from the middle of the stack, so Doubly Linked List can be used.

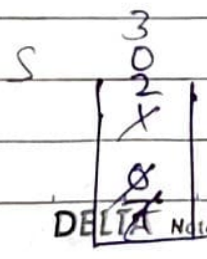
7 0 1 2 0 3 0 4 2 3 – – –



④ Least Frequently used (LFU)

⑤ Most Frequently used (MFU)

Neither LFU & MFU replacement is common

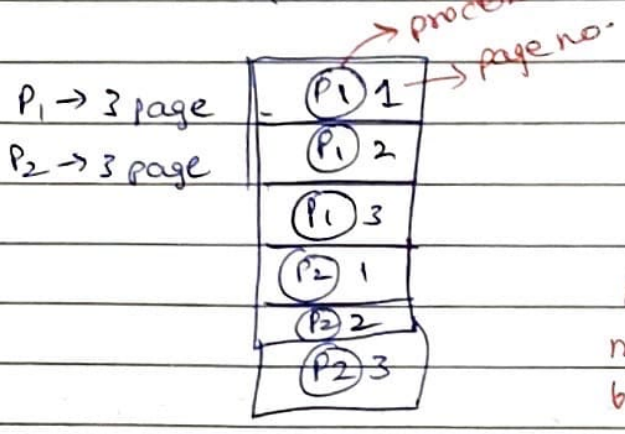


LECTURE 30

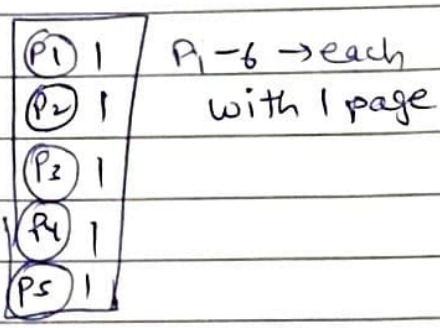
Date
Page No.

Thrashing ⇒

Situation 1

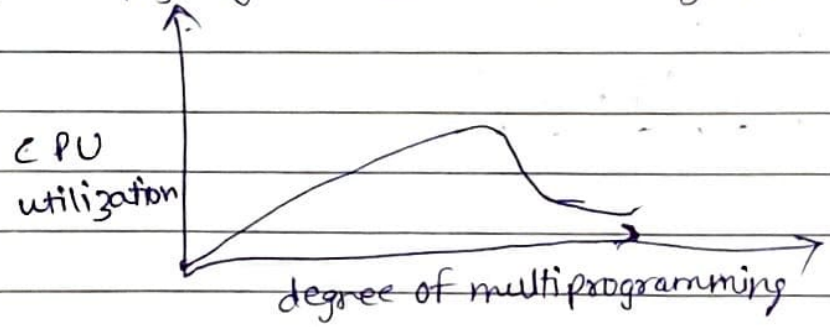


Situation 2



High degree of multithreading, but Thrashing

- situation 2 will have more no. of page faults.
- CPU will be more busy servicing page faults.
- If the process doesn't have the no. of frames it needs to support pages in active use, it will quickly page fault. At this point, it must replace some page. However, since all its pages are in active use, it must a page that will be needed again right away. Consequently, it quickly faults again, and again, and again, replacing pages that it must bring back in immediately.
- This high paging activity is called Thrashing.
- A system is Thrashing when it spends more time servicing the page faults than executing processes.



When less no. of processes are in RAM, degree of mult. is low, and if these processes go to do some I/O, CPU utilization is low.

Date _____
Page No. _____

- Cause of Thrashing ⇒

① Initial Low CPU utilization --- High degree of multip.

② A global page replacement algo, replaces pages without regard to the process.

③ A process may need more frames --- cause faults again.

④ Other processes frames are replaced & they may need those soon.

⑤ As a result CPU utilization decreases.

⑥ CPU schedular now, may increase CPU util. by increasing degree of multip.

⑦ Ultimately, CPU util. drops drastically.

- How to handle Thrashing ⇒

① Working set Model ⇒

- This model is based on the concept of the **Locality** model.

- If we allocate enough frames to a process to accomodate its current locality, it will only fault whenever it moves to some new locality.

② Page Fault Frequency ⇒

- Thrashing has a high page-fault rate.

- We want to control the page fault rate.

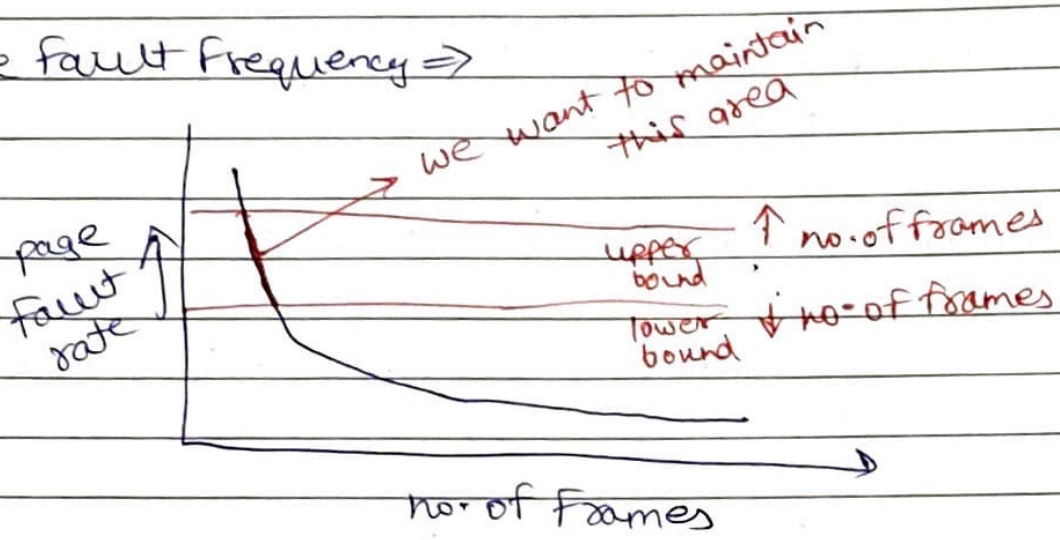
- When it is too high, the process needs more frames. Conversely, if the page fault rate is too low, then the

process may have too many frames.

Date _____
Page No. _____

- we establish upper & lower bounds on the desired page fault rate.

- Page fault Frequency ⇒



⊖ Storage Management ⇒

Self Learning