Page No:
Date: / /

* Operating System [Interface b/w user and hardware]

- OS why? → Resource Management

- OS → software that helps user to make use of hardware
  - → without OS (hardware data can not be used)
  - → software abstracting hardware
  - → (OS shows beautiful view of ugly hardware)
  - → set of utilities to simplify application development/execution (provides some already built fun^{n} that helps software developers)
  - → controls program
  - → acts like a government

Course Structure
1. Introduction
2. Process Management
3. CPU Scheduling
4. Threads and Multi Threading
5. Process Synchro. and Deadlock
6. Memory Management
7. Virtual Memory
8. file System

Services of O.S

1. User Interface
2. Program Execution [Most Imp - use of OS] - Minimum requirement of OS
3.
4. I/O operation [
5. File System Manipulation [Create, Rename, cut, paste folder]
6. Communication (Inter-process communication)

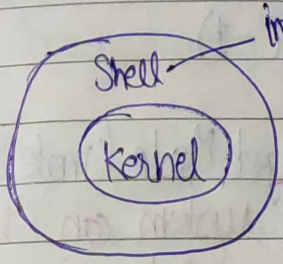
eg [word send request to print to document the communication is done via OS]

7. Error detection
8. Resource Allocation } Everything given in
9. Accounting } Task Manager
10. Protection and Security (firewall, password, ensure process does not interfere each other)

Goals of OS

1. Convinience (User-friendly)
2. Efficiency
3. Portability { Same OS can run on multiple types of devices }
4. Reliability
5. Scalability [To get new update no need to uninstall entire OS, but updation can be done on already installed OS]
6. Robustness [To tackle unexpected error]

Parts of OS



interface, frontend

Kernel → Core part of OS
→ contain all functionalities of O.S. be it way to solve errors, resource management, security etc

Shell → no functionality present
→ provide an interface
→ just frontend to use the functionalities of OS

GUI { Graphical User Interface }

CLI { Command Line Interpreter }

Page No:
Date: / /

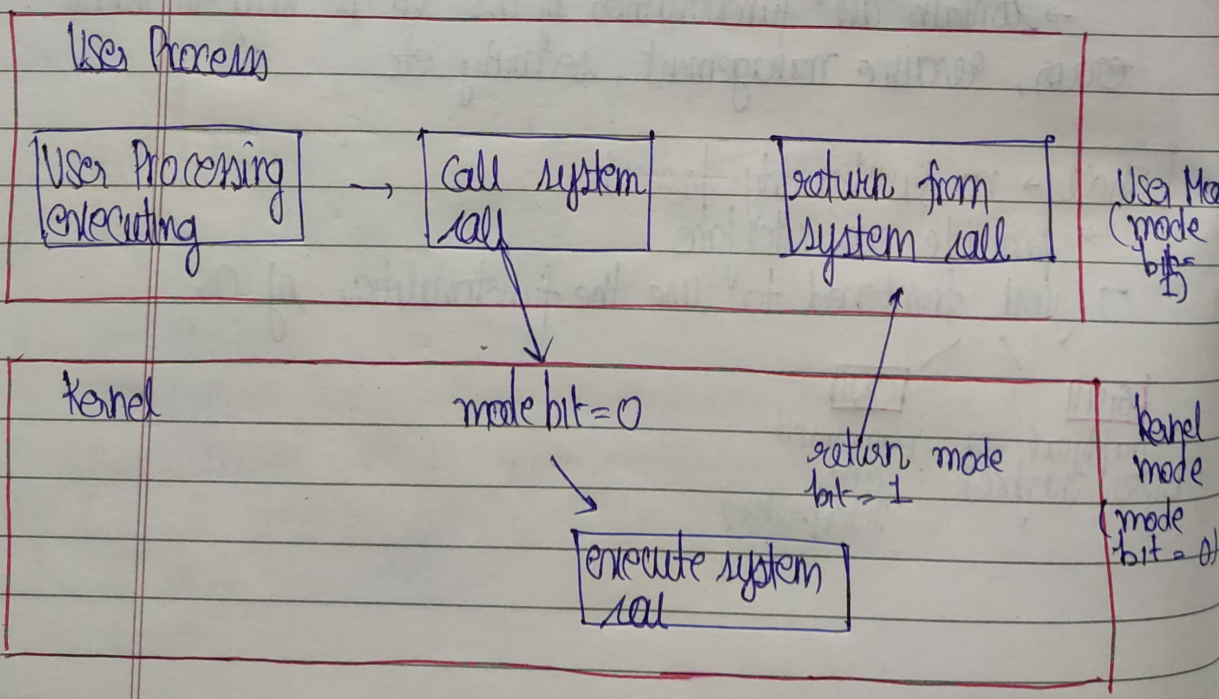
System Call
- is a way for programs to interact with the OS
- without system call can not use the functionality of OS

- Kernel - contain funct^n of each and every operation
  - to perform a operation we need to make a system call

Dual Mode of Operation
→ protection implemented using this
→ some operation can be performed by user / process while some privileged one can only be done using O.S

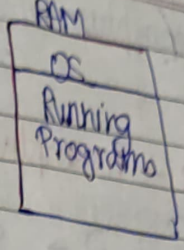
2 modes
① User Mode (mode bit = 1)

② Kernel / System / Supervisor / Privileged Mode (mode bit = 0)
{ If mode bit = 0 then comp system can use the privileged operation under the supervision of O.S }

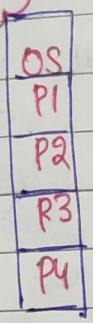


Types of operating systems

1. Uni-programming OS → Apart from OS only one process to reside in the main memory
   - Single process can not keep CPU and I/O devices busy simultaneously (because it can use either of the devices) - resources not used fully
   - Not a good CPU utilization



Multi-programming OS
- Apart from OS, it allows multiple processes to reside in main memory



→ If no of processes increases, the CPU utilization will increase (Better CPU utilization than uni-programming)

→ Degree of Multiprogramming → No. of running programs (process) in main memory (RAM)

- As the degree of multiprogramming increases, CPU utilization also increases (but upto a certain level)

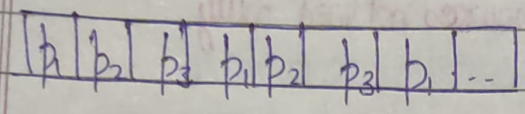
Types of Multiple-programming OS

① Non Preemptive — a process runs on CPU till it's wish
   - when either process terminates
   - process goes for any I/O operation

2. Preemptive - A process can be forcefully taken out of CPU in preemptive system.

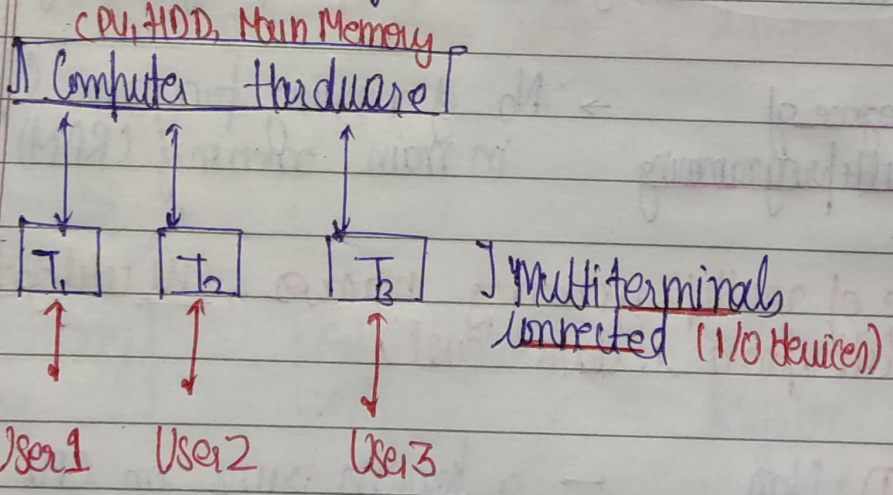
Note - Preemptive system not present in Uni-programming OS

Multi-Tasking OS / Time Sharing OS → extension of multi programming OS in which process execute in round-robin fashion



CPU executes so fast in round-robin fashion that the we can not think that the switching is even taking place (it seems everything taking place in ll fashion)

Multi-User OS → allow multiple users to access single system simultaneously



Note Windows - not multi-user facility
Linux / Unix - have multi-user facility

while types of data

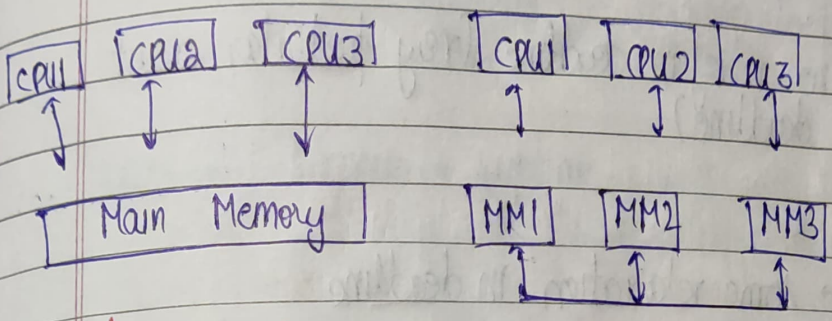
.zon

Page No:
Date: / /

Multi-Processing OS → This OS used in computer system with multiple CPUs.

Types

- **Tightly coupled** (Shared-memory)
- **Loosely coupled** (Distributed)



- Will have common memory
- easy to handle

Embedded OS → Say to make (Refrigerator/AC) intelligent, need to embeed a computer in it and that computer contains embedded OS

→ designed for specific purpose, to increase functionality and reliability for achieving a specific task.

→ User interaction with OS is minimum in the particular OS

Page No:
Date: / /

- Real-Time OS → Real time operating system (RTOS) are used in environments where a large no of events, mostly external to the computer system, must be accepted and processed in a shorter time or within certain deadlines

  → eg Used in rocket launching
  → every process has a deadline

Types ① Hard RTOS → no relief in deadline (very particular about deadline)

② Soft RTOS → some relaxation in deadline

Handheld device OS

- OS in handheld devices

eg IOS
Android

Datawarehouse based on dimensional modelling.

CLASSTIME | Pg. No.
Date / /

Database

Page No:
Date: / /

can use as data structure

{Process Management}

Process → program under execution

→ program + run-time activity = Process

Analogy → Program - Standing Bus
Process = Standing Bus + Conductor / Fuel / Passenger

Process = Program (code) + runtime activity
Program (code) ⇓ Instructions
runtime activity → operands and other info

Process - program under execution
- schedulable / dispatchable unit (CPU)
- Unit of execution (CPU)
- locus of control (OS)
- instance of program

Process as a data structure (how to store data and what operation can be performed on it)

① Definition - code/ only program — compiler while execute this

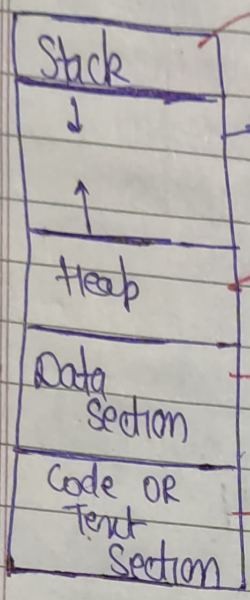
② Representation/ Implementation - how process is actually stored in memory

③ Operations

Created only when funct^n is called and after funct^n execution they get empty

Process

have local variable, funct^n parameters and return addresses



If extra memory needed either for stack/heap

allocate dynamic memory allocation

contain all public and static variables. } motive to put this because the life time of both variable is lifetime

all instructions in this section

Note: The position of the elements in process might change with the OS, but there will be certainly 4 sections.

③ Operations on a process

1. Create (Resource Allocation)
2. Schedule, Run
3. Wait / Block
4. Suspend, Resume
5. Terminate (Resource Deallocation)

④ Attribute of a process

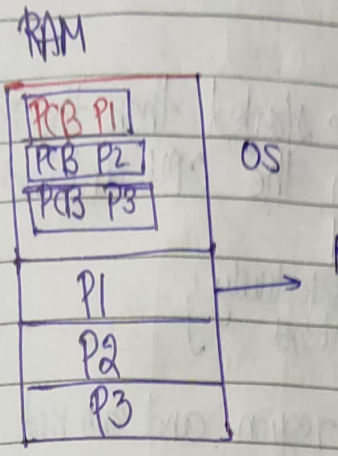
1. PID — Process ID, uniquely identify a process
2. PC — Program counter
3. GPR — General Purpose Register
4. List of Devices
5. Type of process
6. Size of process
7. Memory Limit
8. Priority
9. State
10. List of files

Content of PCB of a process collectively known as context

definition → Representation → Operation → Attribute of process etc.

Date: / /

* All attributes stores in PCB (Process Control Block) PCB remain in control of OS because through PCB, the program / process will controlled by OS

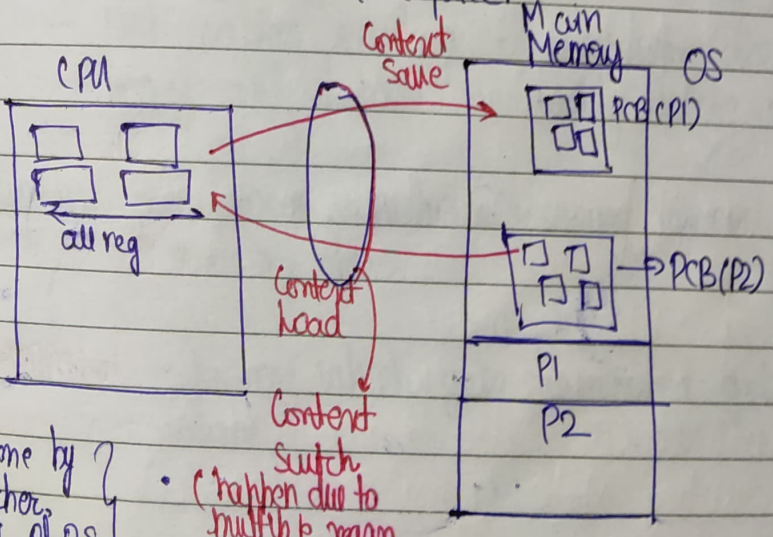


attributes)

Through PCB (contain all detail of a process), OS controls the process

- When suppose we need to pause an process and then a while we CPU need to start it (Resume it), then with the help of values of PC, GPR and other values, these values will be stored in PCB of that process and then resume can take place.

PC



Content Switch

- (happen due to multiple reason but done by OS only)
- Stop a running process and run other process

It is done by dispatcher a part of OS

Page No:
Date: / /

Context Save - Suppose P1 is paused, then the current values of attributes of a process which is in CPU will be saved in the PCB (P1) this is called context save.

Context load → When a process needs to be started then the PCB value need to be loaded in the CPU and that process called context load
{content of PCB collectively known as context}

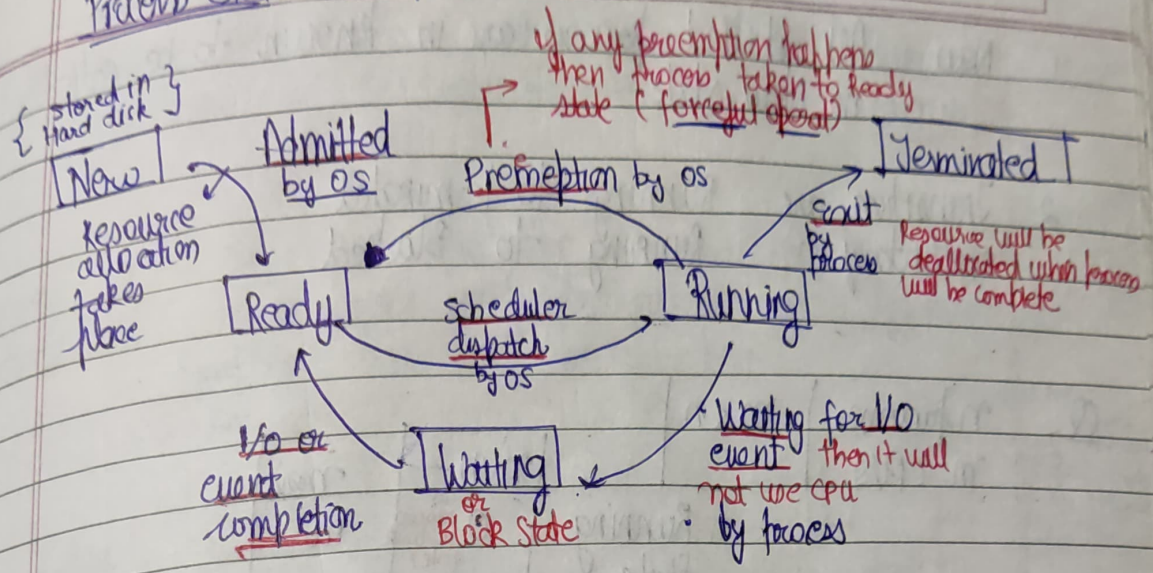
Note · PCB stored in OS, protected region and can not be accessed by even the process

· set of instructions to be executed on a comp called a program

↳ forcefully taking process out of CPU

Page No.
Date: / /

Process State : Preemptive



1. New → When you install a application, and is not running currently then it is called that they are in new state
   - store in hard disk

2. Ready → All those process which are ready to run in CPU is called ready state
   - Not called running because some othe process is running and process in ready state need to wait

3. Running → process whichever running in CPU have the running state

4. Terminated → process which gets terminated gets terminated state

5

1. New : All installed process are in known to be called in New state

2. Transition are voluntary : Running to terminated
Running to Blocked

Q. n process | m < n
m CPUs

	min	max
1. Running State	0 (can process in blocked state)	m
2 Ready State	0	n
3 Blocked State	0	n

CPU vs IO Bound Process

1. CPU Bound Process → A process which spend more time on CPU called CPU bound process
→ eg
{Anti Virus Scanner
Code Editor}

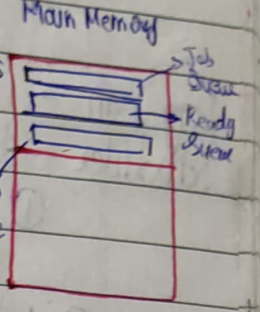
2. IO Bound Process → If process is intensive in terms of IO operations
→ eg : Watching movie
: Listening music
It will use CPU also because from running state to waiting state transition done only if the process was already in running state (using CPU)

Page No:
Date: / /

Note • Process which has just terminated but has to relinquish it's resources called Zombie Process

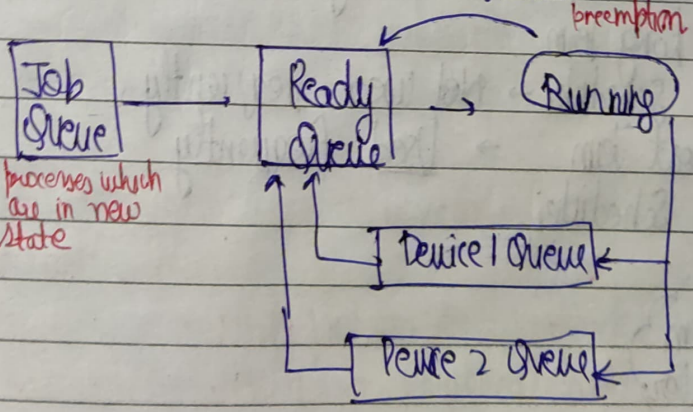
Process Scheduling → Needed so that the resources can be used to a maximum level (better resource utilization)
→ Response time will be effective
→ done by OS

Scheduling Queues
1. Job Queue
2. Ready Queue
3 Device Queue
} these will be present in OS



- Job Queue (Queue which are in new state) - processes which are in new state remain in Job Queue
- Ready Queue - processes which are in ready state
- Device Queue - processes which are waiting for a specific device
  - each device will have unique device queue ~~else~~ जितने device उतनी Queue

- Cycle



Page No:
Date: / /

Note - All queues will have PCBs (Process Control Block) of process

Types of Schedulers
1. Long Term Scheduler (Job) new → ready
2. Short Term Scheduler (CPU) ready → running
3. Mid-Term Scheduler (Medium-Term)

Long Term Scheduler → 1. bring process from new to ready state
Initiated in two ways
① When user double taps on any application (basically opening a software)
② When CPU bit free, then long term scheduler on it's own start some background OS operations

Resource Allocation happens

Short Term Scheduler
- selects one of all ready process to run of on the CPU

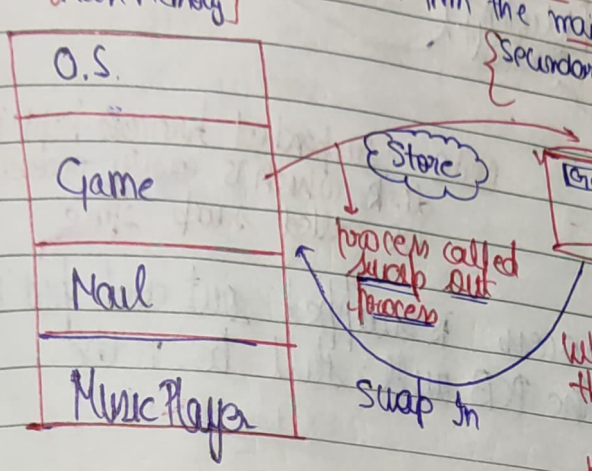
Note
Have names because of their their usage
Long Term Scheduler → Not used frequently
Short Term Scheduler → Used frequently

Mid-Term Scheduler

Page No:
Date: / /

Wide types of data

- When the space in main memory (RAM) gets finished and need more memory to open new application, then the program which is inactive state for long period of time will be saved in hard disk (Secondary Memory) and removed from the main memory



- Mid-Term Scheduler (Medium Term) performs the swapping (both swap in and swap out).

Imp Swapping is known as rolling when the process takes out based on priority.
least priority program will be removed first

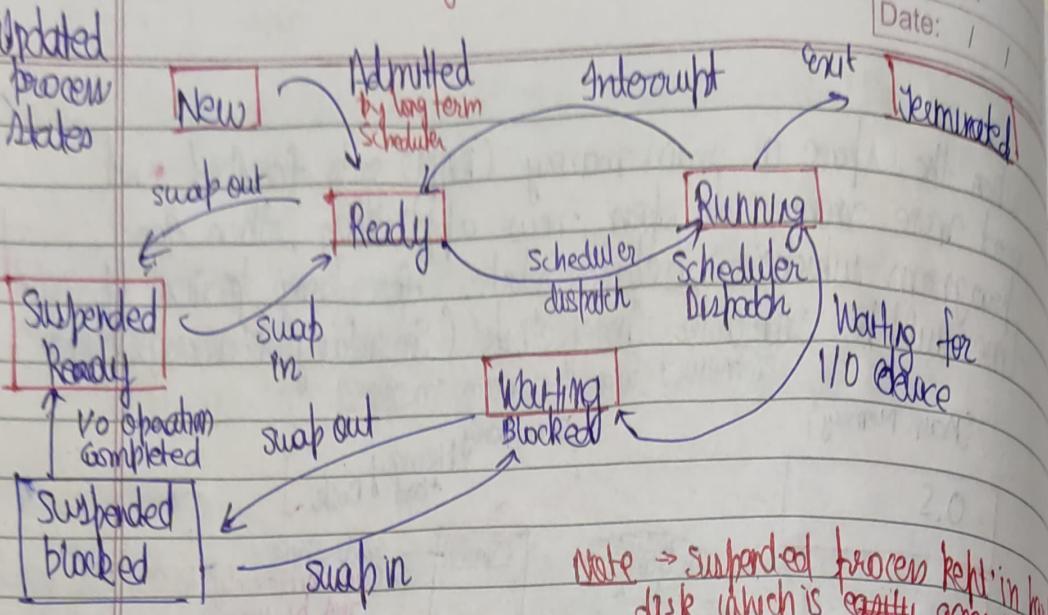
- When swap out takes place, the process saved in that part of Hard disk which is accessible by the OS only but not by the user. The place called Swap space

- If the process is swapped out then the process is said to be in suspended state

Imp Note:- Running program can not go in suspended state, only from ready and waiting (blocked) state the process can go to suspended state

Either from ready or waiting block then process can go in Suspension block

Page No:
Date: / /



Note → Suspended process kept in hard disk which is easily accessible by OS called swap space

Note- Even the when the process is swapped out and put in secondary memory, the PCB will still be in the OS

- Swap out also known as Sleep
  Swap in " " " resume

- Program selected from ready to running by the short-term scheduler but the process is done by dispatcher (ready to CPU)
  context switch also by dispatcher

CPU Scheduling → funct^n
Makes a selection one of of the process selected from ready state to run on CPU

Goal of Scheduler
→ Minimize wait time and turn-around time
→ Maximize CPU utilization (Throughput)
→ Fairness (Programs should be fairly selected)

Page No.
Date: / /

Imp Which scheduler reduces degree of multiprogramming? (No of processes in main memory)

Sol Long Term - controls max degree of programming
Mid Term - reduces degree of multiprogramming

Q Do new process in New state have PCB. If yes where it is stored in OS (main memory) or Harddisk (OS)?

Scheduling Times

1. Arrival Times (AT)
2. Burst Time (BT)
3. Completion time (CT)
4. Turnaround Time (TAT)
5. Waiting Time (WT) → (TAT - BT)
6. Response Time (RT)
7. Deadline (D)
8. Scheduling length (L)
9. Throughput

Arrival Time - time at which the process arrives in the system

Burst Time - amount of time which process runs on CPU (considering no I/O operation)
- how many minutes, seconds ?

Completion Time
- also known as exit time
- when process completes it's execution (time at which process completes execution)
- amount of

Turnaround Time → time from arrival to completion
→ total time

{TAT = Completion time - Arrival Time}

(duration for which process in OS)

eg

Waiting Time → amount of time process is in waiting state
{WT = Turnaround Time - Burst Time}

Page No:
Date: / /

eg. Reach college → 10.00 am → Arrival Time
Leave College - 6.00 pm → Completion Time
Turnaround time → time spend in college → 8 hr (10 to 6)
Burst Time → amount of time u study → Out of 8 hrs spend in college only 5 hrs of study (5 hr → Burst Time)
Waiting Time → Time spend other than study → 3 hrs (in case) in college called the waiting time

6. Response Time → Amount of time from arrival till first time process gets the CPU

8. Scheduling length → Max of completion time - Min (Arrival time)
max (CT) - Min (AT)

9. Throughput → measurement of processes
→ usually in no. of processes / time
→ no. of processes executed per unit time

{ Total processes / Scheduling length }

CPU Scheduling < Preemptive / Non-Preemptive

- Preemptive → If process is in running state, then the OS has the power to take the process out of the CPU forcibly
- Non Preemptive → If process is in running state, OS can not take out the CPU process from CPU until process wants

Note: Every process which we are going to execute has no I/O operation [Assumption for all the future algorithm] → To measure the performance of algorithm

Process out from the CPU
1. Process terminated
2. Process is preemptive
3. Process waiting for I/O operation {Not possible, because of assumption}

First Come First Serve (FCFS) → Process जो पहले arrive हुआ वो execute पहले होगा
→ Process with min arrival time will be executed first

Note - If two processes have same arrival time
→ Tie Breaker → Process with smaller process id (unique of a process) will be executed first

Imp → Type → Non-Preemptive (Non-forcible)

Note - Processes when arrived are kept in Ready Queue

- to calculate
- first time process gets CPU from the time of arrival

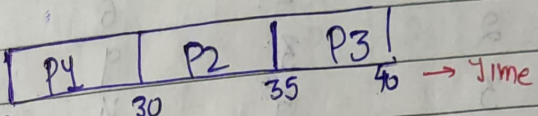
Page No:
Date: / /

Process	AT	Burst Time	Completion Time	Turnaround time	Waiting Time	Response time
P1	0	30	30	30	0	0
P2	0	5	35	35	30	30
P3	0	5	40	40	35	35

Q

Gantt Chart - Timeline diagram for when to when a process must execute
→ always starts from 0

{Gantt Chart of above Q}



Ready Queue → P_1, P_2, P_3 (All have same Arrival time) hence process-id will be used

- Since non-preemptive, hence will be in CPU, until process finished (terminated)

Avg Turnaround time = $\frac{30+35+40}{3} \Rightarrow 35$

Avg Waiting time = $\frac{0+30+35}{3} = \frac{65}{3} \rightarrow 21.67$

- Response Time → first time process gets into the CPU
Time from arrival time to first time it enters CPU

Note In Non Preemptive System
↳ Waiting Time = Response Time

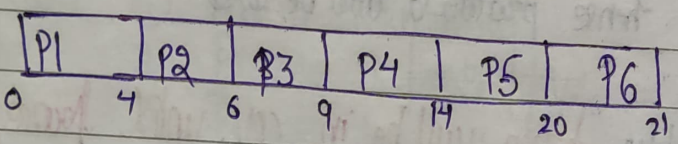
{Scheduling length} = max(CT) − min(AT)
40 − 0 = 40

Throughput (per unit time) = No. of processes / Scheduling length = 3/40

From chart → CT − AT → To calculate

Process	Arrival Time	Burst Time	Completion Time	Turn around	Waiting Time
P_1	0	4	4	4	0
P_2	1	2	6	5	3
P_3	2	3	9	7	4
P_4	3	5	14	11	6
P_5	4	6	20	16	10
P_6	5	1	21	16	15

Gantt Chart



{Avg Turnaround time} = 59/6 → 9.833

Throughput → 6/21

{Avg Waiting Time} = 38/6 → 6.333

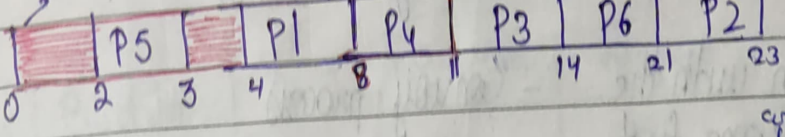
* Scheduling length = max(CT) − min(AT)
21 − 0 = 21

Page No:
Date: / /

AT - CT

Process	Arrival Time	Burst Time	Completion time	Turnaround time	Waiting Time
P1	4	4	8	4	0
P2	8	2	23	15	13
P3	6	3	14	8	5
P4	5	3	11	6	3
P5	2	1	3	1	0
P6	7	7	21	14	7

check based on first come first serve criteria



Gantt Chart

completion time — Arrival time

Avg T.T ⇒ $\frac{48}{6}$ → 8

Scheduler length = max(CT) − min(AT)

Not 0

Avg WT = $\frac{28}{6}$ → 4.66

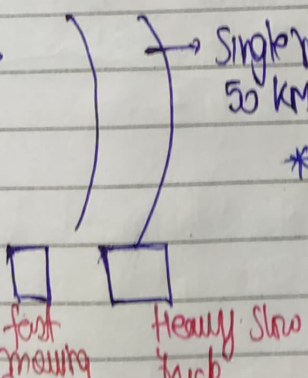
23 − 2

21

Throughput ⇒ $\frac{6}{21}$

Response Time ⇒ Check the gantt chart time and the arrival time from the table

Convoy effect →



→ Single road (no overtake) 50 km long road

* If heavy vehicle (process with ↑ burst time gets into CPU) then the whole process of processing becomes slow

- If truck goes first, then we have entire traffic mo
slow
- Similarly if process taking long time executed first
then that will take more time and slow sma
process need to wait for long time
(This is called convoy effect in FCFS)
↳ Decreases the efficiency

If large process is scheduled first then it slows down
system's process (only in FCFS)

Problem with the - convoy process
First come first
serve

Affiliate scales

* FCFS (First come first serve)
* SJF (Smallest burst time) - Ready Queue
* SRTF

so that we can predict the future

Lecture-7

Direction
Date

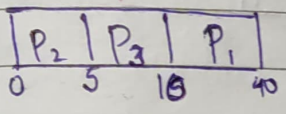
SJF (Shortest Job first) - will schedule the job which is having smallest burst time first
- then will have more efficiency in FCFS

- Criteria - Smallest burst time process first
- Tie Breaker → Implement FCFS
- Type → Non-Preemptive (will leave when the task completed)
  Assumption - No IO device used

Q

Process	Burst time	Arr time	Completion time	CT-AT Turnaround time	TAT-BT Waiting time
P_1	30	0	40	40	10
P_2	5	0	5	5	0
P_3	5	0	10	10	5

Gantt Chart



Need to create Ready Queue also

Time	Process
0	P_1, P_2, P_3
5	P_1, P_3

- P_2 choosen because of less period id

- Avg TAT = $\frac{55}{3}$ = 18.33
- Avg WT → $\frac{15}{3}$ = 5 → less waiting time compared to FCFS

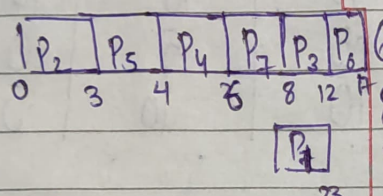
Note - The processes which are arrived can only be scheduled not the future processes

{CT - AT} { TT - BT }

Process	Arrival Time	Burst Time	Completion time	Turnaround Time	Waiting time
P1	0	6	23	23	17
P2	0	3	3	3	0
P3	1	4	12	11	7
P4	2	2	6	4	2
P5	3	1	4	1	0
P6	4	5	17	13	8
P7	6	2	8	2	0

Time	Ready Queue
0	P1, P2
3	P1, P3, P4, P5
4	P1, P3, P4, P6
6	P1, P3, P6, P7
8	P1, P3, P6
12	P1, P6
17	P1

Gantt Chart



Avg Turnaround time = $\frac{57}{7}$

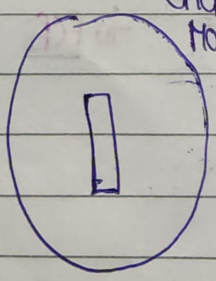
Avg WT = $\frac{34}{7}$

Scheduling length = max (CT) - min (AT)

23 - 0 ⇒ 23

Example

Cricket Match



① Opponent team all 11 members present
2. My team only 3 members present

- Since need to start the match, toss begun and my team won and we choose to bat first
- & send player on field which is best among 3 player

FCFS SJF SRTF

Remaining Time First

Shortest Job first

SJF - Non-preemptive
SRTF - Preemptive

* Shortest Remaining Time first

- In SJF (Shortest Job first), if new better player arrives then that is not possible to take out the existing player from the field and send more good batsman
  - ↳ Non Preemptive

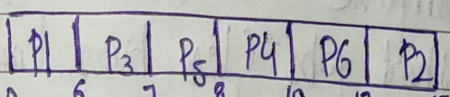
- Shortest Remaining Time first (SRTF)
  - shorter process could able to preempt the longer process
  - Criteria :- Burst Time
  - Tie Breaker : First come first serve (FCFS)
  - Type : ~~Non~~ Preemptive (Imp)

Solve using SRTF

* Get from the gantt chart (when process gets CPU for the first time)

Process	Arrival Time	Burst Time	Completion Time	Turnaround Time	Waiting Time	Response Time
P1	0	6 ✓	17	17	11	0
P2	1	4	9	8	4	0
P3	2	1 ✓	3	1	0	0
P4	3	2	5	2	0	0
P5	4	1 ✓	6	2	1	1
P6	5	3	12	7	4	4

SJF Gantt Chart



* Just for comparision

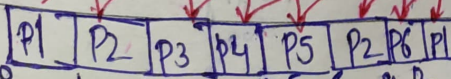
Note - for any Preemptive system need to seperately write process and burst time

	Remaining Burst Time
P1	6 5
P2	4 3
P3	1 X
P4	2 1 X
P5	1 *
P6	3

* Ready Queue

At Time	
0	P1
1	P1, P2
2	P1, P2, P3
3	P1, P2, P4
4	P1, P2, P4, P5
5	P1, P2, P5, P6
6	P1, P2, P6

No of context switch = 7



SRTF - preempted longer jobs for shorter jobs
SJF - favours shorter job

Page No:
Date: / /

- In SRTF, waiting time less compared to the SJF

Note- Preemption only takes place if there is less burst time process

* Avg TAT = 37/6 Avg WT = 20/6

Problem with SJF and SRTF
1. Practical implementation not possible
(Because before the process runs fully we can not know the burst time (how much time process will remain in CPU)

2. Starvation → If shorter process keep on coming then longer process needs to keep on waiting
→ Indefinite waiting
→ because of this there is no fairness

(3) HRRN (Highest Response Ratio Next) → favours shorter jobs and also solves problem of indefinite waiting of longer process
non preemptive

Objective :- Not only favour short jobs (less burst time) but decreases the waiting of longer jobs

Criteria :- Response Ratio
Tie Breaker :- Burst Time
Type :- Non Preemptive

Note → FCFS - focus on arrival time (convoy effect)

SJF
- Non Preemptive
- Focus on burst time
- Ready Queue Needed
- Can not taken out if longer process in CPU

SRTF
- Preemptive
- Focus on burst time
- Ready Queue + Burst time keeps on changing

* SJF & SRTF same remember only then Preemption

FCFS
SJF - need to create ready queue
SRTF - Preemptive, need to create ready queue
HRRN - Highest Response Ratio Next

Date: / /

Response Ratio = (W+S)/S

W = wait time
S = Service / Burst Time

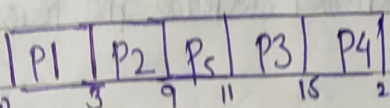
→ If less burst time, then higher Response ratio
→ RR ∝ Waiting Time ∝ 1/Service Time (decreases the waiting time of long burst time process)

Note - Process with high HRRN will be scheduled fast

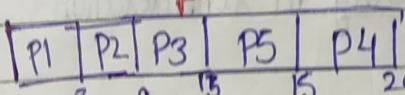
Q.

Process	Arrival Time	Burst Time
P1	0	3
P2	2	6
P3	4	4
P4	6	5
P5	8	2

Solve for HRNN and compare it with SRT SJF

SJF - 

- Shortest Job First

HRRN 

executed for a because waiting for a longer period of time

New process in queue

At time	RR@	W (Waiting Time)	S (BT)	Response
9	P3	5	4	2.25
9	P4	3	5	1.6
9	P5	1	2	1.5
13	P4	7	5	2.4
13	P5	5	2	3.5

Ready Queue

At time	Ready Queue
0	P1
3	P2
9	P3, P4, P5

Need to also complete table

CT - AT

TT - BT

Process	Arrival Time	Burst time	Completion Time	Turnaround Time	Waiting time
P1	0	3	3	3	0
P2	2	6	9	7	1
P3	4	4	13	9	5
P4	6	5	20	14	9
P5	8	2	15	7	5

HRRN - good but it uses burst time which is unpredictable

Page No:
Date: / /

Priority Based Scheduling Algorithm (Priority set by OS)

Criteria: Based on priority (Priority set by OS)
Tie Breaker: Never occur, but it happens then FCFS (first come first serve)
↳ Priority - Arrival Time
Type → Preemptive
→ Non-Preemptive

	Priority
P1	10
P2	15
P3	6

Note - can not tell which having high priority because it is in the Q, where it is mentioned higher/lower which one to consider to have higher priority

Priority → Static - fixed priority (can not be changed)
→ Dynamic - may increase/decrease the priority

Note - By default static priority in consideration

Q a) Non Preemptive

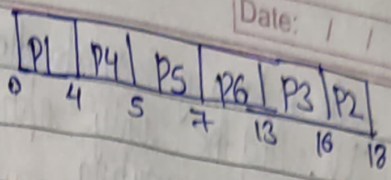
Process	Arrival	Burst	Priority	Completion	Turnaround	WT
P1	0	4	4	4	4	0
P2	1	2	5	18	17	15
P3	2	3	6	16	14	11
P4	3	1	10 HIGHEST	5	2	1
P5	4	2	9	7	3	1
P6	5	6	7	13	8	2

- Avg TAT = $\frac{48}{6} \Rightarrow 8$
- Avg WT = $\frac{30}{6} \Rightarrow 5$

• Ready Queue

Time	Process
0	P1
4	P2, P3, P4, P5
5	P2, P3, P5, P6
7	P2, P3, P6

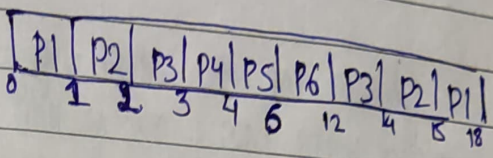
Gantt chart



(b) Preemptive scheduling

Time	Process
0	P1
1	P1, P2
2	P1, P2, P3
3	P1, P2, P3, P4
4	P1, P2, P3, P5
5	P1, P2, P3, P5, P6
6	P1, P2, P3, P6
12	P1, P2, P3
14	P1, P2
15	P1

Gantt chart



Remaining	Burst Time	Priority
P1	~~4~~ 3	4
P2	~~2~~ 1 0	5 ✓
P3	~~8~~ 2 0	6 ✓
P4	1 0	10 ✓
P5	2 1 0	9 ✓
P6	~~8~~ 0	7 ✓

Note - Priority based Scheduling Algo also suffer from Starvation
- If higher priority keeps on arriving then the lower priority process are not getting the CPU, fairness not present (process wait for indefinite time)

- Priority set by the OS

Solution of Starvation → 1. All the waiting process, priority will increase by 1 after certain duration (Aging)

→ Aging - only for dynamic

Page No:
Date: / /

eg - In a family, a toddler given least priority to watch the T.V. And the only option he has is to increase its age by waiting
Similarly we have Aging
Priority of waiting process will increase by 1 after predefined time period
(This is applicable only on dynamic priority system)

FCFS - Amazon
SJF
SRTF (Preemptive)
HRRN (Non Preemptive)
Priority based (Easy)
Round Robin (Preemptive)

to give the user the feel that all processes are running, this algo is best

Round Robin Scheduling

Criteria: Arrival Time + Quant Time
- Time for which we will run a process for maximum time on CPU, after that the process will be preempted automatically

Tie-Breaker: Process Id

Type: Preemptive

Objective: provide interactiveness (gives fairness)

Note - In gully cricket, each player gets 12 balls no matter what and this keeps on going. This is the round robin fashion

Q = 2 = Quant Time

Process	Arrival	Burst	CT	Turn around	WT	Response Time
P1	0	3	9	9	6	0
P2	0	6	17	17	11	2
P3	0	4	13	13	9	4
P4	0	5	18	18	13	6

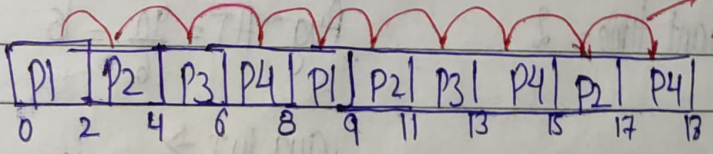
When process gets CPU for the first time

What all element present

Note - Not maintaining ready queue, because all process arrived at 0 and based on process id scheduling needs to be done

Process	Burst Time	
P1	3 1	0
P2	6 4	2
P3	4 2	0
P4	5 3	1

Gantt chart



No of context switches

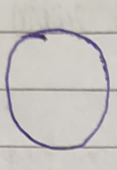
No of context switches = 9

Page No:
Date: / /

Note - Robin Robin does not give less waiting time
- But it is highly effective in multi-tasking and CPU, in which we need response of all CPUs
- But achieves fairness
→ Burst time of processes need not to be done beforehand

Req
Q Process - Arrival Time

eg. In Round Robin fashion, when the arrival time of all the processes not same then



* you, F1, F2
F1, F2, you
F2, you, F1
] after each quant time

Suppose your two friends come then the sequence will be

F2, you, F1, F3, F4

Q Imp

Process	Arrival Time	Burst Time	Completion Time	Turnaround	Waiting Time	Response Time
P1	0	4	8	8	4	0
P2	1	5	20	19	14	2
P3	2	6	24	22	16	4
P4	3	3	19	16	13	8
P5	4	2	12	8	6	10
P6	5	4	22	17	13	14

* Quant time = 2

Avg TAT = $\frac{90}{6} = 15$

Avg WT →

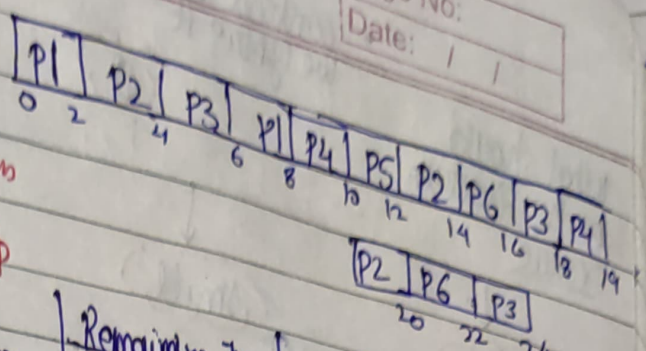
Ready Queue

	Ready Queue
at t=0	P1
2	~~P1~~, P2, P3, P1
4	P3, P1, P4, P5, P2
6	P1, P4, P5, P2, P6, P3
8	P4, P5, P2, P6, P3
10	

current running process added in end

Jmp

Also here no need to maintain ready queue

* 

*

	Remaining Time
P1	4 2 0
P2	5 3 1 0
P3	6 4 2 0
P4	3 1 0
P5	2 0
P6	1 2 0

{No of context switch = 12}

Trick "Only for round robin fashion"

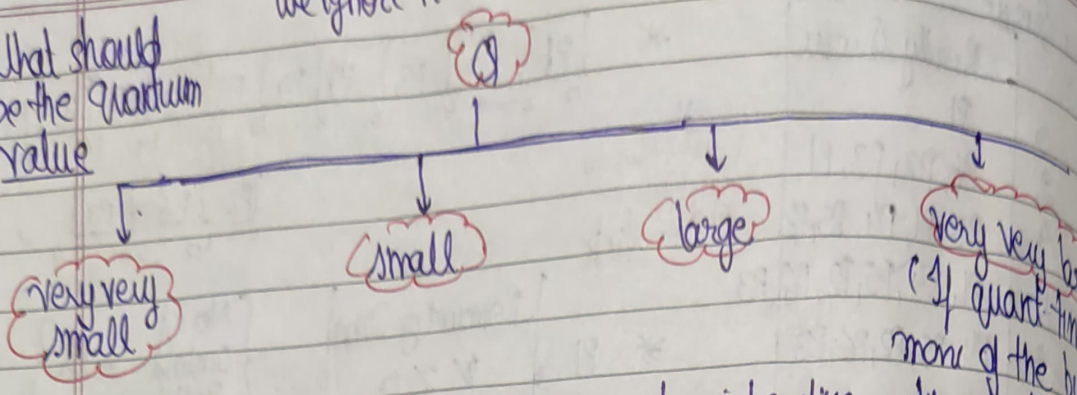
Jmp Since we are given : Q = 2 (here)

	Burst	How many time in CPU
P1	4	2
P2	5	3
P3	6	3
P4	3	2
P5	2	1
P6	1	1
		13

No of context switch => 13 - 1 = 12

* context switch time is negligible hence we ignore it

What should be the quantum value



- Process will take more time in context switch less time in execution of process
- CPU output / efficiency will be ϕ (null)
- CPU spends more time in context switch and less time in process execution

- Interactive performance

- less interactive

- Round Robin degrades to the First come First serve (FCFS)

Note - eg → If you take book from library and it took 5 min, but you read it for just few seconds.
(More time in context switch, less in execution of process) → Low efficiency

- Q → small (Interactive) vs Q → large (less interactive)

When you go to take breakfast, if shopkeeper listening to everyone for short duration then everyone feels yes that the shopkeeper is interactive but if the same person just takes long time to interact with a single person then in becomes less interactive

manner so that we predict the future

Page No.
Date: / /

Process	Arrival	Completion
A	0	4
B	0	1
C	0	8
D	0	1

Round Robin

Q = 1 Completion Time of A = ?

Gantt → | A | B | C | D | A | C | A | C | A | C |

0 1 2 3 4 5 6 7 8 9 10

A 4 3
B 1 0
C 8 7
D 8 0

Ans → 9

Q2

Process	ARRIVAL	Burst Time	FCFS CT	FCFS TT	FCFS WT	Round Robin CT	Round Robin TT	Round Robin WT	Non Pr. SJF CT	Non Pr. SJF TT	Non Pr. SJF WT
P1	0	10	10	10	0	10	10	0	20	20	10
P2	0	29	39	39	10	61	61	32	61	61	32
P3	0	3	42	42	39	23	23	20	3	3	0
P4	0	7	49	49	42	30	30	23	10	10	3
P5	0	12	61	61	49	52	52	40	32	32	20

52
63
115

FCFS | P1 | P2 | P3 | P4 | P5 |

0 10 39 42 49 61

Avg WT ⇒ 140/5 ⇒ 28 ms

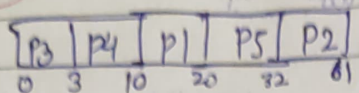
Round Robin Q=10 | P1 | P2 | P3 | P4 | P5 | P2 | P5 | P2 |

0 10 20 23 30 40 50 52 61

Process	Burst	
P1	~~10~~	0
P2	29	~~19~~ 9
P3	~~3~~	0
P4	~~7~~	0
P5	12	~~2~~ 0

Avg WT ⇒ 115/5 ⇒ 23 ms

Non Preemptive → A/c to burst Time
SJF

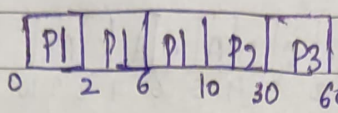


avg WT → 65/5 → 13 ms

Q

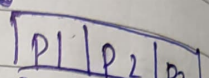
	Arrival	Burst
P1	0	10
P2	2	20
P3	6	30

SRTF



Ready Queue

		Burst Time Remain	
0	P1	P1	10 8 4 0
2	P1, P2	P2	20
6	P1, P2, P3	P3	30



No of Context Switch ⇒ 2

→ Preemption happen when new arriving process has shortest BT than current running process
Here both arriving process are having larger BT, hence no preemption

Q When I/O operation included

If the waiting time for process is p (fraction of time) and there are n processes in the memory then CPU utilization is?

Sol · If no I/O operation CPU utilization ⇒ 100%

· Waiting time means ⇒ I/O operation

Total fraction of waiting time = p^n

If 1 process → p (WT)
2 process → p × p
n process → p × p ... p (n times)

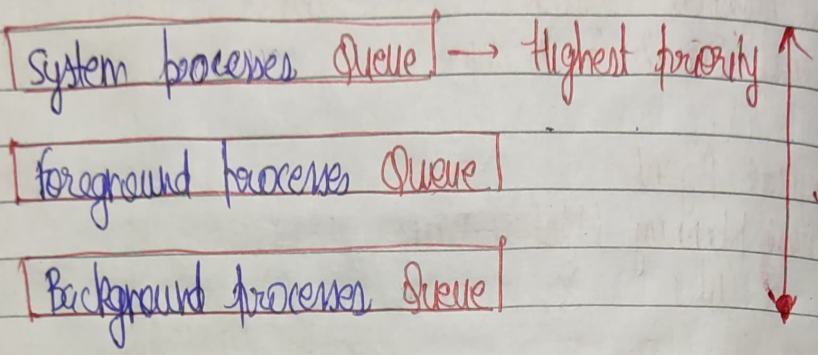
Imp CPU utilization ⇒ $1 - p^n$

Page No:
Date: / /

Lecture-11
Multilevel Queue

Multilevel Queue (MLQ) Scheduling
- Different processes have different requirement in terms of scheduling and hence multilevel queue scheduling is introduced

↔ Hence segregation of different processes in done in different queues



- Priority are now not on the process but on the queue
- Different queue according to their requirement have the specific scheduling algorithm.
- Once all the system processes complete < foreground processes background processes

Scheduling among Queues
1. fixed priority preemptive scheduling method
2. Time slicing (eliminate Starvation)

* Fixed priority preemptive scheduling method
- Nature - Preemptive
* Based on the priority level of queue the processes will be executed, if low priority queue process is executing and a new process arrives in higher priority queue then the preemption of low priority process will take place

. In priority based scheduling we might have starvation problem that only the high priority process will be executed not the less one.

2 Time slicing → based on fraction of time

eg Q1 - 60% of time
Q2 - 30% of time
Q3 - 10% of time
} avoid the problem of starvation

Note: have multiple ready queue and maintained by the main memory

1 Queue 1 ⇒ RR (Q=2) → High priority fixed priority preemptive
Queue 2 ⇒ FCFS

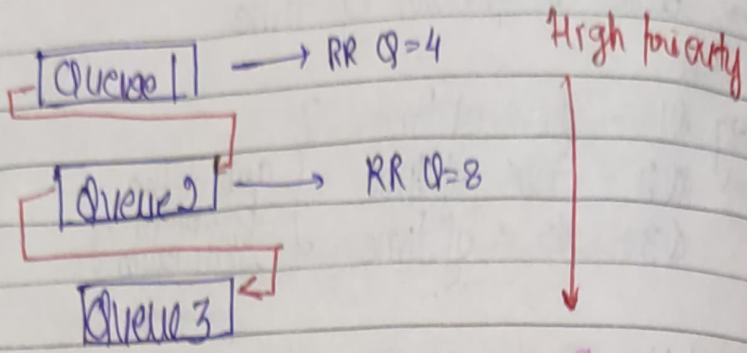
Process	AT	BT	Queue
P1	0	4 2	1
P2	0	3 1	1
P3	0	9 7	2
P4	9	4	1

P1	P2	P1	P2	P3	P4	P3
0	2	4	6	7	9	13	20

Ready Queue 1: 0 | P1, P2 ; 7 | empty ; 9 | P4
Queue 2: 0 | P3

MLQ scheduling disadvantage
- Some process may starve for CPU if some higher priority queues are never becoming empty
- It is inflexible in nature because certain type of process need to present in that particular queue only

Multilevel Feedback Queue (MLFQ)



- Initially all the processes in queue 1, if process completes in one CPU allocation well and good and if not then the priority of that process decreases and it shifted to queue 2 and subsequently to queue 3 (if process not finished after queue 2 scheduling process)

process

- Drawback → Some CPU may starve for CPU if some higher priority queue never getting empty

→ Flexible (eliminate problem of Multilevel queue)

Note Solution of starvation → by moving a process to higher level queue by help of aging.
(If process present in Queue 3 for long time, then the process can be shifted to high priority queue)

Adv and disad of Scheduling algo

FCFS → Adv → Easy to implement → No starvation
→ No complex logic

Disadv → No option of preemption
→ Convoy effect makes the system slow
(If large system / process schedule first)

Amazon
cabs

SJF

Adv → Better throughput in continue run
Imp → Minimum avg waiting time among non preemptive scheduling

Disadv
- No option of preemption
- Longer process may suffer from starvation
- No practical implementation because (burst time not known)

SRTF → preemptive type of SJF

Adv → Min avg WT among all scheduling algo
Imp → Better throughput in continue run

Dis
- No practical implementation (because burst time)
- Longer process may suffer from starvation

Priority based algo

Adv → Better response for real time situation

Dis → ~~Longer~~ low priority may suffer from starvation

Round Robin Algo → Adv → All process execute one by one, no starvation (Quant time)
→ Better interactiveness
→ Burst time not needed

Disdd → Can degrade to FCS (If quant time > max (Burst time)
→ Avg WT and Turnaround time more

Page No:
Date: / /

Part - 2 Multithreading and Threads

Thread - component of process
OR
Lightweight Process

- provide a way to improve application performance through parallelism

Analogy

- If a builder builds a flat whenever he recieves a request of hosted room then that will be wastage of space, money. Because for a student (living room, bed room, Kitchen will be made)

- 3rd person when recieve request then he does not make whole flat but just the bed room (have shared kitchen, living room)
  ↳ Thread (less resource wastage, more utilization)

More ROI

same

little doubt

- Similarly when the process need to be executed with multiple instances then the whole process not replicated again only the instance specific to that process will be executed else things will be shared

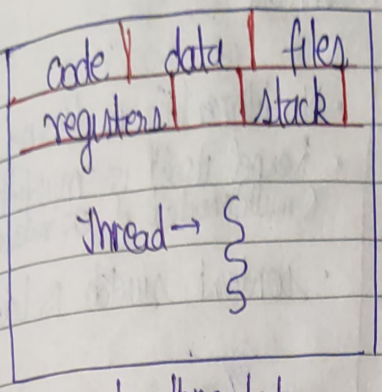
- We do not create multiple instances of a process but create multiple threads within a process

* Shared among thread	Unique for each thread
• Code section (Instructions) • Data section (global and static variable • OS Resources • Open files and signals	• Thread ID • Register set • stack • Program counter

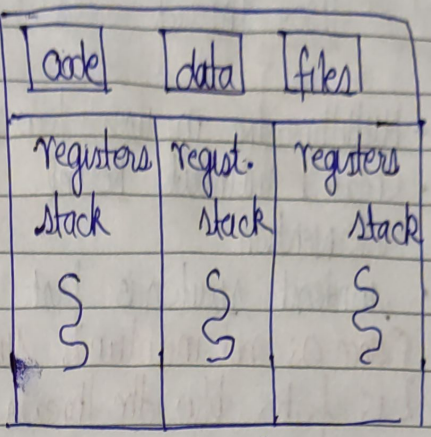
- If process replicate then slow process but thread make it faster
- Thread managed by the process

Page No:
Date: / /

Thread



single-threaded process



multi-thread process

→ Thread just a part of process.

Adv of Multithreading

1. Responsiveness increase
2. faster context switch
3. Resource sharing
4. Economy

{Multithreading cheaper than Multiprocessing}

5. Communication
6. Utilization of Multiprocessor architecture
- Parallelism comes into play

Types of thread

(i) User level thread → if multithreading in user processes
(ii) Kernel level thread → if multithreading in OS related process
OS → OS aware about the os level thread

User level → OS don't know about the User level thread

↓
Multithreading in User process

Page No.
Date: / /

User Thread

- Multithreading in user process
- Created without kernel intervention
- context switch is fast {since os not involved, switching is faster b/w the threads}
- If one thread is blocked, OS blocks the entire process (even if process want for I/O operation then also whole process block because kernel don't know of multiple thread)
- Generic and can run on any OS
- faster to create and manage

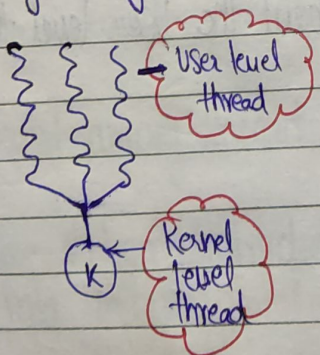
Kernel Thread

- Multithreading in kernel process
- Kernel itself is multithreaded (multithreaded of OS related process)
- context switch is low
- Individual thread can be blocked
- Specific to OS
- Slow to create and manage

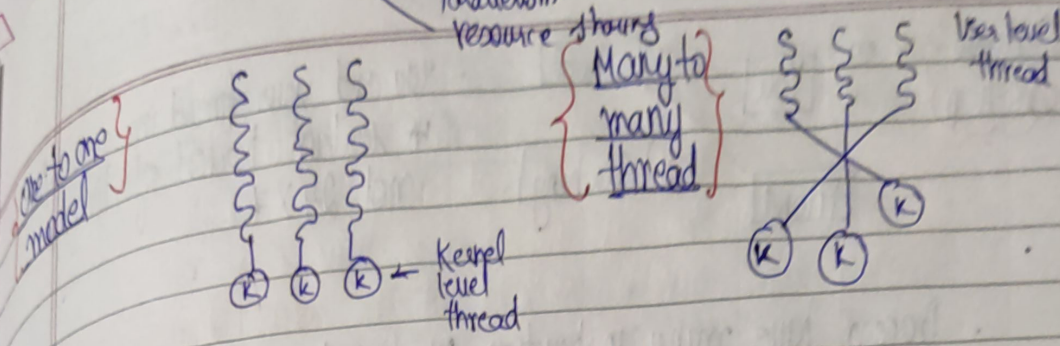
Multithreading model → how many user level thread will be communicating with kernel level thread

1. Many to one model
2. One to one model
3. Many to many model

Many to one - many user level thread are implemented and performed by single kernel level thread (communicating associated with)



Multi threading
- Robustness
- fast
- very computational intensive
- parallelism
- resource sharing



Multithreading → to implement same processes with diff instances without replicating whole process but by sharing of resources

SYNCHRONISATION

Types of processes
1. Independent
2. Cooperating / Coordinating / Communicating = have communication may be affected and may affect some process

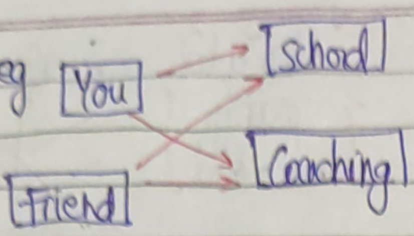
Independent processes
→ no communication with other process (no sync. required)
→ no sharing / sending / recieving of data
→ will not be affected and will not affect any process

Note - Communication b/w the process should be synchronized (two parties decided on one point and implemented what has been decided)

Need of synchronization in communicating process

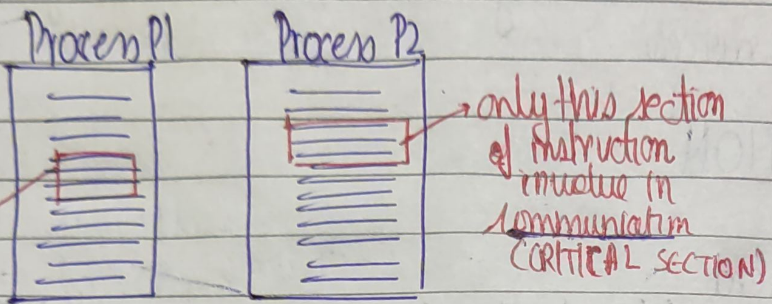
Problems
(i) Inconsistency in data
(ii) loss of data
(iii) deadlock

Critical section → eg



- You and your friend in syn for certain period of time, not 24 x 7.

- Process have many instruction, we do not need synchronization for each instruction (because each instruction not involve in communication)



(CRITICAL SECTION) → The part/section of process which involve communication there only synchronization required
→ The section which involve communication/synchronization called critical section

* The critical section is a code segment where the shared variables can be accessed — communication happens

"Main is communication" then only synchronization needed.

Process has two sections
- → critical section – here only synchronization required
- → Remainder "

Race Condition – It is an undesirable situation, it occurs when the final result of concurrent process depend on the sequence in which the process complete their execution

→ occurs because of lack of synchronization

a = 5
Main Memory

Amazon
online
↳ sales

Critical section Area and Solution
↳ where the shared variables are present

Process P1

```
R1 = a
R1 = R1+2
a = R1
```

• R1=5
• R1=7

Process P2

```
R2 = a
R2 = R2+5
a = R2
```

• R2=5
• R2=10

* R1 and R2 are registers

* If R1 executed first 5 → 7 → 10
  R2 " " 5 → 10 → 7
  } Undesirable result because the actual ans we want is 12. (Due to lack of synchron)

Critical Section Problem and Solution

• Critical section of the process requires the synchronization and to solve that problem we have entry and exit section

Process
- entry section
- critical "
- exit section

- Entry section required because if one process using the critical section (C.S.) then it ensures no other process enters the C.S. (All the three requirement)
- Exit section does the announcement that the process has used C.S, other processes can now use it (Announcement)

Requirement of the solution which is solving critical section problem

(1) Mutual Exclusion
(2) Progress
(3) Bounded Waiting

• Mutual Exclusion - If one process using the C.S then the other process can not use that critical section (C.S)
→ The shared resource can be used only by one process at a time so that the inconsistency and race condition does not take place

• Progress → The solution should be designed such that if the C.S is free and the process is waiting then there should be no blockage (it should be allowed)

Big data ...

Bounded waiting
Progress
Mutual exclusion

Page No:
Date: / /

3 Bounded waiting
* Bounded means limited waiting time.
* If a process is using C.S again and again then that should not be allowed
* All the waiting process have a bounded (limited) waiting time, which should not be exceeded

* waiting of process should be bounded (process)
* waiting is fine because of waiting process but because of already process which has used C.S is not fine

4. Should not depend on hardware architecture

Solution 1: Using lock

lock variable

Boolean lock = false;

while (true) {
entry section → while (lock); → while(condition){ } → empty block
lock = true;
CS
exit section → lock = false;
RS;
}

Nothing execution will remain in loop until lock is true

hoc

while (true) {
while (lock);
lock = true;
CS
lock = false;
RS;
}

Checking if it follows conditions

- Not follow mutual exclusion - If a process is preempted after while (lock) statement
- Progress is satisfied - If lock = false then the entering in any of the process is possible

Drawback → Need to check by preempting at each step whether mutual exclusion followed or not

Solution 2: Using Turn

```
int turn = 0;
P0                          P1
while (true)                while (true){
{ while (turn != 0);          while (turn != 1);   => Entry Section
      CS                          CS
  turn = 1;                   turn = 0;
  RS;                         RS;
}                           }
```

Note:
- Mutual exclusion is possible
- Progress not there; initially turn = 0 and if P1 tries to enter then that won't be possible – two process will run in strict alternation
- Bounded waiting present

Solution 3 Peterson's Solution (Satisfy all the three process) → Mutual exclusion satisfied, Bounded waiting, Progress

```
Boolean flag [2];          flag[0] = false
int turn;                  flag[1] = false
```

P0

```
while (true) {
entry { flag[0] = true;
        turn = 1;
        while (flag [1] && turn == 1);
            CS
        flag [0] = false;
            RS;
}
```

P1

```
while true {
    flag [1] = true;
    turn = 0;
    while (flag [0] && turn == 0
        CS
    flag [1] = false;
        RS;
}
```