Operating Systems

⇒ Mainframe Systems ⟶ were the first computers to tackle many commercial and scientific applications

Batch Systems

→ Early computers were enormous machines run from a console
→ Common input devices : card readers and tape drives
Common output devices : printers, tape drives and card punches

→ User did not interact directly with computer system
↓
prepared a job (which consisted of program, data and some control information about nature of job)
↓
submitted to computer operator
↓ (job was usually in the form of punch cards)
↓
At some later time the output appeared
↓ consisted of
result of the program,
dump of final memory and register contents for debugging

→ To speed up processing operators batched together jobs with similar needs and ran them through the computer as a group.

Programmers left their programs with the operator ⟶ Operator would sort programs into batches with similar requirements ⟶ computer would run each batch

output of each batch would be sent to appropriate programmer

→ CPU is often idle, because the speeds of mechanical I/O devices are intrinsically slower than those of electronic devices

→ Introduction of disk technology allowed the operating system to keep all jobs on a disk rather than serial card reader. With direct access to several jobs the operating system could perform job scheduling.

Multiprogrammed Systems

→ Multiprogramming increases CPU utilization. It is an imp aspect of job scheduling

→ operating system keeps several jobs in the memory simultaneously. This set of jobs is the subset of jobs in job pool

Since the number of jobs that can be kept simultaneously in memory is usually much smaller than the no of jobs that can be in the job pool

→ All the jobs that enter the system are kept in the job pool This pool consists of all processes residing on disk awaiting allocation of main memory

→ If several jobs are ready to be brought to memory and if there is not enough room for them, then the system must choose among them. This is called job scheduling.

If several jobs are ready to run at the same time the system must choose among them, making this decision is CPU scheduling.

Time Sharing Systems

Multiprogrammed batched systems did not provide user interaction

→ Time sharing (multitasking) is a logical extension of multiprogramming

→ CPU executes multiple jobs by switching among them but the switches occur so frequently that the users can interact with each program while it is running

→ Interactive Computer System (Hands-on) : user directly communicates with the system using a keyboard or a mouse and waits for immediate results. Response time should be ~~short~~ short typically within 1 second or so.

→ Time shared operating system uses CPU scheduling and multiprogramming.

A program loaded into the memory and executing is commonly referred to as a process.

→ To obtain reasonable response time, jobs may have to be swapped in and out of main memory to the disk that now serves as a backing store for main memory. A common method for achieving this goal is virtual memory which is a technique that allows execution of a job that may not be completely in memory.

Advantage - programs can be larger than physical memory.

Separates logical memory.

Multiprocessor Systems

→ graceful degradation

→ fault tolerant

NPTEL Operating System

What is an operating system → Just like a resource manager

⇒ Resources

CPU → Central Processing Unit

⇒ When we compile a program the executable code is stored on the disk (hard disk)

⇒ It has to be brought from hard disk to main memory of the system

⇒ CPU cannot access the data that is stored in secondary storage devices like hard disk, floppy etc
It has to be on the main memory

⇒ Anything that needs to be processed by the CPU must reside in the main memory.

Main Memory

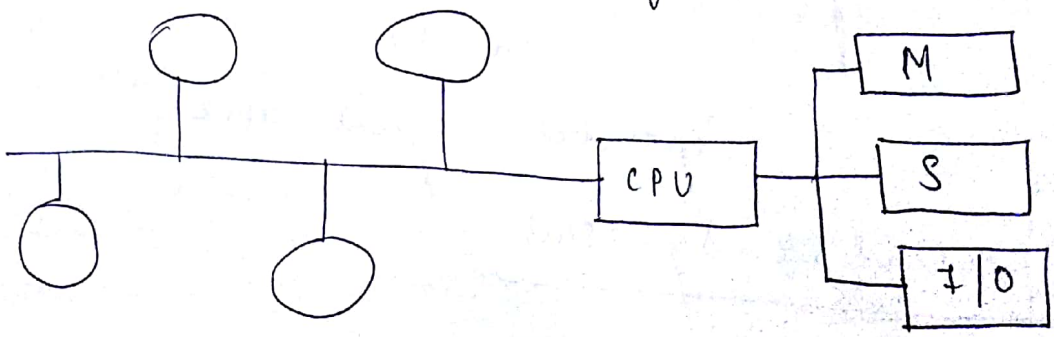
Secondary Storage

↓

Hard disk, magnetic tape, floppy, CD Rom

Input / Output devices keyboard printer monitor

Distributed System → Distributed Computing System



CPU / Process

whenever we execute a program, it become a process

waiting time, turn around time

$t_i - t_0$ → waiting time

t_0 — submitted the job for execution

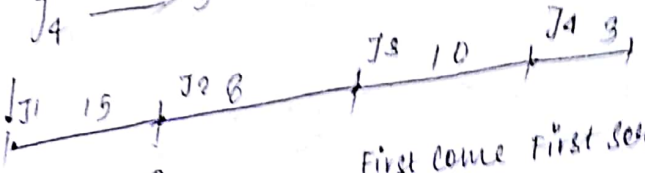
t_i — CPU starts executing

t_f — output is given by CPU

$t_p - t_0$ → turn around time

Scheduling
↳ usually first come first served

Jobs	execution time
J_1	15
J_2	8
J_3	10
J_4	3



First come First served Algorithm

$W_{J_1} = 0$

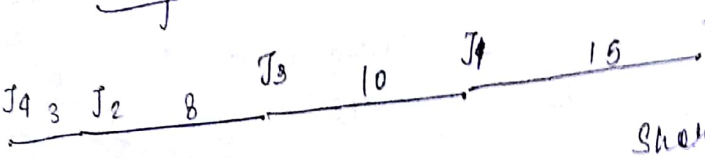
$W_{J_2} = 15$

$W_{J_3} = 23$

$W_{J_4} = +33$

71

$t_{W\,Avg} = 17.75$



Shortest Job First algorithm

$W_{J_4} = 0$

$W_{J_2} = 3$

$W_{J_3} = 11$

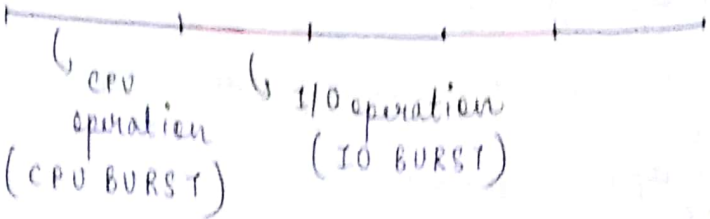
$W_{J_4} = 21$

35

$t_{WA} = 8.75$.

2. Process Management - I

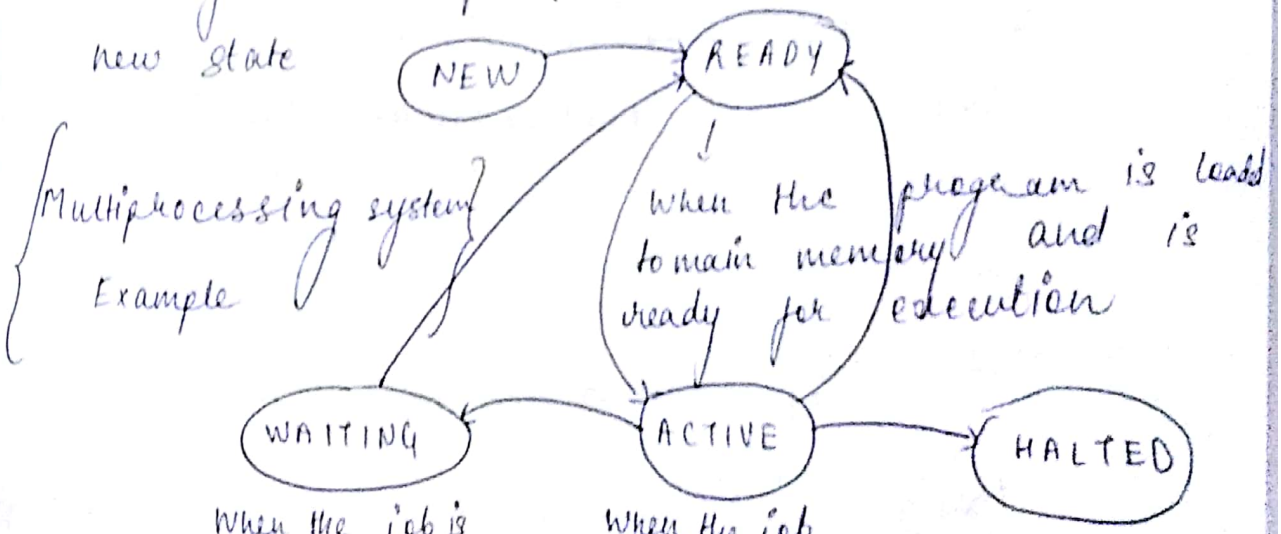
⇒ A process is nothing but a program in execution



⇒ The first and last one must be a CPU BURST when the program is initiated and when the program is terminated

⇒ When we try to execute a program after compilation it first has to be loaded to main memory by my loader then it is executed by CPU.

⇒ When you initiate a program for execution it is called a new job or the process enters a state called the new state



STATE DIAGRAM

We cannot move the job from waiting state to active because the ~~I/O~~ CPU BURST of some other job might not have completed

⇒ For efficient CPU scheduling, we need to have idea about the time of the next CPU BURST.

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

- τ_{n+1} → predicted next CPU BURST time
- t_n → previous CPU BURST time
- τ_n → predicted CPU BURST time

$0 < \alpha < 1$

$\tau_2 = \alpha t_1 + (1-\alpha)\tau_1$ (First come first served)

↳ For first time the job is executed on FCFS basis

⇒ PRIORITY SCHEDULING (Shortest Job First)

$P(I) = \frac{1}{\tau_{n+1}(I)}$ ⟶ STF Scheduling (special case of priority Scheduling)

FCFS, SJF, PRIORITY } Current CPU BURST must execute NON-PREEMPTIVE

PREEMPTIVE

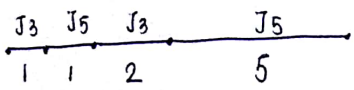
A job can be preempted before its natural completion

J_1	15
J_2	9
J_3	3
J_4	5
J_5	1

SJF ⟶ modify to preemptive.

J_3 3

↳ remaining CPU BURST time is 2 time units But a job J_5 has been added with CPU BURST time of 1 time unit which is less than 2.



SRT → Shortest Remaining Time

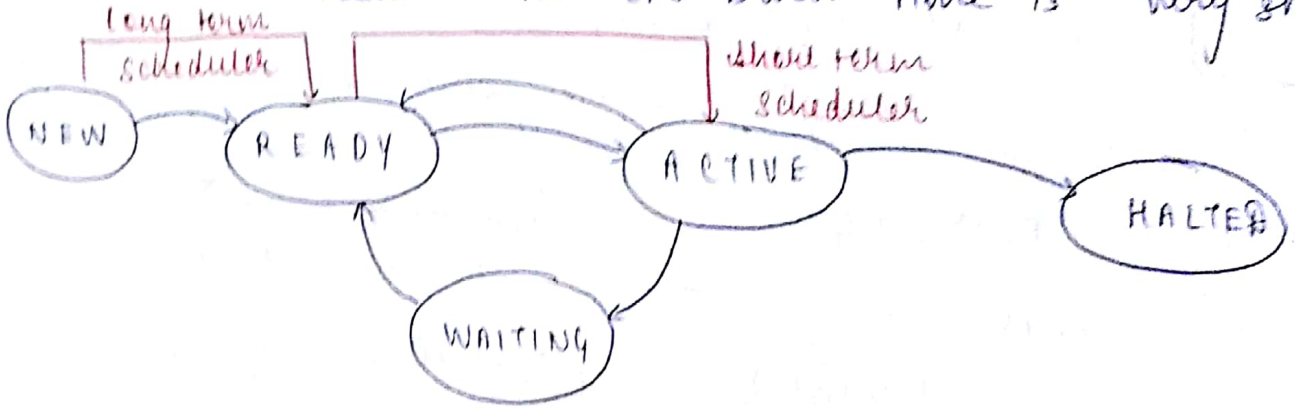
CPU bound jobs ⟶ Require more of CPU BURST time

I/O bound jobs ⟶ Require more of I/O time

⇒ There should be a scheduler that puts jobs from NEW to READY state

⇒ There should be a proper mix of CPU bound and I/O bound jobs in the READY queue.

⇒ There is also a scheduler which puts jobs from READY to ACTIVE state. This scheduler cannot take much time because the CPU Burst time is very small.

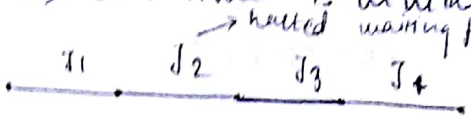


Round Robin Scheduling

2 msec

J1	4
J2	2
J3	3
J4	6

⇒ CPU time is divide



Process Scheduling Chapter 5

→ Process Execution consists of a cycle of CPU execution and I/O wait. Processes alternate b/w these 2 states.

→ **CPU Scheduler :** whenever the CPU becomes idle, the operating system must select one of the processes in ready queue to be executed. The selection process is carried out by short term scheduler or CPU scheduler.

{ The records in the queues are basically PCBS of their processes }

Preemptive Scheduling

CPU scheduling decisions may take place under the following four circumstances :

1. When a process switches from running to waiting state (I/O request or an invocation of wait for termination of one of the child processes)
2. When a process switches from running state to ready state (when an interrupt occurs)
3. When a process switches from waiting to ready state (completion of I/O)
4. When a process terminates.

⇒ When scheduling takes place under circumstances 1 and 4 we say that the scheduling is non-preemptive or cooperative otherwise it is preemptive

⇒ **Non-preemptive Scheduling :** once the CPU has been allocated to a process the process keeps the CPU until it releases the CPU either by terminating or by switching to waiting state.

⇒ **Dispatcher :** It is the module that gives control of the CPU to the process selected by short term scheduler. This function involves the following :

1. Switching context
2. Switching to user mode
3. Jumping to proper location in the user program to restart that program

⇒ It should be as fast as possible since it is invoked during every process switch.

⇒ The time it takes for the dispatcher to stop one process and start another running is known as dispatch latency.

Scheduling criteria

Factors on which CPU scheduling algorithm has to be decided :

1. CPU utilization — real system → 40% to 90%
2. Throughput : One measure of work is the number of processes that are completed per unit time called Throughput.
3. Turnaround time : The interval from the time of submission of a process to the time of completion is called turnaround time.
4. Waiting time : Waiting time is the sum of periods spent waiting in the ready queue
5. Response Time : Time from the submission of a request until the first response is produced.

Scheduling Algorithms

First Come, First served : FCFS scheduling algorithm

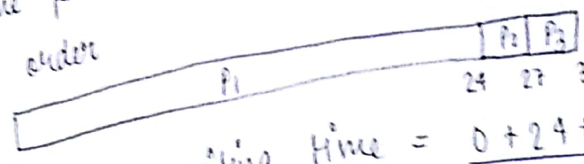
The process that requests the CPU first is allocated CPU first. The implementation is easily managed with FIFO queue. When a process enters the ready queue is PCB is linked onto the tail of the queue. When the CPU is free it is allocated to the process at the head of the queue. The running process is then removed from the queue

The average waiting time under FCFS policy is quite long

Gantt Chart : Bar chart that illustrates a particular schedule.

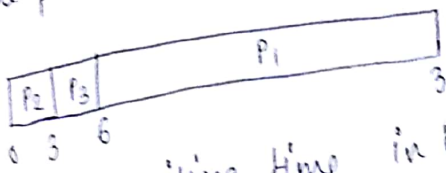
Process	Burst Time
P_1	24
P_2	3
P_3	3

If the processes arrive in order P_1, P_2, P_3 and are served in FCFS order



Average waiting time = $\frac{0 + 24 + 27}{3}$ = 17 milliseconds

If the processes arrive in order P_2, P_3, P_1



Average waiting time in this case = $(3 + 6 + 0)/2 = 3$ ms

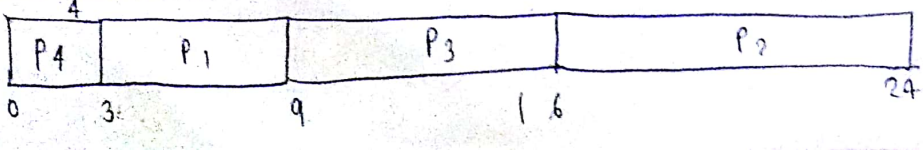
Convoy Effect : FCFS algorithm is non-preemptive in nature, ie, once CPU has been allocated to a process, other processes can get CPU time only after the current {process has finishes}.
This property leads to a situation called { Convoy Effect.
{ refer geeks for geeks → Convoy Effect } → either by terminating or by requesting I/O

Shortest Job First Scheduling

This algorithm associates with each process the length of process's next CPU Burst. When the CPU is available, it is assigned to the process that has the smallest next CPU Burst. If the next CPU Bursts of 2 processes are same FCFS scheduling is used to break the tie.

Process	Burst Time
P_1	6
P_2	8
P_3	7
P_4	3

Avg waiting time = $\frac{3 + 16 + 9 + 0}{4}$ = 7 ms



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

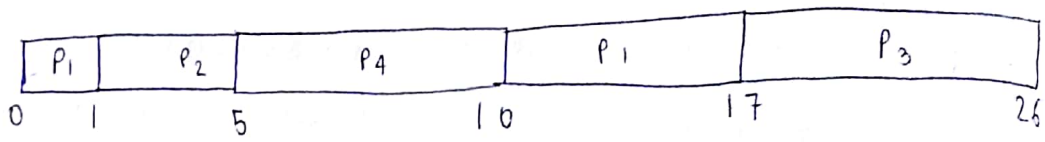
$t_n \longrightarrow$ Length of n^{th} CPU Burst

$\tau_{n+1} \longrightarrow$ Predicted value of next CPU Burst

The SJF Algorithm can either be pre-emptive or non-preempti

⇒ Preemptive SJF Scheduling is sometimes called shortest-remaining-time-first scheduling.

Process	Arrival Time	Burst Time
P_1	0	8
P_2	1	4
P_3	2	9
P_4	3	5



Average waiting time :

$$[(10-1) + (1-1) + (17-2) + (5-3)] / 4 = \frac{26}{4} = 6.5 \text{ ms}$$

($(10-1)$ for P_1, $(1-1)$ for P_2, $(17-2)$ for P_3, $(5-3)$ for P_4)

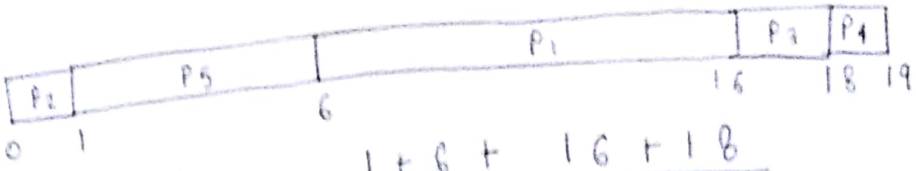
Priority Scheduling

⇒ SJF Algorithm is special case of priority scheduling algorithm

⇒ Priorities are indicated by some fixed range of numbers such as 0 to 7 or 0 to 4095.

There is no general agreement on whether 0 is the highest or lowest priority. It depends on the system

Process	Burst Time	Priority
P_1	10	3
P_2	1	1
P_3	2	4
P_4	1	5
P_5	5	2



Waiting time : $\frac{1 + 6 + 16 + 18}{5}$

$= 8.2$ ms

Priority scheduling can either be pre-emptive or non pre-emptive.

⇒ Major problem with priority scheduling algorithm is indefinite blocking or starvation.
A process that is ready to run but waiting for the CPU can be considered blocked.

Solution to this problem

Aging → It is a technique of gradually increasing the priority of processes that wait in the system for a long time.

Round Robin Scheduling

⇒ It is designed specially for time-sharing systems

⇒ Similar to FCFS scheduling but preemption is added to enable the system to switch b/w processes.

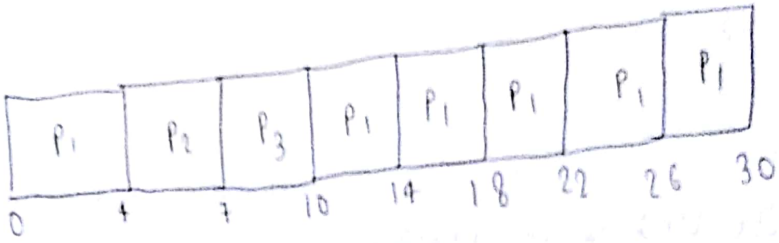
A small unit of time called the time quantum or time slice is defined. A time quantum is generally 10 to 100 milliseconds in length. The ready queue is treated as circular queue. The CPU scheduler goes around the ready queue allocating the CPU to each process for a time interval of up to 1 time quantum.

To implement RR scheduling, we keep the ready queue as FIFO queue of processes. New processes are added to the tail of the ready queue. The CPU scheduler picks the first process from the ready queue, sets a timer to interrupt after 1 time quantum and dispatches the process.

=>

Process	Burst Time
P_1	24
P_2	3
P_3	3

Time Quantum = 4 sec



P_1 (waiting time) = 10 - 4 = 6 ms

P_2 = 4 ms

P_3 = 7 ms

Avg waiting time = 17/3 = 5.66 ms

RR scheduling algorithm is preemptive

! => If there are n processes in the ready queue and time quantum is q

= Each process must wait no longer than (n-1) x q time units until its next time quantum

=> If 5 processes time quantum → 20 ms each process will get up to 20 ms every 100 ms.

⇒ If time quantum is extremely small, the RR approach is called processor sharing and (in theory) creates the appearance that each of the n processes has its own processor running at 1/n the speed of real processor

↳ Used in CDC (Control Data Corporation)

⇒ Time Quantum should be large wrt Context switch time

If the context switch time is approximately 10 percent of the time quantum, then about 10 percent of CPU time will be spent in context switching.

⇒ Turnaround time also depends on the size of time quantum

Time Quantum should be large as compared to context switch time but not too large otherwise it degenerates to FCFS policy. Rule of thumb → 80% of CPU bursts should be shorter than time quantum.

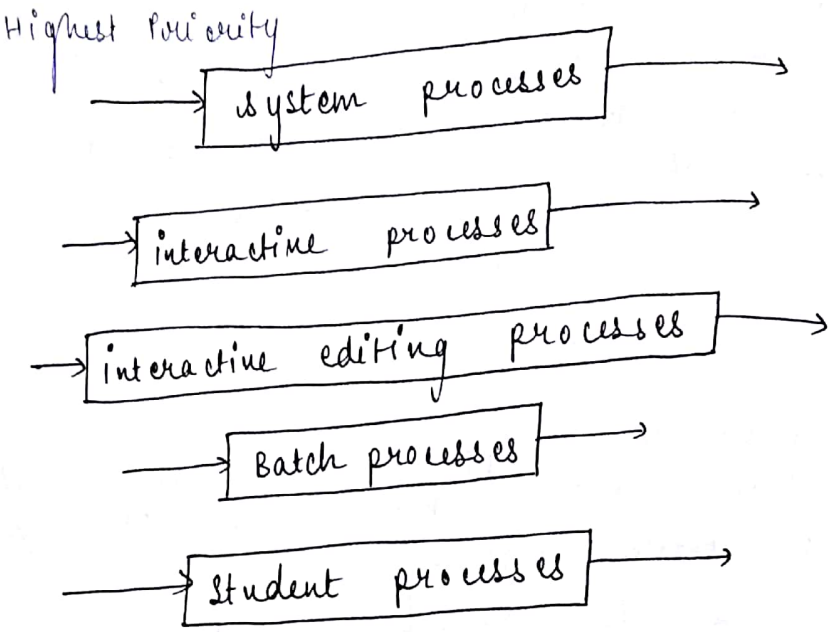
Multilevel queue scheduling

⇒ For situations where the processes are easily classified into different groups:

Foreground (interactive) processes Background (batch) processes

These 2 types of processes have different response time requirements and so may have different scheduling needs. Foreground processes may have priority (externally defined) over background processes.

A multilevel queue scheduling algorithm partitions the ready queue into several separate queues.



Each queue has its own scheduling algorithm. In addition, there must be scheduling among the queues, which is commonly implemented as fixed priority pre-emptive scheduling.
advantage – low scheduling overhead but inflexible

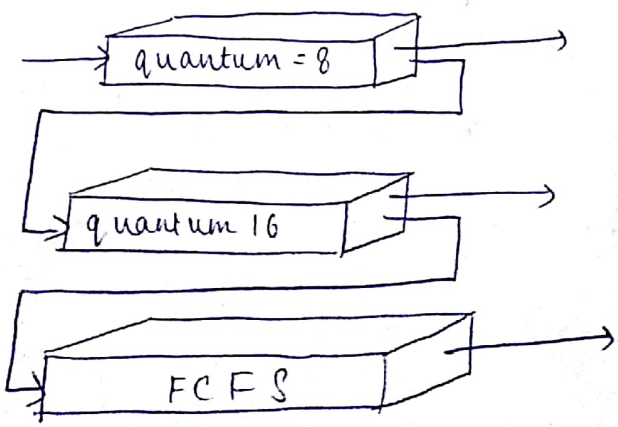
Multilevel feedback queue scheduling

⇒ Allows process to move b/w queues
⇒ Separate processes according to the characteristics of their CPU Bursts

Pv

⇒ Refer Pg 198 A process entering a ready queue is put in queue 0. - - - - - - - .

⇒



It is it defined by following parameters

1. No of queues
2. Scheduling Algo for each queue
3. Method used to determine when to upgrade a process to a higher priority queue
4. Method used to determine when to demote a process to lower priority queue
5. Method used to determine which queue a process will enter when the process needs service.