CPU Scheduling

CPU Scheduling

CPU scheduling is a process which allows one process to use the CPU while the execution of another process is on hold (in waiting state) due to unavailability of any resource like I/O etc, thereby making full use of CPU. The aim of CPU scheduling is to make the system efficient, fast and fair.

Scheduling Criteria

There are many different criteria to check when considering the "best" scheduling algorithm :

- **CPU utilization**

  To make out the best use of CPU and not to waste any CPU cycle, CPU would be working most of the time (Ideally 100% of the time). Considering a real system, CPU usage should range from 40% (lightly loaded) to 90% (heavily loaded.)

- **Throughput**

  It is the total number of processes completed per unit time or rather say total amount of work done in a unit of time. This may range from 10/second to 1/hour depending on the specific processes.

- **Turnaround time**

  It is the amount of time taken to execute a particular process, i.e. The interval from time of submission of the process to the time of completion of the process (Wall clock time).

- **Waiting time**

  The sum of the periods spent waiting in the ready queue amount of time a process has been waiting in the ready queue to acquire get control on the CPU.

- **Load average**

  It is the average number of processes residing in the ready queue waiting for their turn to get into the CPU.

- **Response time**

  Amount of time it takes from when a request was submitted until the first response is produced. Remember, it is the time till the first response and not the completion of process execution (final response).

In general CPU utilization and Throughput are maximized and other factors are reduced for proper optimization.

Basic Concepts

In a single-processor system, only one process can run at a time. Others must wait until the CPU is free and can be rescheduled. The objective of multiprogramming is to have some process running at all times, to maximize CPU utilization. A process is executed until it must wait, typically for the completion of some I/O request. In a simple computer system, the CPU then just sits idle. All this waiting time is wasted; no useful work is accomplished. With Multiprogramming, we try to use this time productively. Several processes are kept in memory at one time.

When one process has to wait, the operating system takes the CPU away from that process and gives the CPU to another process. Every time one process has to wait, another process can take over use of the CPU. Scheduling of this kind is a fundamental operating-system function. Almost all computer resources are scheduled before use. The CPU is, of course, one of the primary computer resources. Thus, its **scheduling is central to operating-system design**.

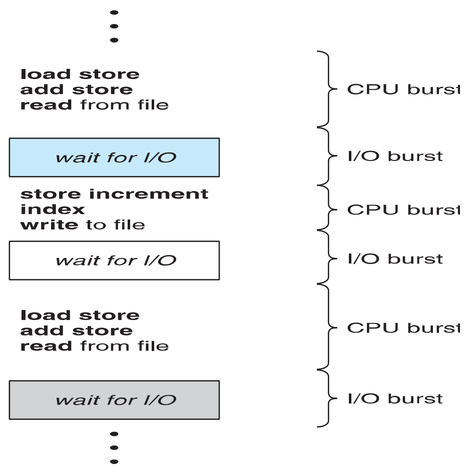


Figure 6.1 Alternating sequence of CPU and I/O bursts.

CPU Burst: While scheduling, each process gets to use the CPU for it's slice. The slice that it gets, is called the CPU burst. In simple terms, the duration for which a process gets control of the CPU is the CPU burst time, and the concept of gaining control of the CPU is the CPU burst.

The time when the process is being executed in the CPU, i.e. CPU is the resource being used by the process at that time.

I/O Burst : I/O burst- The time when the process requests for I/O and is using I/O as a resource i.e. I/O burst.

CPU–I/O Burst Cycle

Process execution consists of a **cycle** of CPU execution and I/O wait. Processes alternate between these two states. Process execution begins with a **CPU burst**. That is followed by an **I/O burst**, which is followed by another CPU burst, then another I/O burst, and so on. Eventually, the final CPU burst ends with a system request to terminate execution (Figure 6.1).

The durations of CPU bursts have been measured extensively. Although they vary from process to process and from computer to computer, they tend to have a frequency curve similar to that shown in Figure 6.2. The curve is generally characterized as exponential or hyperexponential, with a large number of short CPU bursts and a small number of long CPU bursts. An I/O-bound program typically has many short CPU bursts. A CPU-bound program might have a few long CPU bursts. This distribution can be important in the selection of an appropriate CPU-scheduling algorithm.

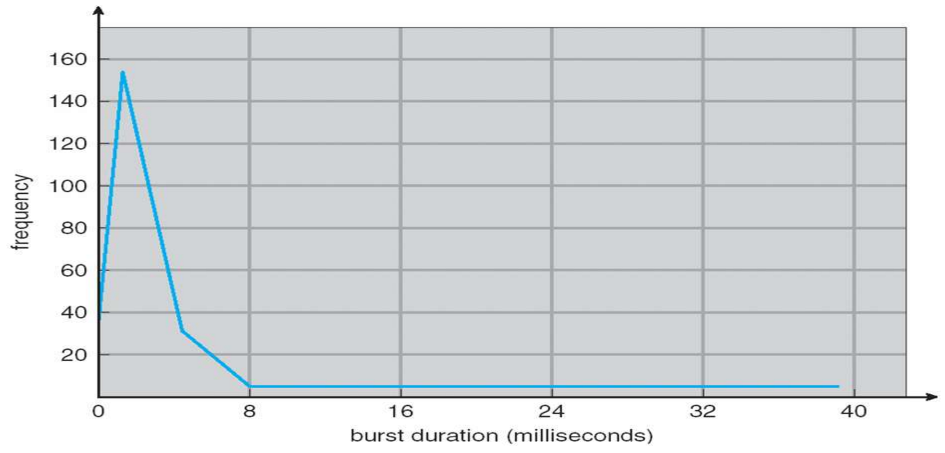


Figure 6.2 Histogram of CPU-burst durations.

CPU Scheduler:

CPU scheduler selects a process among the processes that are ready to execute and allocates CPU to one of them.

Whenever the CPU becomes idle, the operating system must select one of the processes in the ready queue to be executed. The selection process is carried out by the **short-term scheduler**, or CPU scheduler. The scheduler selects a process from the processes in memory that are ready to execute and allocates the CPU to that process.

Commented [A1]: Scheduling Queues: Job Queue, Ready Queue, Device Queue

Commented [A2R1]:

Commented [A3]:

1. Short Term Scheduler. It is also called as CPU scheduler. CPU scheduler selects a process among the processes that are ready to execute and allocates CPU to one of

them. Short-term schedulers, also known as dispatchers, make the decision of which process to execute next.

2. Long Term Scheduler: Selects which process should be brought in the ready queue.

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

1. When a process switches from the running state to the waiting state (for example, as the result of an I/O request or an invocation of wait () for the termination of a child process)

2. When a process switches from the running state to the ready state (for example, when an interrupt occurs)
3. When a process switches from the waiting state to the ready state (for example, at completion of I/O)
4. When a process terminates

When scheduling takes place only under circumstances 1 and 4, we say that the scheduling scheme is non-preemptive or cooperative. Otherwise, it is preemptive.

- All other scheduling is **preemptive** (require special hardware)

  - Preemptive scheduling can result in race conditions when data are shared among several processes.
  - Consider preemption while in kernel mode
  - Consider interrupts occurring during crucial OS activities
  - Need synchronization

- **Dispatcher**
  Another component involved in the CPU-scheduling function is the dispatcher. The dispatcher is the module that gives control of the CPU to the process selected by the short-term scheduler. This function involves the following:
  • Switching context
  • Switching to user mode
  •Jumping to the proper location in the user program to restart that program
  •The dispatcher 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 the dispatch latency.

Scheduling Criteria

- **CPU utilization** – keep the CPU as busy as possible
- **Throughput** – # of processes that complete their execution per time unit
- **Turnaround time** – amount of time to execute a particular process
- **Waiting time** – amount of time a process has been waiting in the ready queue

- **Response time** – amount of time it takes from when a request was submitted until the first response is produced, not output (for time-sharing environment)
- Response time is the amount of time after which a process gets the CPU for the first time after entering the ready queue.
- Response Time = Time at which process first gets the CPU – Arrival time

Scheduling Algorithm Optimization Criteria

- Max CPU utilization
- Max throughput
- Min turnaround time
- Min waiting time
- Min response time

Scheduling Algorithms

We'll discuss four major scheduling algorithms here which are the following :

1. First Come First Serve(FCFS) Scheduling
2. Shortest-Job-First(SJF) Scheduling
3. Priority Scheduling
4. Round Robin(RR) Scheduling
5. Multilevel Queue Scheduling

First-Come, First-Served (FCFS) Scheduling

- Simplest CPU scheduling algorithm
- The process that requests the CPU first is allocated the CPU first
- Managed by a FIFO queue
- When a process enters the ready queue, its 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

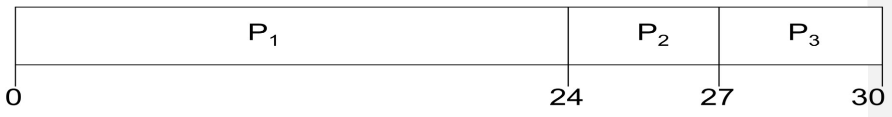
1. **Problem**
   - Average waiting time often quite long

Process	Burst Time
P_1	24

P_2	3
P_3	3

- Suppose that the processes arrive in the order: P_1 , P_2 , P_3

  The Gantt Chart for the schedule is:

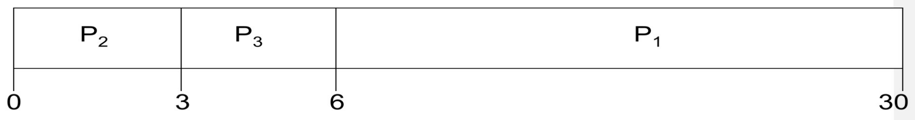


- Waiting time for P_1 = 0; P_2 = 24; P_3 = 27
- Average waiting time: (0 + 24 + 27)/3 = 17

Suppose that the processes arrive in the order:

P_2 , P_3 , P_1

- The Gantt chart for the schedule is:



- Waiting time for P_1 = 6; P_2 = 0; P_3 = 3
- Average waiting time: (6 + 0 + 3)/3 = 3
- Much better than previous case
- **Convoy effect** - short process behind long process
  - Consider one CPU-bound and many I/O-bound processes

There is a convoy effect as all the other processes wait for the one big process to get off the CPU. This effect results in lower CPU and device utilization than might be possible if the shorter processes were allowed to go first.

Shortest-Job-First (SJF) Scheduling

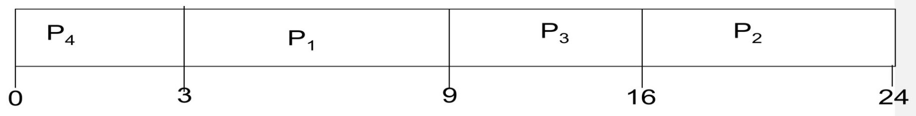
- Associate with each process the length of its next CPU burst
  - Use these lengths to schedule the process with the shortest time
- SJF is optimal – gives minimum average waiting time for a given set of processes
  - The difficulty is knowing the length of the next CPU request
  - Could ask the user

2. Problem

Process	Burst Time
P_1	6
P_2	8
P_3	7

P_4	3

- SJF scheduling chart



- Average waiting time = (3 + 16 + 9 + 0) / 4 = 7

Shortest Job first (with Arrival time and Burst Time):

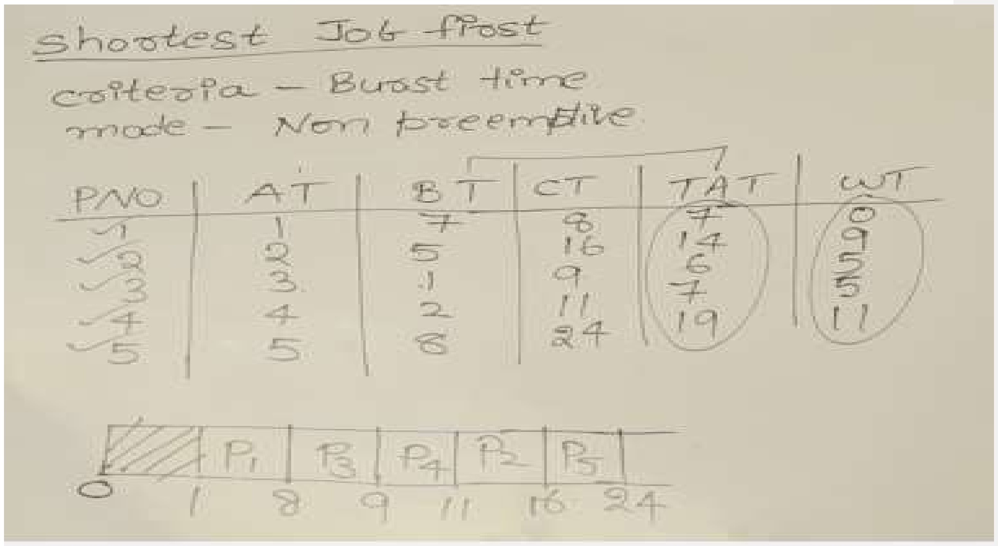
3. Problem

Process	Arrival Time	Burst Time
P_1	1	7
P_2	2	5
P_3	*3*	*1*
P_4	4	2
P4	5	8

AT= Arrival Time , BT= Burst Time ,CT= Completion Time, TAT=Turn around Time WT= Waiting Time.

CT gets from figure P1=8, p2=16, p3=9,p4=11 and p5=24

TAT=CT-AT; WT=TAT-BT



Shortest Job first

criteria – Burst time

mode – Non preemptive

P.NO	AT	BT	CT	TAT	WT
✓1	1	7	8	7	0
✓2	2	5	16	14	9
✓3	3	1	9	6	5
✓4	4	2	11	7	5
✓5	5	8	24	19	11

▨	P1	P3	P4	P2	P5

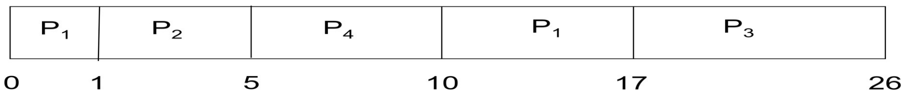
0 1 8 9 11 16 24

Example of Shortest-remaining-time-first (Preemptive)

- Now we add the concepts of varying arrival times and preemption to the analysis

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

- *Preemptive* SJF Gantt Chart

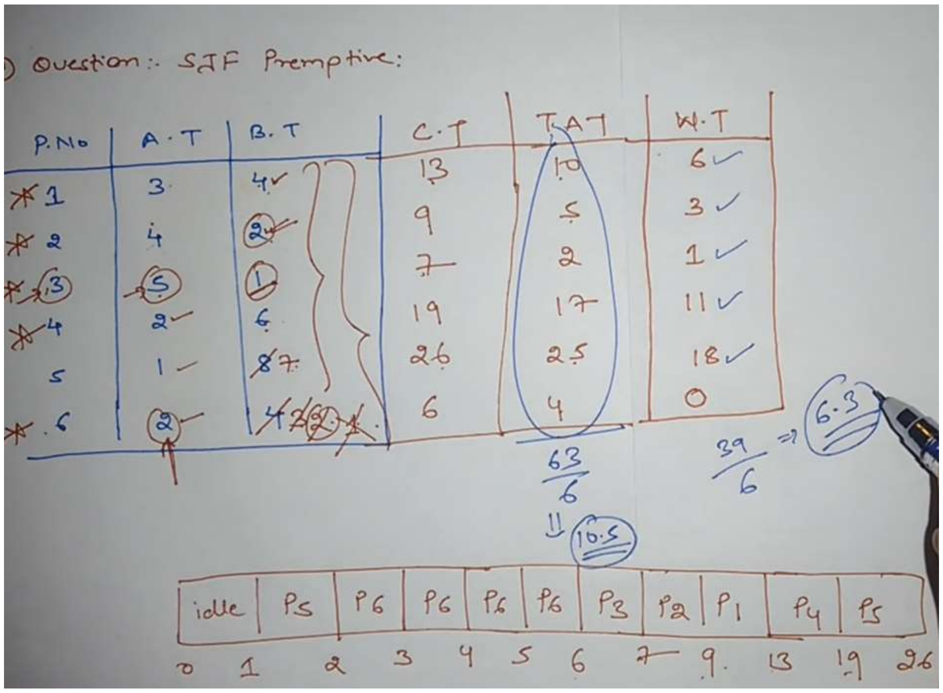


CT: P1=17,p2=5,p3=26,p4=10

TAT=CT-AT

WT=TAT-BT

Average waiting time= 9 +0+15+2=26/4=6.5 msec



Question :- SJF Premptive:

P.No	A.T	B.T	C.T	TAT	W.T
* 1	3	4 ✓	13	10	6 ✓
* 2	4	2 ~~4~~	9	5	3 ✓
* 3	5	~~1~~	7	2	1 ✓
* 4	2 ✓	6	19	17	11 ✓
5	1 ✓	~~8~~ 7	26	25	18 ✓
* 6	2 ✓	~~4~~ ~~3~~ 2 ~~1~~	6	4	0

$\frac{63}{6} = 10.5$

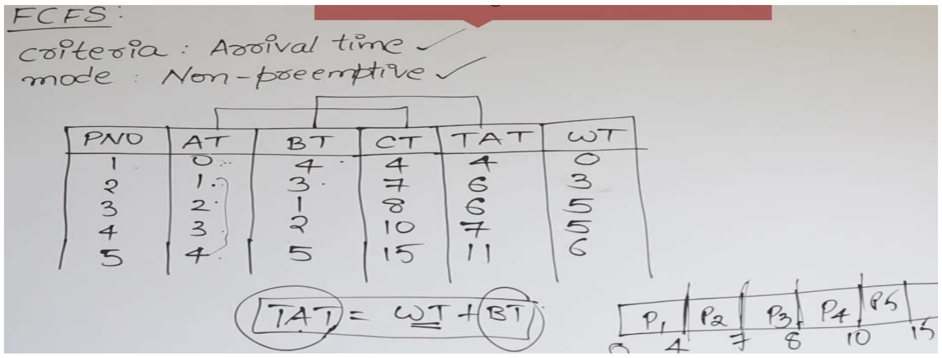
$\frac{39}{6} \Rightarrow 6.3$

idle	P5	P6	P6	P6	P6	P3	P2	P1	P4	P5

0 1 2 3 4 5 6 7 9 13 19 26

Examples:

FCFS algorithm:



FCFS:

criteria : Arrival time ✓

mode : Non-preemptive ✓

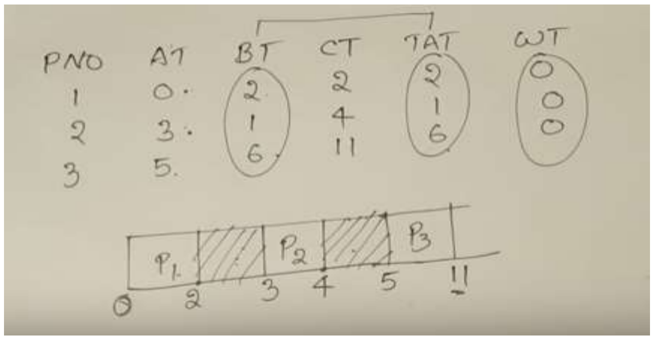
PNO	AT	BT	CT	TAT	WT
1	0	4	4	4	0
2	1	3	7	6	3
3	2	1	8	6	5
4	3	2	10	7	5
5	4	5	15	11	6

$\boxed{TAT} = WT + BT$

P1	P2	P3	P4	P5

0 4 7 8 10 15

FCFS with CPU idle time:

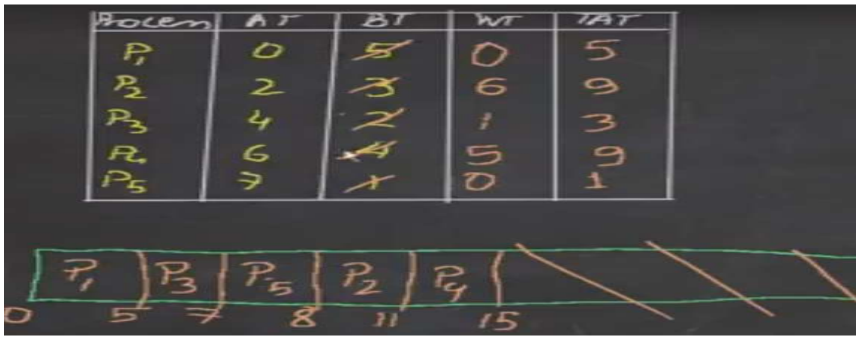


PNO	AT	BT	CT	TAT	WT
1	0	2	2	2	0
2	3	1	4	1	0
3	5	6	11	6	0

P1	▨	P2	▨	P3

0 2 3 4 5 11

SJF- Non-preemptive:

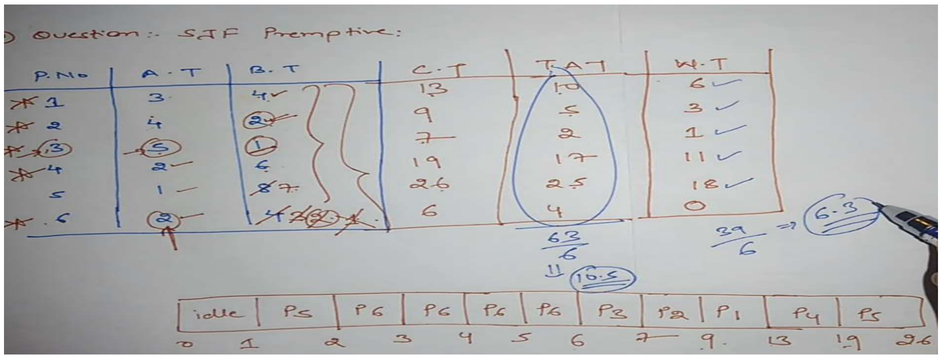


Process	AT	BT	WT	TAT
P1	0	~~5~~	0	5
P2	2	~~3~~	6	9
P3	4	~~2~~	1	3
P4	6	~~4~~	5	9
P5	7	~~1~~	0	1

P1	P3	P5	P2	P4

0 5 7 8 11 15

SJF---preemptive:



Question :- SJF Premptive:

P.No	A.T	B.T	C.T	T.A.T	W.T
1	3	4	13	10	6 ✓
2	4	2	9	5	3 ✓
3	5	1	7	2	1 ✓
4	2	6	19	17	11 ✓
5	1	~~8~~ 7	26	25	18 ✓
6	2	~~4~~ ~~2~~ ~~1~~	6	4	0

$\frac{63}{6} = \boxed{10.5}$

$\frac{39}{6} \Rightarrow \boxed{6.3}$

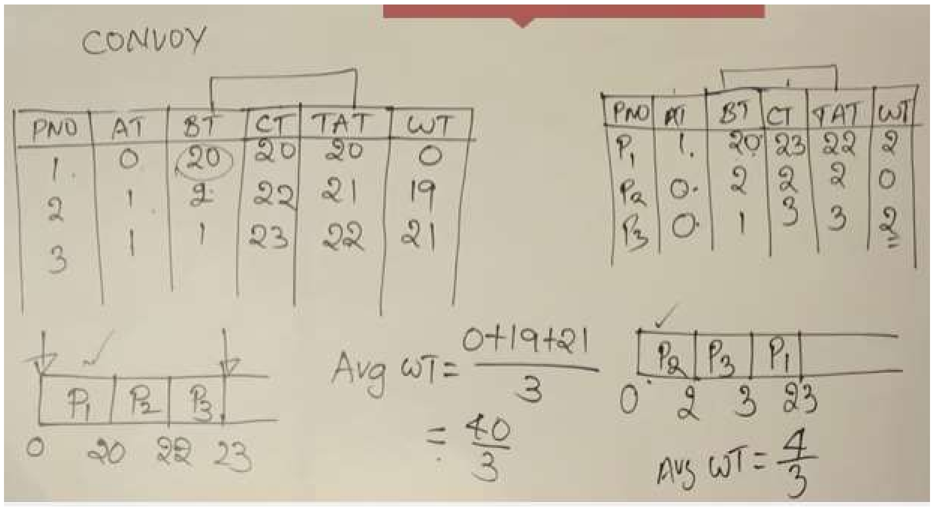
idle	P5	P6	P6	P6	P6	P3	P2	P1	P4	P5

0 1 2 3 4 5 6 7 9 13 19 26

What is convoy effect?

Convoy effect is slowing down of the whole operating system because of few slow processes. It's usually used in context of vehicle traffics. Basically, a **convoy** affected region is the one where the slow moving traffic slows down the speed of all the vehicles and makes the whole process lengthier.

It is the disadvantage of FCFS algorithm.
SJF algorithm used to solved the convoy effect.


CONVOY
P1 P2 P3
0 20 22 23
P2 P3 P1
0 2 3 23

PNO	AT	BT	CT	TAT	WT
1.	0	20	20	20	0
2	1	2	22	21	19
3	1	1	23	22	21

$$\text{Avg } WT = \frac{0+19+21}{3} = \frac{40}{3}$$

PNO	AT	BT	CT	TAT	WT
P_1	1.	20	23	22	2
P_2	0.	2	2	2	0
P_3	0.	1	3	3	2

$$\text{Avg } WT = \frac{4}{3}$$

What is difference between kernel and operating system?

The **operating system** is the software package that communicates directly to the hardware and our application.
The **kernel** is the lowest level of the **operating system**. The **kernel** is the main part of the **operating system** and is responsible for translating the command into something that can be understood by the computer.

Priority Scheduling

- A priority number (integer) is associated with each process
- The CPU is allocated to the process with the highest priority (smallest integer ≡ highest priority)
  - Preemptive
  - Non-preemptive
  - SJF is priority scheduling where priority is the inverse of predicted next CPU burst time
- Problem ≡ **Starvation** – low priority processes may never execute
- Solution ≡ **Aging** – as time progresses increase the priority of the process

Starvation: Starvation is resource management problem where a process does not get the resources it needs for a long time because the resources are being allocated to other processes.

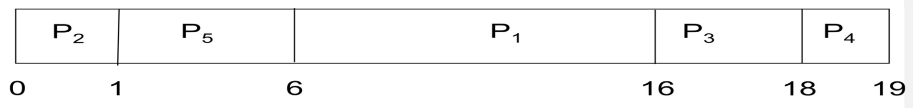
Starvation is the name given to the indefinite postponement of a process because it requires some resource before it can run, but the resource, though available for allocation, is never allocated to this process.

Aging: Aging is a technique to avoid starvation in a scheduling system.

Example of Priority Scheduling(Non Preemptive)

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

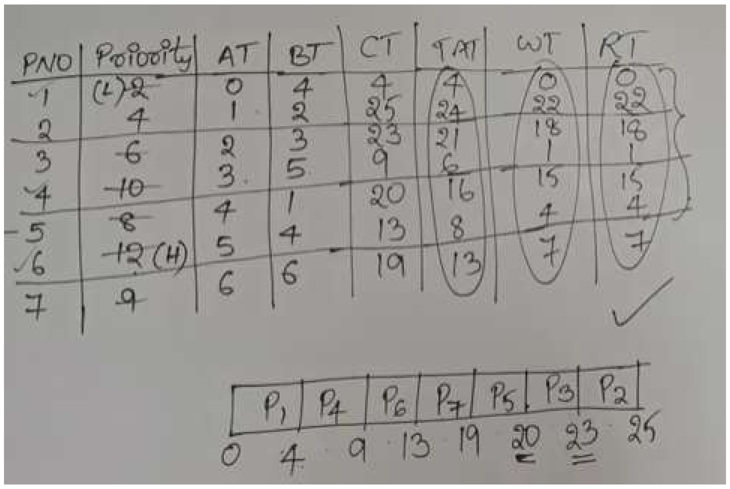
- Priority scheduling Gantt Chart



Average waiting time = 8.2 msec

Example of Priority Scheduling(Non Preemptive)

Process	Priority	Arrival Time	Burst Time
P_1	2	0	4
P_2	4	1	2
P3	6	2	3
P4	10	3	5
P5	8	4	1
p6	12	5	4
p7	9	6	6

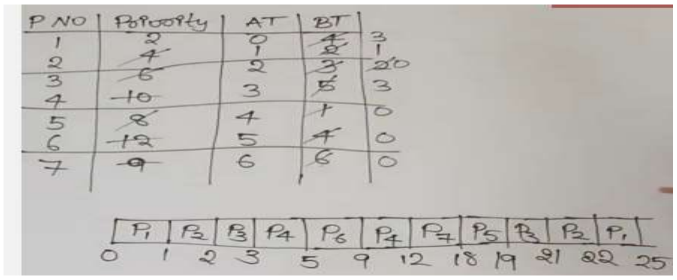


PNO	Priority	AT	BT	CT	TAT	WT	RT
1	(L) ~~2~~	0	4	4	4	0	0
2	4	1	2	25	24	22	22
3	~~6~~	2	3	23	21	18	18
4	~~10~~	3	5	9	6	1	1
5	~~8~~	4	1	20	16	15	15
6	~~12~~ (H)	5	4	13	8	4	4
7	~~9~~	6	6	19	13	7	7

P1	P4	P6	P7	P5	P3	P2

0 4 9 13 19 20 23 25

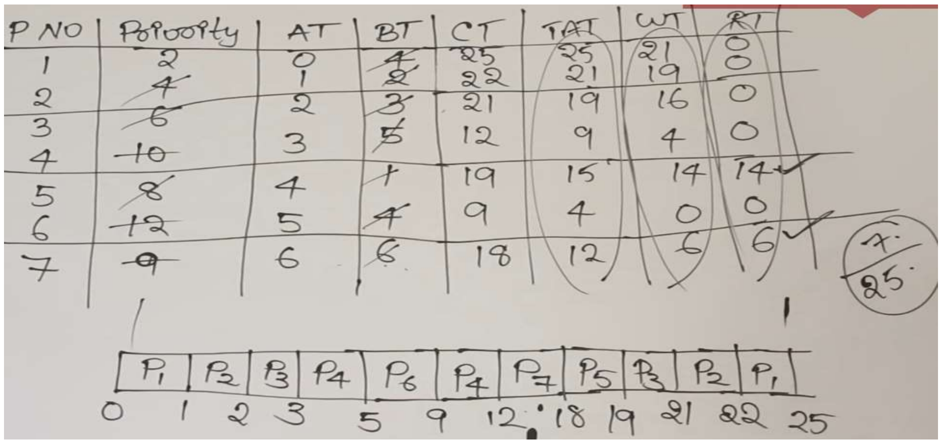
Example of Priority Scheduling(Preemptive)



P NO	Priority	AT	BT	
1	2	0	~~4~~	3
2	~~4~~	1	~~2~~	1
3	~~6~~	2	~~3~~	~~2~~ 0
4	~~10~~	3	~~5~~	3
5	~~8~~	4	~~1~~	0
6	~~12~~	5	~~4~~	0
7	~~9~~	6	~~6~~	0

P1	P2	P3	P4	P6	P4	P7	P5	P3	P2	P1

0 1 2 3 5 9 12 18 19 21 22 25



P NO	Priority	AT	BT	CT	TAT	WT	RT
1	2	0	~~4~~	25	25	21	0
2	~~4~~	1	~~2~~	22	21	19	0
3	~~6~~	2	~~3~~	21	19	16	0
4	~~10~~	3	~~5~~	12	9	4	0
5	~~8~~	4	~~1~~	19	15	14	14 ✓
6	~~12~~	5	~~4~~	9	4	0	0
7	~~9~~	6	~~6~~	18	12	6	6 ✓

$\frac{7}{25}$

P_1	P_2	P_3	P_4	P_6	P_4	P_7	P_5	P_3	P_2	P_1

0 1 2 3 5 9 12 18 19 21 22 25

Round Robin (RR)

- Each process gets a small unit of CPU time (**time quantum** *q*), usually 10-100 milliseconds. After this time has elapsed, the process is preempted and added to the end of the ready queue.
- If there are *n* processes in the ready queue and the time quantum is *q*, then each process gets 1/*n* of the CPU time in chunks of at most *q* time units at once. No process waits more than (*n*-1)*q* time units.
- Timer interrupts every quantum to schedule next process
- Performance
  - *q* large $\Rightarrow$ FIFO
  - *q* small $\Rightarrow$ *q* must be large with respect to context switch, otherwise overhead is too high

Example of RR with Time Quantum = 4

Process	Burst Time
P_1	24
P_2	3
P_3	3

- The Gantt chart is:

P_1	P_2	P_3	P_1	P_1	P_1	P_1	P_1

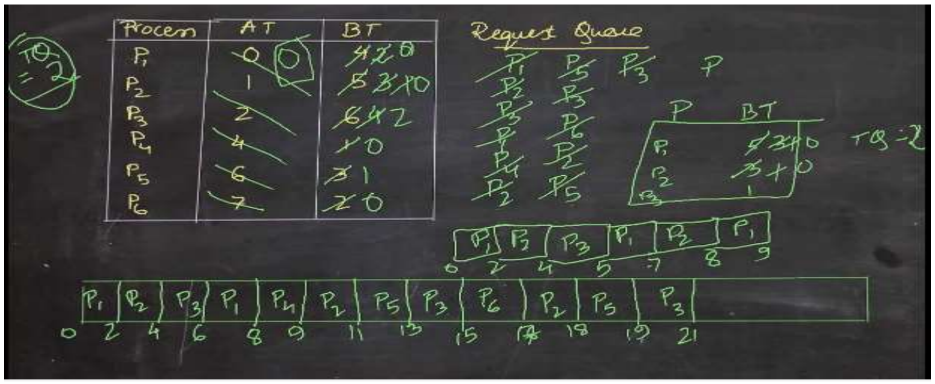
0 4 7 10 14 18 22 26 30

- Typically, higher average turnaround than SJF, but better ***response***
- q should be large compared to context switch time
- q usually 10ms to 100ms, context switch < 10 u sec

What is context switching in operating system?

In computing, a **context switch** is the process of storing and restoring the state (more specifically, the **context**) of a process or thread so that execution can be resumed from the same point at a later time.

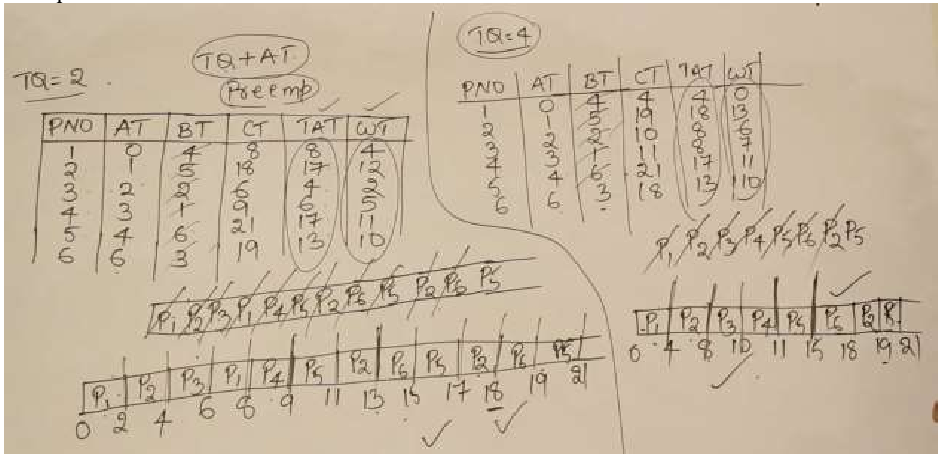
Example:


TQ = 2
Request Queue
P1 P2 P3 P1 P4 P2 P5 P3 P6 P2 P5 P3 P
P BT
P1 ~~4~~ ~~3~~ ~~1~~ 0
P2 ~~3~~ ~~1~~ 0
P3 1
TQ = 2
P1 P2 P3 P1 P2 P1
0 2 4 5 7 8 9

Process	AT	BT
P1	~~0~~ 0	~~4~~ ~~2~~ 0
P2	1	~~5~~ ~~3~~ ~~1~~ 0
P3	~~2~~	~~6~~ ~~4~~ 2
P4	~~4~~	~~1~~ 0
P5	~~6~~	~~3~~ 1
P6	~~7~~	~~2~~ 0

P1	P2	P3	P1	P4	P2	P5	P3	P6	P2	P5	P3	
0	2	4	6	8	9	11	13	15	17	18	19	21

Example:


TQ+AT
Preemp
P1 P2 P3 P1 P4 P5 P2 P6 P5 P2 P6 P5
P1 P2 P3 P4 P5 P6 P2 P5

TQ = 2

PNO	AT	BT	CT	TAT	WT
1	0	4	8	8	4
2	1	5	18	17	12
3	2	2	6	4	2
4	3	1	9	6	5
5	4	6	21	17	11
6	6	3	19	13	10

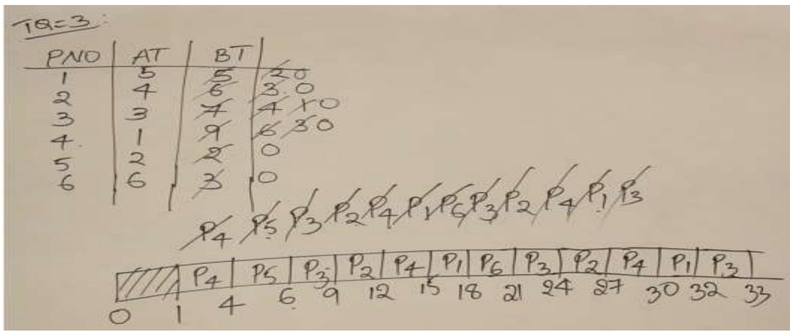
P1	P2	P3	P1	P4	P5	P2	P6	P5	P2	P6	P5	
0	2	4	6	8	9	11	13	15	17	18	19	21

TQ=4

PNO	AT	BT	CT	TAT	WT
1	0	4	4	4	0
2	1	5	19	18	13
3	2	2	10	8	6
4	3	1	11	8	7
5	4	6	21	17	11
6	6	3	18	13	10

P1	P2	P3	P4	P5	P6	P2	P5	
0	4	8	10	11	15	18	19	21

Example:



TQ=3:

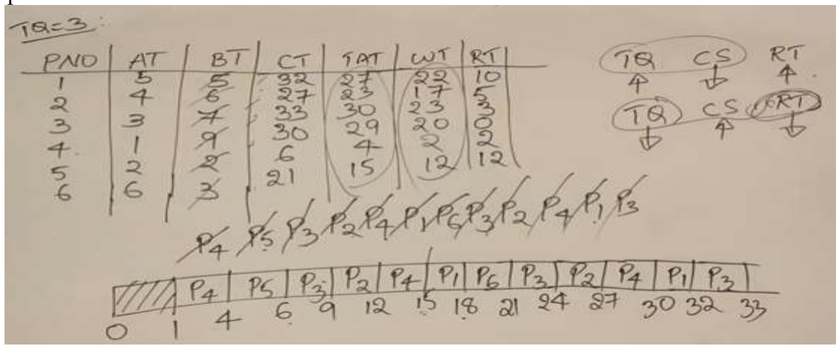
PNO	AT	BT	
1	5	~~5~~	~~2~~ 0
2	4	~~6~~	~~3~~ 0
3	3	~~7~~	~~4~~ ~~1~~ 0
4	1	~~9~~	~~6~~ ~~3~~ 0
5	2	~~2~~	0
6	6	~~3~~	0

~~P4~~ ~~P5~~ ~~P3~~ ~~P2~~ ~~P4~~ ~~P1~~ ~~P6~~ ~~P3~~ ~~P2~~ ~~P4~~ ~~P1~~ ~~P3~~

▨	P4	P5	P3	P2	P4	P1	P6	P3	P2	P4	P1	P3

0 1 4 6 9 12 15 18 21 24 27 30 32 33

Example:



TQ=3:

PNO	AT	BT	CT	TAT	WT	RT
1	5	~~5~~	32	27	22	10
2	4	~~6~~	27	23	17	5
3	3	~~7~~	33	30	23	3
4	1	~~9~~	30	29	20	0
5	2	~~2~~	6	4	2	2
6	6	~~3~~	21	15	12	12

TQ ↑ CS ↓ RT ↑

TQ ↓ CS ↑ RT ↓

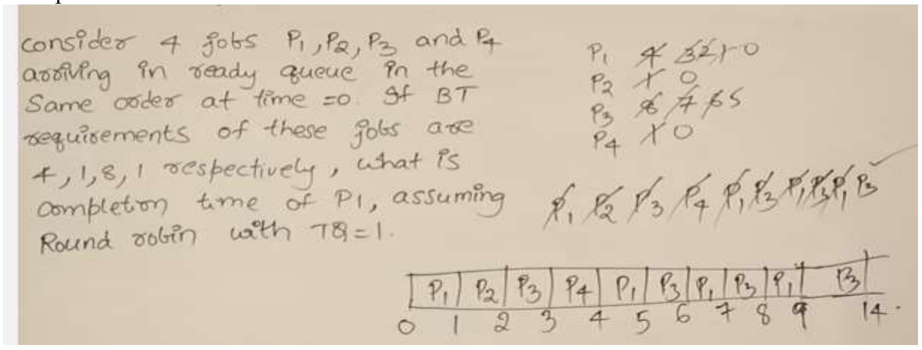
~~P4~~ ~~P5~~ ~~P3~~ ~~P2~~ ~~P4~~ ~~P1~~ ~~P6~~ ~~P3~~ ~~P2~~ ~~P4~~ ~~P1~~ ~~P3~~

▨	P4	P5	P3	P2	P4	P1	P6	P3	P2	P4	P1	P3

0 1 4 6 9 12 15 18 21 24 27 30 32 33

Time quantum is increased: - Context switching is decreased and Response time is increased.
Time quantum is decreased: Context switching is increased and Response time is decreased.

Example:



consider 4 jobs P_1, P_2, P_3 and P_4 arriving in ready queue in the Same order at time =0. If BT requirements of these jobs are 4, 1, 8, 1 respectively, what is completion time of P1, assuming Round robin with TQ=1.

P_1	~~4~~ ~~3~~ ~~2~~ ~~1~~ 0
P_2	~~1~~ 0
P_3	~~8~~ ~~7~~ ~~6~~ 5
P_4	~~1~~ 0

~~P_1~~ ~~P_2~~ ~~P_3~~ ~~P_4~~ ~~P_1~~ ~~P_3~~ ~~P_1~~ ~~P_3~~ ~~P_1~~ P_3

P_1	P_2	P_3	P_4	P_1	P_3	P_1	P_3	P_1	P_3

0 1 2 3 4 5 6 7 8 9 14.

Multilevel Queue

- Ready queue is partitioned into separate queues, eg:
    - **foreground** (interactive)
    - **background** (batch)
- Process permanently in a given queue
    - Each queue has its own scheduling algorithm:
    - foreground – RR
    - background – FCFS
    - Scheduling must be done between the queues:
    - Fixed priority scheduling; (i.e., serve all from foreground then from background). Possibility of starvation.
    - Time slice – each queue gets a certain amount of CPU time which it can schedule amongst its processes; i.e., 80% to foreground in RR
    - 20% to background in FCFS

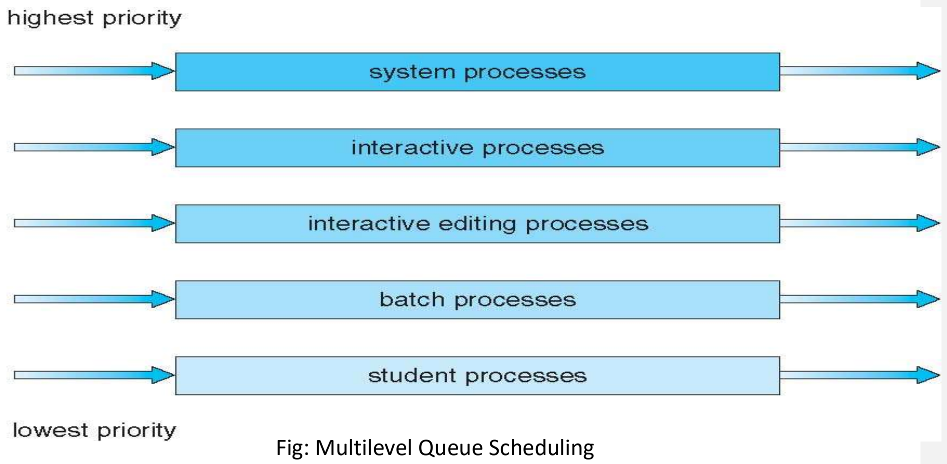
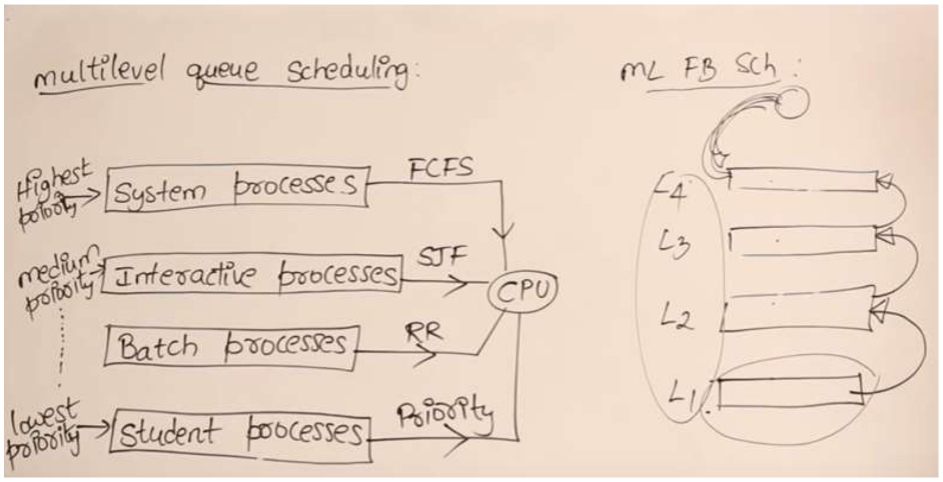


Fig: Multilevel Queue Scheduling



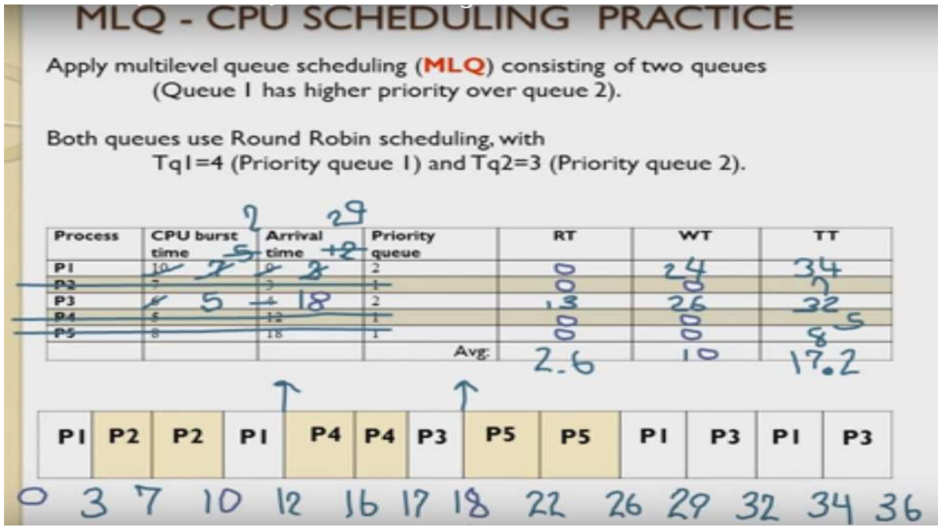
System Process: In computing, a **process** is an instance of a computer program that is being executed. It contains the program code and its current activity. Depending on the operating **system**(OS), a **process** may be made up of multiple threads of execution that execute instructions concurrently.

Example: OS
Interactive Processes: When we playing online game.
Batch processes: submit lots of jobs

What is a batch processing system?
Batch processing is the execution of a series of jobs in a program on a computer without manual intervention (non-interactive). Strictly speaking, it is a **processing** mode: the execution of a series of programs each on a set or "**batch**" of inputs, rather than a single input (which would instead be a custom job).

Batch processing: collecting jobs in a single batch and then execute them without further interaction what the user. Interactive processing: Allows a program being executed to carry on a dialogue with the user through remote terminals. Requires real-time processing.

Student process: It might have executed some program.



MLQ - CPU SCHEDULING PRACTICE

Apply multilevel queue scheduling (**MLQ**) consisting of two queues
(Queue 1 has higher priority over queue 2).

Both queues use Round Robin scheduling, with
Tq1=4 (Priority queue 1) and Tq2=3 (Priority queue 2).

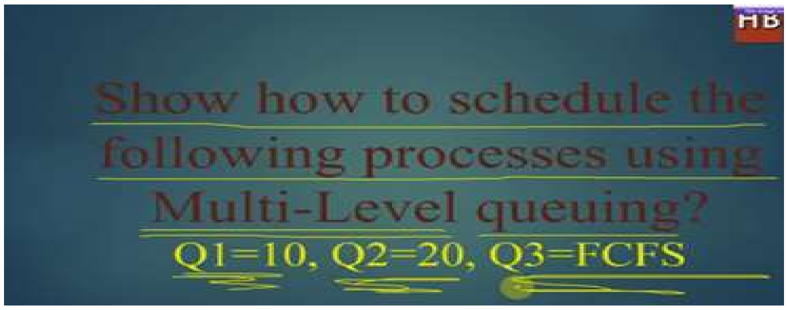
2 29

Process	CPU burst time −5	Arrival time +2	Priority queue	RT	WT	TT
P1	~~10~~ 7	~~0~~ 8	2	0	24	34
~~P2~~	~~7~~	~~3~~	~~1~~	0	0	7
P3	~~6~~ 5	~~4~~ 18	2	13	26	32
~~P4~~	~~5~~	~~12~~	~~1~~	0	0	5
~~P5~~	~~8~~	~~18~~	~~1~~	0	0	8
			Avg:	2.6	10	17.2

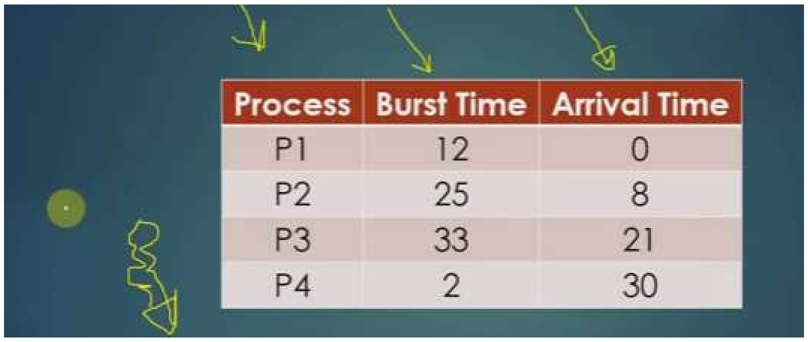
P1	P2	P2	P1	P4	P4	P3	P5	P5	P1	P3	P1	P3

0 3 7 10 12 16 17 18 22 26 29 32 34 36

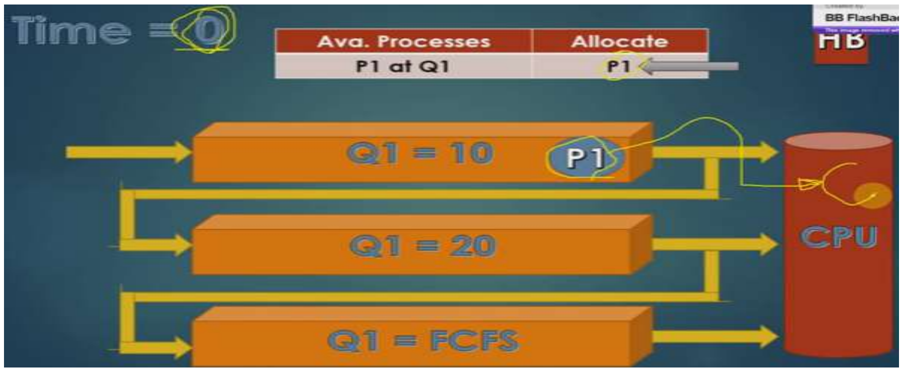
Example:



Show how to schedule the following processes using Multi-Level queuing?
Q1=10, Q2=20, Q3=FCFS

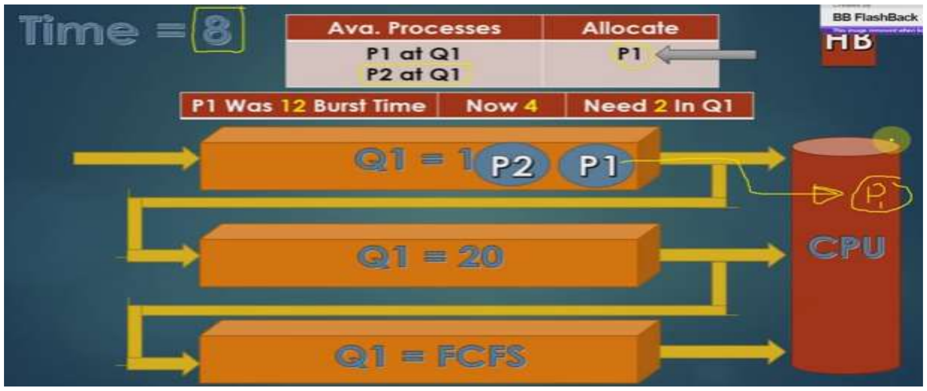


Process	Burst Time	Arrival Time
P1	12	0
P2	25	8
P3	33	21
P4	2	30



Time = 0

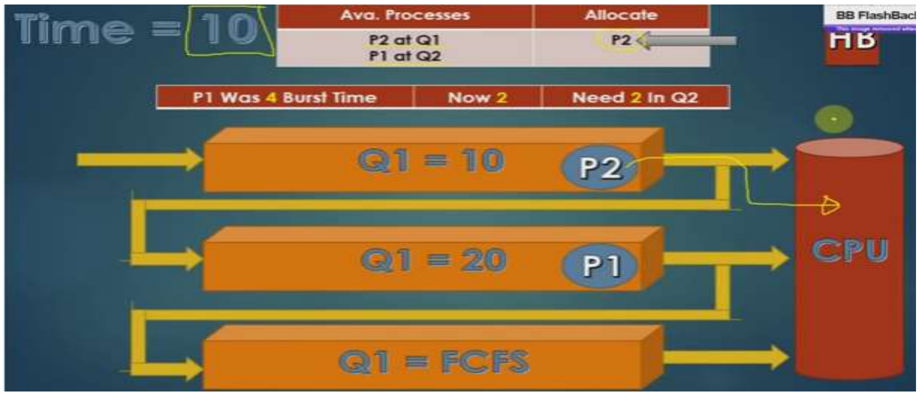
Ava. Processes	Allocate
P1 at Q1	P1



Time = 8

Ava. Processes	Allocate
P1 at Q1 P2 at Q1	P1

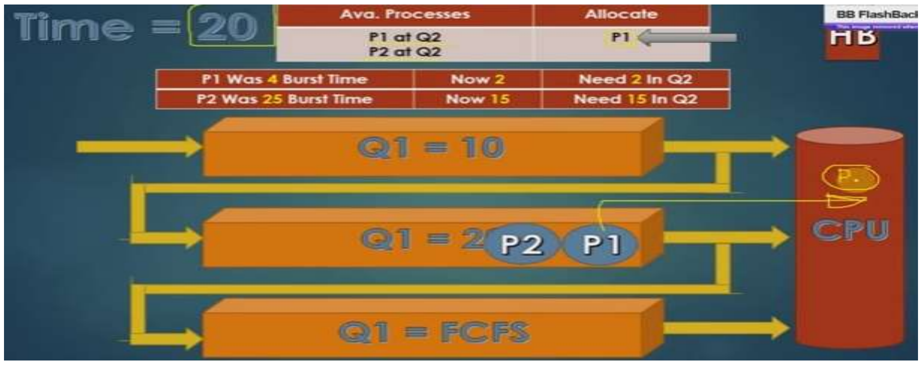
P1 Was 12 Burst Time	Now 4	Need 2 In Q1



Time = 10

Ava. Processes	Allocate
P2 at Q1 P1 at Q2	P2

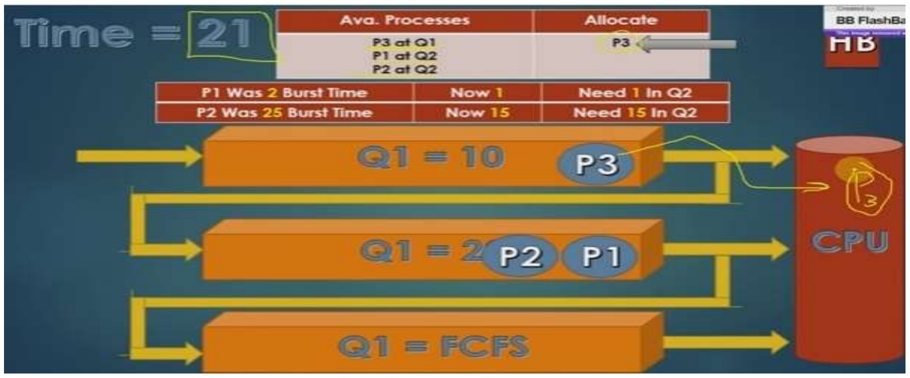
P1 Was 4 Burst Time	Now 2	Need 2 In Q2



Time = 20

Ava. Processes	Allocate
P1 at Q2 P2 at Q2	P1

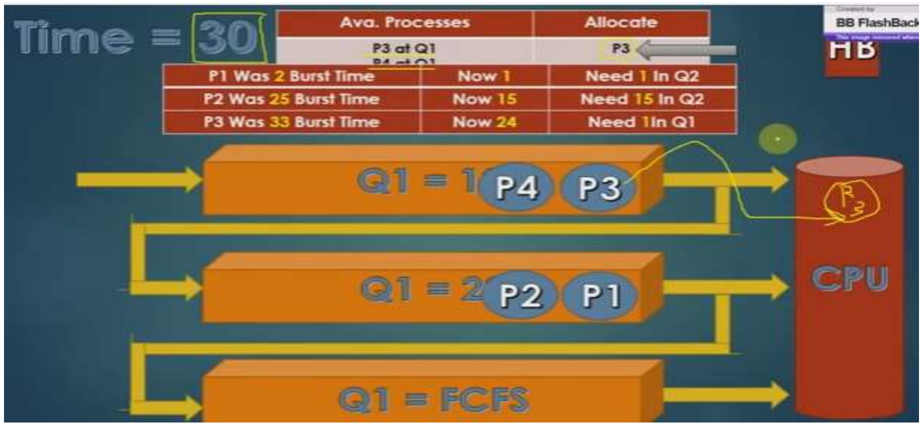
P1 Was 4 Burst Time	Now 2	Need 2 In Q2
P2 Was 25 Burst Time	Now 15	Need 15 In Q2



Time = 21

Ava. Processes	Allocate
P3 at Q1 P1 at Q2 P2 at Q2	P3

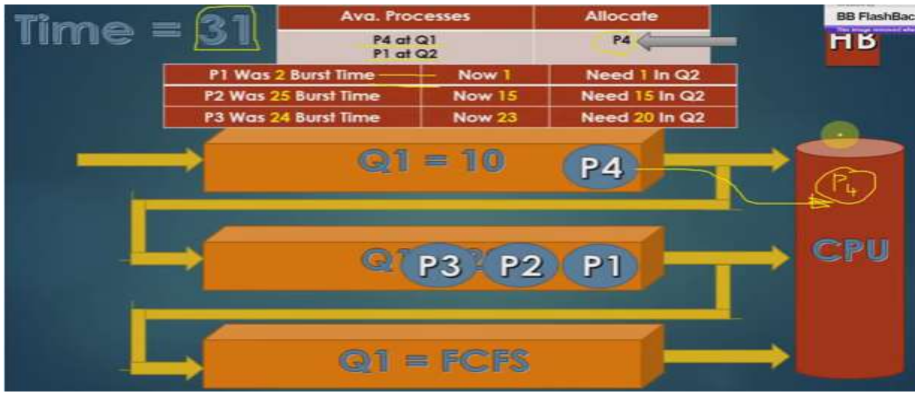
P1 Was 2 Burst Time	Now 1	Need 1 In Q2
P2 Was 25 Burst Time	Now 15	Need 15 In Q2



Time = 30

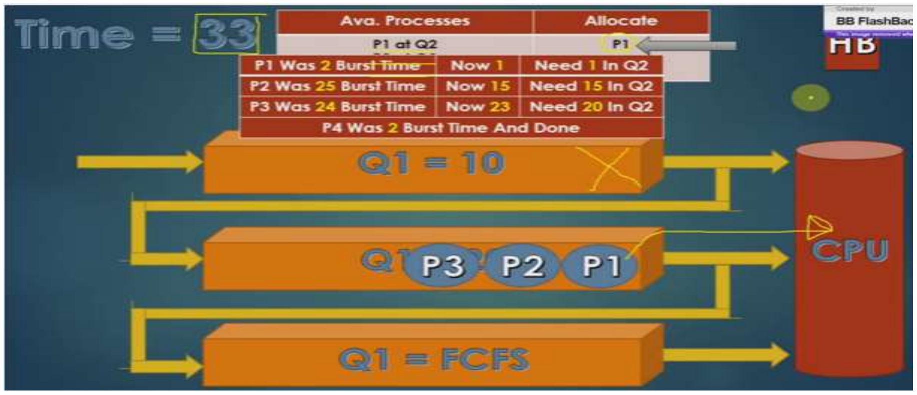
Ava. Processes	Allocate
P3 at Q1 P4 at Q1	P3

P1 Was 2 Burst Time	Now 1	Need 1 In Q2
P2 Was 25 Burst Time	Now 15	Need 15 In Q2
P3 Was 33 Burst Time	Now 24	Need 1In Q1


Time = 31

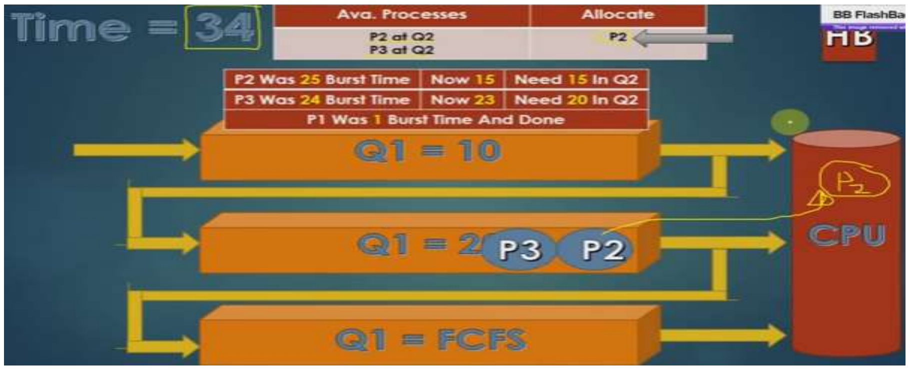
Ava. Processes	Allocate
P4 at Q1 P1 at Q2	P4

P1 Was 2 Burst Time	Now 1	Need 1 In Q2
P2 Was 25 Burst Time	Now 15	Need 15 In Q2
P3 Was 24 Burst Time	Now 23	Need 20 In Q2


Time = 33

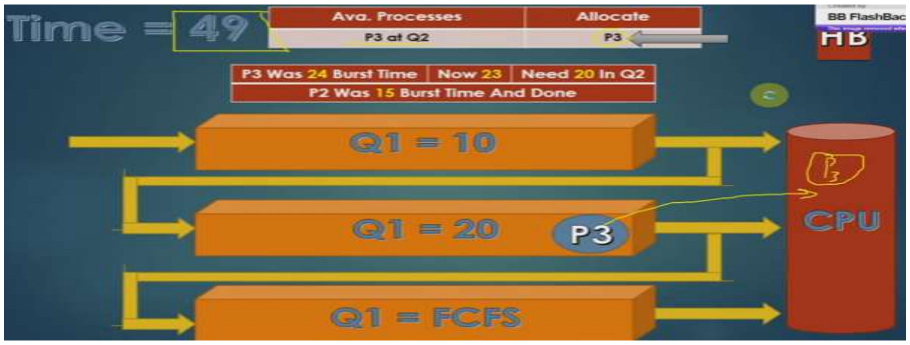
Ava. Processes	Allocate
P1 at Q2	P1

P1 Was 2 Burst Time	Now 1	Need 1 In Q2
P2 Was 25 Burst Time	Now 15	Need 15 In Q2
P3 Was 24 Burst Time	Now 23	Need 20 In Q2
P4 Was 2 Burst Time And Done		


Time = 34

Ava. Processes	Allocate
P2 at Q2 P3 at Q2	P2

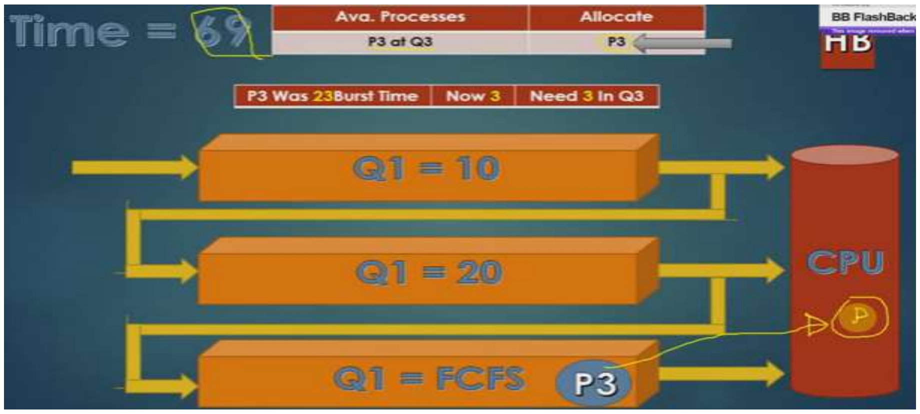
P2 Was 25 Burst Time	Now 15	Need 15 In Q2
P3 Was 24 Burst Time	Now 23	Need 20 In Q2
P1 Was 1 Burst Time And Done		


Time = 49

Ava. Processes	Allocate
P3 at Q2	P3

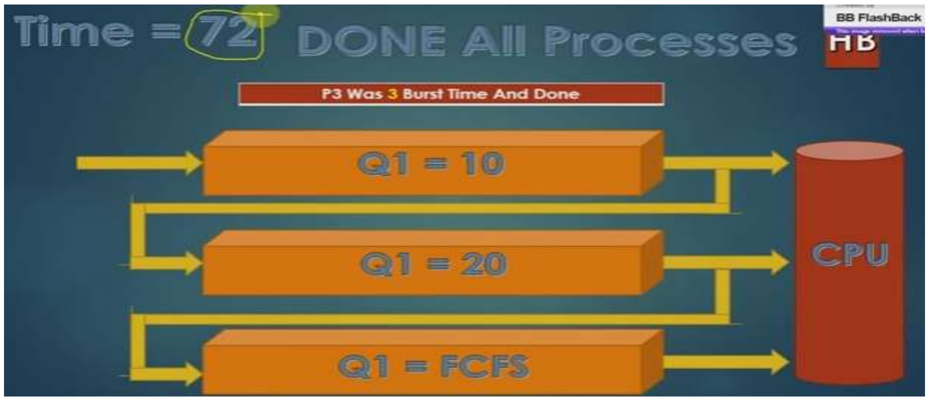
P3 Was 24 Burst Time | Now 23 | Need 20 In Q2

P2 Was 15 Burst Time And Done

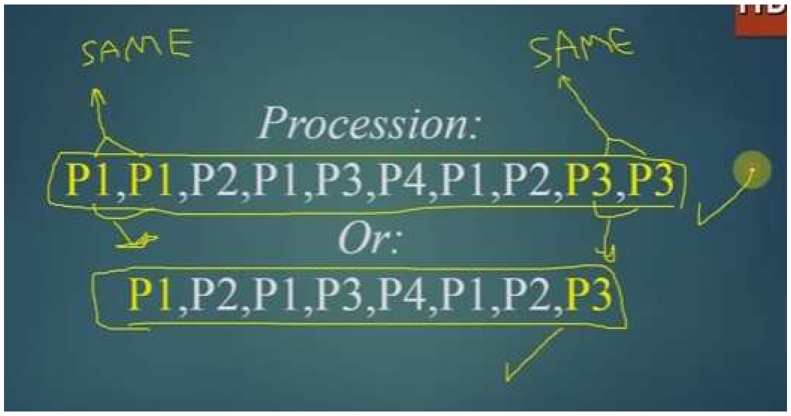

Time = 69

Ava. Processes	Allocate
P3 at Q3	P3

P3 Was 23Burst Time | Now 3 | Need 3 In Q3


Time = 72 DONE All Processes

P3 Was 3 Burst Time And Done



Procession:

P1,P1,P2,P1,P3,P4,P1,P2,P3,P3

Or:

P1,P2,P1,P3,P4,P1,P2,P3