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**.

6_01.pdf

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

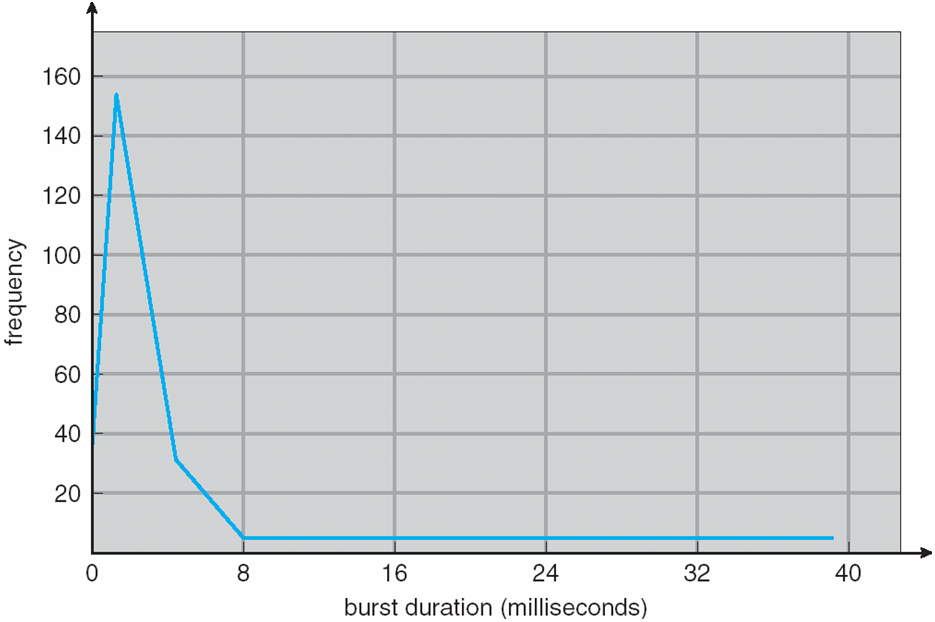
the process at that time.

**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.

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

*P1* 24

*P2* 3

*P3* 3

* Suppose that the processes arrive in the order: *P1* , *P2* , *P3*The Gantt Chart for the schedule is:
* Waiting time for *P1* = 0; *P2* = 24; *P3* = 27
* Average waiting time: (0 + 24 + 27)/3 = 17

Suppose that the processes arrive in the order:

*P2* , *P3* , *P1*

* The Gantt chart for the schedule is:
* Waiting time for *P1 =* 6*;P2* = 0*; P3 =* 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

1. **Problem**

Process Burst Time

*P1* 6

*P2*  8

*P3* 7

*P4* 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

*P1* 1 7

*P2*  2 5

*P3 3 1*

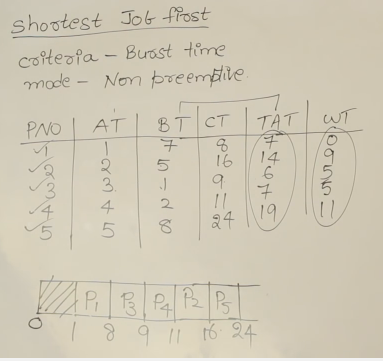
*P4* 4 2

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



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

*P1* 0 8

*P2* 1 4

*P3* 2 9

*P4* 3 5

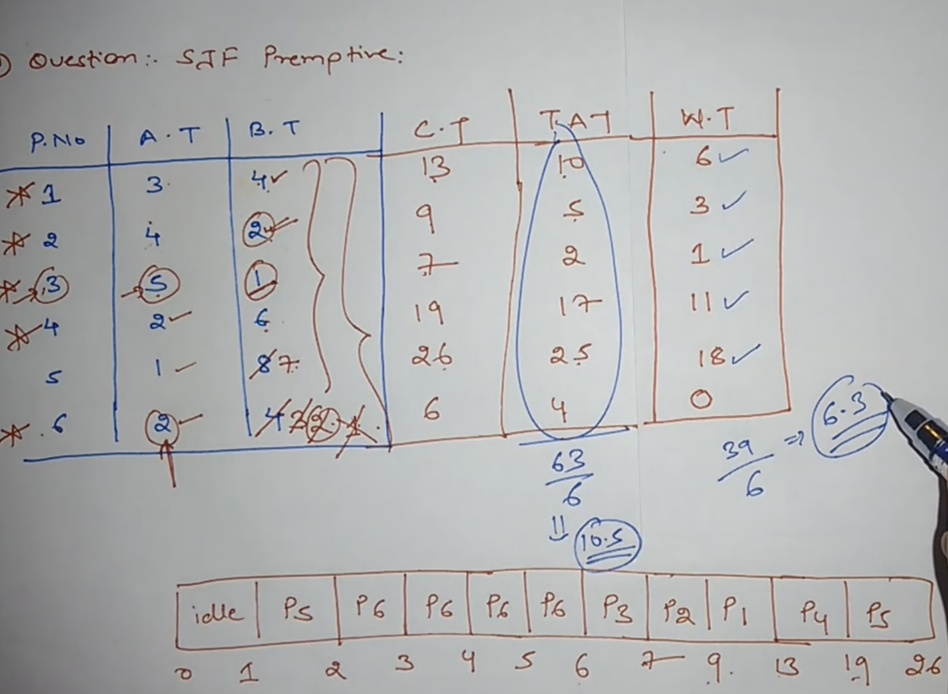
* *Preemptive* SJF Gantt Chart

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

TAT=CT-AT

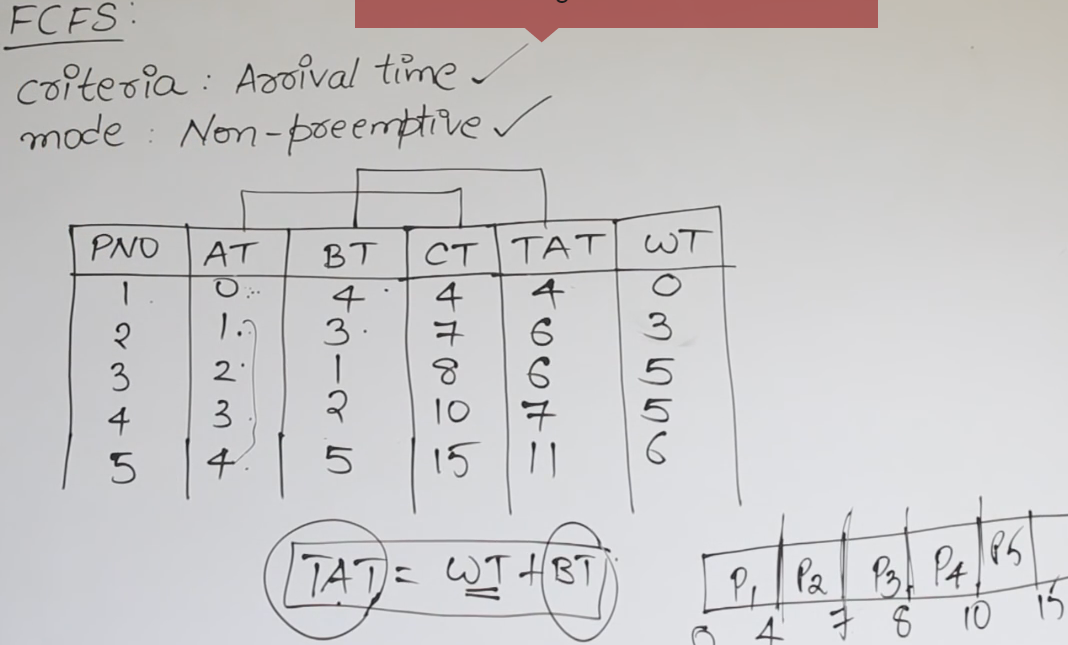
WT=TAT-BT

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



Examples:

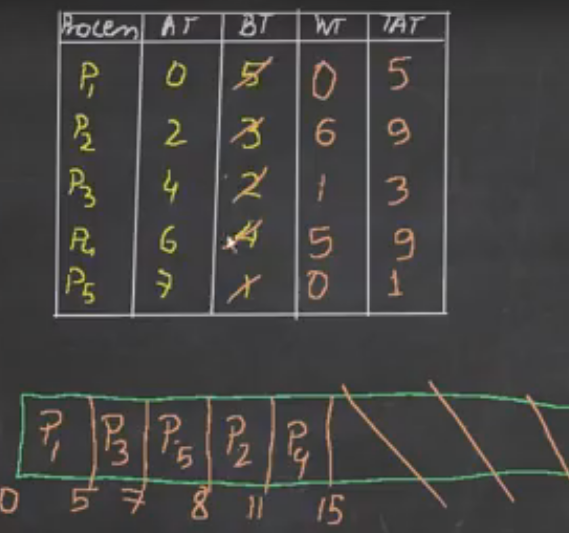
FCFS algorithm:



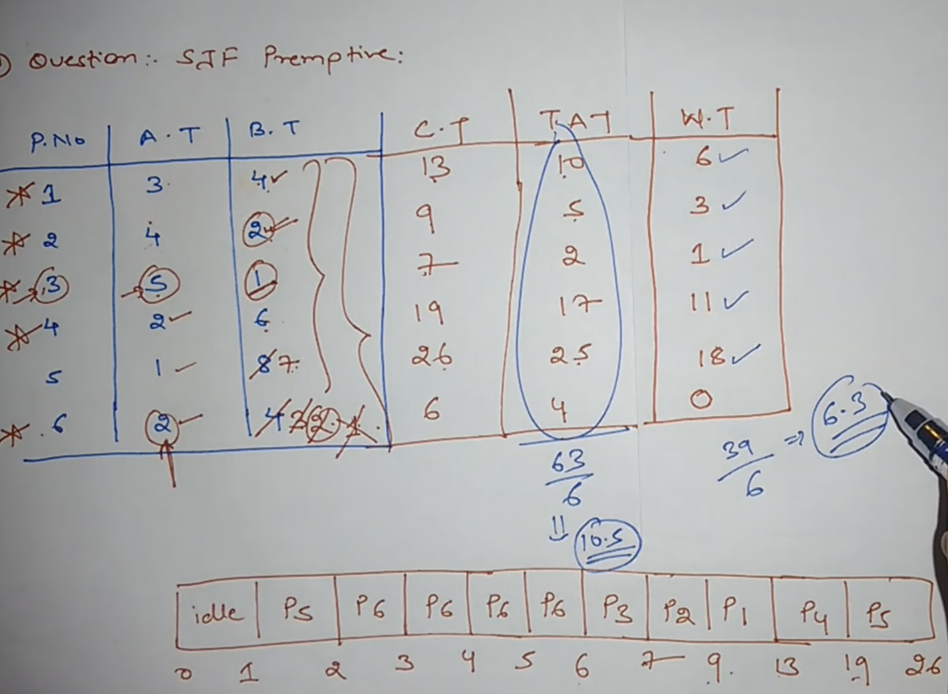
FCFS with CPU idle time:



SJF- Non-preemptive:



SJF---preemptive:

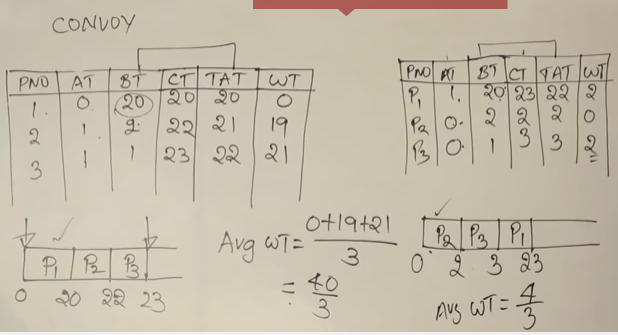


What is convoy effect?

**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.



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

*P1* 10 3

*P2* 1 1

*P3* 2 4

*P4* 1 5

*P5* 5 2

* Priority scheduling Gantt Chart

Average waiting time = 8.2 msec

Example of Priority Scheduling(Non Preemptive)

Process Priority Arrival Time Burst Time

*P1* 2 0 4

*P2*  4 1 2

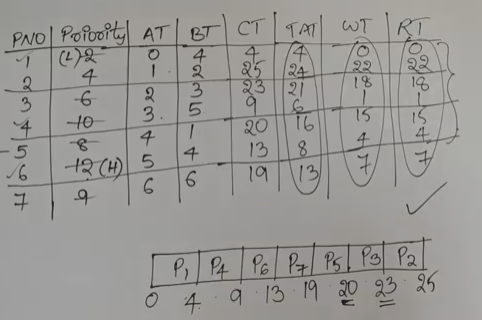
P3 6 2 3

P4 10 3 5

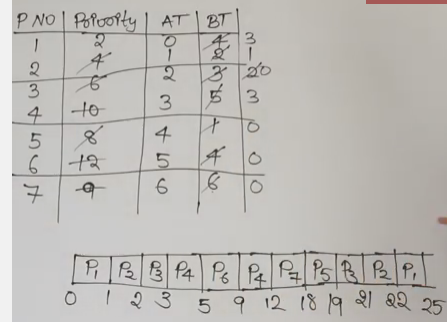
P5 8 4 1

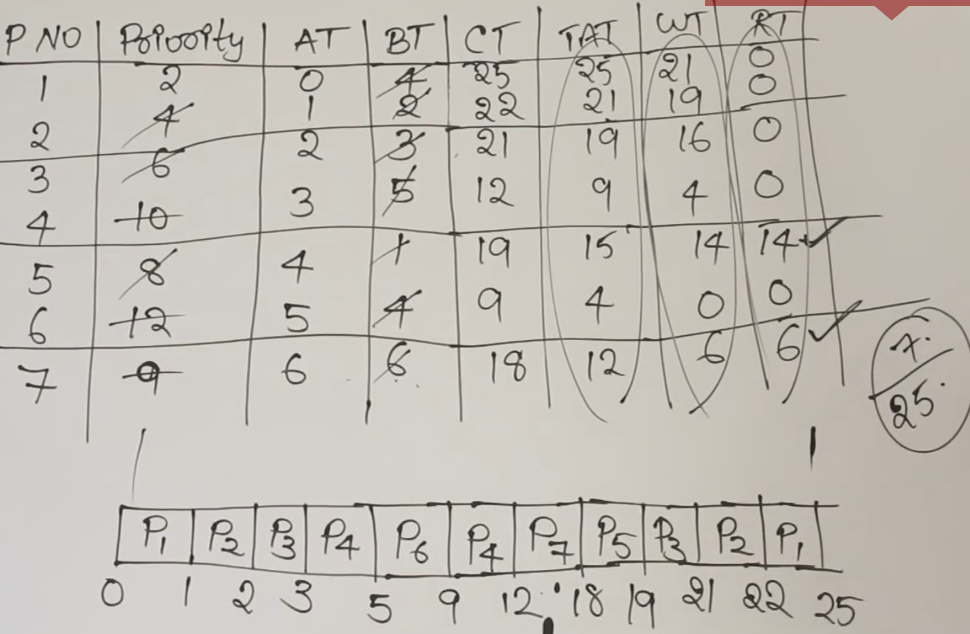
p6 12 5 4

p7 9 6 6



Example of Priority Scheduling(Preemptive)





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 ⇒ FIFO
  + *q* small ⇒ *q* must be large with respect to context switch, otherwise overhead is too high

Example of RR with Time Quantum = 4

Process Burst Time

*P1* 24

*P2* 3

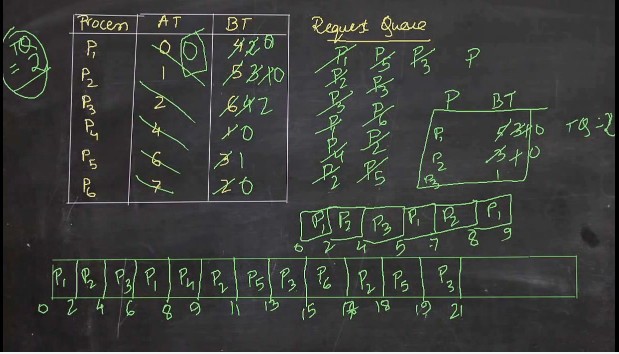
*P3* 3

* The Gantt chart is:
* 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

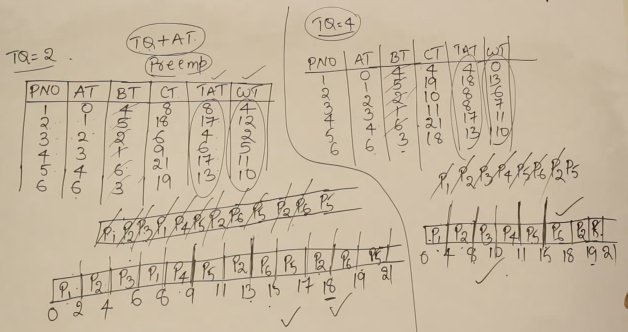
What is context switching in operating system?

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.

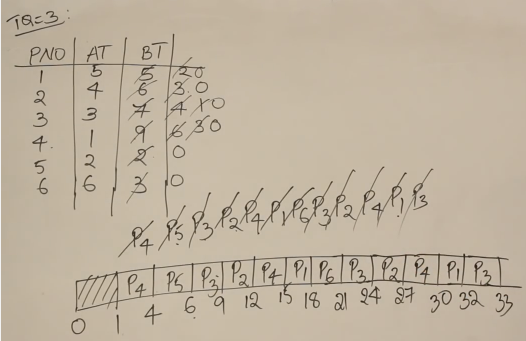
Example:



Example:



Example:

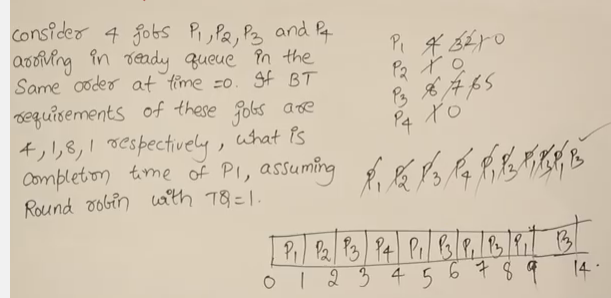


Example:



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: 

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

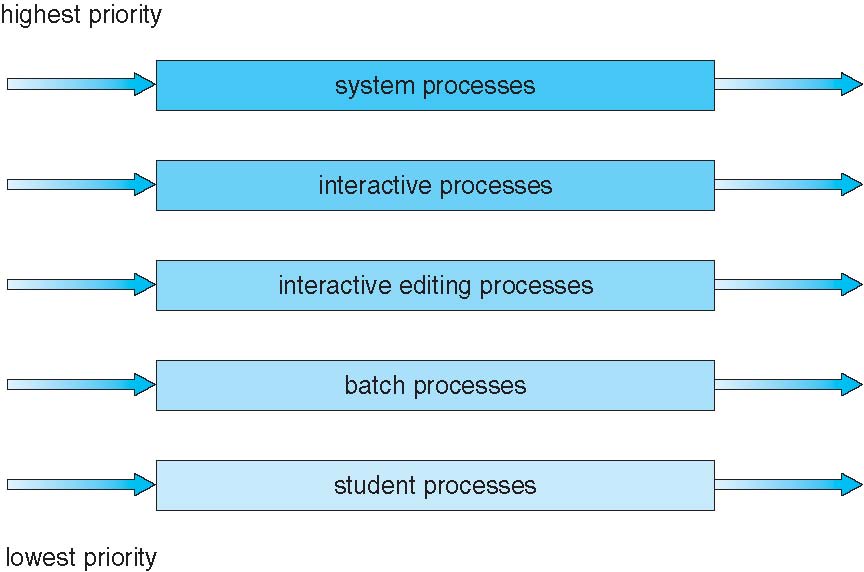
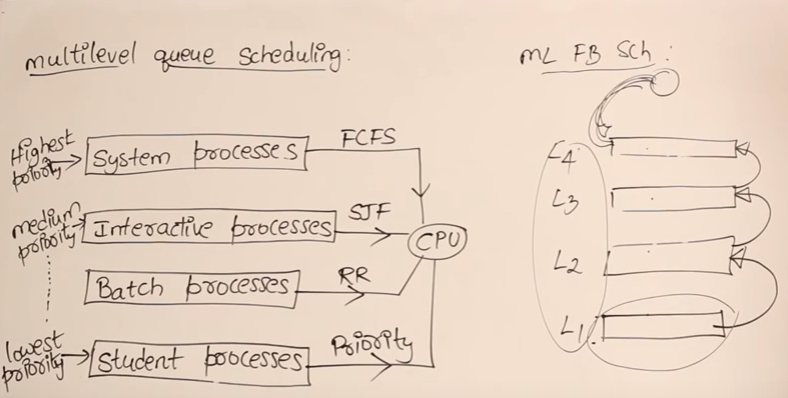


Fig: Multilevel Queue Scheduling



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.

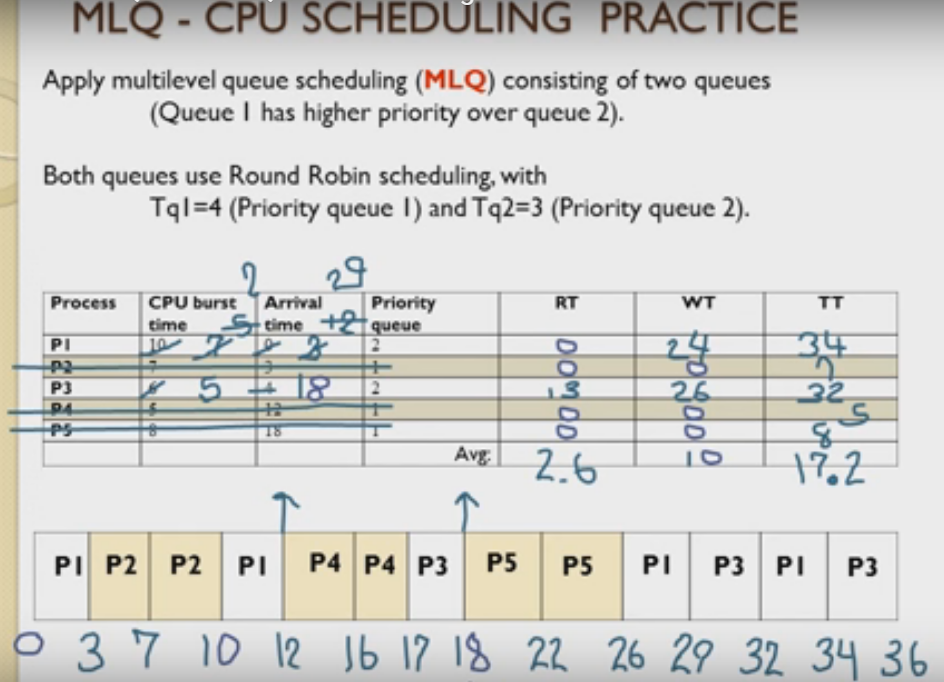
Batch processes: submit lots of jobs

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.



Example:

