SATHYABAMA
INSTITUTE OF SCIENCE AND TECHNOLOGY
(DEEMED TO BE UNIVERSITY)
Accredited "A" Grade by NAAC | 12B Status by UGC | Approved by AICTE
www.sathyabama.ac.in

SCS1501 - OPERATING SYSTEM

UNIT – 2
PROCESS MANAGEMENT

Mrs. C.Kavitha
ASSISTANT PROFESSOR
DEPARTMENT OF CSE
SCHOOL OF COMPUTING

SYLLUBUS

		L	T	P	Credits	Total Marks
SCSA1501	OPERATING SYSTEMS	3	0	0	3	100

UNIT 2 PROCESS MANAGEMENT 9 Hrs.

Processes - Process concepts - Process scheduling - Operations on processes - Cooperating processes - CPU scheduling - Basic concepts - Scheduling criteria - Scheduling algorithms - Preemptive strategies - Non-preemptive strategies.

COURSE OUTCOMES

SCS1501.1	Understand the fundamental components of a computer operating system and how computing resources are managed by the operating system
SCS1501.2	Apply the concepts of CPU scheduling, synchronization and deadlocks in real computing problems
SCS1501.3	Demonstrate the different memory and I/O management techniques used in Operating systems.
SCS1501.4	Have practical exposure to the concepts of semaphores and monitors for process synchronization
SCS1501.5	Create design and construct the following OS components: Schedulers, Memory management systems in the modern operating system.
SCS1501.6	Understand file system structure and implement a file system such as FAT

INTRODUCTION TO PROCESSES

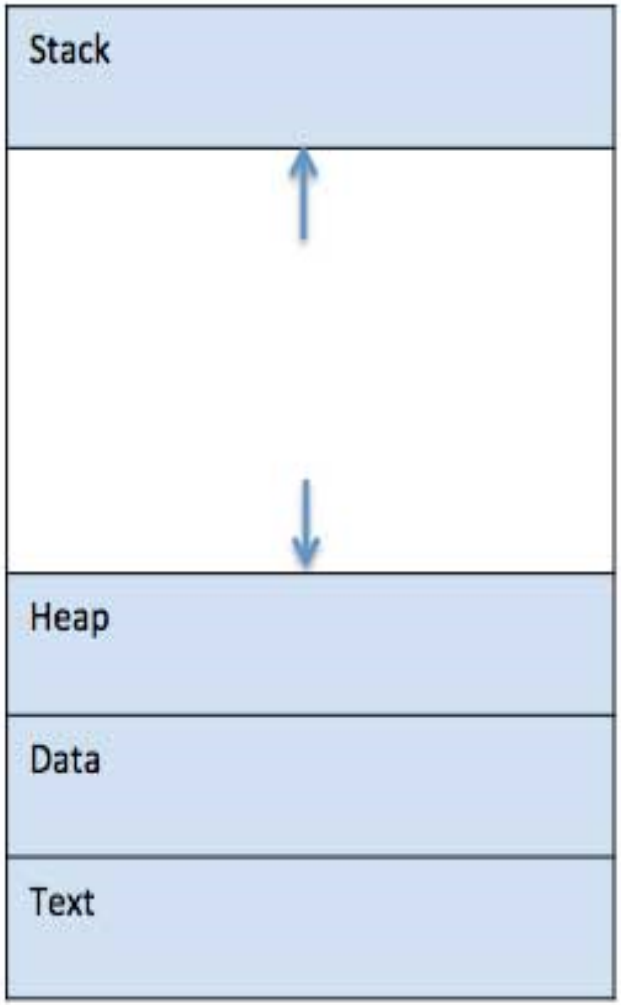
- Early computer systems allowed only one program to be executed at a time.
- **Definition :** A process is defined as an entity which represents the basic unit of work to be implemented in the system.
- A process is basically a program in execution. The execution of a process must progress in a sequential fashion.
- A process will need certain resources—such as CPU time, memory, files, and I/O devices —to accomplish its task.
- A process is the unit of work in most systems.

- Although traditionally a process contained only a single thread of control as it ran, most modern operating systems now support processes that have multiple threads.
- A process is the unit of work in a modern time-sharing system.
- The more complex the operating system is, the more it is expected to do on behalf of its users.
- Systems consist of a collection of processes:
  - Operating-system processes execute system code, and
  - User processes execute user code.

PROCESSES

- A process is mainly a program in execution where the execution of a process must progress in a sequential order or based on some priority or algorithms.
- In other words, it is an entity that represents the fundamental working that has been assigned to a system.
- When a program gets loaded into the memory, it is said to as process. This processing can be categorized into 4 sections. These are:
  - Heap
  - Stack
  - Data
  - Text

PROCESSES



PROCESSES

Process memory is divided into four sections for efficient working :

- **Text section** is made up of the compiled program code, read in from non-volatile storage when the program is launched.
- **Data section** is made up the global and static variables, allocated and initialized prior to executing the main.
- **Heap** is used for the dynamic memory allocation, and is managed via calls to new, delete, malloc, free, etc.
- **Stack** is used for local variables. Space on the stack is reserved for local variables when they are declared.

What is a Process?

- A process is a program in execution.
- Process is not as same as program code but a lot more than it.
- A process is an 'active' entity as opposed to program which is considered to be a 'passive' entity.
- Attributes held by process include hardware state, memory, CPU etc.

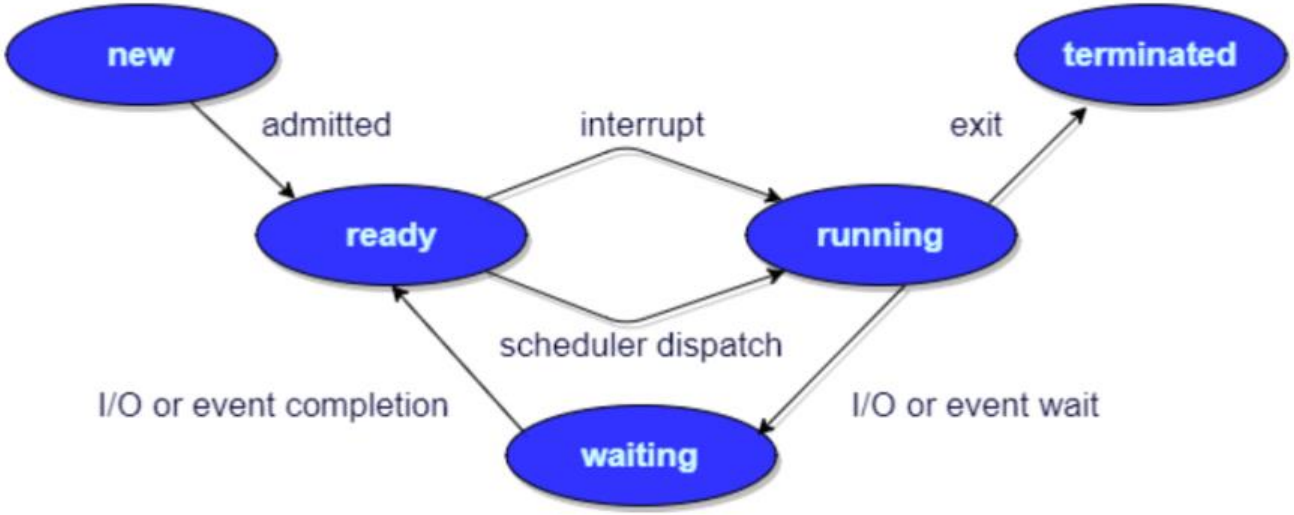
PROCESS CONCEPTS

- A question that arises in discussing operating systems involves what to call all the CPU activities.
- A batch system executes jobs, whereas a time-shared system has user programs, or tasks.
- Even on a single-user system such as Microsoft Windows, a user may be able to run several programs at one time: a word processor, a web browser, and an e-mail package.

PROCESS CONCEPTS

- **Process** – a program in execution; process execution must progress in sequential fashion.
- A process is a program in execution. A process is more than the program code, which is sometimes known as the text section.
- A process includes:
  - Counter
  - Program stack
  - Data section
    - The current activity, as represented by the value of the program counter and the contents of the processor's registers.
    - The process stack contains temporary data (such as function parameters, return addresses, and local variables)
    - A data section, which contains global variables.
- Process may also include a heap, which is memory that is dynamically allocated during process run time.

PROCESS STATE / PROCESS LIFE CYCLE



PROCESS STATE

- When a process executes, it passes through different states.
- These stages may differ in different operating systems, and the names of these states are also not standardized.
- In general, a process can have one of the following five states at a time.
  - New/Start
  - Running
  - Waiting
  - Ready
  - Terminated

PROCESS STATE

- **START / NEW**
  - This is the initial state when a process is first started / created.
- **READY**
  - The process is waiting to be assigned to a processor.
  - Ready processes are waiting to have the processor allocated to them by the operating system so that they can run.
  - Process may come into this state after Start state or while running it by but interrupted by the scheduler to assign CPU to some other process.
- **RUNNING**
  - Once the process has been assigned to a processor by the OS scheduler, the process state is set to running and the processor executes its instructions.

PROCESS STATE

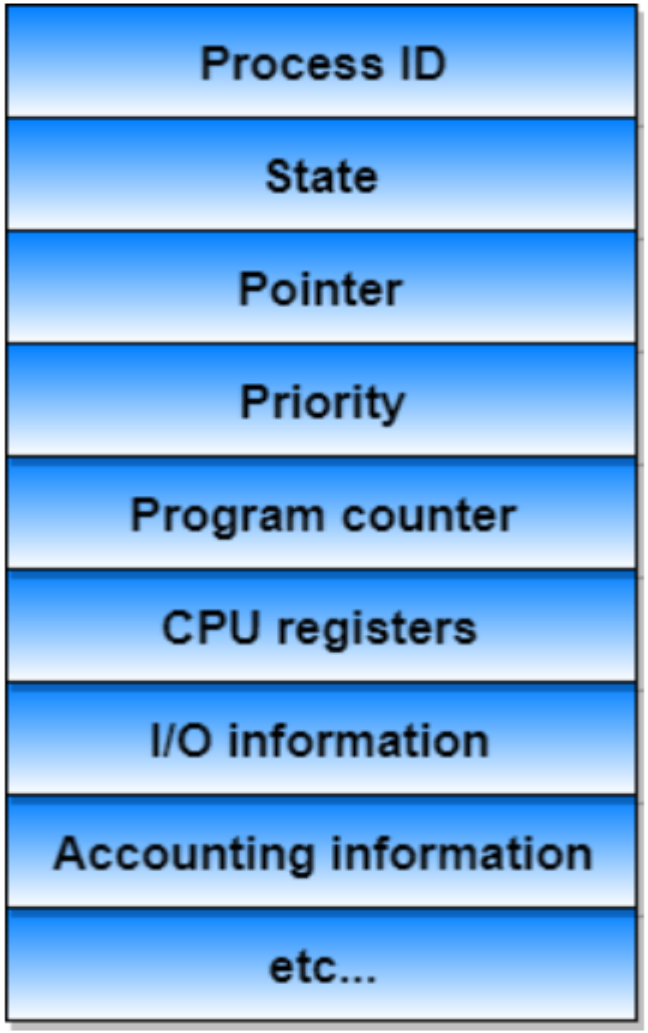
- **WAITING**
  - Process moves into the waiting state if it needs to wait for a resource, such as waiting for user input, or waiting for a file to become available.
- **TERMINATED OR EXIT**
  - Once the process finishes its execution, or it is terminated by the operating system, it is moved to the terminated state where it waits to be removed from main memory.

These names are arbitrary, and they vary across operating systems.

PROCESS STATE

- **NEW-** The process is being created.
- **READY-** The process is waiting to be assigned to a processor.
- **RUNNING-** Instructions are being executed.
- **WAITING-** The process is waiting for some event to occur(such as an I/O completion or reception of a signal).
- **TERMINATED-** The process has finished execution.

PROCESS CONTROL BLOCK (PCB)



PCB

- There is a Process Control Block for each process, enclosing all the information about the process. It is a data structure, which contains the following:
- **Process State**
  - It can be running, waiting etc.
- **Process Privileges**
  - This is required to allow/disallow access to system resources.
- **Process ID**
  - Unique identification for each of the process in the operating system**.**

PCB

- **Pointer**
  - A pointer to parent process.
- **Program Counter**
  - Program Counter is a pointer to the address of the next instruction to be executed for this process**.**
- **CPU Registers**
  - Various CPU registers where process need to be stored for execution for running state.
- **CPU Scheduling Information**
  - Process priority and other scheduling information which is required to schedule the process.

PCB

- **Memory Management Information**
  - This includes the information of page table, memory limits, Segment table depending on memory used by the operating system.
- **Accounting Information**
  - This includes the amount of CPU used for process execution, time limits, execution ID etc.
- **IO Status Information**
  - This includes a list of I/O devices allocated to the process.

PCB

- The architecture of a PCB is completely dependent on Operating System and may contain different information in different operating systems. (shown in the diagram)
- The PCB is maintained for a process throughout its lifetime, and is deleted once the process terminates.

PROCESS SCHEDULING

- The act of determining which process is in the **ready** state, and should be moved to the **running** state is known as **Process Scheduling**.
- **AIM :** The process scheduling system is to keep the CPU busy all the time and to deliver minimum response time for all programs.
- For achieving this, the scheduler must apply appropriate rules for swapping processes **IN** and **OUT** of CPU

PROCESS SCHEDULING

- Scheduling fell into one of the two general categories:
- **Non Pre-emptive Scheduling:** When the currently executing process gives up the CPU voluntarily.
- **Pre-emptive Scheduling:** When the operating system decides to favour another process, pre-empting the currently executing process.

What are Scheduling Queues?

- All processes, upon entering into the system, are stored in the **Job Queue**.
- Processes in the Ready state are placed in the **Ready Queue**.
- Processes waiting for a device to become available are placed in **Device Queues**. There are unique device queues available for each I/O device.
- The OS maintains all PCBs in Process Scheduling Queues.
- The OS maintains a separate queue for each of the process states and PCBs of all processes in the same execution state are placed in the same queue.

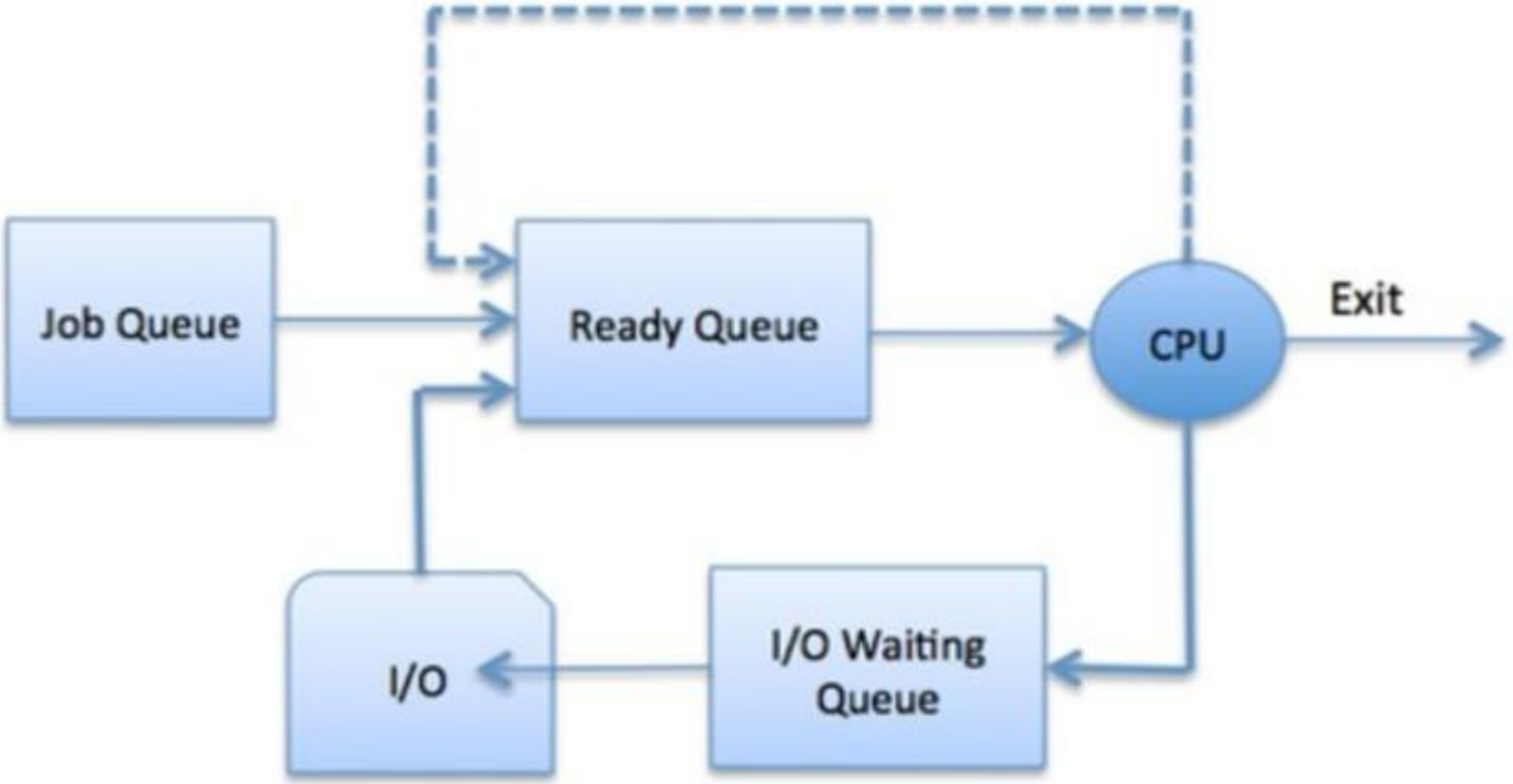
Scheduling Queues?

- Once the process is assigned to the CPU and is executing, one of the following several events can occur:
  - The process could issue an I/O request, and then be placed in the I/O queue.
  - The process could create a new sub-process and wait for its termination.
  - The process could be removed forcibly from the CPU, as a result of an interrupt, and be put back in the ready queue.

Scheduling Queues?

- Process migration between the various queues:
  - Job queue
  - Ready queue
  - Device queues
- **JOB QUEUE –** This queue keeps all the processes in the system.
- **READY QUEUE –** This queue keeps a set of all processes residing in main memory, ready and waiting to execute. A new process is always put in this queue.
- **DEVICE QUEUES –** The processes which are blocked due to unavailability of an I/O device constitute this queue.

Scheduling Queues?

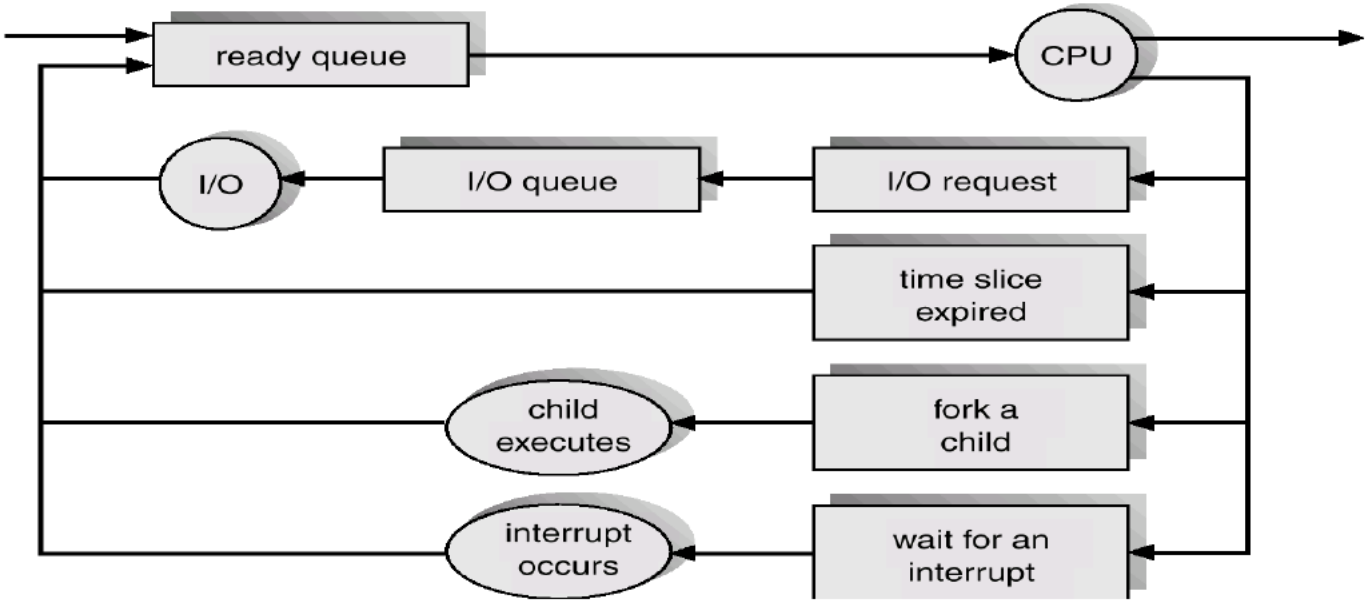


- The OS can use different policies to manage each queue **(FIFO, Round Robin, Priority, etc.)**

TWO STATE PROCESS MODEL

- Two-state process model refers to running and non-running states which are described below
- **RUNNING**
  - When a new process is created, it enters into the system as in the running state.
- **NOT RUNNING**
  - Processes that are not running are kept in queue, waiting for their turn to execute. Each entry in the queue is a pointer to a particular process. Queue is implemented by using linked list.

QUEUING DIAGRAM REPRESENTATION FOR PROCESS SCHEDULING



PROCESS SCHEDULING

- Each rectangular box represents a queue.
- Two types of queues are present:
  - the ready queue and
  - a set of device queues
- The circles represent the resources that serve the queues, and the arrows indicate the flow of processes in the system.
- A new process is initially put in the ready queue.
- It waits there until it is selected for execution, or is dispatched.

SCHEDULERS

- Schedulers are special system software which handles the process scheduling in various ways.
- Their main task is to select the jobs to be submitted into the system and to decide which process to run.
- Schedulers are of three types –
  - Long-Term Scheduler
  - Short-Term Scheduler
  - Medium-Term Scheduler

LONG TERM SCHEDULER

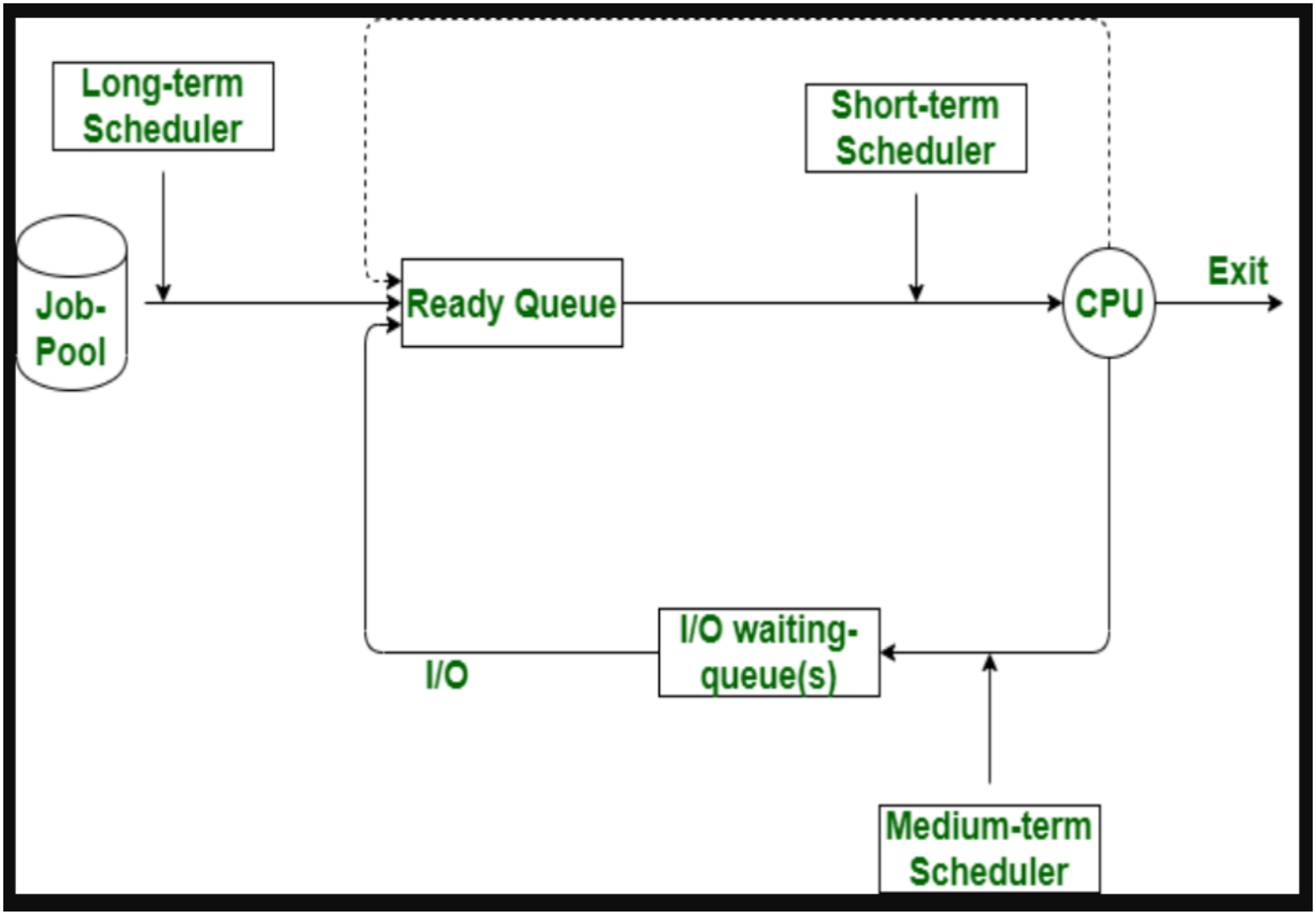
- It is also called a job scheduler.
- A long-term scheduler determines which programs are admitted to the system for processing.
- It selects processes from the queue and loads them into memory for execution.
- Process loads into the memory for CPU scheduling.

SHORT TERM SCHEDULER

- It is also called as CPU scheduler.
- Its main objective is to increase system performance in accordance with the chosen set of criteria.
- It is the change of ready state to running state of the process.
- 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.

MEDIUM TERM SCHEDULER

- Medium-term scheduling is a part of swapping.
- It removes the processes from the memory.
- The medium-term scheduler is in-charge of handling the swapped out-processes.
- A running process may become suspended if it makes an I/O request.
- A suspended process cannot make any progress towards completion.


Long-term Scheduler
Short-term Scheduler
Job-Pool
Ready Queue
CPU
Exit
I/O
I/O waiting-queue(s)
Medium-term Scheduler

S.N.	Long-Term Scheduler	Short-Term Scheduler	Medium-Term Scheduler
1	It is a job scheduler	It is a CPU scheduler	It is a process swapping scheduler.
2	Speed is lesser than short term scheduler	Speed is fastest among other two	Speed is in between both short and long term scheduler.
3	It controls the degree of multiprogramming	It provides lesser control over degree of multiprogramming	It reduces the degree of multiprogramming.
4	It is almost absent or minimal in time sharing system	It is also minimal in time sharing system	It is a part of Time sharing systems.
5	It selects processes from pool and loads them into memory for execution	It selects those processes which are ready to execute	It can re-introduce the process into memory and execution can be continued.

CONTEXT SWITCH

- Switching the CPU to another process requires **saving** the state of the old process and **loading** the saved state for the new process. This task is known as a **Context Switch**.
- The **context** of a process is represented in the **Process Control Block(PCB)** of a process; it includes the value of the CPU registers, the process state and memory-management information.
- Context switch time is **pure overhead**, because the **system does no useful work while switching**. Its speed varies from machine to machine, depending on the memory speed.

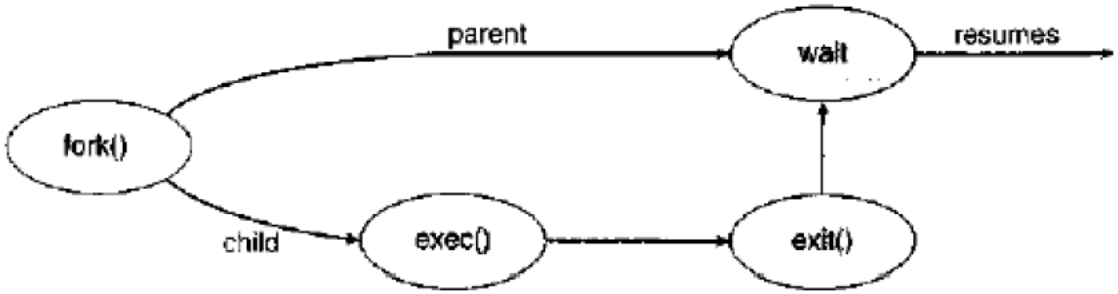
OPERATIONS ON PROCESSES

- The processes in the system can execute concurrently, and they must be created and deleted dynamically
- The two major operation **Process Creation** and **Process Termination**.

PROCESS CREATION

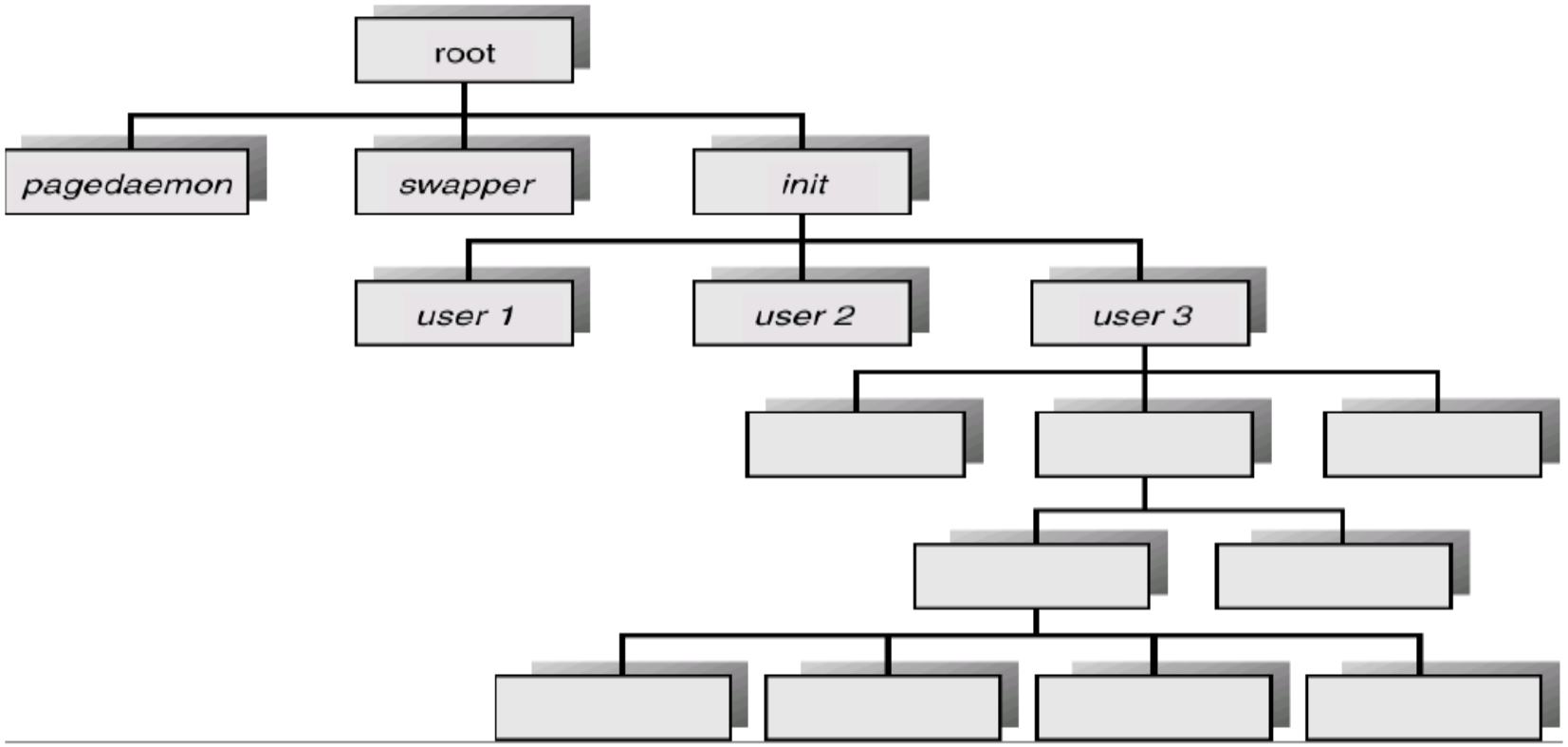
- Through appropriate system calls, such as fork or spawn, processes may create other processes.
- The process which creates other process, is termed the **parent** of the other process, while the created sub-process is termed its **child**.
- Parent process creates children processes, which, in turn create other processes, forming a tree of processes.
- **Resource sharing**
  - Parent and children share all resources.
  - Children share subset of parent's resources.
  - Parent and child share no resources.
- **Execution**
  - Parent and children execute concurrently.
  - Parent waits until children terminate.

PROCESS CREATION



Process creation

PROCESS CREATION



A Tree of Processes On A Typical UNIX System

PROCESS TERMINATION

- By making the **exit** (system call), typically returning an **int**, processes may request their own termination.
- This **int** is passed along to the parent if it is doing a **wait()**, and is typically zero on successful completion and some non-zero code in the event of any problem.
- Processes may also be terminated by the system for a variety of reasons, including :
  - The inability of the system to deliver the necessary system resources.
  - In response to a KILL command or other unhandled process interrupts.

COOPERATING PROCESSES

- Processes executing concurrently in the operating system may be either independent processes or cooperating processes.
- A process is independent if it cannot affect or be affected by the other processes executing in the system.
- Any process that does not share data with any other process is independent.
- There are several reasons for providing an environment that allows process cooperation:
  - **INFORMATION SHARING**
  - **COMPUTATION SPEEDUP**
  - **MODULARITY**
  - **CONVENIENCE**
  - **BUFFER**

COOPERATING PROCESSES

- **INFORMATION SHARING:**
  - Since several users may be interested in the same piece of information (for instance, a shared file), we must provide an environment to allow concurrent access to such information.
- **COMPUTATION SPEEDUP:**
  - If we want a particular task to run faster, we must break it into subtasks, each of which will be executing in parallel with the others.
- **MODULARITY:**
  - We may want to construct the system in a modular fashion, dividing the system functions into separate processes or threads
- **CONVENIENCE**
  - Even an individual user may work on many tasks at the same time

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)
- This is 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.
- The aim of CPU scheduling is to make the system efficient, fast and fair
- Whenever the CPU becomes idle, the operating system must select one of the processes in the **ready queue** to be executed.

CPU SCHEDULING: 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:
  - Switching context
  - Switching to user mode
  - Jumping to the proper location in the user program to restart that program from where it left last time.

CPU SCHEDULING: DISPATCHER

- The dispatcher should be as fast as possible, given that it is invoked during every process switch. The time taken by the dispatcher to stop one process and start another process is known as the **Dispatch Latency**.
- **Types of CPU Scheduling**
  - When a process switches from the **running** state to the **waiting** state (for I/O request or invocation of wait for the termination of one of the child processes).
  - When a process switches from the **running** state to the **ready** state (for example, when an interrupt occurs).
  - When a process switches from the **waiting** state to the **ready** state (for example, completion of I/O).
  - When a process **terminates**.

CPU SCHEDULING: SCHEDULING CRITERIA

- Different CPU scheduling algorithms have different properties, and the choice of a particular algorithm may favor one class of processes over another.
- Many criteria have been suggested for comparing CPU scheduling algorithms.
- Which characteristics are used for comparison can make a substantial difference in which algorithm is judged to be best.
  - **CPU UTILIZATION**
  - **THROUGHPUT**
  - **TURNAROUND TIME**
  - **WAITING TIME**
  - **LOAD AVERAGE**
  - **RESPONSE TIME**

CPU SCHEDULING: SCHEDULING CRITERIA

- **CPU UTILIZATION**
  - We want to keep the CPU as busy as possible.
  - Conceptually, CPU utilization can range from 0 to 100 percent.
- **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.
- **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).

CPU SCHEDULING: SCHEDULING CRITERIA

- **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).

PREEMPTIVE AND NON – PREEMPTIVE

Preemptive Scheduling	Non-Preemptive Scheduling
Processor can be preempted to execute a different process in the middle of execution of any current process.	Once Processor starts to execute a process it must finish it before executing the other. It cannot be paused in middle.
CPU utilization is more compared to Non-Preemptive Scheduling.	CPU utilization is less compared to Preemptive Scheduling.
Waiting time and Response time is less.	Waiting time and Response time is more.
The preemptive scheduling is prioritized. The highest priority process should always be the process that is currently utilized.	When a process enters the state of running, the state of that process is not deleted from the scheduler until it finishes its service time.
If a high priority process frequently arrives in the ready queue, low priority process may starve.	If a process with long burst time is running CPU, then another process with less CPU burst time may starve.
Preemptive scheduling is flexible.	Non-preemptive scheduling is rigid.
Ex:- SRTF, Priority, Round Robin, etc.	Ex;- FCFS, SJF, Priority, etc.

SCHEDULING ALGORITHMS

- CPU scheduling deals with the problem of deciding which of the processes in the ready queue is to be allocated the CPU.
- There are many different CPU scheduling algorithms:
  - **First Come First Serve (FCFS) Scheduling**
  - **Shortest-Job-First (SJF) Scheduling**
  - **Priority Scheduling**
  - **Shortest Remaining Time (SRT) Scheduling**
  - **Round Robin (RR) Scheduling**
  - **Multilevel Queue Scheduling**
  - **Multilevel Feedback Queue Scheduling**

FCFS - FIRST COME FIRST SERVE SCHEDULING

- In the "First come first serve" scheduling algorithm, as the name suggests, the process which arrives first, gets executed first
- We can say that the process which requests the CPU first, gets the CPU allocated first.
- Jobs are executed on first come, first serve basis.
- It is a non-preemptive, pre-emptive scheduling algorithm.
- Easy to understand and implement.
- Its implementation is based on FIFO queue.
- Poor in performance as average wait time is high.

SJF – SHORTEST JOB FIRST SCHEDULING

- This is also known as shortest job next, or SJN
- This is a non-preemptive, pre-emptive scheduling algorithm.
- Best approach to minimize waiting time.
- Easy to implement in Batch systems where required CPU time is known in advance.
- Impossible to implement in interactive systems where required CPU time is not known.
- The processer should know in advance how much time process will take.

PRIORITY SCHEDULING

- The SJF algorithm is a special case of the general priority scheduling algorithm.
- A priority is associated with each process, and the CPU is allocated to the process with the highest priority.
- Equal-priority processes are scheduled in FCFS order.
- An SJF algorithm is simply a priority algorithm where the priority (p) is the inverse of the (predicted) next CPU burst.
- The larger the CPU burst, the lower the priority, and vice versa.
- There is no general agreement on whether 0 is the highest or lowest priority.
- Some systems use low numbers to represent low priority; others use low numbers for high priority.
- Here, we assume that low numbers represent high priority.

ROUND ROBIN SCHEDULING

- Round Robin is a CPU scheduling algorithm where each process is assigned a fixed time slot in a cyclic way.
- It is pre-emptive as processes are assigned CPU only for a fixed slice of time at most.
- The CPU scheduler goes around the ready queue allocating the CPU to each process for a time interval up to some time quantum.
- The CPU scheduler picks the process from the ready queue, sets a timer to interrupt after that time quantum and dispatches the process

1. **Assume the following workload in a system**

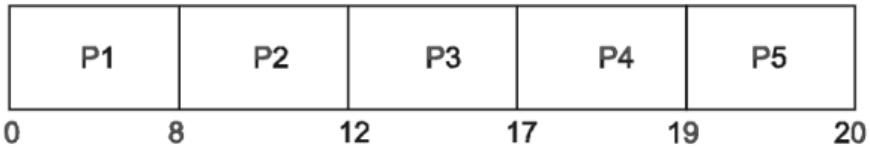
Process	Burst Time (BT)	Priority
P1	8	2
P2	4	1
P3	5	4
P4	2	2
P5	1	3

Draw the Gantt chart, Illustrate the execution of these jobs using FCFS, Round Robin (4 ms), Non-Preemptive Priority and Non Premptive SJF.

Process	Burst Time (BT)	Priority
P1	8	2
P2	4	1
P3	5	4
P4	2	2
P5	1	3

(i) FCFS

Gantt chart:



Waiting Time (WT) = [Completion Time (CT) – Burst Time (BT)] – Arrival Time (AT)

Turn Around Time (TAT) = Completion (CT) – Arrival Time (AT)

Waiting time and Turn around time:

$$AT = 0$$

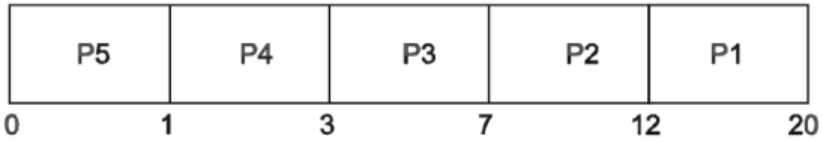
Process	CT	Waiting Time (WT)	Turn Around Time (TAT)
P1	8	0	8
P2	12	8	12
P3	17	12	17
P4	19	17	19
P5	20	9	20

Average WT = (0 + 8 + 12 + 17 + 9) / 5 = 12 ms

Average TAT = (8 + 12 + 17 + 19 + 20) / 5 = 15.2 ms

Process	Burst Time (BT)	Priority
P1	8	2
P2	4	1
P3	5	4
P4	2	2
P5	1	3

(ii) SJF



Waiting time and Turn around time:

$$AT = 0$$

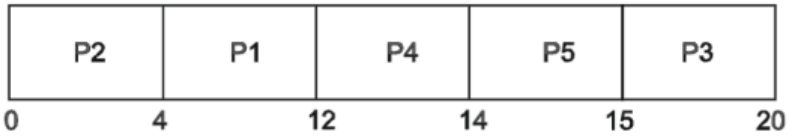
Process	CT	Waiting Time (WT) WT = (CT – BT) – AT	Turn Around Time (TAT) TAT = CT – AT
P1	20	12	20
P2	7	3	7
P3	12	7	12
P4	3	1	3
P5	1	6	1

Average WT = (12 + 3 + 7 + 1 + 6) / 5 = 4.6 ms

Average TAT = (20 + 7 + 12 + 3 + 1) / 5 = 8.6 ms

Process	Burst Time (BT)	Priority
P1	8	2
P2	4	1
P3	5	4
P4	2	2
P5	1	3

(iii) Priority (Non- Pre-emptive)



Waiting time and Turn around time:

$$AT = 0$$

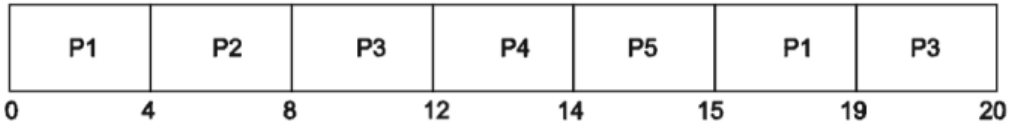
Process	CT	Waiting Time (WT) WT = (CT – BT) – AT	Turn Around Time (TAT) TAT = CT – AT
P1	12	4	12
P2	14	0	14
P3	20	15	20
P4	14	12	14
P5	15	14	15

Average WT = (4 + 0 + 15 + 12 + 14) / 5 = 9 ms

Average TAT = (12 + 14 + 20 + 14 + 15) / 5 = 13 ms

Process	Burst Time (BT)	Priority
P1	8	2
P2	4	1
P3	5	4
P4	2	2
P5	1	3

(iv) Round Robin



Waiting time and Turn around time:

$$AT = 0$$

Process	CT	Waiting Time (WT) WT = (CT – BT) – AT	Turn Around Time (TAT) TAT = CT – AT
P1	19	11	19
P2	8	4	8
P3	20	15	20
P4	14	12	14
P5	15	14	15

Average WT = (11 + 4 + 15 + 12 + 14) / 5 = 11.2 ms

Average TAT = (19 + 8 + 20 + 14 + 15) / 5 = 15.2 ms

2. **Assume the following process arrive for execution at the time indicated and also mention with length of the CPU burst time given:**

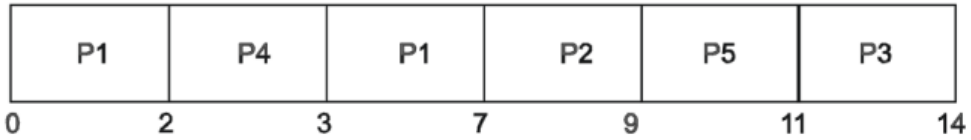
Process	Burst Time (BT)	Priority	Arrival Time (AT)
P1	6	2	0
P2	2	2	1
P3	3	4	1
P4	1	1	2
P5	2	3	2

Draw Gantt chart and find the waiting time and turn around time for Pre-emptive and Non-preemptive priority scheduling.

Process	Burst Time (BT)	Priority	Arrival Time (AT)
P1	6	2	0
P2	2	2	1
P3	3	4	1
P4	1	1	2
P5	2	3	2

(i) Priority (Preemptive):

Gantt chart:



Waiting time and Turn around time:

Process	Waiting Time (WT) WT = (CT – BT) – AT	Turn Around Time (TAT) TAT = CT – AT
P1	1 – 0 = 1	7 – 0 = 7
P2	7 – 1 = 6	9 – 1 = 8
P3	11 – 1 = 10	14 – 1 = 13
P4	2 – 2 = 0	3 – 2 = 1
P5	9 – 2 = 7	11 – 2 = 9

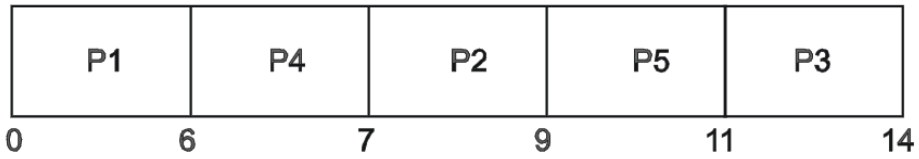
Average WT = (1 + 6 + 10 + 0 + 7) / 5 = 4.8 ms

Average TAT = (7 + 8 + 13 + 13 + 9) / 5 = 7.6 ms

Process	Burst Time (BT)	Priority	Arrival Time (AT)
P1	6	2	0
P2	2	2	1
P3	3	4	1
P4	1	1	2
P5	2	3	2

(ii) Priority (Non-Preemptive):

Gantt chart:



Waiting time and Turn around time:

Process	Waiting Time (WT) WT = (CT – BT) – AT	Turn Around Time (TAT) TAT = CT – AT
P1	0 – 0 = 0	6 – 0 = 6
P2	7 – 1 = 6	9 – 1 = 8
P3	11 – 1 = 0	14 – 1 = 13
P4	6 – 2 = 4	7 – 2 = 5
P5	9 – 2 = 7	11 – 2 = 9

3. **Assume the following process arrive for execution at the time indicated and also mention with length of the CPU burst time given:**

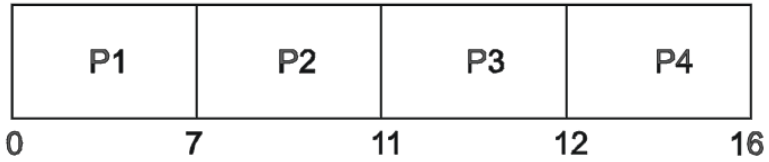
Process	Burst Time (BT)	Arrival Time (AT)
P1	7	0.0
P2	4	2.0
P3	1	4.0
P4	4	5.0

Draw Gantt chart and find the waiting time and turn around time for FCFS, RR, Pre-emptive and Non-preemptive SJF scheduling.

Process	Burst Time (BT)	Arrival Time (AT)
P1	7	0.0
P2	4	2.0
P3	1	4.0
P4	4	5.0

(i) FCFS

Gantt chart



Waiting time and Turn around time

Process	CT	Waiting Time (WT) WT = (CT – BT) – AT	Turn Around Time (TAT) TAT = CT – AT
P1	7	0 – 0 = 0	7 – 0 = 7
P2	11	7 – 2 = 5	11 – 2 = 9
P3	12	11 – 4 = 7	12 – 4 = 8
P4	16	12 – 5 = 7	16 – 5 = 11

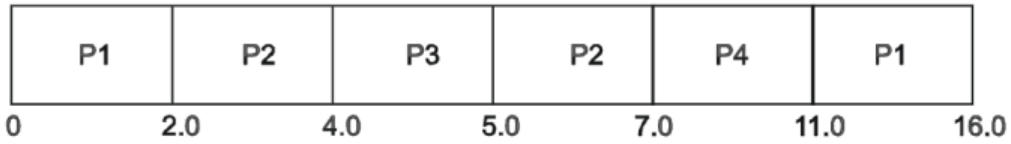
Average WT = (0 + 5 + 7 + 7)/4 = 4.75 ms

Average TAT = (7 + 9 + 8 + 11)/4 = 8.75 ms

Process	Burst Time (BT)	Arrival Time (AT)
P1	7	0.0
P2	4	2.0
P3	1	4.0
P4	4	5.0

(ii) SJF (Preemptive):

Gantt chart:



Waiting time and Turn around time:

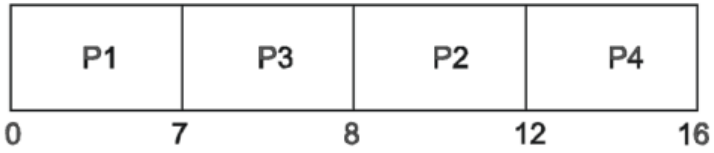
Process	CT	Waiting Time (WT) WT = (CT – BT) – AT	Turn Around Time (TAT) TAT = CT – AT
P1	16	(0 – 2) + 11 – 0 = 9	16 – 0 = 16
P2	7	5 – 4 = 1	7 – 2 = 5
P3	5	4 – 4 = 0	5 – 4 = 1
P4	11	7 – 5 = 2	11 – 5 = 6

Average WT = (9 + 1 + 0 + 2) / 4 = 3 ms

Average TAT = (16 + 5 + 1 + 6) / 4 = 7 ms

Process	Burst Time (BT)	Arrival Time (AT)
P1	7	0.0
P2	4	2.0
P3	1	4.0
P4	4	5.0

(iii) SJF (Non-Preemptive):



Gantt chart:

Waiting time and Turn around time:

Process	CT	Waiting Time (WT) WT = (CT – BT) – AT	Turn Around Time (TAT) TAT = CT – AT
P1	7	0 – 0 = 0	7 – 0 = 7
P2	12	8 – 2 = 6	12 – 2 = 10
P3	8	7 – 4 = 3	8 – 4 = 4
P4	16	12 – 5 = 7	16 – 5 = 11

Average WT = (0 + 6 + 3 + 7) / 4 = 4 ms

Average TAT = (7 + 10 + 4 + 11) / 4 = 8 ms

Round Robin

Process	AT	BT
P1	0	5
P2	1	3
P3	3	6
P4	5	1
P5	6	4

Request queue:

TQ = 3

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

TQ = 3

Request queue:

~~P1~~ ~~P3~~

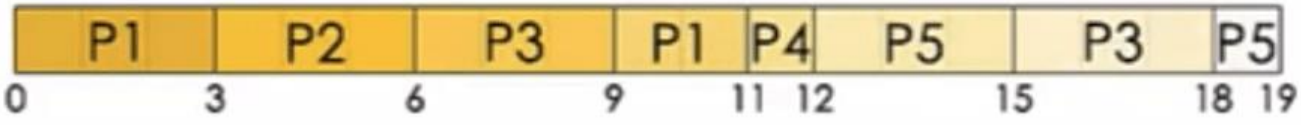
~~P2~~ ~~P5~~

~~P3~~

~~P1~~

~~P4~~

~~P5~~



Process	AT	BT
P1	0	5
P2	1	3
P3	3	6
P4	5	1
P5	6	4

TQ = 3

Request queue:

(a) (i) Consider the following set of processes with the length of CPU-burst time given in milliseconds.

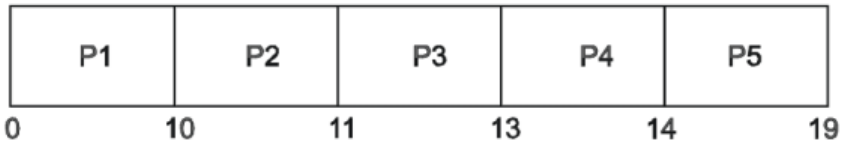
Process	Burst time	Priority	Arrival time
P_1	10	3	0
P_2	1	1	1
P_3	2	3	2
P_4	1	4	1
P_5	5	2	2

Draw the Gantt chart for the execution of these processes using FCFS, SJF, SRTS, pre-emptive and non-pre-emptive priority, round robin with time slice of 2 ms. Find the average waiting and turnaround time using each of the methods.

Process	Burst time	Priority	Arrival time
P_1	10	3	0
P_2	1	1	1
P_3	2	3	2
P_4	1	4	1
P_5	5	2	2

FCFS

Gantt chart



Turn around time (TAT) = Completion time (CT) – Arrival Time (AT)

Process	CT	WT	TAT
P_1	10	0 – 0 = 0	10 – 0 = 10
P_2	11	10 – 1 = 9	11 – 1 = 10
P_3	13	11 – 2 = 9	13 – 2 = 11
P_4	14	13 – 1 = 12	14 – 1 = 13
P_5	19	14 – 2 = 12	19 – 2 = 17

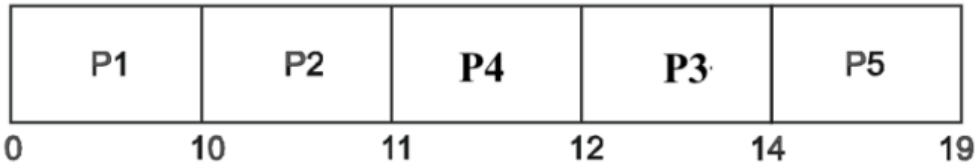
Average WT = (0 + 9 + 9 + 12 + 12)/5 = 8.4 ms

Average TAT = (10 + 10 + 11 + 13 + 17/5 = 12.2 ms.

Process	Burst time	Priority	Arrival time
P_1	10	3	0
P_2	1	1	1
P_3	2	3	2
P_4	1	4	1
P_5	5	2	2

SJF (Non-Premption)

Gantt Chart



Process	CT	WT	TAT
P_1	10	$0 - 0 = 0$	$10 - 0 = 10$
P_2	11	$10 - 1 = 9$	$11 - 1 = 10$
P_3	14	$12 - 2 = 10$	$14 - 2 = 12$
P_4	12	$11 - 1 = 10$	$12 - 1 = 11$
P_5	19	$14 - 2 = 12$	$19 - 2 = 17$

Average WT $= (0 + 9 + 10 + 10 + 12)/5 = 8.2$ ms

Process	Burst time	Priority	Arrival time
P_1	10	3	0
P_2	1	1	1
P_3	2	3	2
P_4	1	4	1
P_5	5	2	2

SJF (Pre-emption)

Gantt Chart

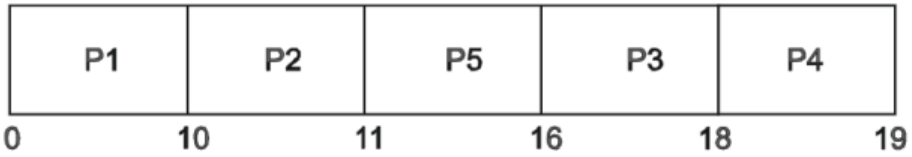
Process	CT	WT	TAT
P_1	10	0 – 0 = 0	10 – 0 = 10
P_2	11	10 – 1 = 9	11 – 1 = 10
P_3	14	12 – 2 = 10	14 – 2 = 12
P_4	12	11 – 1 = 10	12 – 1 = 11
P_5	19	14 – 2 = 12	19 – 2 = 17

Average WT $= (9 + 0 + 1 + 1 + 3)/5 = 2.8$ ms

Average TAT $= (19 + 1 + 3 + 2 + 8)/5 = 6.6$ ms

Process	Burst time	Priority	Arrival time
P_1	10	3	0
P_2	1	1	1
P_3	2	3	2
P_4	1	4	1
P_5	5	2	2

Priority (Non-Premption)



Process	CT	WT	TAT
P_1	10	$0 - 0 = 0$	$10 - 0 = 10$
P_2	11	$10 - 1 = 9$	$11 - 1 = 10$
P_3	18	$16 - 2 = 14$	$18 - 2 = 16$
P_4	19	$18 - 1 = 17$	$19 - 1 = 18$
P_5	16	$11 - 2 = 9$	$16 - 2 = 14$

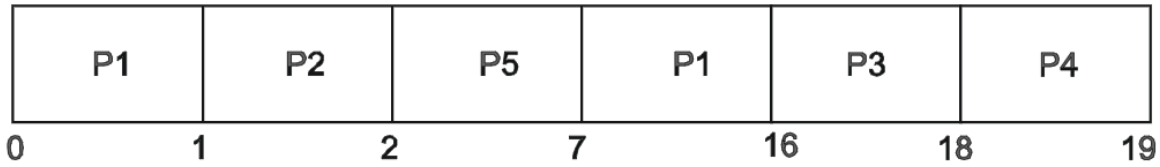
Average WT $= (0 + 9 + 14 + 17 + 9)/5 = 9.2$ ms

Average TAT $= (10 + 10 + 16 + 18 + 14)/5 = 13.6$ ms

Process	Burst time	Priority	Arrival time
P_1	10	3	0
P_2	1	1	1
P_3	2	3	2
P_4	1	4	1
P_5	5	2	2

Priority (Pre-emption)

Gantt Chart



Process	CT	WT	TAT
P_1	16	$6 - 0 = 0$	$16 - 0 = 16$
P_2	2	$1 - 1 = 0$	$2 - 1 = 1$
P_3	18	$16 - 2 = 14$	$18 - 2 = 16$
P_4	19	$18 - 1 = 17$	$19 - 1 = 18$
P_5	7	$2 - 2 = 0$	$7 - 2 = 5$

Average WT $= (6 + 0 + 14 + 17 + 0)/5 = 7.4$ ms

Average TAT $= (16 + 1 + 16 + 18 + 5)/5 = 11.2$ ms