OPERATING SYSTEM

UNIT 2

PROCESS MANAGEMENT

INTRODUCTION TO PROCESSES:

- Early computer systems allowed only one program to be executed at a time.
- This program had complete control of the system and had access to all the system's resources.
- In contrast, current-day computer systems allow multiple programs to be loaded into memory and executed concurrently.
- This evolution required firmer control and more compartmentalization of the various programs; and these needs resulted in the notion of a process, which is a program in execution.
- 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.
- These resources are allocated to the process either when it is created or while it is executing.
- **Definition :** A process is defined as an entity which represents the basic unit of work to be implemented in the system.
- A process is the unit of work in most systems.
- Systems consist of a collection of processes:
  - Operating-system processes execute system code, and
  - User processes execute user code.
- 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.
- Although its main concern is the execution of user programs, it also needs to take care of various system tasks that are better left outside the kernel itself.
- A system therefore consists of a collection of processes: operating system processes executing system code and user processes executing user code.

PROCESSES:

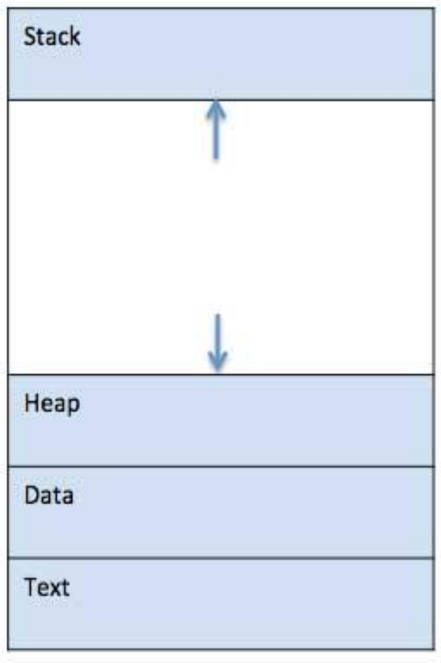
- A process is mainly a program in execution where the execution of a process must progress in a sequential order or based on some priority or algorithms.
- In other words, it is an entity that represents the fundamental working that has been assigned to a system.
- When a program gets loaded into the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

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.
- Even if the user can execute only one program at a time, the operating system may need to support its own internal programmed activities, such as memory management.
- In many respects, all these activities are similar, so we call all of them processes.
- The terms job and process are used almost interchangeably used.
- Although we personally prefer the term process, much of operating-system theory and terminology was developed during a time when the major activity of operating systems was job processing.
- It would be misleading to avoid the use of commonly accepted terms that include the word job (such as job scheduling) simply because process has superseded job.

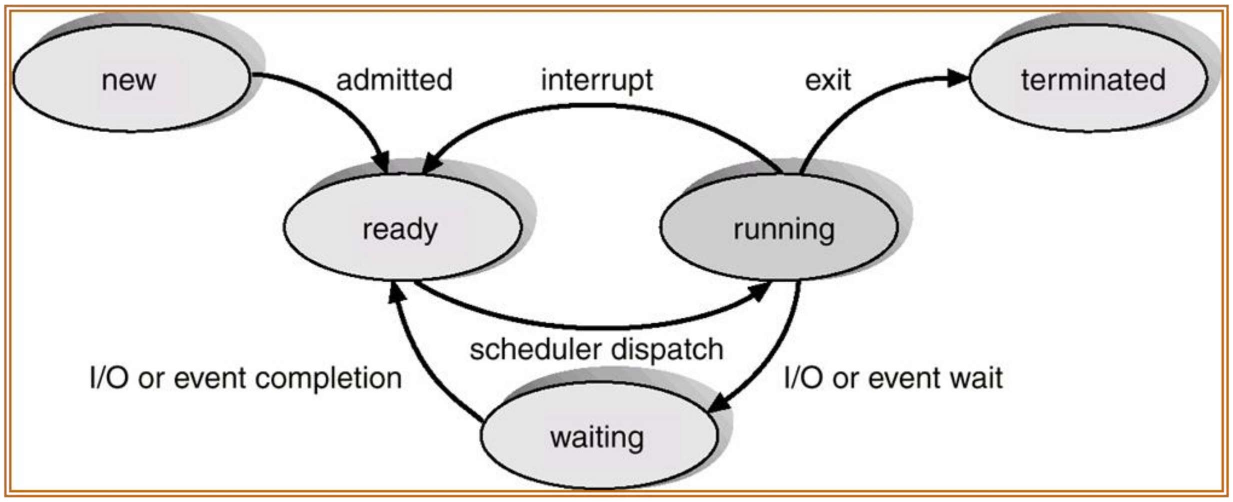
THE PROCESS:

- An operating system executes a variety of programs:
  - Batch system – jobs
  - Time-shared systems – user programs or tasks
- We use the terms job and process almost interchangeably.
- 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.



- **STACK -** The process Stack contains the temporary data such as method/function parameters, return address and local variables.
- **HEAP -** This is dynamically allocated memory to a process during its run time.
- **TEXT -** This includes the current activity represented by the value of Program Counter and the contents of the processor's registers**.**
- **DATA -** This section contains the global and static variables.
- We emphasize that a program by itself is not a process; a program is a passive entity, such as a file containing a list of instructions stored on disk (often called an executable file).
- Whereas a process is an active entity, with a program counter specifying the next instruction to execute and a set of associated resources.
- A program becomes a process when an executable file is loaded into memory.

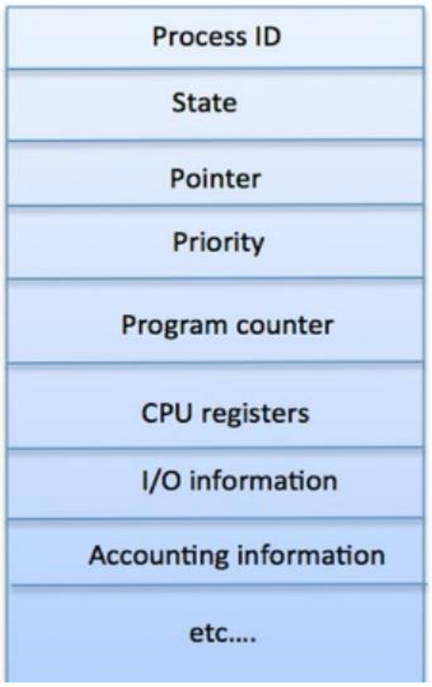
PROCESS STATE / PROCESS LIFE CYCLE:



- 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

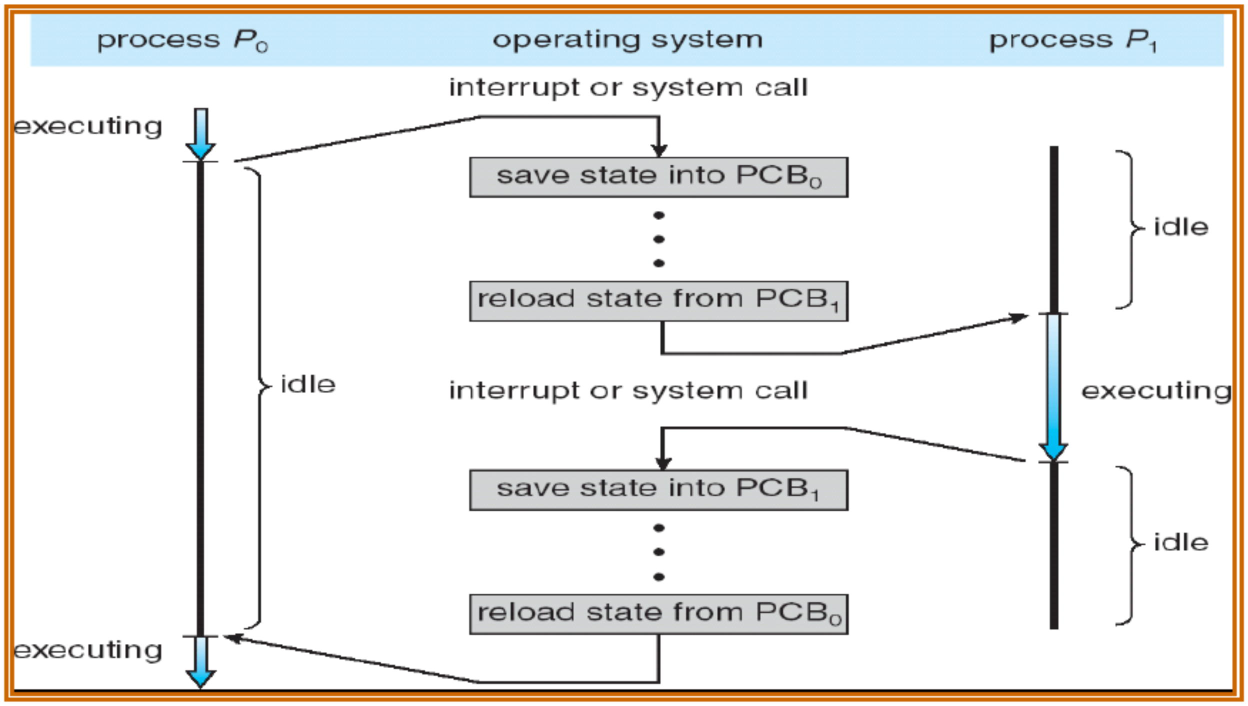
- **START / NEW –**
    - This is the initial state when a process is first started / created.
- **READY –**
    - The process is waiting to be assigned to a processor.
    - Ready processes are waiting to have the processor allocated to them by the operating system so that they can run.
    - Process 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.
- **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 CONTROL BLOCK (PCB):



- A Process Control Block is a data structure maintained by the Operating System for every process.
- The PCB is identified by an integer process ID (PID).
- A PCB keeps all the information needed to keep track of a process as listed below ,
- **PROCESS STATE**
  - The current state of the process i.e., whether it is ready, running, waiting, or whatever.
- **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**.**
- **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.
- **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.
- The architecture of a PCB is completely dependent on Operating System and may contain different information in different operating systems. (shown in above diagram)
- The PCB is maintained for a process throughout its lifetime, and is deleted once the process terminates.

DIAGRAM SHOWING CPU SWITCH PROCESS TO PROCESS



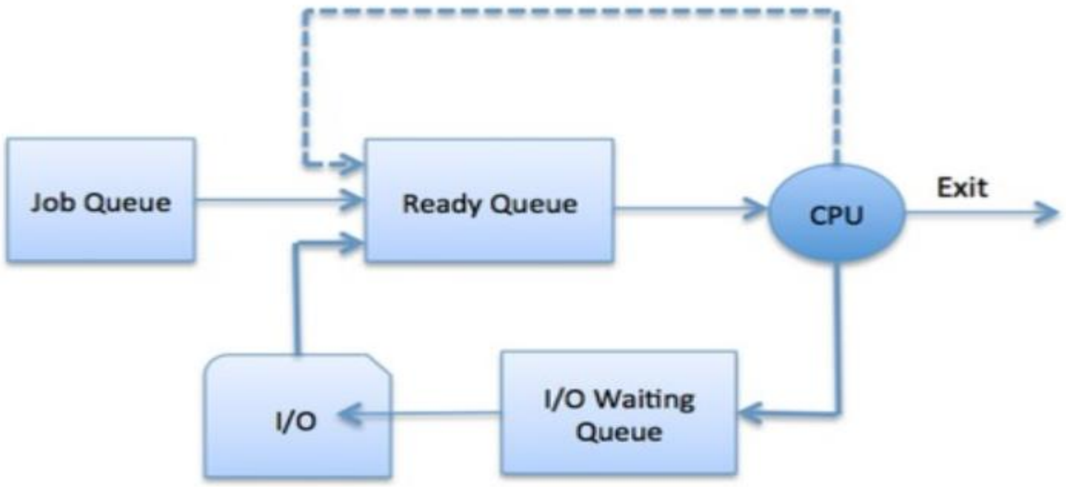
PROCESS SCHEDULING:

- A uniprocessor system can have only one running process. If more process exist, the rest 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.
- The objective of time sharing is to switch the CPU among processes so frequently that users can interact with each program while it is running.
- To meet these objectives, the process scheduler selects an available process (possibly from a set of several available processes) for program execution on the CPU.

PROCESS SCHEDULING QUEUES:

- 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.
- When the state of a process is changed, its PCB is unlinked from its current queue and moved to its new state queue.

- Process migration between the various queues:
    - Job queue
    - Ready queue
    - Device queues



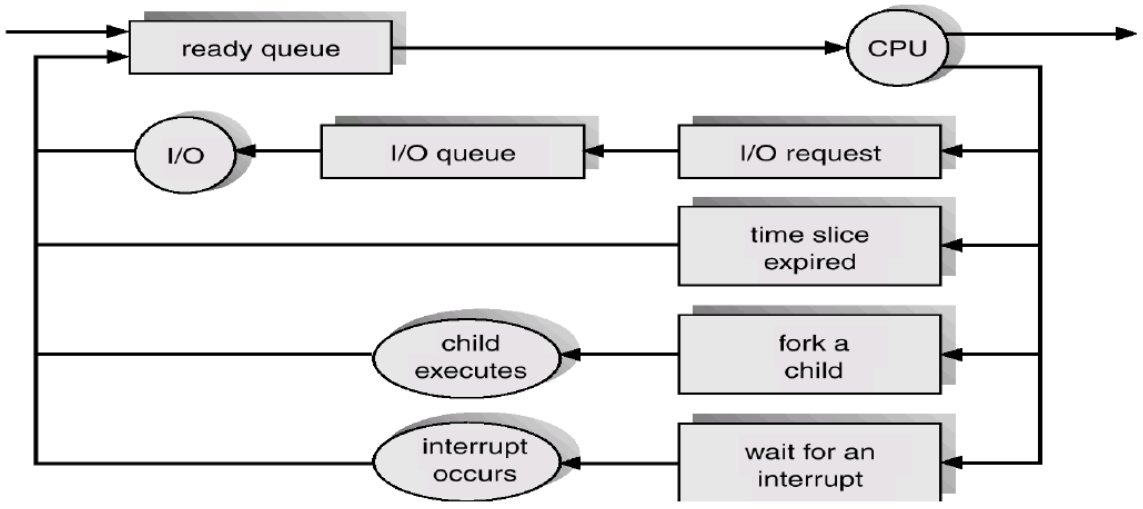
- **JOB QUEUE –** This queue keeps all the processes in the system.
- **READY QUEUE –** This queue keeps a set of all processes residing in main memory, ready and waiting to execute. A new process is always put in this queue.
- **DEVICE QUEUES –** The processes which are blocked due to unavailability of an I/O device constitute this queue.
- The OS can use different policies to manage each queue (FIFO, Round Robin, Priority, etc.).
- The OS scheduler determines how to move processes between the ready and run queues which can only have one entry per processor core on the system; in the above diagram, it has been merged with the CPU.

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



- 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.
- Once the process is allocated the CPU and is executing, one of several events could occur:
    - The process could issue an I/O request and then be placed in an I/O queue.
    - The process could create a new sub process and wait for the sub process's termination.
    - The process could be removed forcibly from the CPU, as a result of an
    - Interrupt, and be put back in the ready queue.
- In the first two cases, the process eventually switches from the waiting state to the ready state and is then put back in the ready queue.

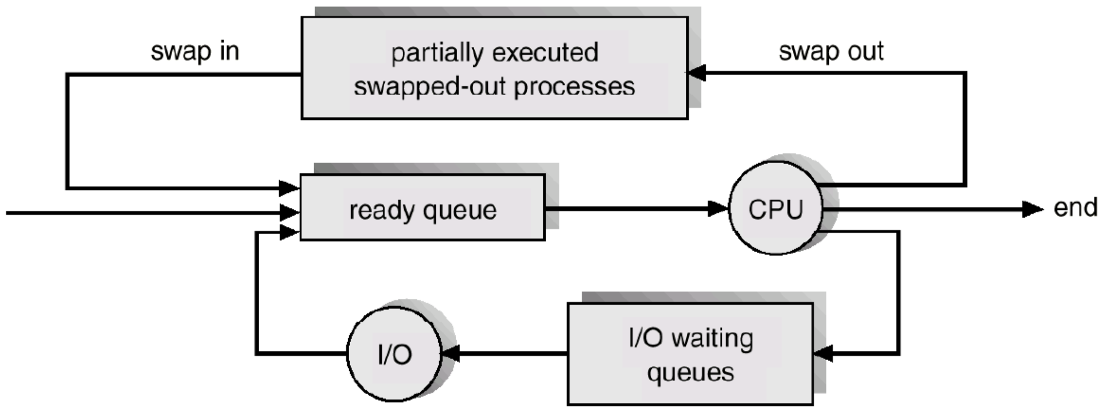
- A process continues this cycle until it terminates, at which time it is removed from all queues and has its PCB and resources de-allocated.

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

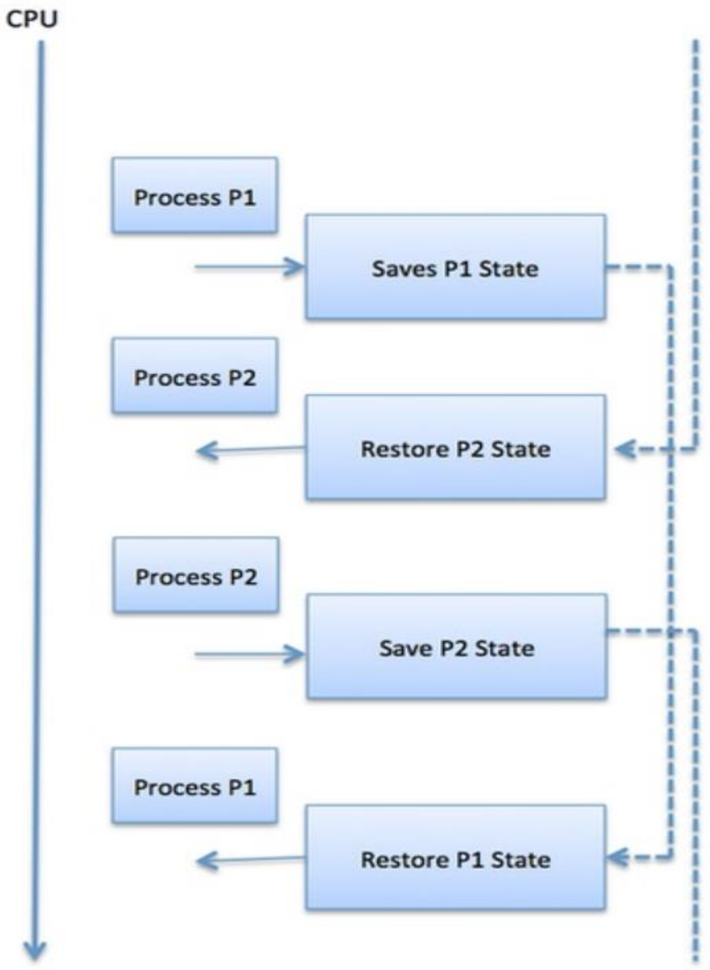
S.N.	Long-Term Scheduler	Short-Term Scheduler	Medium-Term Scheduler
1	It is a job scheduler	It is a CPU scheduler	It is a process swapping scheduler.
2	Speed is lesser than short term scheduler	Speed is fastest among other two	Speed is in between both short and long term scheduler.
3	It controls the 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.

- **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.
    - The primary objective of the job scheduler is to provide a balanced mix of jobs, such as I/O bound and processor bound.
    - It also controls the degree of multiprogramming. If the degree of multiprogramming is stable, then the average rate of process creation must be equal to the average departure rate of processes leaving the system.
- **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.
    - Short-term schedulers are faster than long-term schedulers.
- **MEDIUM TERM SCHEDULER:**
    - Medium-term scheduling is a part of swapping.
    - It removes the processes from the memory.
    - It reduces the degree of multiprogramming. 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.
    - In this condition, to remove the process from memory and make space for other processes, the suspended process is moved to the secondary storage. This process is called swapping, and the process is said to be swapped out or rolled out. Swapping may be necessary to improve the process mix.



Addition of Medium Term Scheduling

CONTEXT SWITCH:



- A context switch is the mechanism to store and restore the state or context of a CPU in Process Control block so that a process execution can be resumed from the same point at a later time.

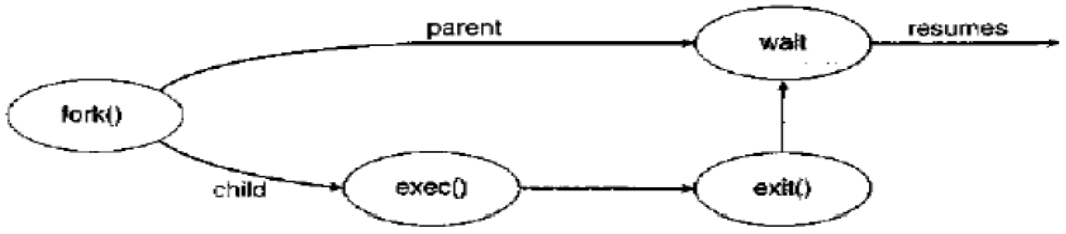
- Using this technique, a context switcher enables multiple processes to share a single CPU.
- Context switching is an essential part of a multitasking operating system features.
- Context switches are computationally intensive since register and memory state must be saved and restored.
- To avoid the amount of context switching time, some hardware systems employ two or more sets of processor registers.
- When the process is switched, the following information is stored for later use.
    - Program Counter
    - Scheduling information
    - Base and limit register value
    - Currently used register
    - Changed State
    - I/O State information
    - Accounting information

OPERATIONS ON PROCESSES:

- The processes in the system can execute concurrently, and they must be created and deleted dynamically.
- Thus, the operating system must provide a mechanism (or facility) for process creation and termination.
- In this section, we explore the mechanisms involved in creating processes and illustrate process creation on UNIX and Windows systems.

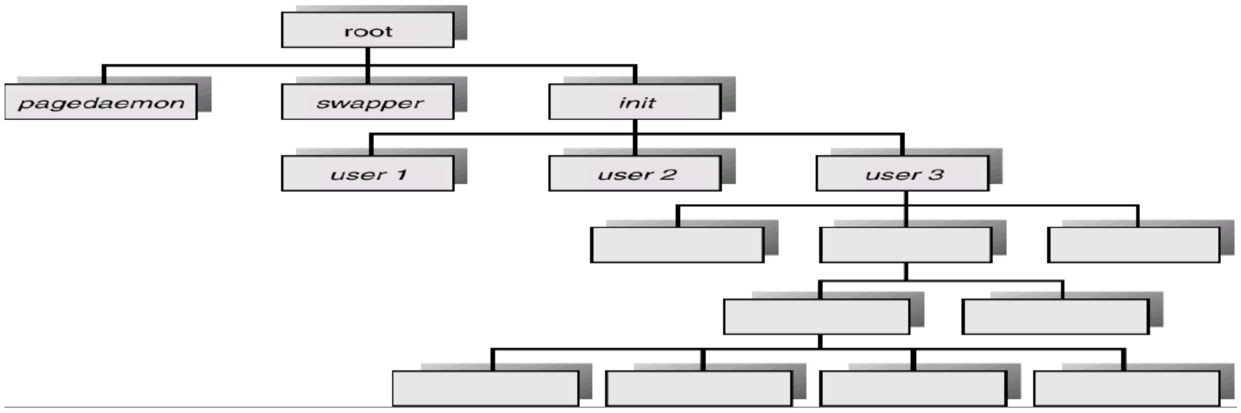
PROCESS CREATION:

- Parent process creates children processes, which, in turn create other processes, forming a tree of processes.



Process creation

- 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.
- A process may create several new processes, via a create-process system call, during the course of execution.
- The creating process is called a parent process, and the new processes are called the children of that process.



A Tree of Processes On A Typical UNIX System

- Each of these new processes may in turn create other processes, forming a tree of processes.
- When a process creates a sub process, that sub process may be able to obtain its resources directly from the operating system, or it may be constrained to a subset of the resources of the parent process.
- When a process creates a new process, two possibilities exist in terms of execution:
  - The parent continues to execute concurrently with its children.
  - The parent waits until some or all of its children have terminated.
- There are also two possibilities in terms of the address space of the new process:
  - The child process is a duplicate of the parent process (it has the same program and data as the parent).

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.
- A process is cooperating if it can affect or be affected by the other processes executing in the system.
- There are several reasons for providing an environment that allows process cooperation:
- **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.
- Concurrent execution of cooperating processes requires mechanisms that allow processes to communicate with one another and to synchronize their actions.
- To illustrate the concept of cooperating processes, let's consider the producer-consumer problem, which is a common paradigm for cooperating processes.
- A producer process produces information that is consumed by a consumer process.
- To allow producer and consumer processes to run concurrently, we must have available **A BUFFER** of items that can be filled by the producer and emptied by the consumer.
- This **BUFFER** will reside in a region of memory that is shared by the producer and consumer processes.
- The producer and consumer must be synchronized, so that the consumer does not try to consume an item that has not yet been produced.

- Two types of buffers can be used:
    - Unbounded buffer
    - Bounded buffer
- **UNBOUNDED BUFFER** –
    - Places no practical limit on the size of the buffer.
    - The consumer may have to wait for new items, but the producer can always produce new items.
- **BOUNDED BUFFER –**
    - It assumes a fixed buffer size.
    - In this case, the consumer must wait if the buffer is empty, and the producer must wait if the buffer is full.

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
- 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).

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.
- 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.**

TYPES OF CPU SCHEDULING:

- CPU scheduling decisions may take place under the following four circumstances:
- 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.
- In choosing which algorithm to use in a particular situation, we must consider the properties of the various algorithms.
- 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.
- The criteria include the following
- **CPU UTILIZATION:**
  - We want to keep the CPU as busy as possible.
  - Conceptually, CPU utilization can range from 0 to 100 percent.
  - In a real system, it should range from 40 percent (for a lightly loaded system) to 90 percent (for a heavily used system).
- **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).

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:

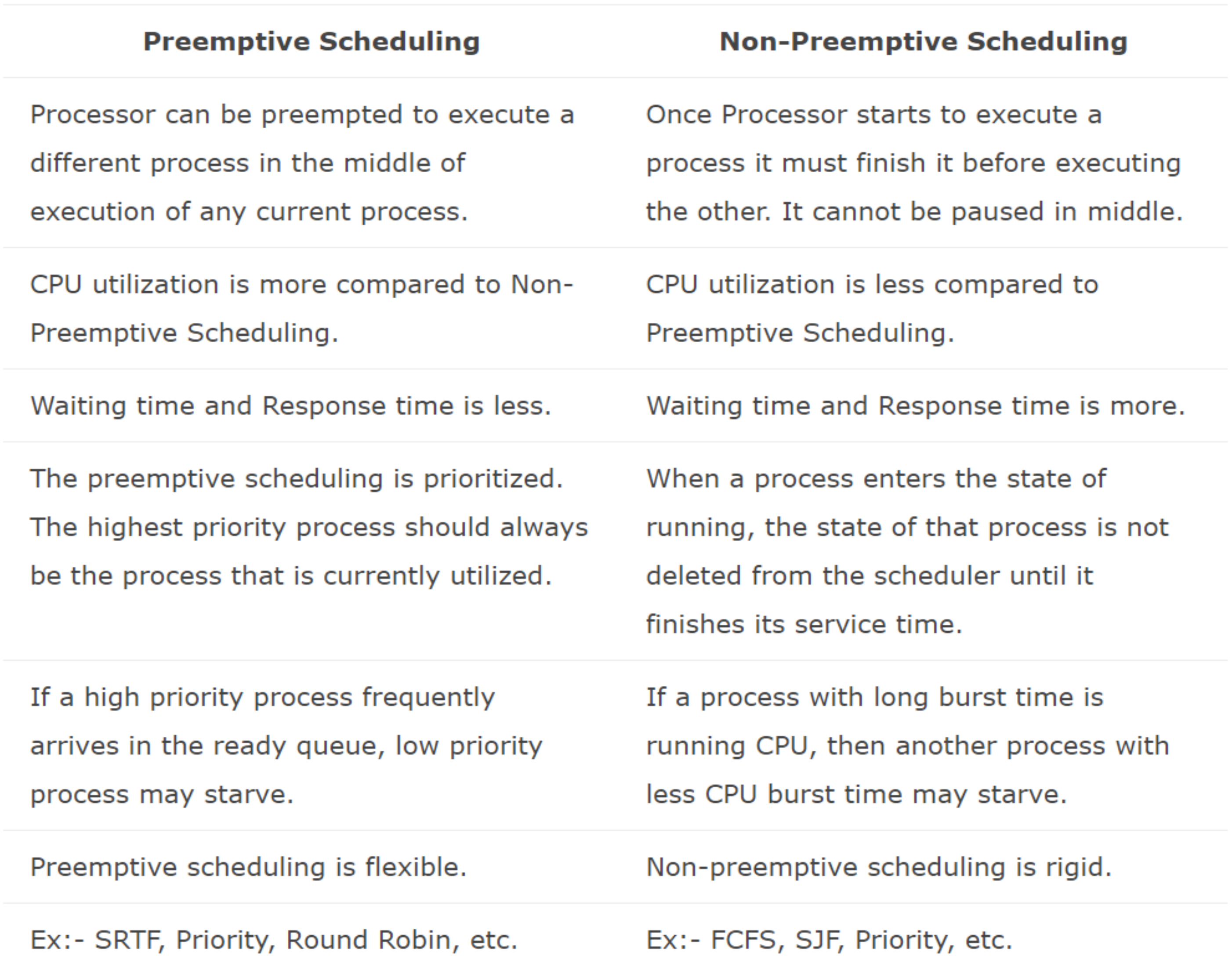
- In the "First come first serve" scheduling algorithm, as the name suggests, the process which arrives first, gets executed first, or we can say that the process which requests the CPU first, gets the CPU allocated first.
- Jobs are executed on first come, first serve basis.
- It is a non-preemptive, 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.

- **CHECK THE ANOTHER PDF FOR THE CALCULATION OF AVERAGE Turnaround Time – TAT & Waiting Time – WT**

COMMON DEFINITIONS INVOLVED IN CALCULATION PROCESS

- **ARRIVAL TIME:** Time taken for the arrival of each process in the CPU Scheduling Queue.
- **COMPLETION TIME:** Time taken for the execution to complete, starting from arrival time.
- **TURN AROUND TIME:** Time taken to complete after arrival. In simple words, it is the difference between the Completion time and the Arrival time.
- **WAITING TIME:** Total time the process has to wait before it's execution begins. It is the difference between the Turn Around time and the Burst time of the process**.**

DIFFERENCE BETWEEN PREEMPTIVE AND NON – PREEMPTIVVE



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.

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 BASED SCHEDULING:

- Priority scheduling is a non-preemptive algorithm and one of the most common scheduling algorithms in batch systems.
- Each process is assigned a priority. Process with highest priority is to be executed first and so on.
- Processes with same priority are executed on first come first served basis.
- Priority can be decided based on memory requirements, time requirements or any other resource requirement**.**

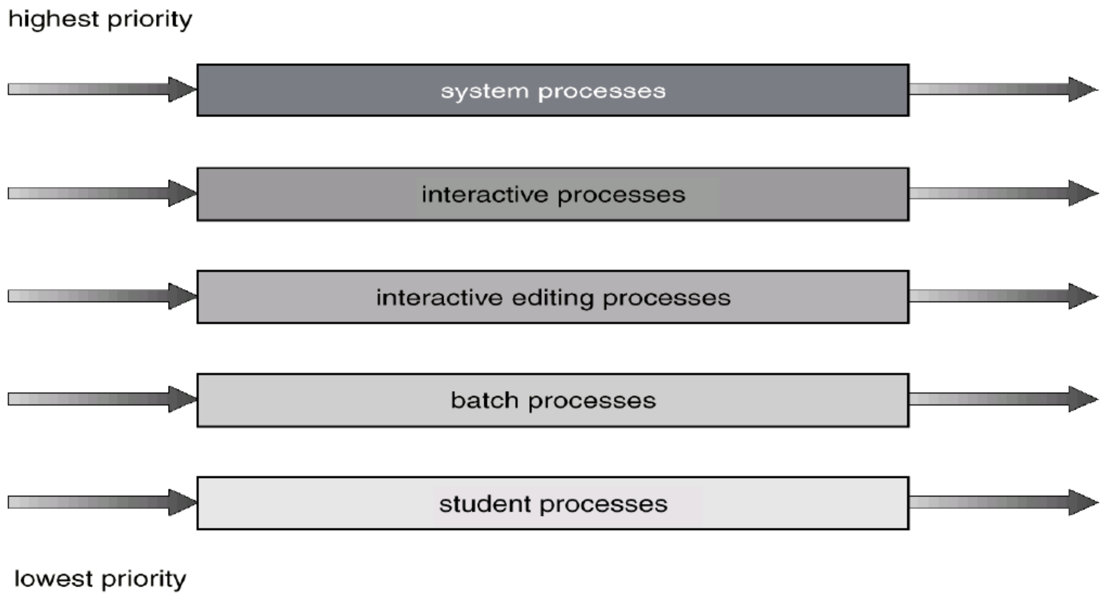
SRT – SHORTEST REMAINING TIME SCHEDULING:

- Shortest remaining time (SRT) is the preemptive version of the SJN algorithm.
- The processor is allocated to the job closest to completion but it can be preempted by a newer ready job with shorter time to completion.
- Impossible to implement in interactive systems where required CPU time is not known.
- It is often used in batch environments where short jobs need to give preference.

ROUND ROBIN SCHEDULING:

- Round Robin is the preemptive process scheduling algorithm.
- Each process is provided a fix time to execute, it is called a quantum.
- Once a process is executed for a given time period, it is preempted and other process executes for a given time period.
- Context switching is used to save states of preempted processes.

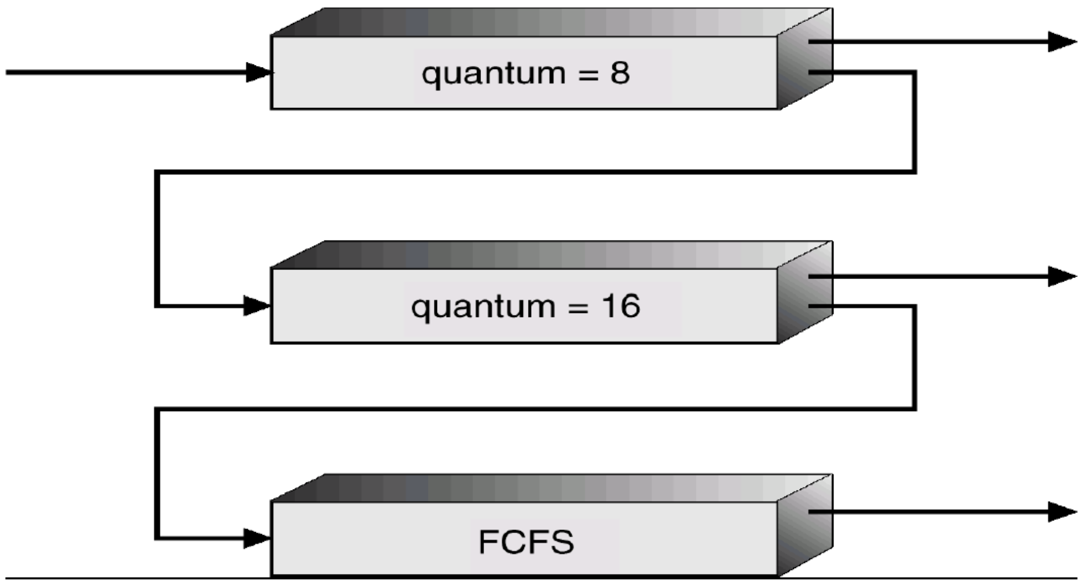
MULTIPLE-LEVEL QUEUES SCHEDULING:



- Multiple-level queues are not an independent scheduling algorithm. They make use of other existing algorithms to group and schedule jobs with common characteristics.
- Multiple queues are maintained for processes with common characteristics.
- Each queue can have its own scheduling algorithms.
- Priorities are assigned to each queue**.**

MULTILEVEL FEEDBACK QUEUE SCHEDULING:

- Normally, in the multilevel queue scheduling algorithm, processes are permanently assigned to a queue when they enter to the system.
- The multilevel feedback-queue scheduling algorithm, in contrast, allows a process to move between queues.
- The idea is to separate processes according to the characteristics of their CPU bursts.
- If a process uses too much CPU time, it will be moved to a lower-priority queue.
- This scheme leaves I/O-bound and interactive processes in the higher-priority queues.
- Similarly, a process that waits too long in a lower-priority queue may be moved to a higher-priority queue. This form of **AGING** prevents **STARVATION.**



Multilevel feedback queues