Unit - II

process Management

processes :-

A process is a program in execution. It is an active entity, unlike a program, which is passive.
- It represents the basic unit of work that the system must manage. when a program is loaded into memory, it becomes a process, which can be divided into four section: stack, heap, text, and data.

Components of a Process :-

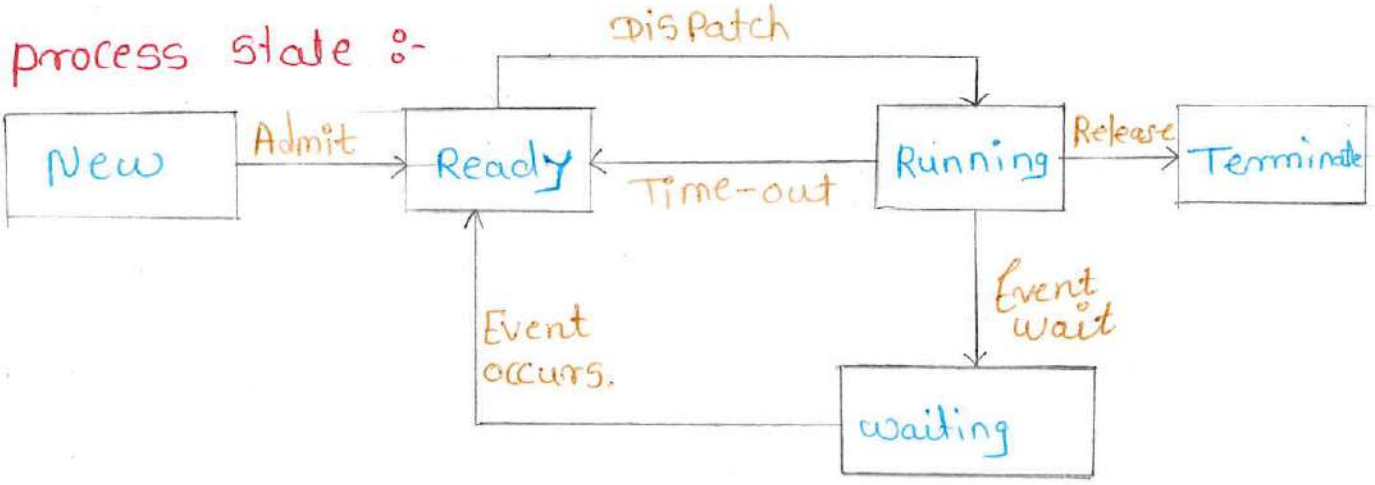
Stack :- This section contains temporary data such as method function parameters, returns addresses, and local variables.

Heap :- This is dynamically allocated memory used by the process during its runtime.

Text :- This includes the current activity represented by the value of the program counter and the contents of the processor's registers.

Data :- This section contains global and static variables.

process state :-



~~New :-~~ A process goes through various state during its life cycle. These states help the OS manage the execution of multiple processes efficiently. process can be execute in the different state like. - New, Ready, Running, waiting, Terminated.

New :- The new state is the ~~int~~ process is being created and initialized by the operating system.

Ready :- The process is loaded in memory and waiting to be assigned to a processor for execution.

Running :- The process is currently executing instruction on the CPU.

Waiting :- The process is waiting for an event to occur, such as I/O operation completion or receiving a signal.

Terminated :- 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.

process Control Block (PCB) :-
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).

process State :- The current state of 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. It is point to another process control block. pointer is used for maintaining the scheduling list.

process Number :- Each process is identified by its process number, called process identification number. Every process has a unique process ID. This process ID is provided by the os.

priority :- Each process is assigned a certain level of priority that corresponds to the relative importance of the event that its Services process priority is the preference of the one process over other process for execution.

program Counter :- The counter indicates the address of the next instruction to be executed for this process.

CPU registers :- The registers vary in number and type, depending on the computer architecture. They include accumulators, index registers, stack pointers, and general-purpose registers, plus any condition-code information.

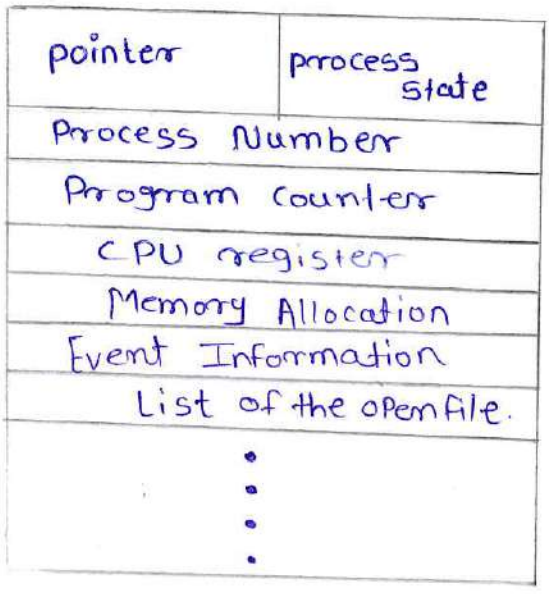
CPU scheduling Information :- This information includes a process priority, pointers to scheduling queues, and any other scheduling parameters.

Memory management Information :- This information include such information as the value of the base and limit registers, the pages tables, or the segment tables, depending on the memory system used by the OS.

Accounting Information :- This information includes the amount of CPU and real time used, time limits, account numbers, job or process numbers, and so on.

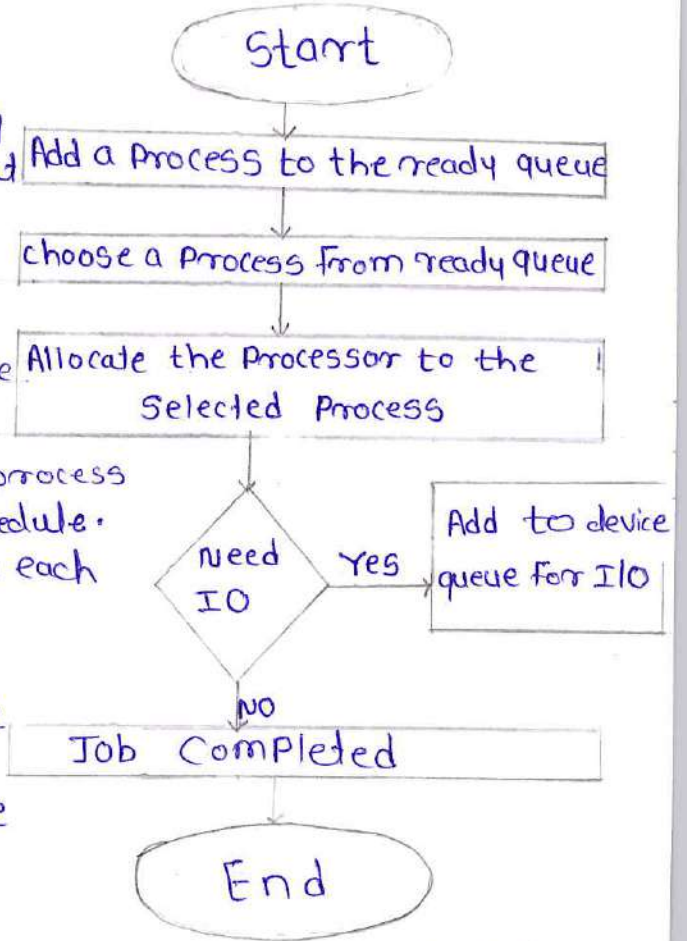
I/O status Information :- This information includes the list of I/O devices allocated to the process, a list of open files, and so on.

File Management :- It includes information about all open files, access rights etc.

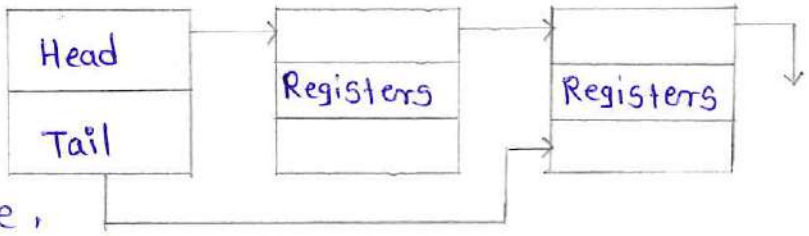


Process Scheduling

- As we know that we can perform many programs at a time on the computer, but there is a single CPU. So, for running all the programs concurrently or simultaneously, then we use the scheduling.
- processes are the small programs those are executed by the user according to their request. CPU executes all the process according to some rules or some schedule.
- Scheduling provides time of CPU to each process.
- Scheduling is that in which each process has some amount of time of CPU.
- The process of scheduling can be explained with the help of fig.



Scheduling queues :- For a uniprocessor system, there will never be more than one running process. If there are more than one processes, the rest will have to wait until the CPU is free and can be rescheduled.



- The processes, which are ready and waiting to execute, are kept on a list called the ready queue.
- The list is generally a linked list. A ready que header will conta-in pointers to the first and last PCB's in the list. Each PCB has a pointer field which points to the next process in the ready queue.
- There are also other queues in the system. When a process is allocated the CPU, it executes for a while and eventually quits, is ~~int err~~ interrupted or waits for the occurrence of a part-ticular event, such as the completion of an I/O request.
- In the case of an I/O request, such as tape drive, or to a shared device such as a disk. Since there are many processes in the system, the disk may be busy with I/O request of some other process.
- The process therefore may have to wait for the disk. The list of processes waiting for a particular I/O device is called a device queue. Each device has its own device queue.

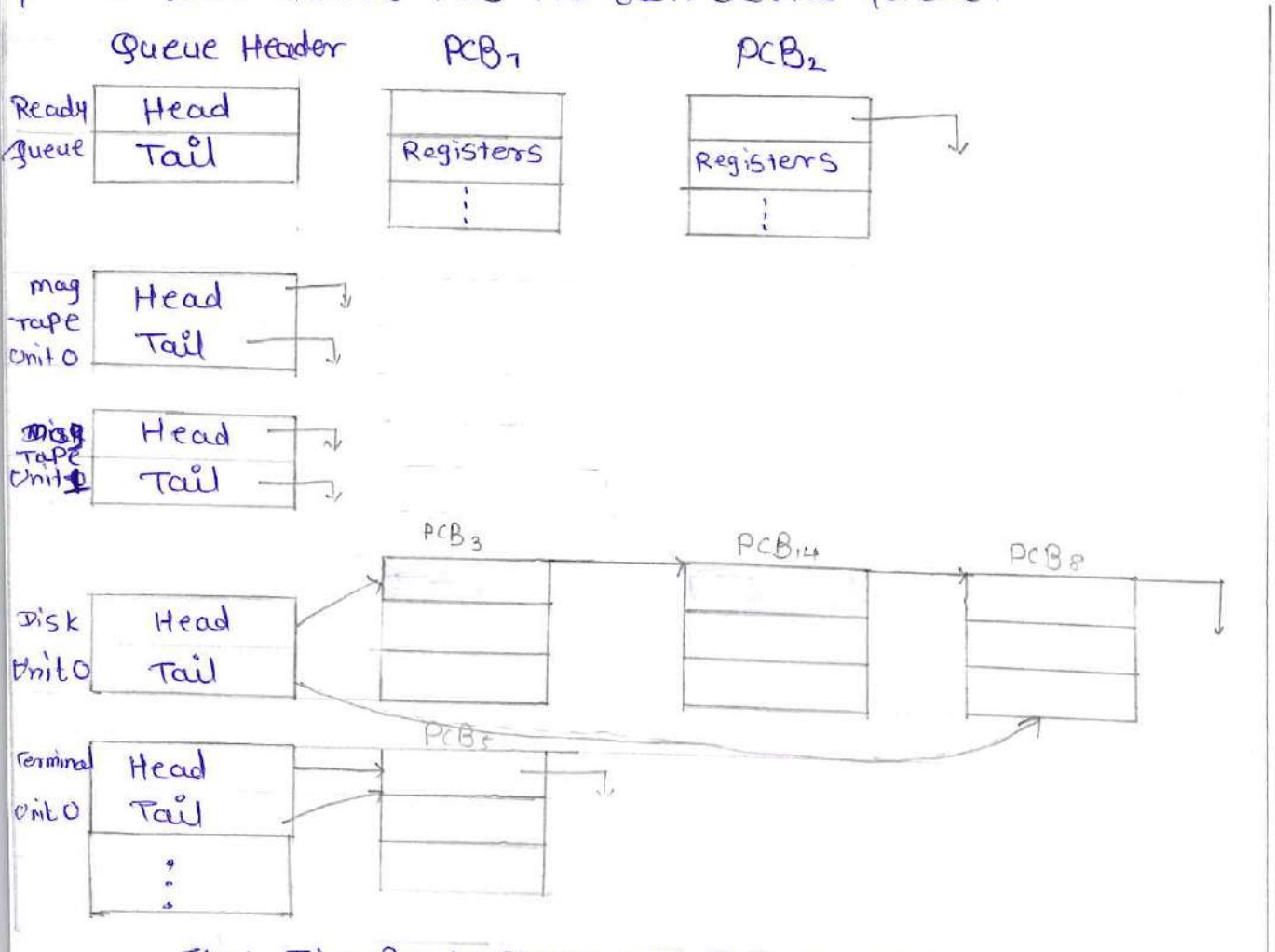


Fig :- The Ready Queue and various I/O Device Queue.

A process enters the system from the outside world and is put in the ready queue. It waits in the ready queue until it is selected for the CPU. After running on the CPU, its waits for an I/O operation by moving to an I/O queue.
Eventually it is served by the I/O device and returned to the ready queue. A process continues this CPU, I/O cycle until it finishes and then it exits from the system.

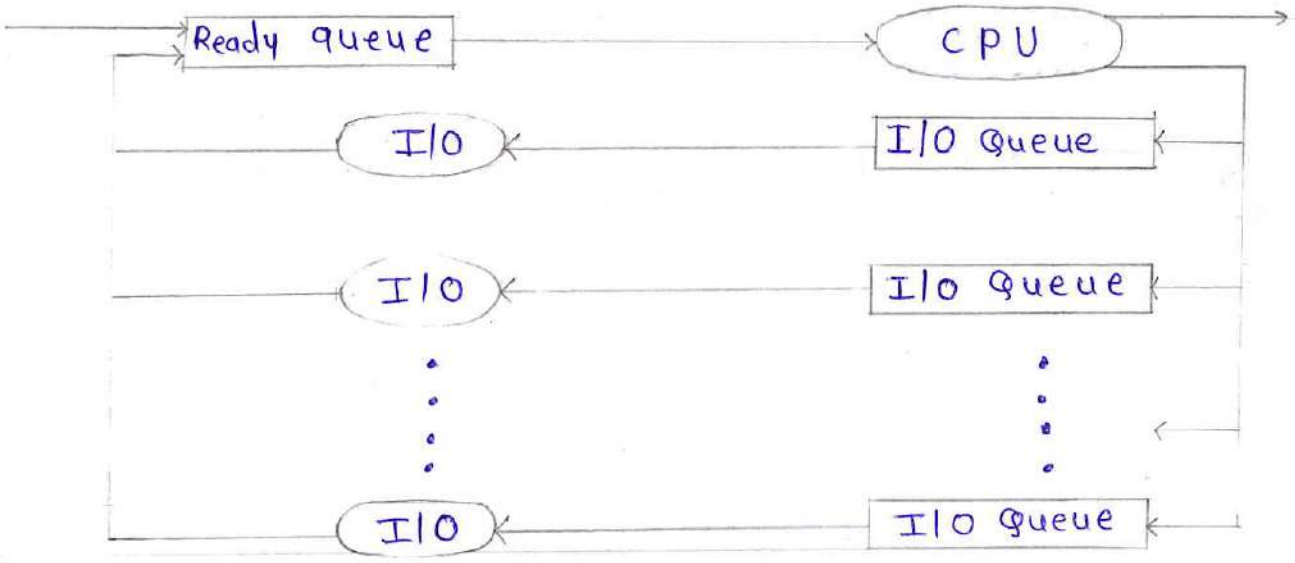


Fig : Queuing diagram representation of CPU scheduling

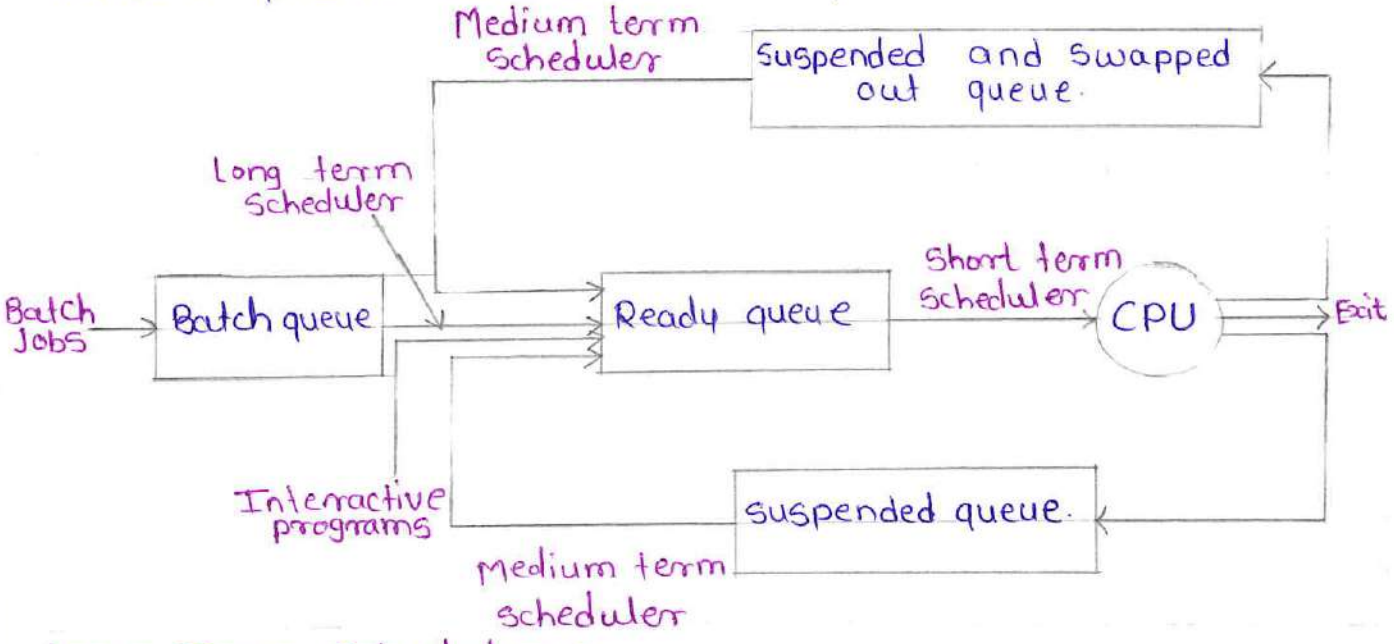
Fig shows the queuing diagram for representing the process scheduling. Each rectangle box represents a queue.
- In the queuing diagram two types of queues are present namely the ready queue and a set of device queues. The circles represent the resources that serve the queues, and the arrows indicate the flow of processes in the system.

AS 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 assigned the CPU and is executing. One of several events could occur.
- The process could create a new sub-process and wait for the termination of the sub-process.
- The process could issue an I/O request and then be placed in an I/O queue.
- The process could be removed forcibly from the CPU, as a result of an interrupt, and again put in the ready queue.

Types Of Schedulers :-

Schedulers in operating system are special system software's which handles process scheduling in various / different ways. Schedulers main task is to select the Jobs to be submitted into the system and to decide which process to run.

In other words, the job of process scheduling is done by a software routine called as Scheduler.
- Schedulers are of three types, namely, Long term scheduler, short Term Scheduler and Medium-Term scheduler.
Various types of schedulers are explained below:



- Long Term Scheduler :-

- The long-term scheduler determines which jobs are admitted to the system for processing.
- In batch system there are more jobs submitted that can be executed immediately. These jobs are spooled to mass storage device, where they are kept for later execution (ready Queue).
- The long-term scheduler selects jobs from this job pool and loads into memory for execution. It is also called job scheduler or admission scheduler
- Long term scheduler determines which programs are admitted to the system for processing. Job scheduler selects processes from the queues and loads them into memory for execution.
- on some systems, the long-term scheduler may not be available or minimal. Time-sharing operating systems have no long-term scheduler. When process changes the state from new to ready, then they is use of long-term scheduler.

- Medium Term Scheduler :-

- On some systems, the long-term scheduler may be absent or minimal. For e.g. time sharing systems such as unix and Microsoft windows systems often have no long-term scheduler but simply put every new process in memory for the short-term scheduler.
- Some OS, such as time-sharing systems, may introduce an additional, intermediate level of scheduling, medium term scheduling is a part of swapping so it is also known swapper.
- The medium-term scheduler temporarily removes processes from main memory and places themes to secondary memory or vice versa.

This is commonly referred to as "swapping out" or "swapping in"

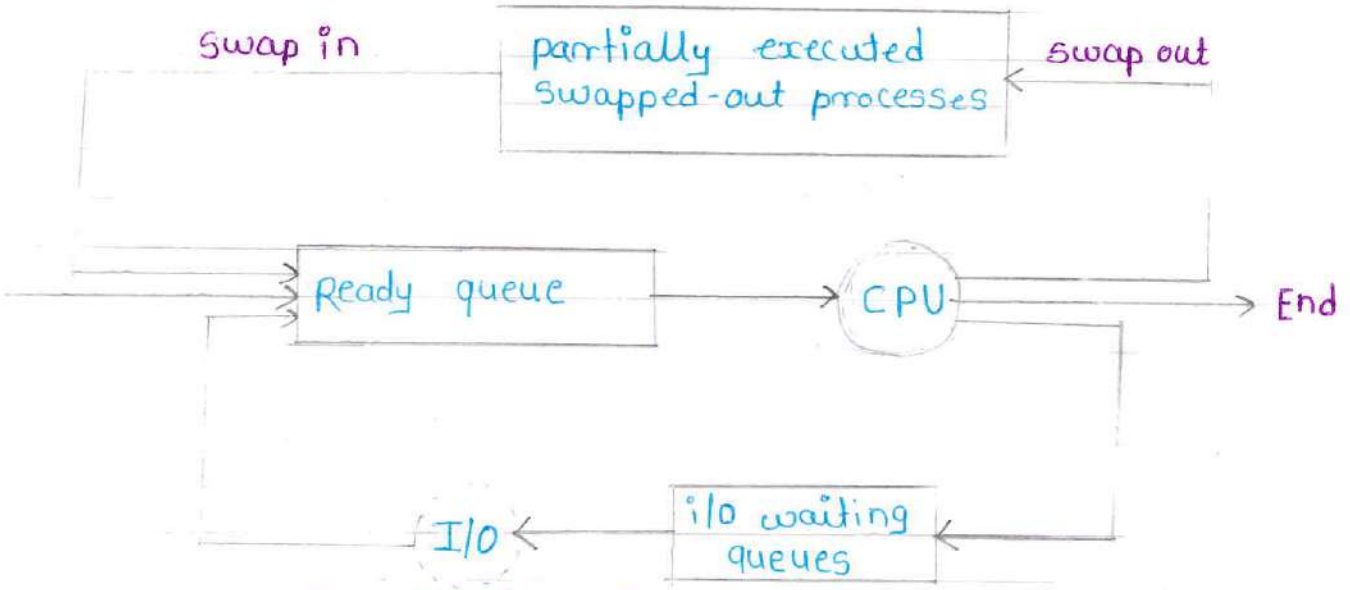


Fig: Addition of medium-Term scheduling

Short Term Scheduler :-

- The short-term scheduler selects from memories, which are ready to execute and allocates the CPU's one of them. It is also called as CPU scheduler or process scheduler.
- The short-term scheduler makes scheduling decision much more frequently than the long term or mid term schedulers. a scheduling decision will at a minimum have to be made after every time slice, and these are very short.
- In most cases short term scheduler is written in assembly because it is a critical part of the operating system.
- This scheduler can be preemptive, implying that it is capable of forcibly removing processes from a CPU when it decides to allocates that CPU to another process, or non-preemptive, in which case the scheduler is unable to "force" processes off the CPU.
- short term scheduler is invoked very frequently. long term scheduler is invoked very infrequently. The long-term scheduler controls the degree of multiprogramming.

- processes can be described as either :-

1) I/O-bound process spends more time doing I/O than computations, many short CPU bursts.

2) CPU-bound process spends more time doing computations; few very long CPU bursts.

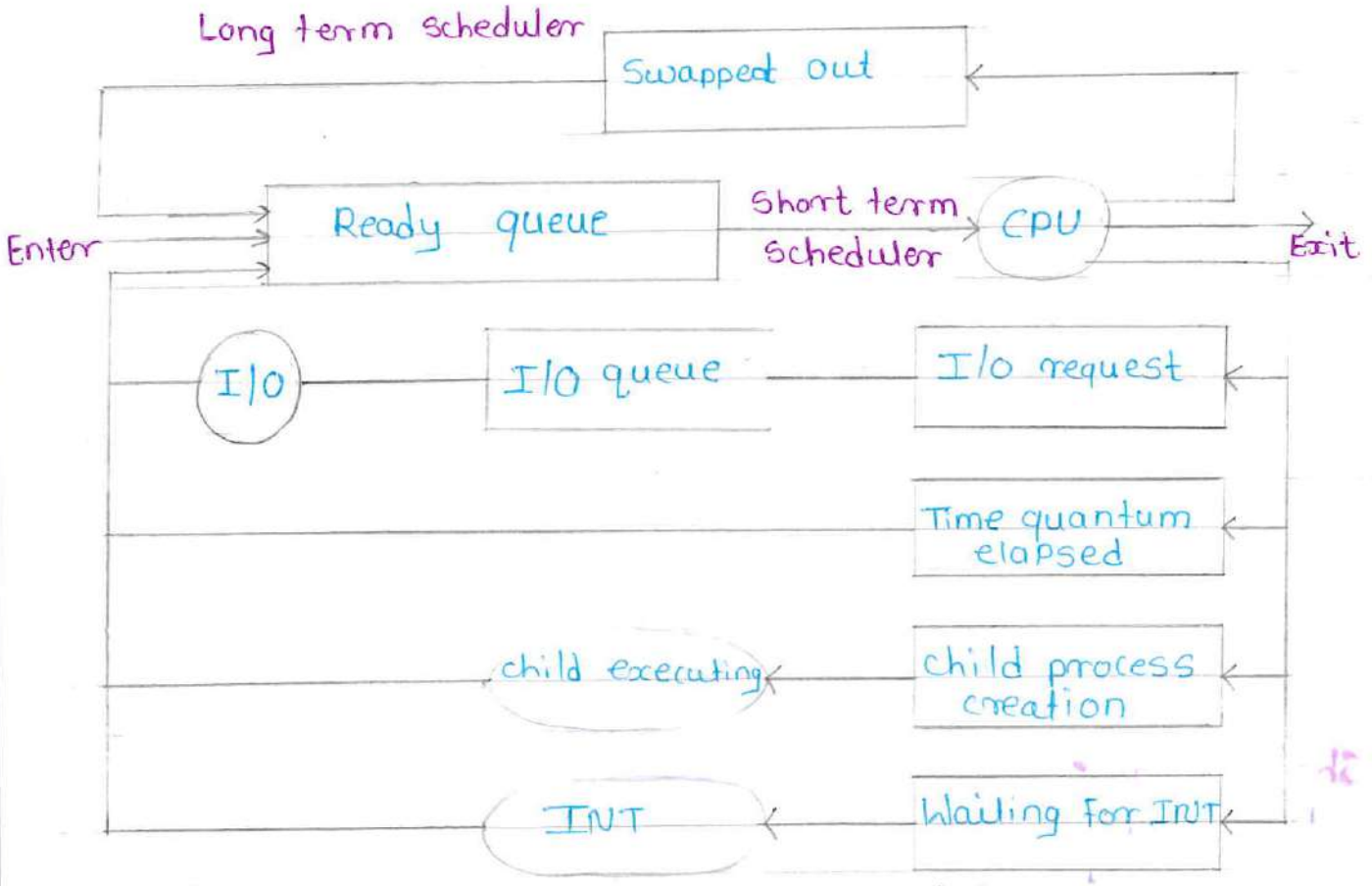


Fig: Representation of process scheduling

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.
- CPU switching from one process to another process is called a context switch.
- A context switch is the computing process of storing and restoring the state of a CPU so that execution can be resumed from the same point at a later time.
- This enables multiple processes to share a single CPU. The context switch is an essential feature of a multitasking operating system.
- The context of process is represented in the PCB of the process, it includes the value of the CPU registers the process state and memory management information
- Context switch times are highly dependent on hardware support. Its speed varies from machine to machine depending on the memory speed, the number of registers that must be copied and the existence of special instructions.
- Typically, the speed ranges from 1 to 1000 microseconds. context switch times are highly dependent on Hlw support.

Some hardware systems employ two or more sets of processor registers to reduce the amount of context switching time. When the process is switched, the following information is stored:

- program counter.
- Scheduling Information
- Base and limit register value
- Currently used register
- changed state
- Ilo state
- Accounting.

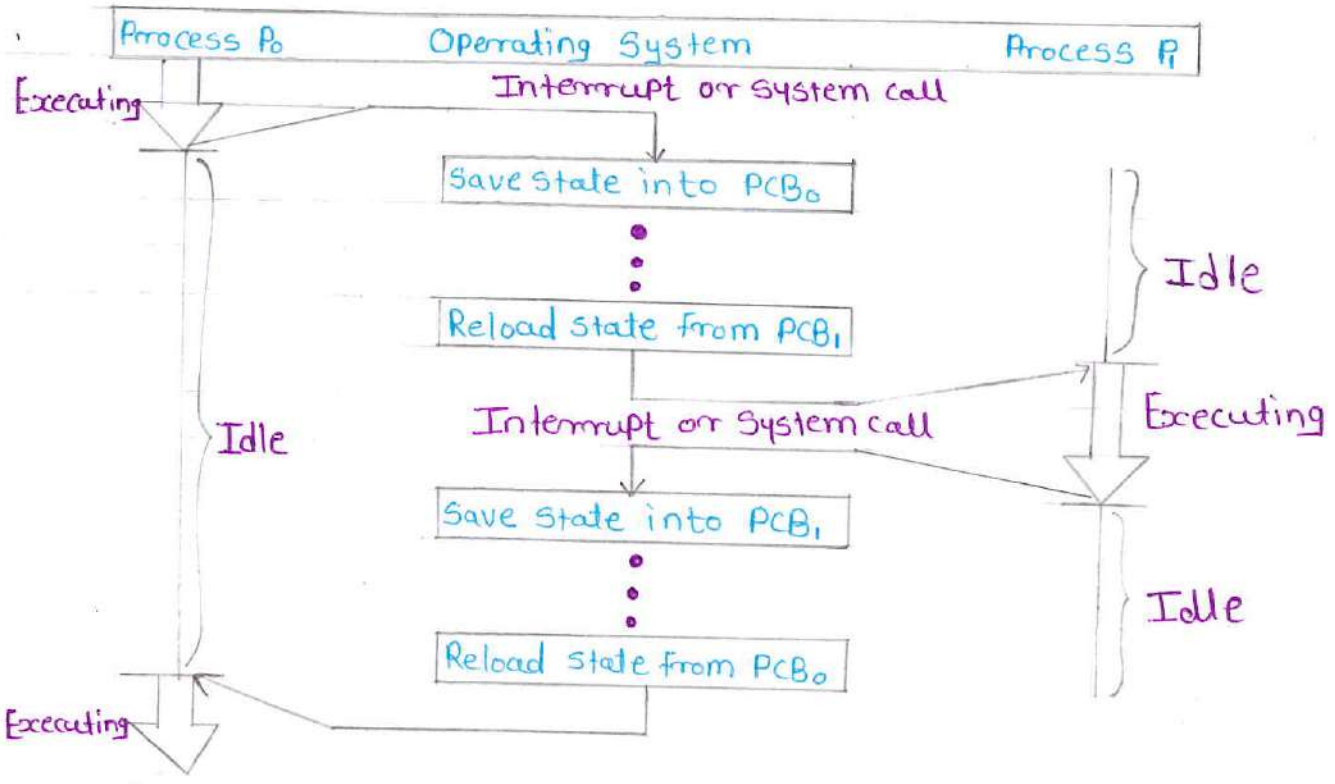


Fig: CPU switch from process to process.

Inter-process Communication (IPC)

- Inter-process Communication or IPC is a mechanism that allows processes to communicate and share data with each other while they are running. Since each process has its own memory space, IPC provides controlled methods for exchanging information and coordinating actions. It helps processes work together efficiently and safely in an operating system.
- There are applications that involve each executing multiple processes concurrently. processes work together to perform application-specific tasks. They are referred to as co-operating processes.
- When multiple processes run on a system concurrently and more than one process requires CPU at the same time, then it becomes essential to select any one process to which the CPU can be allocated.

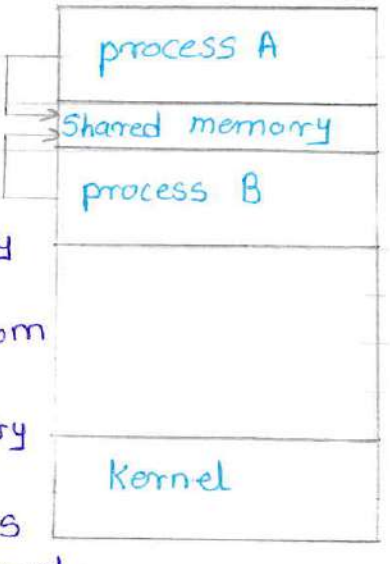
To serve this purpose, scheduling is required.
- Moreover, the multiple processes running on a system also need to intercommunicate in order to reciprocate some data or information.
- This kind of intercommunication between several processes is referred to as Inter process communication (IPC).

Overview Of IPC:

processes executing concurrently in the operating system might be either independent processes or co-operating processes.
Interprocess communication (IPC) is a set of programming interfaces that allow a programmer to coordinate activities among different program processes that can run concurrently in an OS.
- This allows a program to handle many user requests at the same time. Since even a single user request may result in multiple processes running in the OS. on the user's behalf, the processes need to communicate with each other. The IPC interfaces makes this possible.
- The interocess communication is a set of techniques for the exchange of data among multiple processes. IPC techniques divided into methods for synchronization, message passing, stored memory etc.

Two fundamental models allow inter-process communication as explain below:-

Shared Memory Model:-

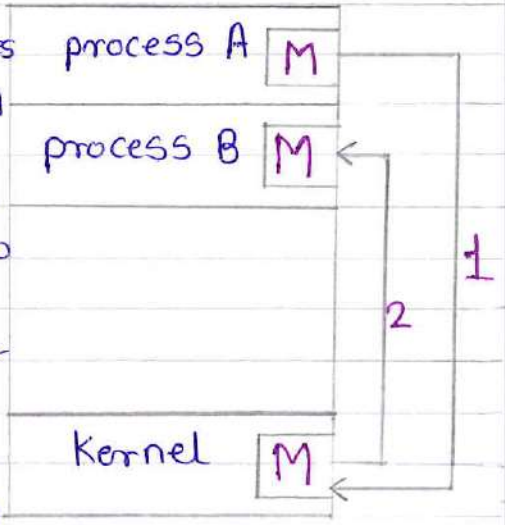


- In the above shared memory model, a common memory space is allocated by the kernel.
- process A writes data into the shared memory region (step1).
- process B can then directly read this data from the same shared memory region (step2).
- Since both processes access the same memory segment, this method is fast but requires synchronization mechanisms to avoid conflicts when multiple processes read/write simultaneously.
- E.g. - Multiple people can edit the document at the same time in shared google doc.
- Communication between processes using shared memory requires processes to shared a memory segment, the implementation of which is handled by the programmer. processes can use

Shared memory for extracting information as a record from another process as well as for delivering any specific information to other processes.

Message Passing :-

Message passing is a method where processes communicate by sending and receiving messages to exchange data. One process sends a message and the other process receives it, allowing them to share information. Message passing can be achieved through different methods like sockets, Message Queues or pipes.



- In the above message passing model, Processes exchanges information by sending and receiving messages through the kernel.
- process A sends a message to the kernel.
- The kernel then delivers the message to process B.
- Here, processes do not share memory directly. Instead, communication happens via system calls (send(), recv(), or similar).
- This method is simpler and safer than shared memory because there's no risk of overwriting shared data, but it incurs more overhead due to kernel involvement.
- E.g.- Multiple people send updates to a group chat, but each message goes through the server before others see it, like processes sending messages instead of directly sharing memory.

Threads :-

- A thread, sometimes called a Light Weight process (LWP), is a basic unit of CPU utilization; it comprises a thread ID, a program counter, a register set and a stack.
- A thread is defined, "as a unit of concurrency within a process and had access to the entire code and data parts of the process". Thus, thread of the same process can share their code and data with one another.
- It shares with other threads belonging to the same process its code section, data section, and other operating-System resources, such as open files and signals.
- Each thread has its own program counter, register set, and stack space.
- Threads share code, data, and operating system resources within the same process.

Advantages of Threads :-

- Threads improve the performance of a program.
- Concurrent operations can be achieving using threads within a process.
- multiple threads are useful in a multiprocessor system where threads run concurrently on separate processors.
- Multiple threads also improve program performance on single-processor systems by permiting the overlap of input and output or other slow operations with computational operations.
- Thread minimizes context switching time.
- A process with multiple threads makes a greate server for example printer server.
- Because threads can share common data, they do not need to use inter process communication.
- context switching are fast when working with threads.

Benefits Of Threads :-

A thread is a single sequence stream within a process. Because threads have some of the properties of processes, they are sometimes called lightweight process.

- The benefits of multiithreaded programming are categories as:
- Responsiveness :- Multiithreading an interactive application may allow a program to continue running even if part of it is blocked or is performing a lengthy operation, thereby increasing responsiveness to the user.

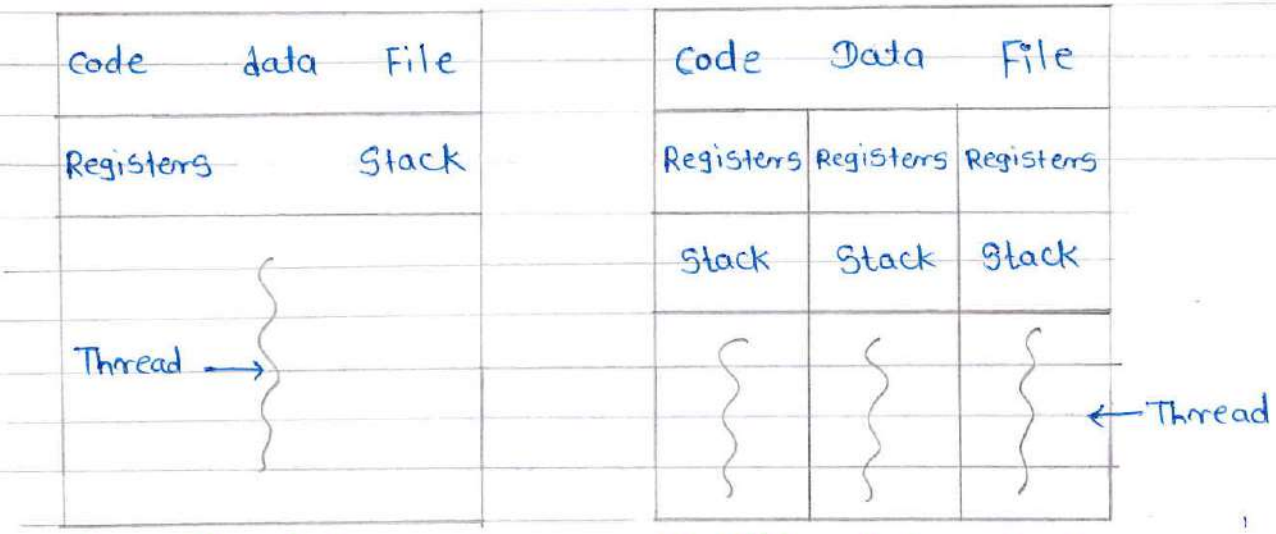
Resource sharing :- By default, threads share the memory and the resources of the process to which they belong. The benefit of code sharing is that it allows an application to have several different threads of activity all ~~which~~ within the same address space.

Economy :- Allocating memory and resources for process creation is costly. Alternatively, because threads share resources of the process to which they belong, it is more economical to create and context switch threads. It can be difficult to gauge empirically the difference in overhead for creating and maintaining a process rather than a thread, but in general it is much more time consuming to create and manage processes than threads.

Utilization of multiprocessor Architectures :- The benefits of multiithreading can be greatly increased in a multiprocessor architecture, where each thread may be running in parallel on a different processor. A single threaded process can only run on one CPU, no matter how many are available. multi threading on a multi-CPU machine increase concurrency.

Concept Multithreaded Programming :-

- A thread is the fundamental unit of CPU utilization. A thread is a program execution that uses the resources of a process.
- A traditional or heavyweight process has a single thread of control comprises a single thread of control i.e., it can execute one task at a time and thus, is reffered to as a single-threaded process.
- If a process has multiple threads of control, it can perform more than one task at a time, such a process is known as multithreaded process.
- Fig shows the single threaded and multithreaded processes



single threaded process

Multithreaded process.

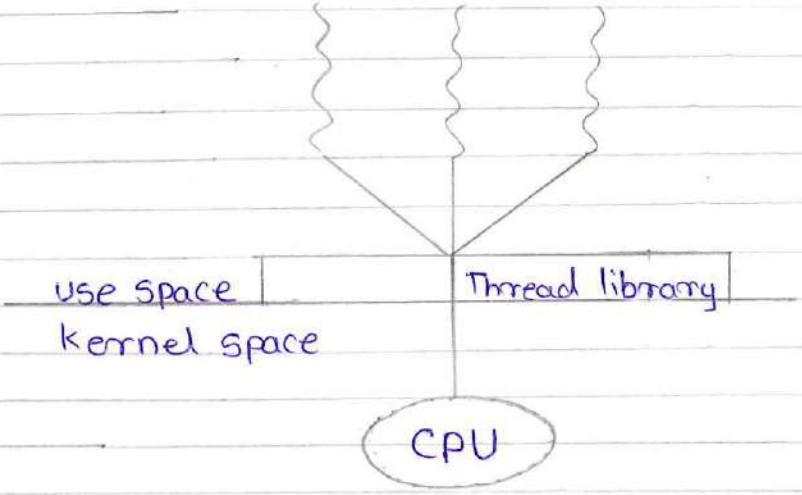
Types Of Threads :-

There are two types of threads.

- User level Thread.
- Kernel level Thread.

User level Threads -

- Managed entirely in user space using a thread library, the kernel is unaware of them.
- Switching between ULTs is fast since only program counter, registers, and stack need to be saved / restored.
- Do not require system calls for creation or management, making them lightweight.
- If one thread makes a blocking system call, the entire process is blocked.
- scheduling is done by the application itself, which may not be as efficient as kernel-level scheduling.
- cannot fully utilize multiprocessor systems because the kernel schedules processes, not individual user-level threads.



Kernel-Level Threads -

- Managed directlly by the operating system kernel; each thread has an entry in the kernel's thread table.
- The kernel schedules each thread indepe ndently, allowing true parallel execution on multiple CPUs/cores.
- Handles blocking system calls efficiently; if one thread blocks, the kernel can run another thread from the same process.
- provides better load balancing across pro cessors since the kernel controls all threads.
- Context switching is slower compared to ULTs because it requires switching between user mode and kernel mode.
- Implementation is more complex and requires frequent interact -ion with the kernel.

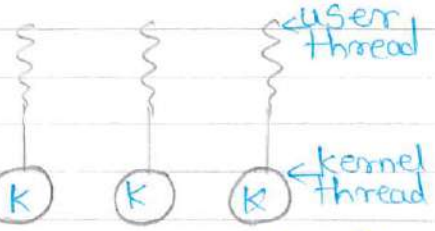
Multithreading Models :-

Many systems provide support for both user and kernel threads, resulting in different multithreading models. Three types of multi -threading models implemented.

One-to-One Model :-

The one-to-one model maps each user thread to a kernel thread.

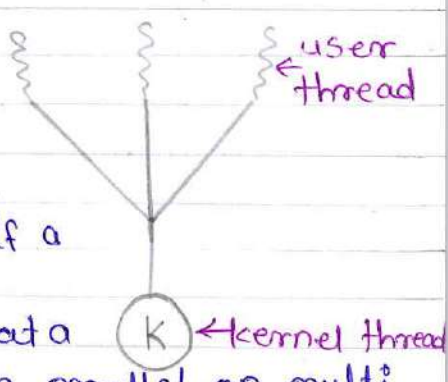
- It provides more concurrency than the many-to-one model by allowing another thr -ead to run when a thread makes a blocking system call; it also allows multiple threads to run in parallel on multipr -ocessors.



Advantage of One-to-one model :-
- More concurrency because of multiple threads can run in parallel on multiple CPUs.
- Multiple threads can run parallel.
- Less complication in the processing.

Disadvantages of one-to-one model-
- Thread creation involves Light weight process creation.
- Every time with user's thread, kernel thread is created.
- Limiting the number of total threads.
- kernel thread is an overhead.
- It reduces the performance of system.

Many to One Model -

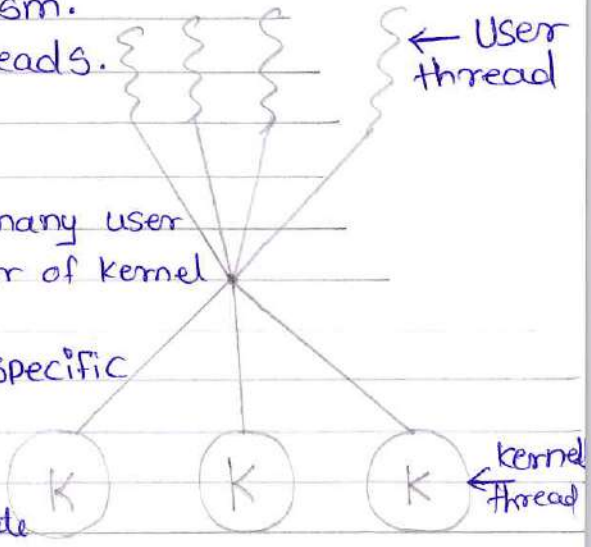


- The many to one model maps many user level threads to one kernel thread. Thread management is done in user space, so it is efficient, but the entire process will block if a thread makes a blocking system call.
- only one thread can access the kernel at a time. Multiple threads are unable to run in parallel on multi processors.

~~- Green threads a~~

Advantages of Many-to-one Model :-
- Totally portable.
- Easy to do with few systems dependencies.
- Mainly used in language systems, portable liabraries.
- Efficient system in terms of performance.
- one kernel thread controls multiple user threads.

Disadvantages of Many-to-one Model :-
- cannot take advantage of parallelism.
- one block call blocks all user threads.



Many to Many Model

- The many-to-many model multiplexes many user level threads to a smaller or equal number of kernel threads.
- The number of kernel threads may be specific to either a particular application or a particular machine.
- This model allows developer to create as many threads.

Concurrency is not gained because the kernel can schedule only one thread at a time.

-The one-to-one model allows for greater concurrency, but the developer has to be careful not to create too many threads within an application.

Advantages of many-to-many model:-
- Many threads can be created as per user's requirement.
- Multiple kernel or equal to user threads can be created.

Disadvantages of many-to-many model:-
- True concurrency cannot be achieved
- Multiple threads of kernel are an overhead for operating system.
- performance is less.

Execute process commands like: top, PS, kill, wait, ~~spt~~ sleep, exit, nice.

PS command:-
- When a program runs on the system, it's referred to as a process Linux is a multitasking operating system, which means that more than one process can be active at once.
- To know the status of process PS command is used. PS stands for process status. The PS command is used to display the characteristics of a process.

Syntax: PS [option]

e.g. default PS command show only processes that belong to the current user and that are running on the current terminal

```
$PS
  PID  TTY  TIME  CMD
   30  01   0.03  Sh
   56  01   0.00  Ps
$.
```

top command:-
- top stand for table of processes. As the name suggests, it displays a list of processes in table form
- top command is used to show the Linux processes. It provides a dynamic real-time view of the running system. Usually, this command shows the summary information of the system and the list of processes or threads which are currently managed by the Linux Kernel.

Syntax top.

Kill command:-

The kill command is the most commonly used command to terminate a process. The kill command will kill a process using the kill

signal and PID given by the user.
The kill command allows you to send signals to processes based on their process ID.
- The command used to one or more PID numbers as its arguments.
Syntax- kill [signal] PID

Wait command :-

Wait is a built-in shell command which wait for a given process to complete, and returns its exit status. wait waits for the process identified by process ID Pid, and reports its terminati-on status.
Syntax - wait PID.
E.g. wait 2112
- wait for process 2112 to terminate, and return its exit status.

Exit Command :- The exit command exits the current terminal or shell. The exit command terminates a script just as in a C program.
- Syntax exit
E.g. The simplest use of the exit command is without any parameters. When executed, it will close the current terminal session.

nice Command :-

- The linux nic command allows us to launch processes with altered priorities, which affects process scheduling.
The nice command is used to start a new process with a specific priority, known as the nice value. "A higher nice value lowers the process's priority, while a lower nice value increases it.
Syntax :- nice -n [nice value]

Unit - II process Management (co 2)

2 - Marks question -

1) Draw the process state diagram. (W-2019)
2) Draw a neat labelled diagram of process state (summer2
3) write the syntax of sleep and kill commands. (winter2019)
4) write commands to add delay in a script and terminate a process (Summer-2022)
5) Define process and PCB (winter 2023)
6) List different types of process scheduler. (Summer 2024)

4 - Marks question -

1) Explain process control Block (PCB) with neat diagram (W19
2) Differentiate shared memory and message passing IPC. (S-22)
3) Explain scheduling queues with suitable diagram (S-22)
4) Explain PS command with any four option (w-19)
5) Explain ps and wait commands with suitable eg. (S-22)
6) Explain type of process scheduler (w-19)
7) Explain context switching (w-23)

6 - Marks question -

1) Explain Multithreading Models with neat diagram (W-19)
2) Explain one-to-one Multithreding Model with diagram (Summer-22)
3) Explain IPC using shared Memory and message passing with diagram (w-23).
4) Explain PCB and context switching with suitable diagram (S-24)