COURSE NAME : OPERATING SYSTEMS

(COURSE CODE :22516)

COURSE TEACHER : MRS.T.H.GAVHANE

UNIT NO 3 (14Marks)

PROCESS MANAGEMENT

Course Outcome: Execute Process Commands for performing process management operations

- **UNIT OUTCOME**

3a	**Explain functions carried out in the given process state**
3b	**Describe the function of the given components of process stack in PCB**
3c	**Explain characteristics of the given multithreading model**
3d	**Describe method of executing the given process command with example**

3.1 Process : Process State, Process Control Block (PCB)

What do you mean by Process?

A process can be thought of as a program in execution. 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.

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. All these processes may execute concurrently

The Process :

A process is a program in execution. A process is more than the program code, which is sometimes known as the **text section.**

It also includes the current activity, as represented by the value of the **program counter** and the contents of the processor's registers.

A process generally also includes the **process stack,** which contains temporary data (such as function parameters, return addresses, and local variables), and

A data section, which contains global variables.

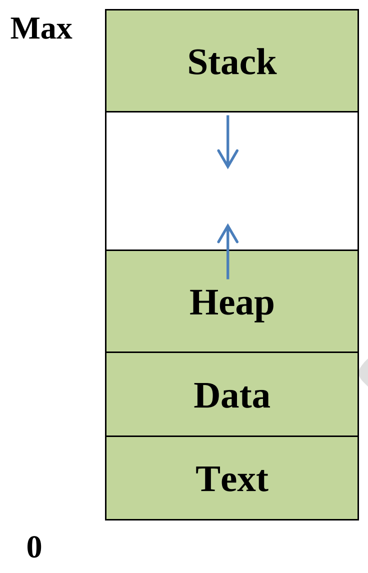
A process may also include **a heap**, which is memory that is dynamically allocated during process run time.

The structure of a process in memory is shown in

A program is a passive entity, such as a file containing a list of instructions stored on disk (often called an executable file).

In contrast, **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. Two common techniques for loading executable files



Process States:

As a process executes, it changes state. The state of a process is defined in part by the current activity of that process.

A process may be in one of the following states:

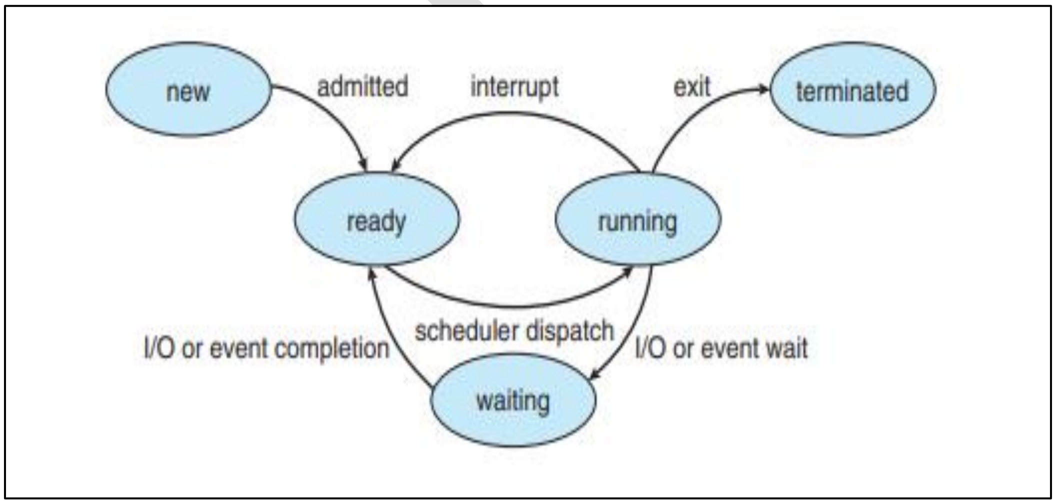
• **New**. The process is being created.

• **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).

• **Ready**. The process is waiting to be assigned to a processor.

• **Terminated**. The process has finished execution.



Process Control Block (PCB) :

Each process is represented in the operating system by a process control block (PCB)—also called a task control block. A PCB is shown in Figure It contains many pieces of information associated with a specific process, including these:

Process state. The state may be new, ready, running, waiting, halted, and so on.

• **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. Along with the program counter, this state information must be saved when an interrupt occurs, to allow the process to be continued correctly afterward

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

• **Memory-management information**. This information may include such items as the value of the base and limit registers and the page tables, or the segment tables, depending on the memory system used by the operating system

• **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.

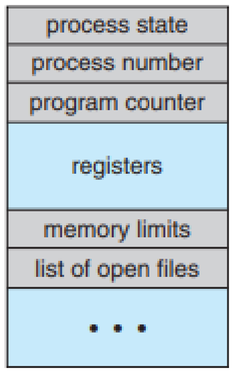


Fig. Process Control Block

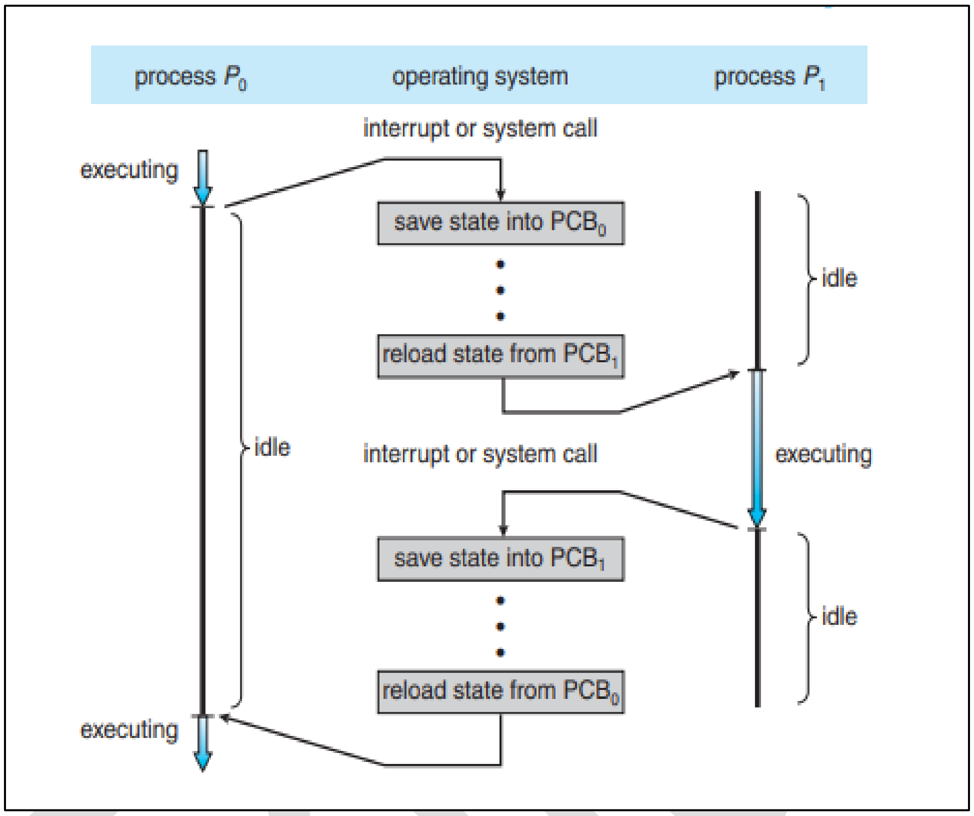


Diagram showing CPU switch from process to process.

3.2 Process Scheduling : Scheduling Queues , Schedulers , Context Switching

What is Process Scheduling?

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

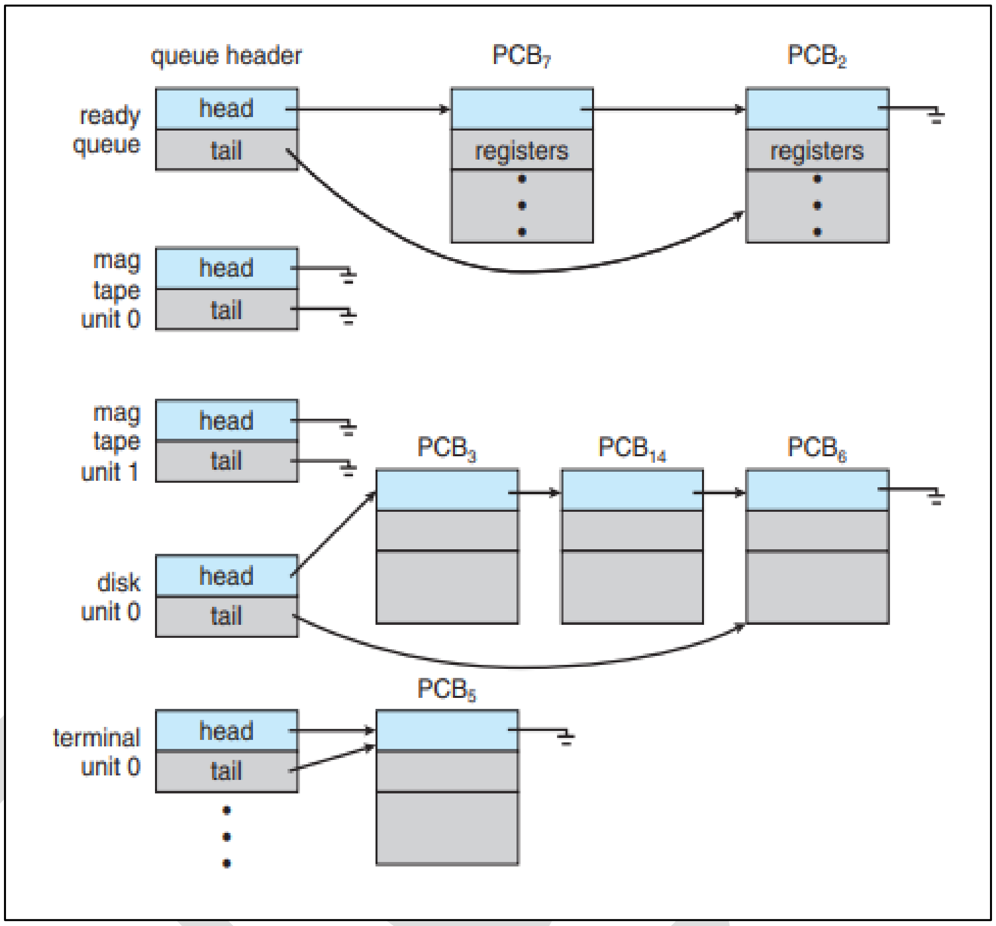


Fig. The ready queue and various I/O device queues.

Scheduling Queues:

As processes enter the system, they are put into **a job queue**, which consists of all processes in the system.

The processes that are residing in main memory and are ready and waiting to execute are kept on a list called the **ready queue.** This queue is generally stored as a linked list. A ready-queue header contains pointers to the first and final PCBs in the list. Each PCB includes a pointer field that points to the next PCB in the ready queue. The system also includes other queues. When a process is allocated the CPU, it executes for a while and eventually quits, is interrupted, or waits for the occurrence of a particular event, such as the completion of an I/O request. Suppose the process makes an I/O request to a shared device, such as a disk. Since there are many processes in the system, the disk may be busy with the 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

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 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 child process and wait for the child'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 deallocated.

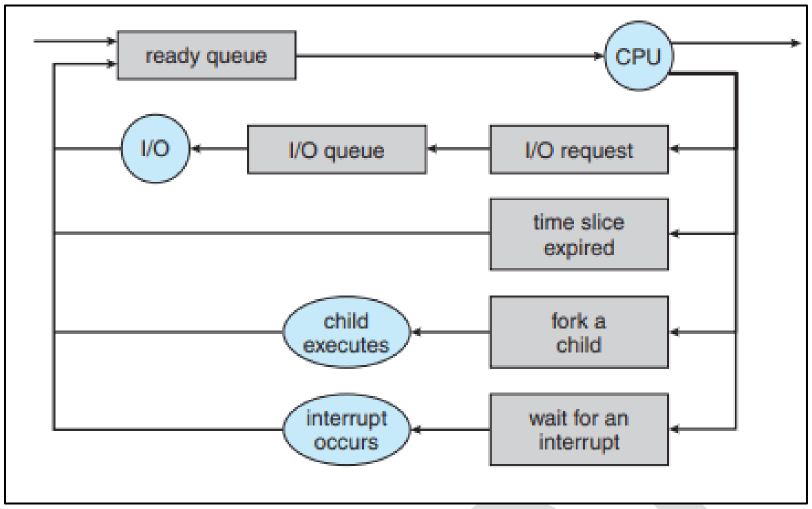


Fig. Queuing-diagram representation of process scheduling.

Schedulers :

Schedulers are special system software which handle 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

1. Long term scheduler

Long term scheduler is also known as job scheduler. It chooses the processes from the pool (secondary memory) and keeps them in the ready queue maintained in the primary memory.

Long Term scheduler mainly controls the degree of Multiprogramming. The purpose of long term scheduler is to choose a perfect mix of IO bound and CPU bound processes among the jobs present in the pool.

If the job scheduler chooses more IO bound processes then all of the jobs may reside in the blocked state all the time and the CPU will remain idle most of the time. This will reduce the degree of Multiprogramming. Therefore, the Job of long term scheduler is very critical and may affect the system for a very long time.

2. Short term scheduler

Short term scheduler is also known as CPU scheduler. It selects one of the Jobs from the ready queue and dispatch to the CPU for the execution.

A scheduling algorithm is used to select which job is going to be dispatched for the execution. The Job of the short term scheduler can be very critical in the sense that if it selects job whose CPU burst time is very high then all the jobs after that, will have to wait in the ready queue for a very long time.

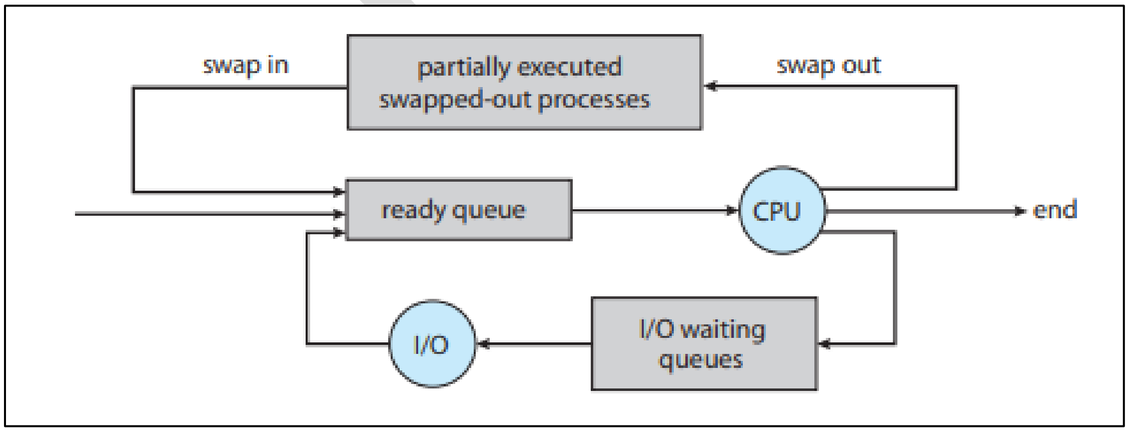
This problem is called starvation which may arise if the short term scheduler makes some mistakes while selecting the job.

3. Medium term scheduler

Medium term scheduler takes care of the swapped out processes. If the running state processes needs some IO time for the completion then there is a need to change its state from running to waiting.

Medium term scheduler is used for this purpose. It removes the process from the running state to make room for the other processes. Such processes are the swapped out processes and this procedure is called swapping. The medium term scheduler is responsible for suspending and resuming the processes.

It reduces the degree of multiprogramming. The swapping is necessary to have a perfect mix of processes in the ready queue.



Addition of medium-term scheduling to the queuing diagram

Context Switch:

The Context switching is a technique or method used by the operating system to switch a process from one state to another to execute its function using CPUs in the system. When switching performs in the system, it stores the old running process's status in the form of registers and assigns the CPU to a new process to execute its tasks. While a new process is running in the system, the previous process must wait in a ready queue. The execution of the old process starts at that point where another process stopped it. It defines the characteristics of a multitasking operating system in which multiple processes shared the same CPU to perform multiple tasks without the need for additional processors in the system.

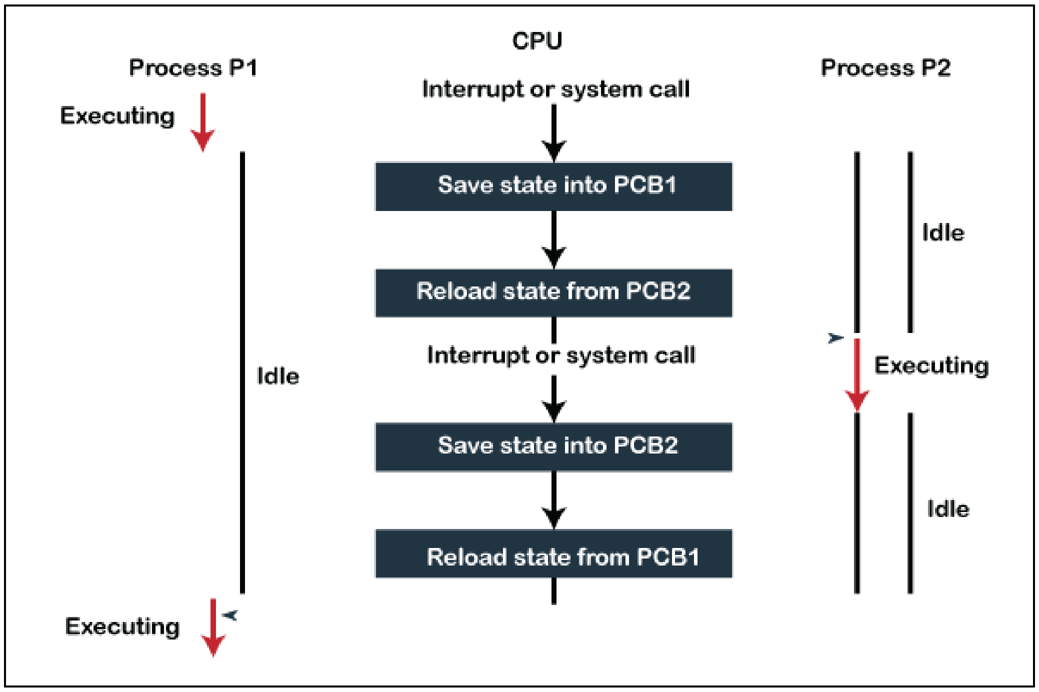
As we can see in the diagram, initially, the P1 process is running on the CPU to execute its task, and at the same time, another process, P2, is in the ready state. If an error or interruption has occurred or the process requires input/output, the P1 process switches its state from running to the waiting state. Before changing the state of the process P1, context switching saves the context of the process P1 in the form of registers and the program counter to the **PCB1**. After that, it loads the state of the P2 process from the ready state of the **PCB2** to the running state.

The following steps are taken when switching Process P1 to Process 2:

1. First, thes context switching needs to save the state of process P1 in the form of the program counter and the registers to the PCB (Program Counter Block), which is in the running state.
2. Now update PCB1 to process P1 and moves the process to the appropriate queue, such as the ready queue, I/O queue and waiting queue.
3. After that, another process gets into the running state, or we can select a new process from the ready state, which is to be executed, or the process has a high priority to execute its task.

4. Now, we have to update the PCB (Process Control Block) for the selected process P2. It includes switching the process state from ready to running state or from another state like blocked, exit, or suspend.
5. If the CPU already executes process P2, we need to get the status of process P2 to resume its execution at the same time point where the system interrupt occurs.

Similarly, process P2 is switched off from the CPU so that the process P1 can resume execution. P1 process is reloaded from PCB1 to the running state to resume its task at the same point. Otherwise, the information is lost, and when the process is executed again, it starts execution at the initial level.



3.3 Inter-Process Communication (IPC): Shared Memory System and Message Passing System

Inter-Process Communication (IPC):

Inter Process Communication is a type of mechanism usually provided by the operating system (or OS). The main aim or goal of this mechanism is to provide communications in between several processes. In short, the intercommunication allows a process letting another process know that some event has occurred.

A process can be of two types:

- **Independent process.**
- **Co-operating process.**

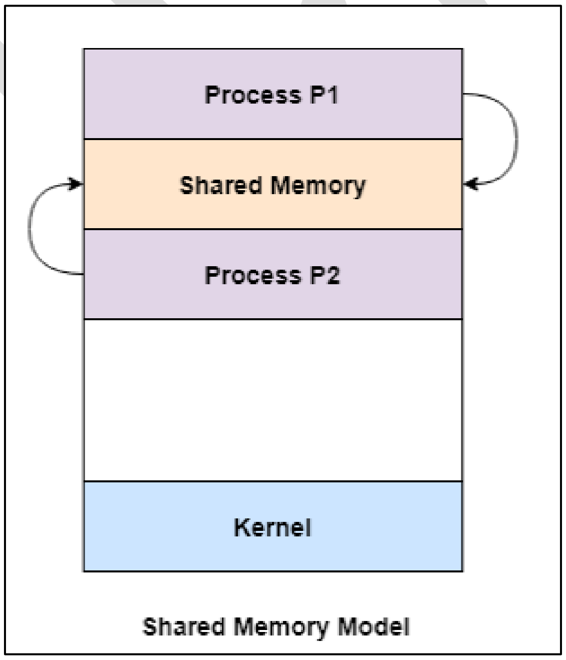
An **Independent process** is not affected by the execution of other processes while a **Co-operating process** can be affected by other executing processes.

Processes can communicate with each other using these two ways:

- **Shared Memory**
- **Message passing**

1. Shared Memory :

The shared memory in the shared memory model is the memory that can be simultaneously accessed by multiple processes. This is done so that the processes can communicate with each other. All POSIX systems, as well as Windows operating systems use shared memory.
A diagram that illustrates the shared memory model of process communication is given as follows −



Working

In the Shared Memory system, the cooperating processes communicate, to exchange the data with each other. Because of this, the cooperating processes establish a shared region in their memory. The processes share data by reading and writing the data in the shared segment of the processes.

Let the two cooperating processes P1 and P2. Both the processes P1 and P2, have their different address spaces. Now let us assume, P1 wants to share some data with P2.

So, P1 and P2 will have to perform the following steps −

Step 1 − Process P1 has some data to share with process P2. First P1 takes initiative and establishes a shared memory region in its own address space and stores the data or information to be shared in its shared memory region.

Step 2 − Now, P2 requires the information stored in the shared segment of P1. So, process P2 needs to attach itself to the shared address space of P1. Now, P2 can read out the data from there.

Step 3 − The two processes can exchange information by reading and writing data in the shared segment of the process.

Advantages :

The advantages of Shared Memory are as follows −

- Shared memory is a faster inter process communication system.
- It allows cooperating processes to access the same pieces of data concurrently.
- It speeds up the computation power of the system and divides long tasks into smaller sub-tasks and can be executed in parallel.
- Modularity is achieved in a shared memory system.
- Users can perform multiple tasks at a time.

Disadvantage of Shared Memory Model :

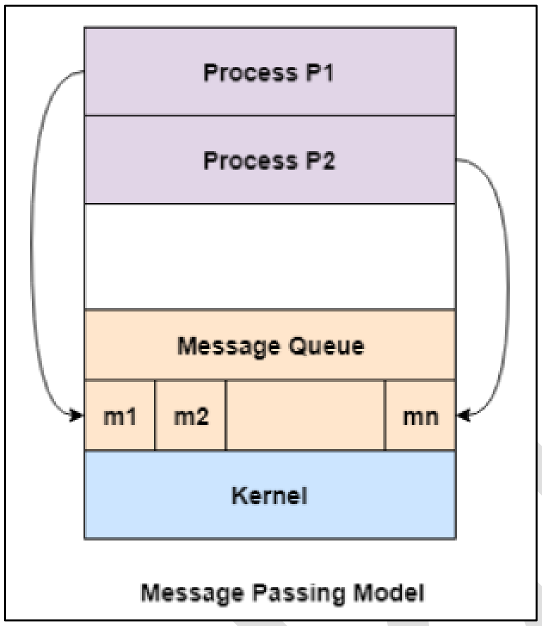
Some of the disadvantages of shared memory model are as follows −

- All the processes that use the shared memory model need to make sure that they are not writing to the same memory location.
- Shared memory model may create problems such as synchronization and memory protection that need to be addressed.

2. Message Passing :

Message passing model allows multiple processes to read and write data to the message queue without being connected to each other. Messages are stored on the queue until their recipient retrieves them. Message queues are quite useful for interprocess communication and are used by most operating systems.

A diagram that demonstrates message passing model of process communication is given as follows −



For example − chat programs on World Wide Web.

Now let us discuss the message passing step by step.

Step 1 − Message passing provides two operations which are as follows −

- Send message
- Receive message

Messages sent by a process can be either fixed or variable size.

Step 2 − For fixed size messages the system level implementation is straight forward. It makes the task of programming more difficult.

Step 3 − The variable sized messages require a more system level implementation but the programming task becomes simpler.

Step 4 − If process P1 and P2 want to communicate they need to send a message to and receive a message from each other that means here a communication link exists between them.

Step 5 − Methods for logically implementing a link and the send() and receive() operations.

Advantages of Message Passing Model :

Some of the advantages of message passing model are given as follows −

- The message passing model is much easier to implement than the shared memory model.
- It is easier to build parallel hardware using message passing model as it is quite tolerant of higher communication latencies.

Disadvantage of Message Passing Model :

The message passing model has slower communication than the shared memory model because the connection setup takes time.

The main differences between the Shared Memory and the Message Passing are as follows:

Shared Memory	Message Passing
It is mainly used for data communication.	It is mainly used for communication.
It offers a maximum speed of computation because communication is completed via the shared memory, so the system calls are only required to establish the shared memory.	It takes a huge time because it is performed via the kernel (system calls).
The code for reading and writing the data from the shared memory should be written explicitly by the developer.	No such code is required in this case because the message passing feature offers a method for communication and synchronization of activities executed by the communicating processes.
It is used to communicate between the single processor and multiprocessor systems in which the processes to be communicated are on the same machine and share the same address space.	It is most commonly utilized in a distributed setting when communicating processes are spread over multiple devices linked by a network.
It is a faster communication strategy than the message passing.	It is a relatively slower communication strategy than the shared memory.

Make sure that processes in shared memory aren't writing to the same address simultaneously.	It is useful for sharing little quantities of data without causing disputes.

3.4 Threads - Benefits, users and kernel threads, Multithreading Models - Manyto One, One to One, Many to Many.

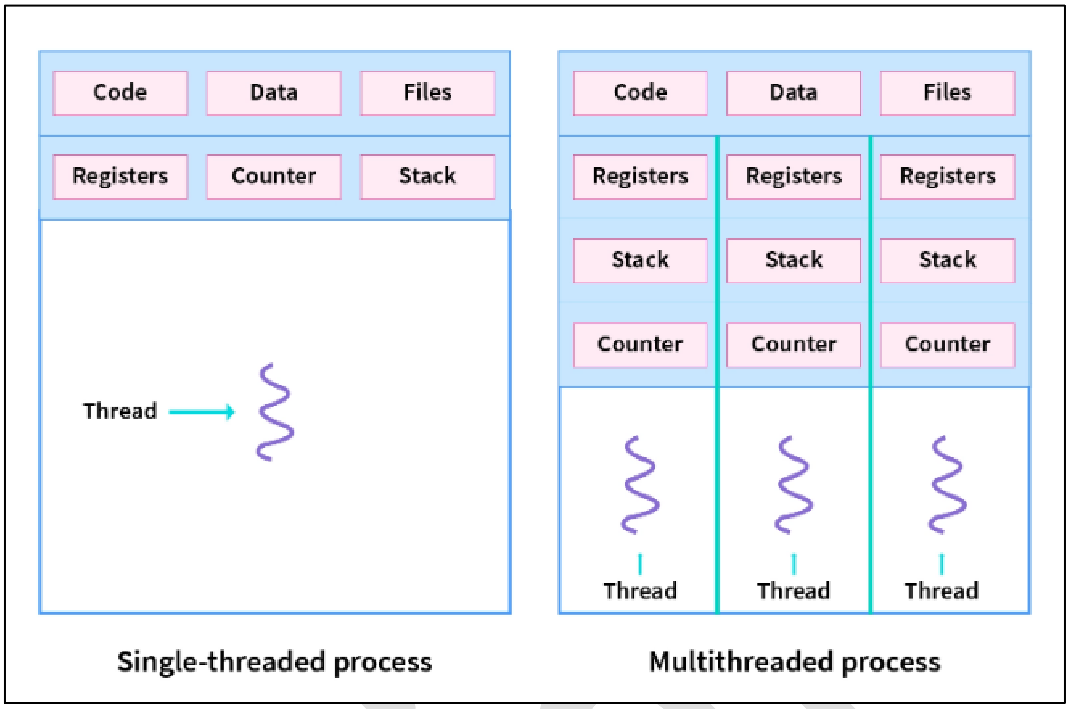
What is Thread :

Thread is a sequential flow of tasks within a process. Threads in OS can be of the same or different types. Threads are used to increase the performance of the applications.

Each thread has its own program counter, stack, and set of registers. But the threads of a single process might share the same code and data/file. **Threads are also termed as lightweight processes as they share common resources**.

Eg: While playing a movie on a device the audio and video are controlled by different threads in the background.

The process can be split down into so many threads. **For example**, in a browser, many tabs can be viewed as threads. MS Word uses many threads - formatting text from one thread, processing input from another thread, etc.



Need of Thread:

- It takes far less time to create a new thread in an existing process than to create a new process.
- Threads can share the common data, they do not need to use Inter- Process communication.
- Context switching is faster when working with threads.
- It takes less time to terminate a thread than a process.

Components of Threads

Any thread has the following components.

1. Program counter
2. Register set
3. Stack space

• Types of Threads

In the operating system, there are two types of threads.

1. User-level thread.
2. Kernel level thread.

1. User-Level Thread :

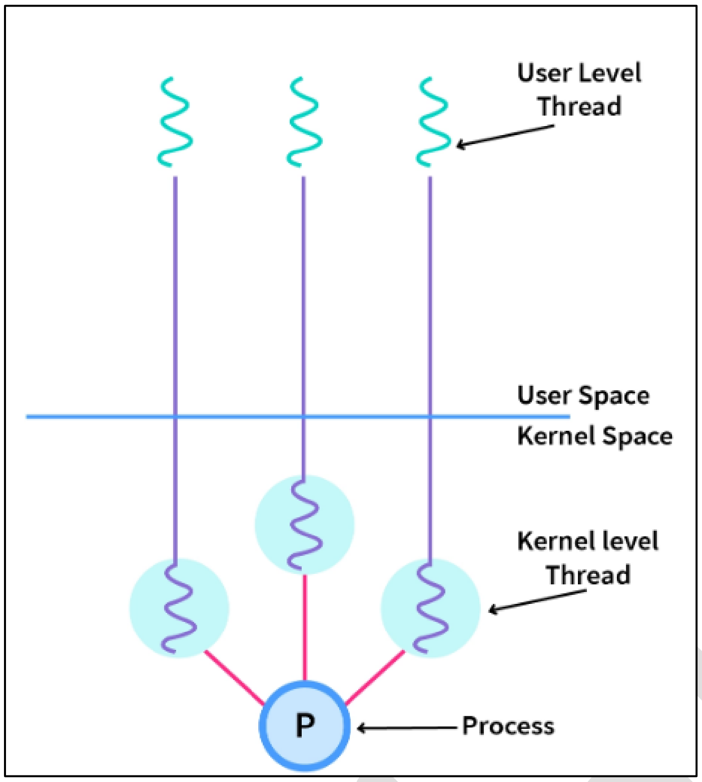
User-level threads are implemented and managed by the user and the kernel is not aware of it.

- User-level threads are implemented using user-level libraries and the OS does not recognize these threads.
- User-level thread is faster to create and manage compared to kernel-level thread.
- Context switching in user-level threads is faster.
- If one user-level thread performs a blocking operation then the entire process gets blocked. Eg: POSIX threads, Java threads, etc.

2. Kernel-Level Thread :

Kernel level threads are implemented and managed by the OS.

- Kernel level threads are implemented using system calls and Kernel level threads are recognized by the OS.
- Kernel-level threads are slower to create and manage compared to user-level threads.
- Context switching in a kernel-level thread is slower.
- Even if one kernel-level thread performs a blocking operation, it does not affect other threads. Eg: Window Solaris.



❖ Benefits of Threads :

- **Enhanced throughput of the system:** When the process is split into many threads, and each thread is treated as a job, the number of jobs done in the unit time increases. That is why the throughput of the system also increases.
- **Effective Utilization of Multiprocessor system:** When you have more than one thread in one process, you can schedule more than one thread in more than one processor.
- **Faster context switch:** The context switching period between threads is less than the process context switching. The process context switch means more overhead for the CPU.
- **Responsiveness:** When the process is split into several threads, and when a thread completes its execution, that process can be responded to as soon as possible.
- **Communication:** Multiple-thread communication is simple because the threads share the same address space, while in process, we adopt just a few exclusive communication strategies for communication between two processes.

- **Resource sharing:** Resources can be shared between all threads within a process, such as code, data, and files. Note: The stack and register cannot be shared between threads. There is a stack and register for each thread.

❖ Multithreading Models :

Multithreading

Multithreading is a phenomenon of executing multiple threads at the same time. To understand the concept of multithreading, you must understand what is a **thread** and a **process**.

A process is a program in execution. A process can further be divided into sub-processes known as **threads**.

Example: Playing a video and downloading it at the same time is an example of multithreading**.**

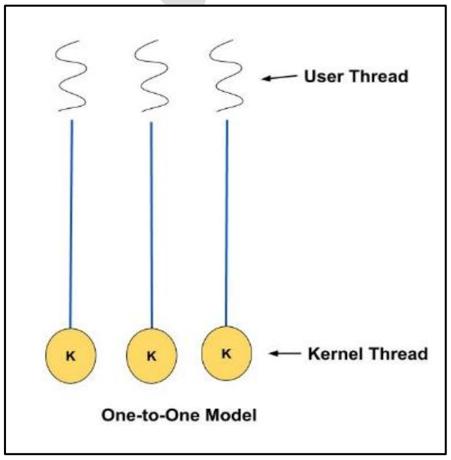
There are three common ways/Models

1. **One-to-One Model**
2. **Many-to-Many Model**
3. **Many-to-One Model**

1. One – to - one Model :

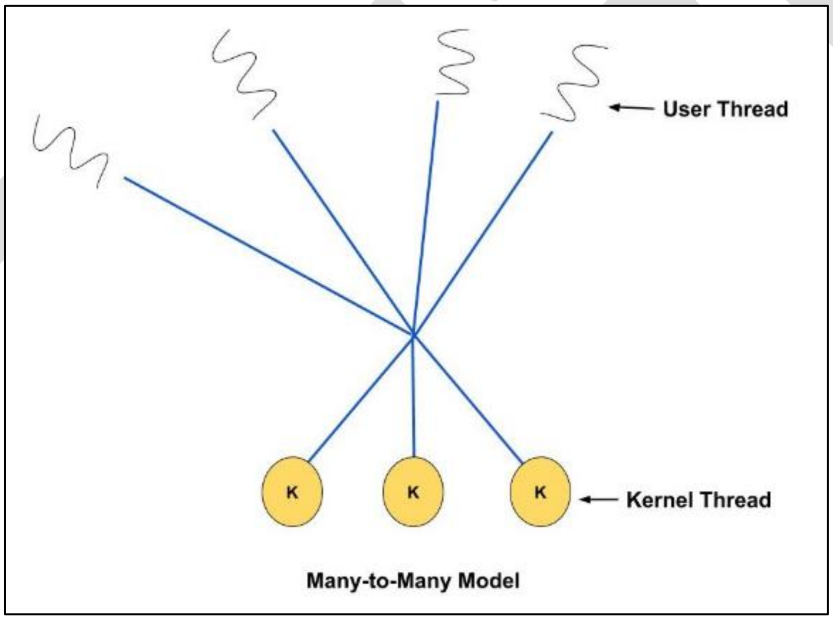
- In this model, one to one relationship between kernel and user thread. In this model multiple thread can run on multiple processor. Problem with this model is that creating a user thread requires the corresponding kernel thread.

- As each user thread is connected to different kernel , if any user thread makes a blocking system call, the other user threads won't be blocked.



2. Many- to - Many Model :

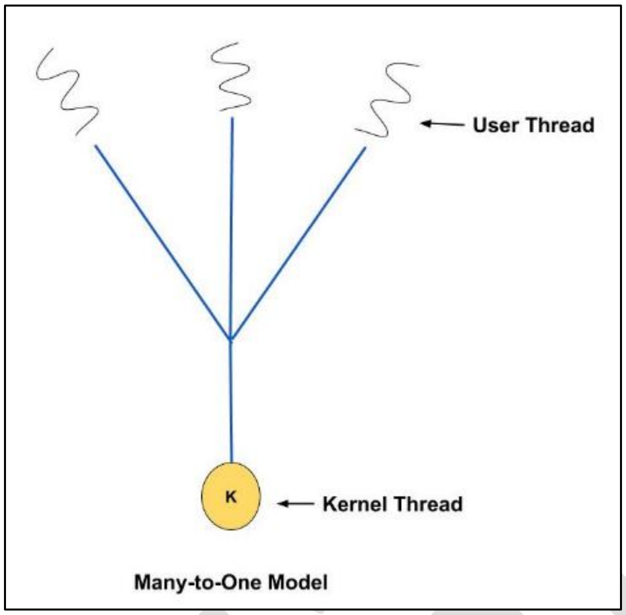
- In this type of model, there are several user-level threads and several kernel-level threads. The number of kernel threads created depends upon a particular application.
- The developer can create as many threads at both levels but may not be the same. The many to many model is a compromise between the other two models. In this model, if any thread makes a blocking system call, the kernel can schedule another thread for execution.
- Also, with the introduction of multiple threads, complexity is not present as in the previous models. Though this model allows the creation of multiple kernel threads, true concurrency cannot be achieved by this model. This is because the kernel can schedule only one process at a time.



3. Many –to - One Model :

- The many to one model maps many user levels threads to one kernel thread. This type of relationship facilitates an effective context-switching environment, easily implemented even on the simple kernel with no thread support.
- The disadvantage of this model is that since there is only one kernel-level thread schedule at any given time, this model cannot take advantage of the hardware

acceleration offered by multithreaded processes or multi-processor systems. In this, all the thread management is done in the userspace. If blocking comes, this model blocks the whole system.



3.5 Execute Process Commands : ps, wait, sleep exit, Kill

A **process** is nothing but a program in execution. It's a running instance of a program. Any command that you execute starts a process.

Types of Processes in Linux

Processes can be of two types:

- **Foreground Processes**: They run on the screen and need input from the user. For example Office Programs
- **Background Processes**: They run in the background and usually do not need user input. For example Antivirus.

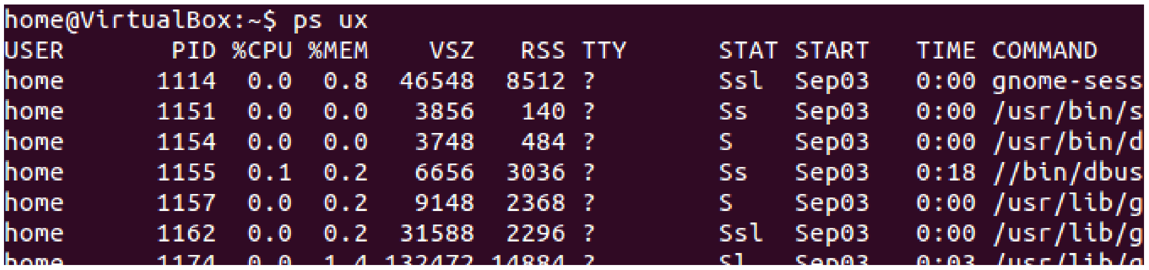
1. ps command:

This command stands for 'Process Status'. It is similar to the "Task Manager" that pop-ups in a Windows Machine when we use Cntrl+Alt+Del.

This command is similar to 'top' command but the information displayed is different.

To check all the processes running under a user, use the command –

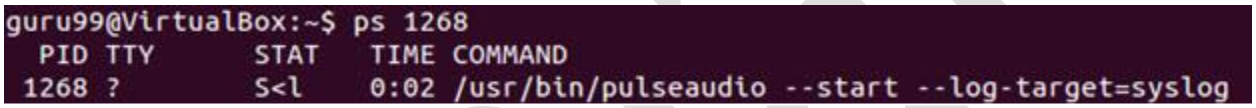
ps ux



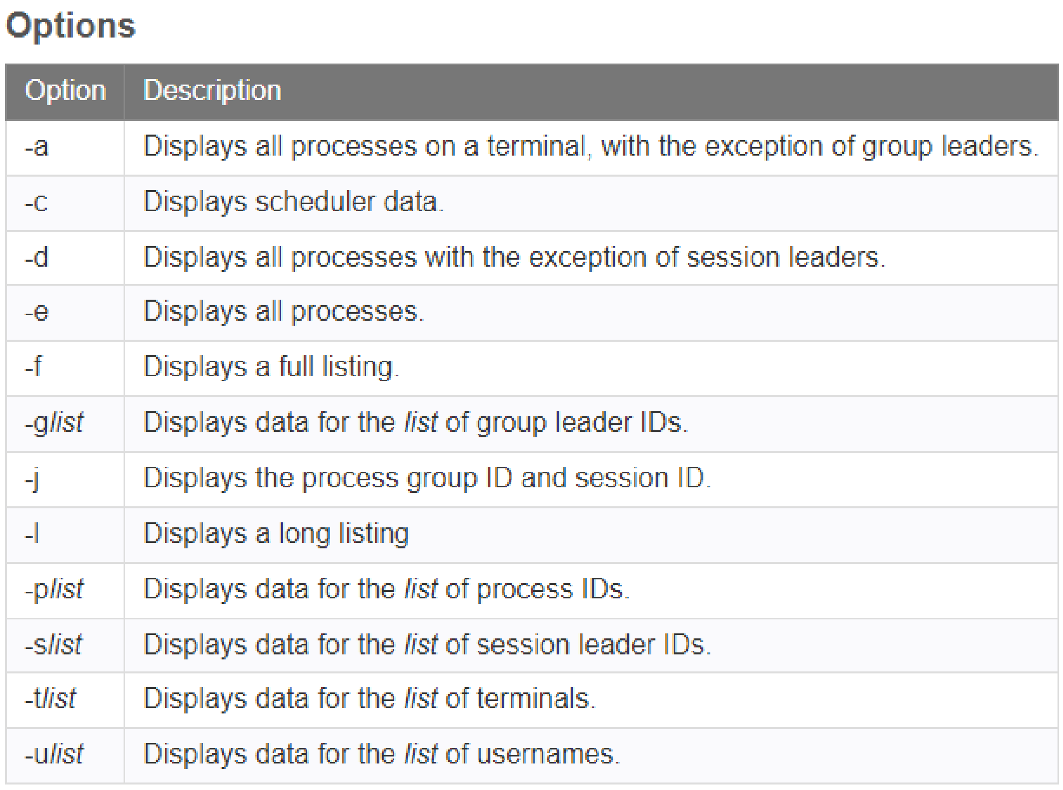
```
home@VirtualBox:~$ ps ux
USER       PID %CPU %MEM    VSZ   RSS TTY      STAT START   TIME COMMAND
home      1114  0.0  0.8  46548  8512 ?        Ssl  Sep03   0:00 gnome-sess
home      1151  0.0  0.0   3856   140 ?        Ss   Sep03   0:00 /usr/bin/s
home      1154  0.0  0.0   3748   484 ?        S    Sep03   0:00 /usr/bin/d
home      1155  0.1  0.2   6656  3036 ?        Ss   Sep03   0:18 //bin/dbus
home      1157  0.0  0.2   9148  2368 ?        S    Sep03   0:00 /usr/lib/g
home      1162  0.0  0.2  31588  2296 ?        Ssl  Sep03   0:00 /usr/lib/g
home      1174  0.0  1.4 132472 14884 ?        Sl   Sep03   0:03 /usr/lib/g
```

We can also check the process status of a single process, use the syntax –

ps PID



```
guru99@VirtualBox:~$ ps 1268
  PID TTY      STAT   TIME COMMAND
 1268 ?        S<l    0:02 /usr/bin/pulseaudio --start --log-target=syslog
```



Options

Option	Description
-a	Displays all processes on a terminal, with the exception of group leaders.
-c	Displays scheduler data.
-d	Displays all processes with the exception of session leaders.
-e	Displays all processes.
-f	Displays a full listing.
-g*list*	Displays data for the *list* of group leader IDs.
-j	Displays the process group ID and session ID.
-l	Displays a long listing
-p*list*	Displays data for the *list* of process IDs.
-s*list*	Displays data for the *list* of session leader IDs.
-t*list*	Displays data for the *list* of terminals.
-u*list*	Displays data for the *list* of usernames.

2. Wait Command :

The **wait** command captures the exit status of a background process after waiting for it to complete.
wait command will suspend execution of the calling thread until one of its children terminate. It will return the exit status of that command

Syntax:

wait [ID]

Here, ID is a **PID** (Process Identifier) which is unique for each running process. To find the process ID of a process you can use the command:
pidof [process_name]

3. Sleep Command :

This command is used in UNIX operating system to suspend execution of the system for the specified time limit mentioned in it as parameter.

The general format of sleep command is

sleep no_of_seconds

The sleep suspends the execution of the shell in UNIX operating system for the no_of seconds specified.

sleep 5

The above suspend the execution of the shell in UNIX operating system for 5 seconds specified.

4. Exit Command :

Linux exit command is used to exit from the current shell. It takes a parameter as a number and exits the shell with a return of status number. If we did not provide any parameter, it would return the status of the last executed command. The exit command closes a script and exits the shell.

If we have more than one shell tab, the exit command will close the tab where it is executed. This is a built-in command, and we cannot find a dedicated manual page for this.

Syntax: exit

5. Kill Command :

- *kill* command in Linux (located in /bin/kill), is a built-in command which is used to terminate processes manually. *kill* command sends a signal to a process which terminates the process.
- If the user doesn't specify any signal which is to be sent along with kill command then default *TERM* signal is sent that terminates the process.

Syntax:

```
kill -SIGNAL PID
```