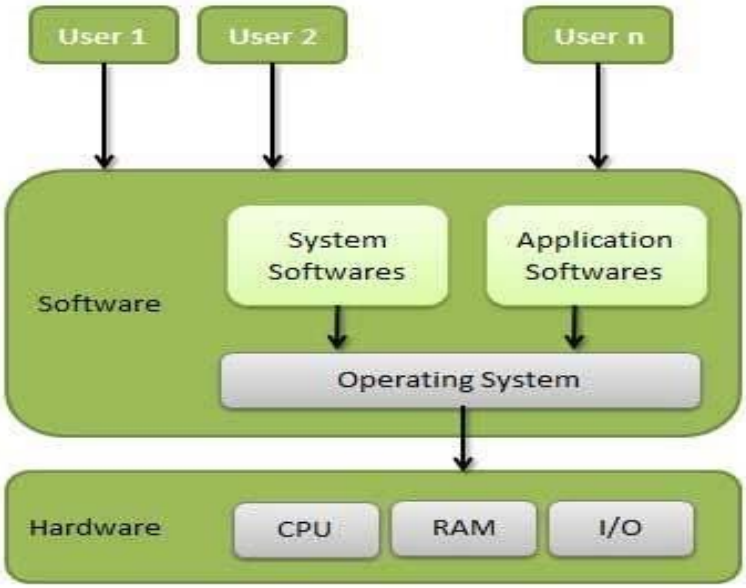
UNIT-1

An **Operating System** (OS) is an interface between a computer user and computer hardware. An operating system is a software which performs all the basic tasks like file management, memory management, process management, handling input and output, and controlling peripheral devices such as disk drives and printers.

Some popular Operating Systems include Linux Operating System, Windows Operating System, VMS, OS/400, AIX, z/OS, etc.

Definition An operating system is a program that acts as an interface between the user and the computer hardware and controls the execution of all kinds of programs.



Functions of an operating System.

1.Memory Management

Memory management refers to management of Primary Memory or Main Memory. Main memory is a large array of words or bytes where each word or byte has its own address.

Main memory provides a fast storage that can be accessed directly by the CPU. For a program to be executed, it must in the main memory. An Operating System does the following activities for memory management −

- Keeps tracks of primary memory, i.e., what part of it are in use by whom, what part are not in use.
- In multiprogramming, the OS decides which process will get memory when and how much.

- Allocates the memory when a process requests it to do so.
- De-allocates the memory when a process no longer needs it or has been terminated.

2 Processor Management

In multiprogramming environment, the OS decides which process gets the processor when and for how much time. This function is called **process scheduling**. An Operating System does the following activities for processor management −

- Keeps tracks of processor and status of process. The program responsible for this task is known as **traffic controller**.
- Allocates the processor (CPU) to a process.
- De-allocates processor when a process is no longer required.

3 Device Management

An Operating System manages device communication via their respective drivers. It does the following activities for device management −

- Keeps tracks of all devices. Program responsible for this task is known as the **I/O controller**.
- Decides which process gets the device when and for how much time.
- Allocates the device in the efficient way.
- De-allocates devices.

4 File Management

A file system is normally organized into directories for easy navigation and usage. These directories may contain files and other directions.

An Operating System does the following activities for file management −

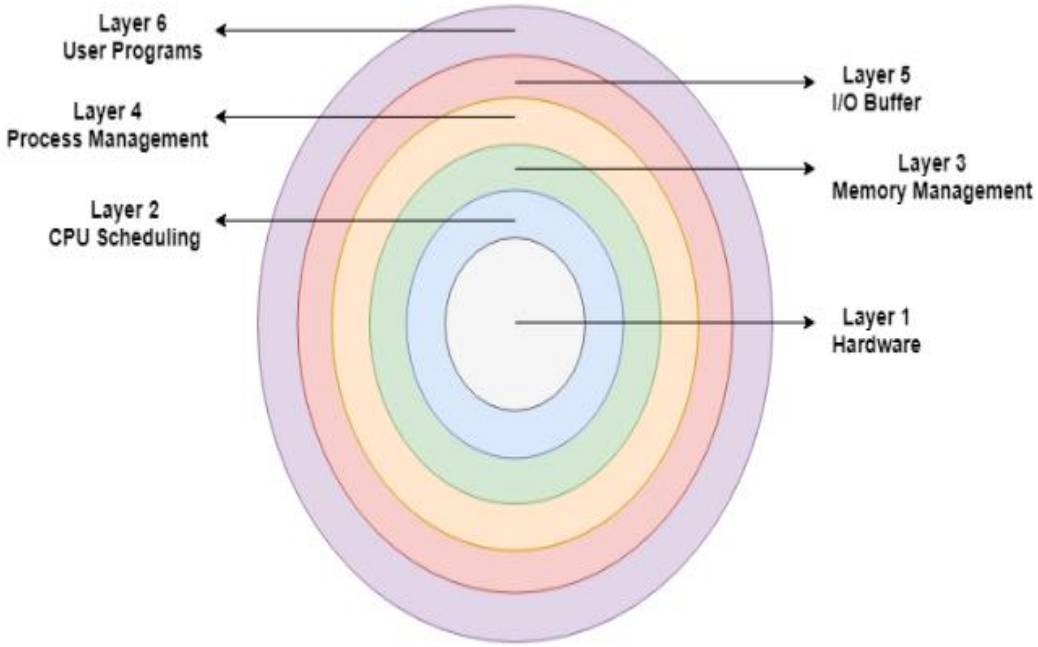
- Keeps track of information, location, uses, status etc. The collective facilities are often known as **file system**.
- Decides who gets the resources.
- Allocates the resources.
- De-allocates the resources.
- The operating system is split into various layers In the layered operating system and each of the layers have different functionalities. This type of operating system was created as an improvement over the early monolithic systems.

Layering in Operating System

- Layering provides a distinct advantage in an operating system. All the layers can be defined separately and interact with each other as required. Also, it is easier to create, maintain and update the system if it is done in the form of layers. Change in one layer specification does not affect the rest of the layers.
- Each of the layers in the operating system can only interact with the layers that are above and below it. The lowest layer handles the hardware and the uppermost layer deals with the user applications.

Layers in Layered Operating System

- There are six layers in the layered operating system. A diagram demonstrating these layers is as follows:



LAYERED OPERATING SYSTEM

Details about the six layers are:

Hardware

- This layer interacts with the system hardware and coordinates with all the peripheral devices used such as printer, mouse, keyboard, scanner etc. The hardware layer is the lowest layer in the layered operating system architecture.

CPU Scheduling

- This layer deals with scheduling the processes for the CPU. There are many scheduling queues that are used to handle processes. When the processes enter the system, they are

put into the job queue. The processes that are ready to execute in the main memory are kept in the ready queue.

Memory Management

- Memory management deals with memory and the moving of processes from disk to primary memory for execution and back again. This is handled by the third layer of the operating system.

Process Management

- This layer is responsible for managing the processes i.e assigning the processor to a process at a time. This is known as process scheduling. The different algorithms used for process scheduling are FCFS (first come first served), SJF (shortest job first), priority scheduling, round-robin scheduling etc.

I/O Buffer

- I/O devices are very important in the computer systems. They provide users with the means of interacting with the system. This layer handles the buffers for the I/O devices and makes sure that they work correctly.

User Programs

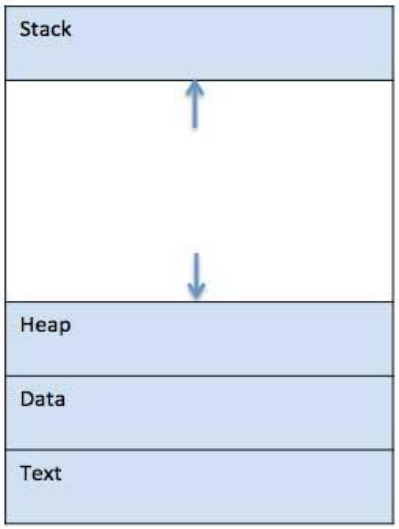
- This is the highest layer in the layered operating system. This layer deals with the many user programs and applications that run in an operating system such as word processors, games, browsers etc.

Process

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

Process memory is divided into four sections for efficient working :

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



components of stack

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.

1. **Start**

This is the initial state when a process is first started/created.

2. **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.

3. **Running**

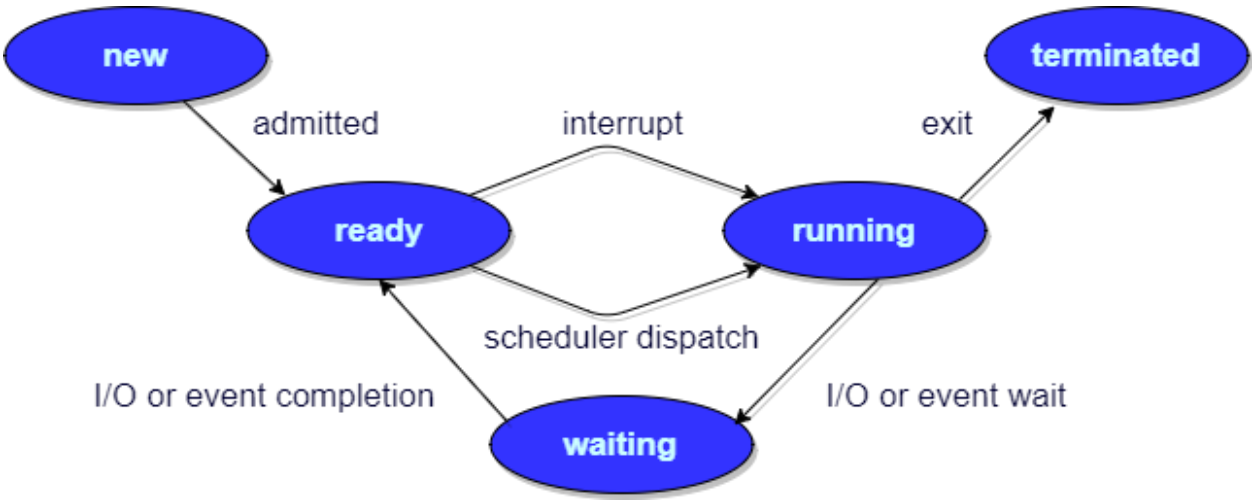
Once the process has been assigned to a processor by the OS scheduler, the process state is set to running and the processor executes its instructions.

4. **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.

5.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.



Process state diagram

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 in the table –

1. **Process State**

The current state of the process i.e., whether it is ready, running, waiting, or whatever.

2. **Process privileges**

This is required to allow/disallow access to system resources.

3. **Process ID**

Unique identification for each of the process in the operating system.

4. **Pointer**

A pointer to parent process.

5.Program Counter

Program Counter is a pointer to the address of the next instruction to be executed for this process.

6. **CPU registers**

Various CPU registers where process need to be stored for execution for running state.

7. **CPU Scheduling Information**

Process priority and other scheduling information which is required to schedule the process.

8. **Memory management information**

This includes the information of page table, memory limits, Segment table depending on memory used by the operating system.

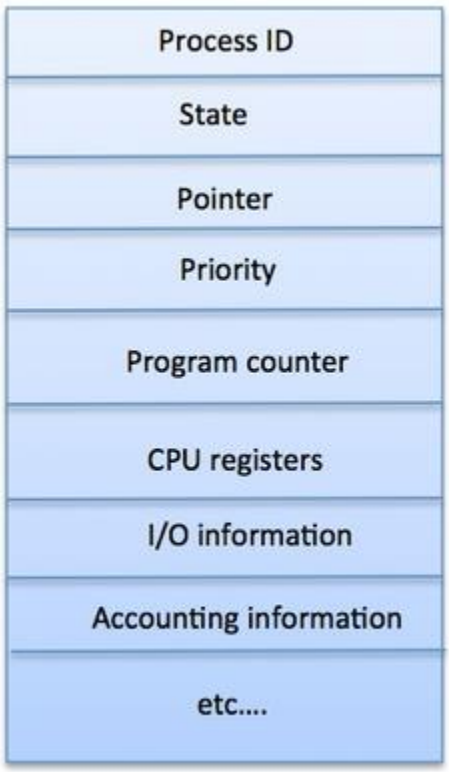
9. **Accounting information**

This includes the amount of CPU used for process execution, time limits, execution ID etc.

10. **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. Here is a simplified diagram of a PCB −



The PCB is maintained for a process throughout its lifetime, and is deleted once the process terminates.

Threads

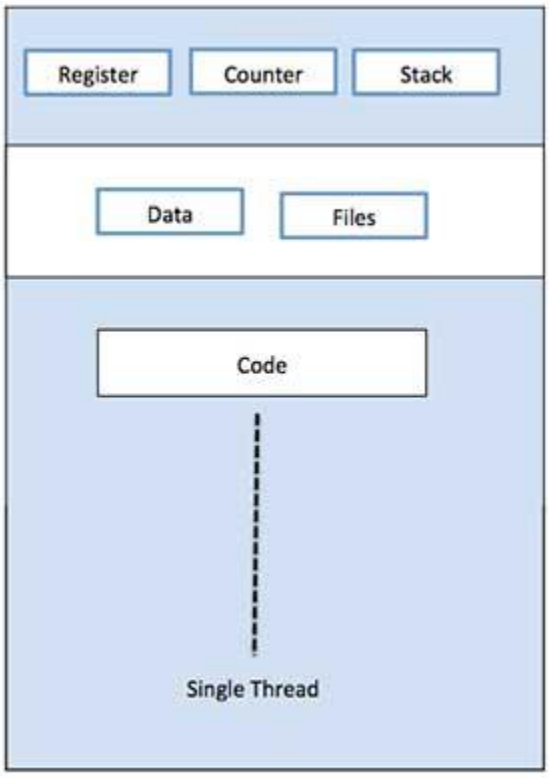
A thread is a flow of execution through the process code, with its own program counter that keeps track of which instruction to execute next, system registers which hold its current working variables, and a stack which contains the execution history.

A thread shares with its peer threads few information like code segment, data segment and open files. When one thread alters a code segment memory item, all other threads see that.

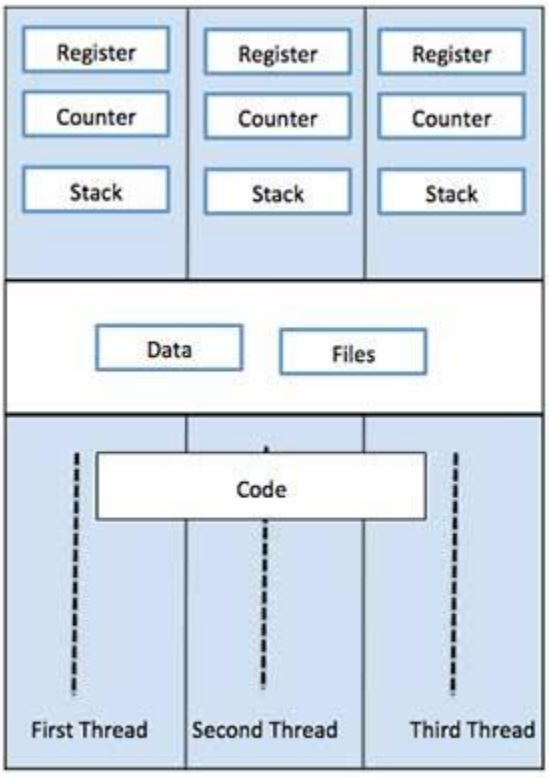
A thread is also called a **lightweight process**. Threads provide a way to improve application performance through parallelism. Threads represent a software approach to improving

performance of operating system by reducing the overhead thread is equivalent to a classical process.

Each thread belongs to exactly one process and no thread can exist outside a process. Each thread represents a separate flow of control. Threads have been successfully used in implementing network servers and web server. They also provide a suitable foundation for parallel execution of applications on shared memory multiprocessors. The following figure shows the working of a single-threaded and a multithreaded process.



Single Process P with single thread



Single Process P with three threads

Difference between Process and Thread

S.N.	Process	Thread
1	Process is heavy weight or resource intensive.	Thread is light weight, taking lesser resources than a process.
2	Process switching needs interaction with operating system.	Thread switching does not need to interact with operating system.

3	In multiple processing environments, each process executes the same code but has its own memory and file resources.	All threads can share same set of open files, child processes.
4	If one process is blocked, then no other process can execute until the first process is unblocked.	While one thread is blocked and waiting, a second thread in the same task can run.
5	Multiple processes without using threads use more resources.	Multiple threaded processes use fewer resources.
6	In multiple processes each process operates independently of the others.	One thread can read, write or change another thread's data.

Advantages of Thread

- Threads minimize the context switching time.
- Use of threads provides concurrency within a process.
- Efficient communication.
- It is more economical to create and context switch threads.
- Threads allow utilization of multiprocessor architectures to a greater scale and efficiency.

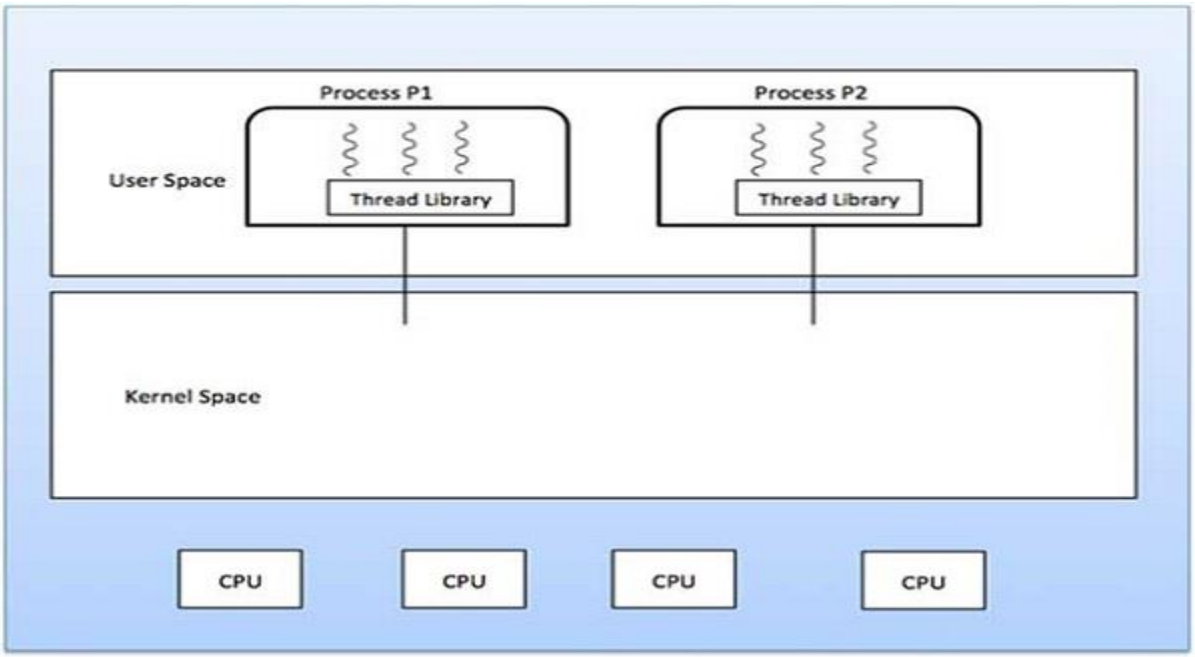
Types of Thread

Threads are implemented in following two ways −

- **User Level Threads** − User managed threads.
- **Kernel Level Threads** − Operating System managed threads acting on kernel, an operating system core.

User Level Threads

In this case, the thread management kernel is not aware of the existence of threads. The thread library contains code for creating and destroying threads, for passing message and data between threads, for scheduling thread execution and for saving and restoring thread contexts. The application starts with a single thread.



Advantages

- Thread switching does not require Kernel mode privileges.
- User level thread can run on any operating system.
- Scheduling can be application specific in the user level thread.
- User level threads are fast to create and manage.

Disadvantages

- In a typical operating system, most system calls are blocking.
- Multithreaded application cannot take advantage of multiprocessing.

Kernel Level Threads

In this case, thread management is done by the Kernel. There is no thread management code in the application area. Kernel threads are supported directly by the operating system. Any application can be programmed to be multithreaded. All of the threads within an application are supported within a single process.

The Kernel maintains context information for the process as a whole and for individuals threads within the process. Scheduling by the Kernel is done on a thread basis. The Kernel performs thread creation, scheduling and management in Kernel space. Kernel threads are generally slower to create and manage than the user threads.

Advantages

- Kernel can simultaneously schedule multiple threads from the same process on multiple processes.
- If one thread in a process is blocked, the Kernel can schedule another thread of the same process.
- Kernel routines themselves can be multithreaded.

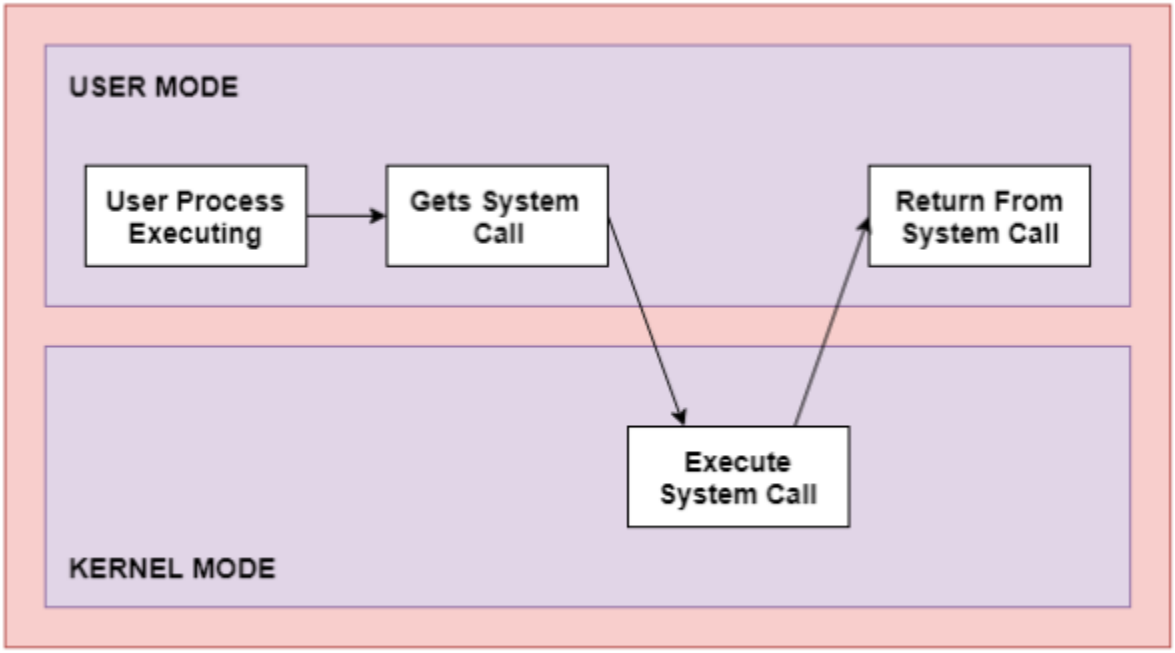
Disadvantages

- Kernel threads are generally slower to create and manage than the user threads.
- Transfer of control from one thread to another within the same process requires a mode switch to the Kernel.

System calls

The interface between a process and an operating system is provided by system calls. In general, system calls are available as assembly language instructions. They are also included in the manuals used by the assembly level programmers. System calls are usually made when a process in user mode requires access to a resource. Then it requests the kernel to provide the resource via a system call.

A figure representing the execution of the system call is given as follows:



As can be seen from this diagram, the processes execute normally in the user mode until a system call interrupts this. Then the system call is executed on a priority basis in the kernel mode. After the execution of the system call, the control returns to the user mode and execution of user processes can be resumed.

In general, system calls are required in the following situations:

- If a file system requires the creation or deletion of files. Reading and writing from files also require a system call.
- Creation and management of new processes.
- Network connections also require system calls. This includes sending and receiving packets.
- Access to a hardware devices such as a printer, scanner etc. requires a system call.

Types of System Calls

There are mainly five types of system calls. These are explained in detail as follows:

Process Control

These system calls deal with processes such as process creation, process termination etc.

File Management

These system calls are responsible for file manipulation such as creating a file, reading a file, writing into a file etc.

Device Management

These system calls are responsible for device manipulation such as reading from device buffers, writing into device buffers etc.

Information Maintenance

These system calls handle information and its transfer between the operating system and the user program.

Communication

These system calls are useful for interprocess communication. They also deal with creating and deleting a communication connection.

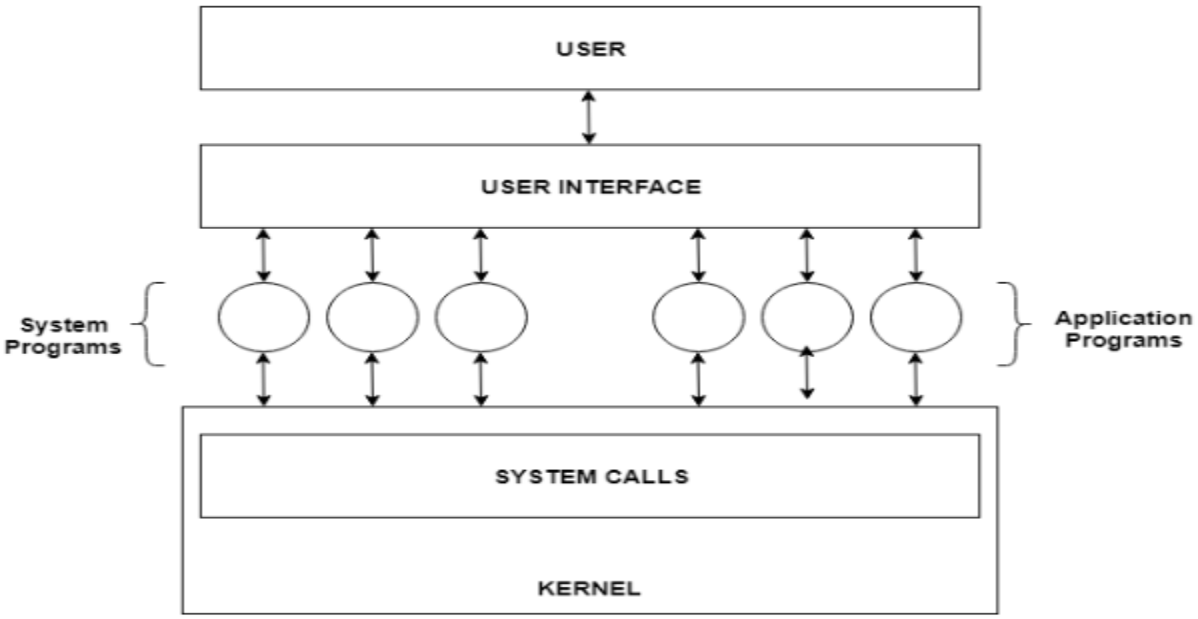
System Programs

System programs provide an environment where programs can be developed and executed. In the simplest sense, system programs also provide a bridge between the user interface and system calls. In reality, they are much more complex. For example, a compiler is a complex system program.

The system program serves as a part of the operating system. It traditionally lies between the user interface and the system calls. The user view of the system is actually defined by system

programs and not system calls because that is what they interact with and system programs are closer to the user interface.

An image that describes system programs in the operating system hierarchy is as follows:



In the above image, system programs as well as application programs form a bridge between the user interface and the system calls. So, from the user view the operating system observed is actually the system programs and not the system calls.

Types of System Programs

System programs can be divided into seven parts. These are given as follows:

Status Information

The status information system programs provide required data on the current or past status of the system. This may include the system date, system time, available memory in system, disk space, logged in users etc.

Communications

These system programs are needed for system communications such as web browsers. Web browsers allow systems to communicate and access information from the network as required.

File Manipulation

These system programs are used to manipulate system files. This can be done using various commands like create, delete, copy, rename, print etc. These commands can create files, delete files, copy the contents of one file into another, rename files, print them etc.

Program Loading and Execution

The system programs that deal with program loading and execution make sure that programs can be loaded into memory and executed correctly. Loaders and Linkers are a prime example of this type of system programs.

File Modification

System programs that are used for file modification basically change the data in the file or modify it in some other way. Text editors are a big example of file modification system programs.

Application Programs

Application programs can perform a wide range of services as per the needs of the users. These include programs for database systems, word processors, plotting tools, spreadsheets, games, scientific applications etc.

Programming Language Support

These system programs provide additional support features for different programming languages. Some examples of these are compilers, debuggers etc. These compile a program and make sure it is error free respectively.

Process scheduling

The act of determining which process is in the **ready** state, and should be moved to the **running** state is known as **Process Scheduling**.

The prime aim of the process scheduling system is to keep the CPU busy all the time and to deliver minimum response time for all programs. For achieving this, the scheduler must apply appropriate rules for swapping processes IN and OUT of CPU.

Scheduling fell into one of the two general categories:

- **Non Pre-emptive Scheduling:** When the currently executing process gives up the CPU voluntarily.
- **Pre-emptive Scheduling:** When the operating system decides to favour another process, pre-empting the currently executing process.

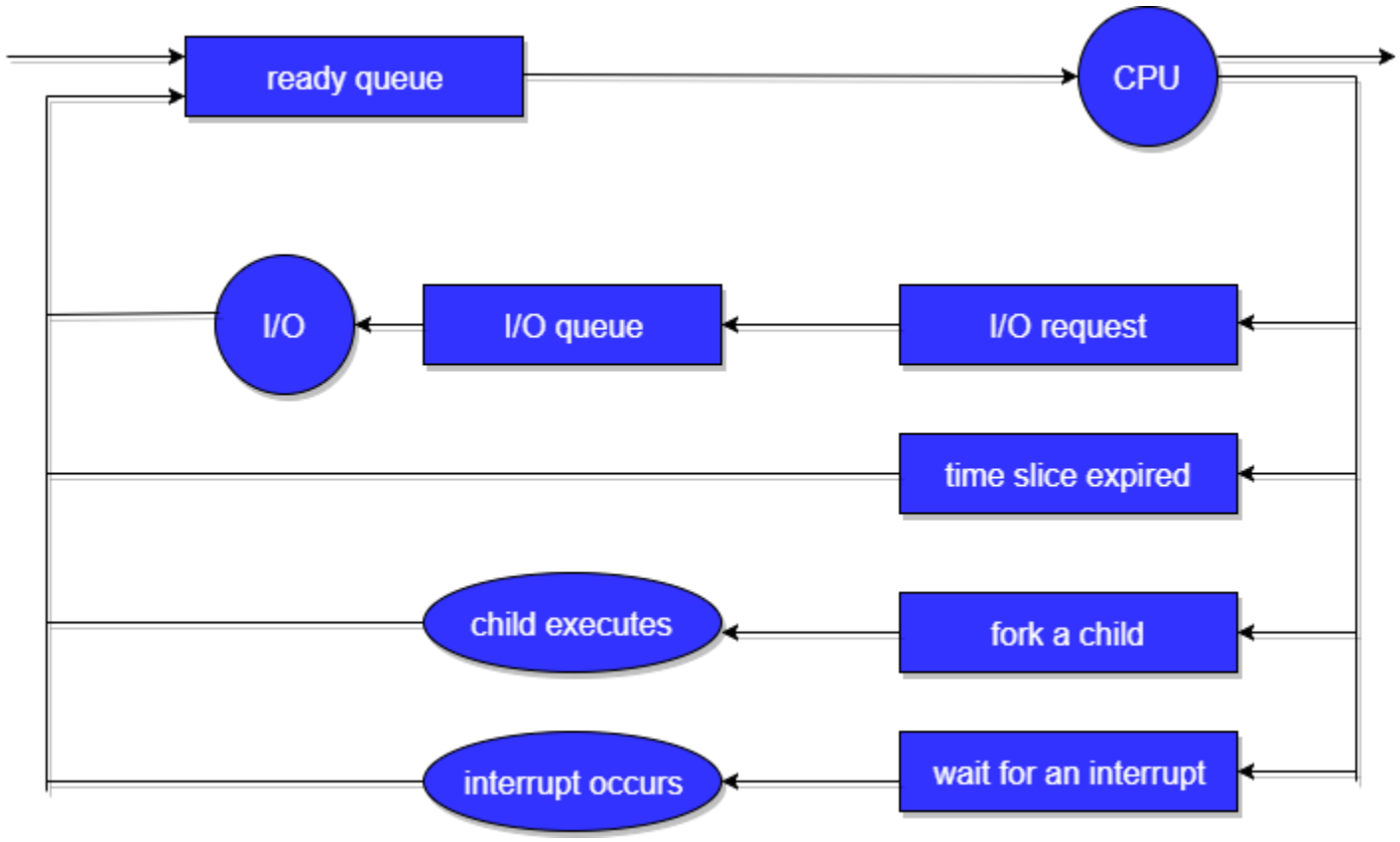
Scheduling queues

- All processes, upon entering into the system, are stored in the **Job Queue**.
- Processes in the Ready state are placed in the **Ready Queue**.

- Processes waiting for a device to become available are placed in **Device Queues**. There are unique device queues available for each I/O device.

A new process is initially put in the **Ready queue**. It waits in the ready queue until it is selected for execution(or dispatched). Once the process is assigned to the CPU and is executing, one of the following several events can occur:

- The process could issue an I/O request, and then be placed in the **I/O queue**.
- The process could create a new subprocess and wait for its termination.
- The process could be removed forcibly from the CPU, as a result of an interrupt, and be put back in the ready queue.



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.

Types of Schedulers

There are three types of schedulers available:

1. Long Term Scheduler
2. Short Term Scheduler
3. Medium Term Scheduler

Long Term Scheduler

Long term scheduler runs less frequently. Long Term Schedulers decide which program must get into the job queue. From the job queue, the Job Processor, selects processes and loads them into the memory for execution. Primary aim of the Job Scheduler is to maintain a good degree of Multiprogramming. An optimal degree of Multiprogramming means the average rate of process creation is equal to the average departure rate of processes from the execution memory.

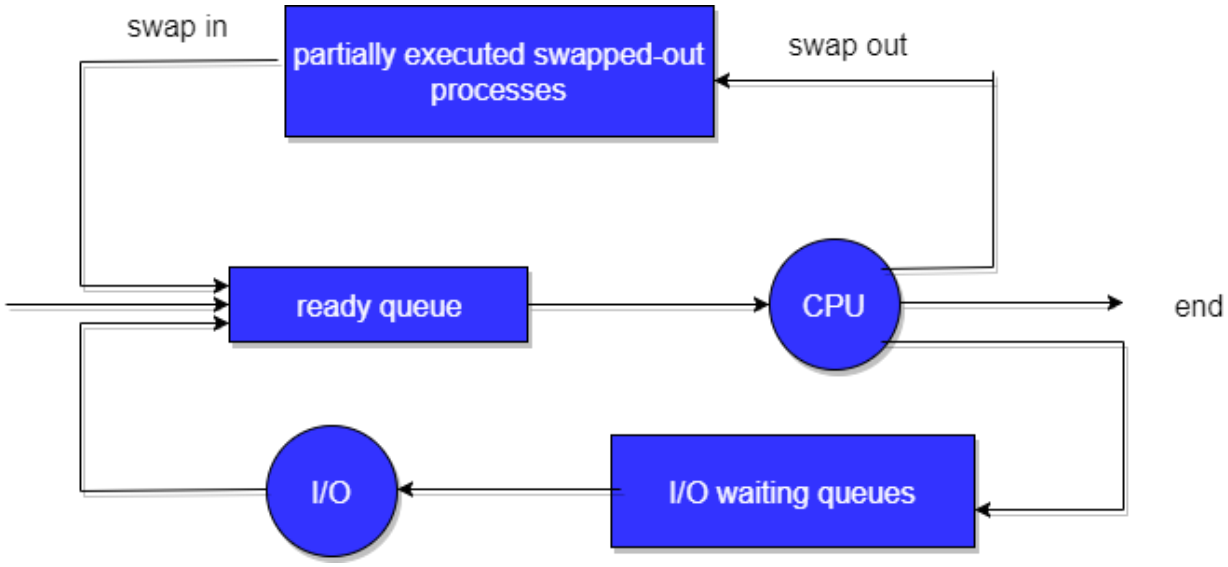
Short Term Scheduler

This is also known as CPU Scheduler and runs very frequently. The primary aim of this scheduler is to enhance CPU performance and increase process execution rate.

Medium Term Scheduler

This scheduler removes the processes from memory (and from active contention for the CPU), and thus reduces the degree of multiprogramming. At some later time, the process can be reintroduced into memory and its execution van be continued where it left off. This scheme is called **swapping**. The process is swapped out, and is later swapped in, by the medium term scheduler.

Swapping may be necessary to improve the process mix, or because a change in memory requirements has overcommitted available memory, requiring memory to be freed up. This complete process is descripted in the below diagram:



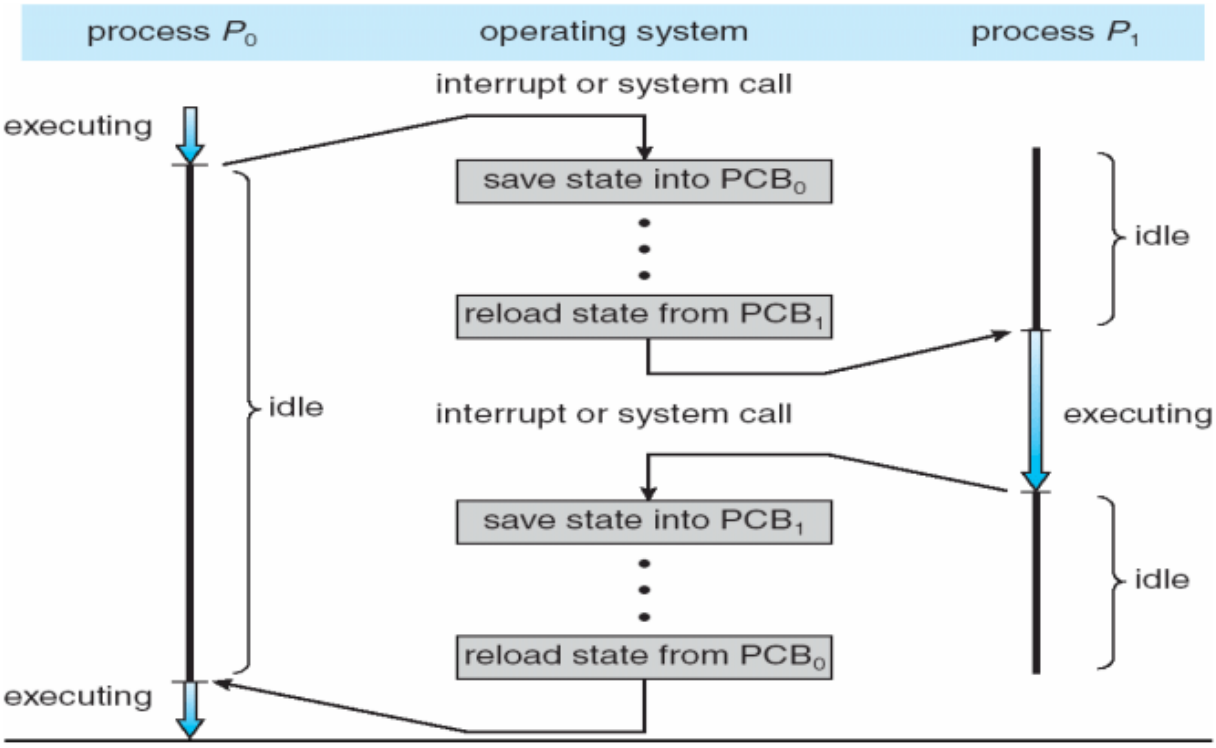
Addition of Medium-term scheduling to the queueing diagram.

Context switch

A context switch occurs when a computer's CPU switches from one process or thread to a different process or thread.

- Context switching allows for one CPU to handle numerous processes or threads without the need for additional processors.
- 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.
- Any operating system that allows for multitasking relies heavily on the use of context switching to allow different processes to run at the same time.

Typically, there are three situations that a context switch is necessary, as shown below.

- **Multitasking** – When the CPU needs to switch processes in and out of memory, so that more than one process can be running.
- **Kernel/User Switch** – When switching between user mode to kernel mode, it may be used (but isn't always necessary).
- **Interrupts** – When the CPU is interrupted to return data from a disk read.
- 

The steps in a full process switch are:

1. **Save the context** of the processor, including program counter and other registers.

2. **Update the process control block** of the process p_0 that is currently in the Running state. This includes changing the state of the process to one of the other states (Ready; Blocked; Ready/Suspend; or Exit). Other relevant fields must also be updated, including the reason for leaving the Running state and accounting information.

3. **Move** the **process control block** of this process to the **appropriate queue** (Ready; Blocked on Event *i ; Ready/Suspend)*.

4. **Select another process** p_1 for execution.
5. Update the process control block of the process p_1 selected. This includes changing the state of this process to Running.

6. Update memory management data structures. This may be required, depending on how address translation is managed.

7. **Restore the context** of the processor to that which existed at the time the selected process p_0 was last switched out of the Running state, by loading in the previous values of the program counter and other registers.