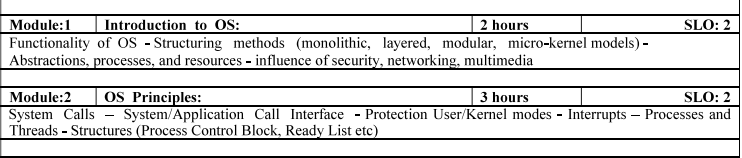
ARYAMAN MISHRA

19BCE1027

DIGITAL ASSIGNMENT 1

Topics Covered:



UNIT-1

***INTRODUCTION TO OS***

**Operating System –** Definition:

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

**TOPIC 1:Functions of Operating system –** Operating system performs three functions:

1. **Convenience:** An OS makes a computer more convenient to use.
2. **Efficiency:** An OS allows the computer system resources to be used in an efficient manner.
3. **Ability to Evolve:** An OS should be constructed in such a way as to permit the effective development, testing and introduction of new system functions at the same time without interfering with service.

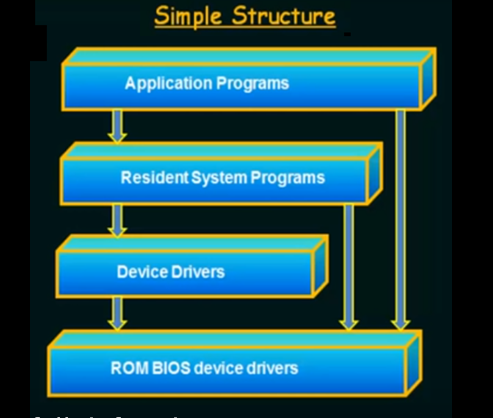
The operating system is a set of special programs that run on a computer system that allows it to work properly. It performs basic tasks such as recognizing input from the keyboard, keeping track of files and directories on the disk, sending output to the display screen and controlling peripheral devices.  
OS is designed to serve two basic purposes:

1. It controls the allocation and use of the computing System’s resources among the various user and tasks.
2. It provides an interface between the computer hardware and the programmer that simplifies and makes feasible for coding, creation, debugging of application programs.

The Operating system must support the following tasks. The task are:

1. Provides the facilities to create, modification of programs and data files using an editor.
2. Access to the compiler for translating the user program from high level language to machine language.
3. Provide a loader program to move the compiled program code to the computer’s memory for execution.
4. Provide routines that handle the details of I/O programming.

**TOPIC 2:STRUCTURING METHODS-SIMPLE STRUCTURE**

****

**There are many operating systems that have a rather simple structure. These started as small systems and rapidly expanded much further than their scope**. A common example of this is MS-DOS. It was designed simply for a niche amount for people. There was no indication that it would become so popular.

**It is better that operating systems have a modular structure, unlike MS-DOS. That would lead to greater control over the computer system and its various applications. The modular structure would also allow the programmers to hide information as required and implement internal routines as they see fit without changing the outer specifications.**

Operating systems such as MS-DOS and the original UNIX did not have well-defined structures.

There was no [CPU Execution Mode](http://faculty.salina.k-state.edu/tim/ossg/Introduction/OSworking.html#mode) (user and kernel), and so errors in applications could cause the whole system to crash.

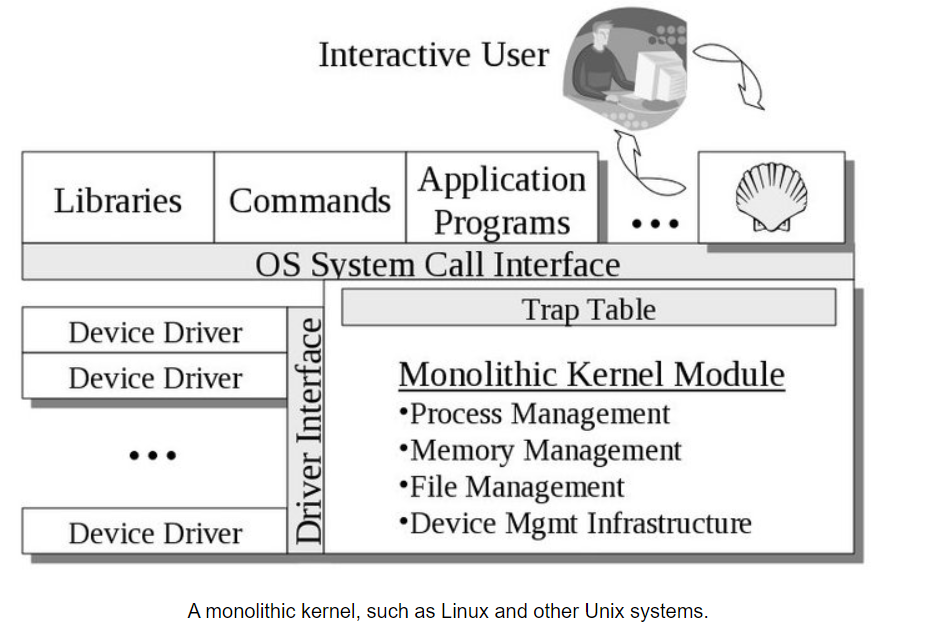
**TOPIC 3:STRUCTURING METHODS-MONOLITHIC**

A monolithic kernel is an operating system architecture where the entire operating system is working in [kernel space](https://en.wikipedia.org/wiki/Kernel_space).

The monolithic model differs from other operating system architectures (such as the [microkernel](https://en.wikipedia.org/wiki/Microkernel) architecture) in that it alone defines a high-level virtual interface over computer hardware. A set of primitives or [system calls](https://en.wikipedia.org/wiki/System_call) implement all operating system services such as [process](https://en.wikipedia.org/wiki/Process_(computing)) management, [concurrency](https://en.wikipedia.org/wiki/Concurrency_(computer_science)), and [memory management](https://en.wikipedia.org/wiki/Memory_management). Device drivers can be added to the kernel as [modules](https://en.wikipedia.org/wiki/Module_(programming)).

The monolithic operating system is a very basic operating system in which file management, memory management, device management, and process management is directly controlled within the kernel. All these components like file management, memory management etc. are located within the kernel.

* Functionality of the OS is invoked with simple function calls within the kernel, which is one large program.
* Device drivers are loaded into the running kernel and become part of the kernel.



**TOPIC 4:STRUCTURING METHODS-LAYERED STRUCTURE**

**An OS can be broken into pieces and retain much more control on system. In this structure the OS is broken into number of layers (levels). The bottom layer (layer 0) is the hardware and the topmost layer (layer N) is the user interface. These layers are so designed that each layer uses the functions of the lower level layers only. This simplifies the debugging process as if lower level layers are debugged and an error occurs during debugging then the error must be on that layer only as the lower level layers have already been debugged.**

The main disadvantage of this structure is that at each layer, the data needs to be modified and passed on which adds overhead to the system. Moreover careful planning of the layers is necessary as a layer can use only lower level layers. UNIX is an example of this structure.

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.

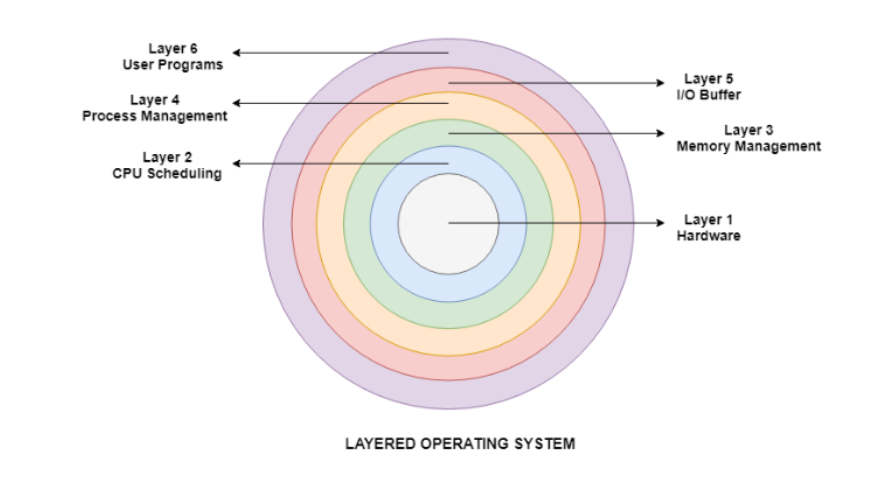
**Why Layering in Operating System?**

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:



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

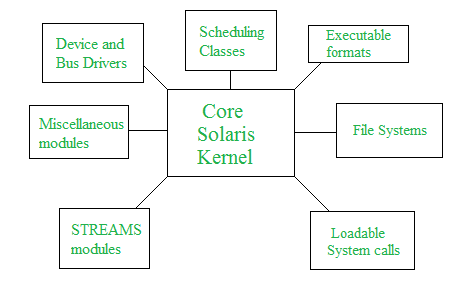
I/O devices are very important in the computer systems. They provide users with the means of interacting with the system. This layer handles the buffers for the I/O devices and makes sure that they work correctly.

### 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.

**TOPIC 5:STRUCTURING METHODS-** **MODULAR STRUCTURE**

**It is considered as the best approach for an OS. It involves designing of a modular kernel. The kernel has only set of core components and other services are added as dynamically loadable modules to the kernel either during run time or boot time. It resembles layered structure due to the fact that each kernel has defined and protected interfaces but it is more flexible than the layered structure as a module can call any other module.  
For example Solaris OS is organized as shown in the figure.**



A modular operating system is built with its various functions broken up into distinct processes, each with its own interface. By contrast, a traditional monolithic operating system uses one static-compiled image and runs in an “all or nothing” mode. If any one element or application within a monolithic operating system fails or needs an update, the entire system must be shut down and restarted, interrupting all packet flows. The primary benefit of the modular approach is that each process operates independently, and if one of them fails or needs an update, it won’t affect any of the other functions. A modular operating system significantly improves a company’s infrastructure uptime to levels approaching the desired 99.999% (five nines) and maximizes the availability of all its [business-critical applications](https://susedefines.suse.com/definition/mission-critical-computing/).

The main elements of a modular operating system are a kernel and a set of dynamically loadable applications with their own discrete memory spaces. The kernel is protected from service and application failures. Each process can be monitored to determine if it is operating properly, and if necessary a faulty process can be restarted dynamically. The overall system can continue to operate throughout the fix. This maintains the [uptime](https://susedefines.suse.com/definition/uptime/) of the underlying infrastructure, system applications and overall operating system. This ability to start and stop an application gracefully, without impacting the entire system, enables a quick response to emerging threats and security vulnerabilities, and makes it easy to manage applications with frequent refresh cycles.

The [Linux kernel](https://susedefines.suse.com/definition/linux-kernel/) is modular, which means it can extend its capabilities through dynamically-loaded kernel modules. Leading [Linux enterprise](https://susedefines.suse.com/definition/enterprise-linux/) distributions such as SUSE [Linux Enterprise Server](https://www.suse.com/products/server/) build on this modular [Linux](https://susedefines.suse.com/definition/linux/) kernel and offer many additional modules that contain software packages which add functionality to the operating system with all the advantages of easy management and [high availability](https://susedefines.suse.com/definition/high-availability/).

**(EXTRA)TOPIC 6:STRUCTURING METHODS-** **MICRO KERNEL MODELS**

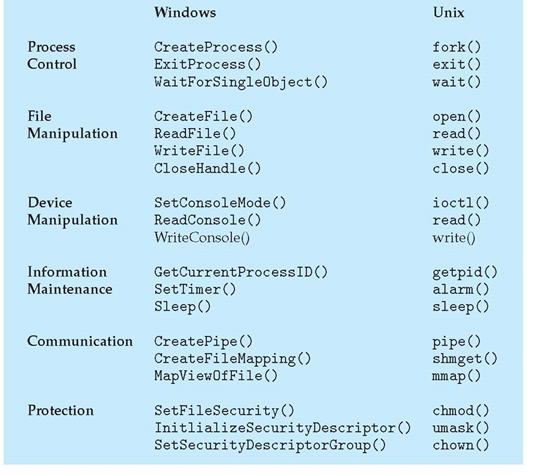
**This structure designs the operating system by removing all non-essential components from the kernel and implementing them as system and user programs. This result in a smaller kernel called the micro-kernel.**  
Advantages of this structure are that all new services need to be added to user space and does not require the kernel to be modified. Thus it is more secure and reliable as if a service fails then rest of the operating system remains untouched. Mac OS is an example of this type of OS.

***UNIT-2***

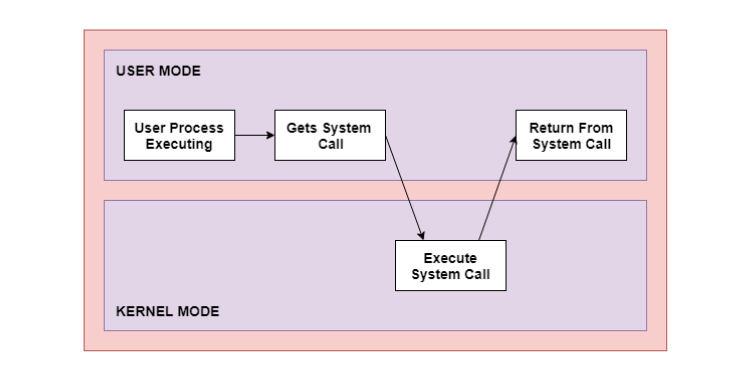
***OS PRINCIPLES***

**TOPIC 1: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.

**TOPIC 2:SYSTEM CALL INTERFACE**

**The system call interface is the programming interface for application programmers. The programmer must live with the interface that T&R have defined. The interface provides the process, interprocess communication, file, tty, and user abstractions.**

The interface and its implemenation are determined by the kernel implementors. Applications that need a different interface, or a different implementation of the interface, cannot run on v6. (Unless, of course, you convince T&R(technical and research) to change v6, which was the model at Bell Labs.)

What applications cannot run on v6, other than network applications? Another way of asking this question is "What is the intended applications of v6?". This question suprisingly hard to answer, because v6 didn't have a set of precise requirements. v6 and its successors are fixed points of what the UNIX developers needed to develop UNIX.

So, what are example of applications that don't run on v6? Databases, because they require transactions, which are impossible to implement on the UNIX file system, because of its weak reliability semantics. (One may be able to use the raw I/O interface by writing blocks directly to /dev/rk0, but every write is synchronous.) Multiple threads within a single address space are difficult to get right, because if one thread performs a system call the whole process is blocked. A process that has an image slightly larger than physical memory. Local servers are difficult to implement, but there is no way to communicate with an arbitrary process (other than through the file system), because pipe require a common anchestor.

If you want to write these kinds of applications, there is only one way out and that is to add system calls. This approach is taken by the modern versions of UNIX; FreeBSD 4.5 has 364 system calls. If an application programmer desires a different interface or different implementation, the programmer is stuck; he has to wait until the next release (assuming he could convince the kernel developers).

Recently monolithic operating systems added support for downloadable kernel modules, which allows a programmer to add its own code to a running kernel (assuming he has superuser privileges). Of course, if the programmer has created a bug in his code, then all other programs on the machine may suffer too. If the programmer wants his code to run on other machines, he has to convince the owners of those machines also to download his code in their kernel.

Although monolithic operating systems are the dominant operating system architecture for desktop and server machines, it is worthwhile to consider alternative architectures, even it is just to understand operating systems better.

**TOPIC 3:USER MODE AND KERNEL MODE**

## User Mode

**The system is in user mode when the operating system is running a user application such as handling a text editor. The transition from user mode to kernel mode occurs when the application requests the help of operating system or an interrupt or a system call occurs.**

**The mode bit is set to 1 in the user mode. It is changed from 1 to 0 when switching from user mode to kernel mode.**

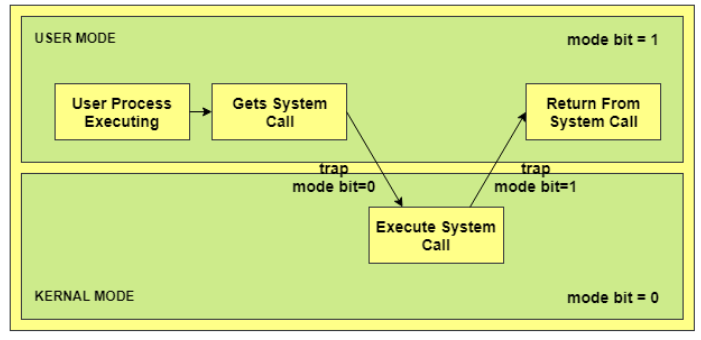
## Kernel Mode

**The system starts in kernel mode when it boots and after the operating system is loaded, it executes applications in user mode. There are some privileged instructions that can only be executed in kernel mode.**

These are interrupt instructions, input output management etc. If the privileged instructions are executed in user mode, it is illegal and a trap is generated.

The mode bit is set to 0 in the kernel mode. It is changed from 0 to 1 when switching from kernel mode to user mode.

An image that illustrates the transition from user mode to kernel mode and back again is −



In the above image, the user process executes in the user mode until it gets a system call. Then a system trap is generated and the mode bit is set to zero. The system call gets executed in kernel mode. After the execution is completed, again a system trap is generated and the mode bit is set to 1. The system control returns to kernel mode and the process execution continues.

**Necessity of Dual Mode (User Mode and Kernel Mode) in Operating System**

The lack of a dual mode i.e user mode and kernel mode in an operating system can cause serious problems. Some of these are −

* A running user program can accidentaly wipe out the operating system by overwriting it with user data.
* Multiple processes can write in the same system at the same time, with disastrous results.

These problems could have occurred in the MS-DOS operating system which had no mode bit and so no dual mode.

**TOPIC 4:INTERRUPTS**

**Interrupts** **are signals sent to the CPU by external devices, normally I/O devices. They tell the CPU to stop its current activities and execute the appropriate part of the operating system**

There are three types of interrupts:

1. **Hardware Interupts** are generated by hardware devices to signal that they need some attention from the OS. They may have just received some data (e.g., keystrokes on the keyboard or an data on the ethernet card); or they have just completed a task which the operating system previous requested, such as transfering data between the hard drive and memory.
2. **Software Interupts** are generated by programs when they want to request a [system call](http://faculty.salina.k-state.edu/tim/ossg/glossary.html#term-system-call) to be performed by the operating system.
3. **Traps** are generated by the CPU itself to indicate that some error or condition occured for which assistance from the operating system is needed.

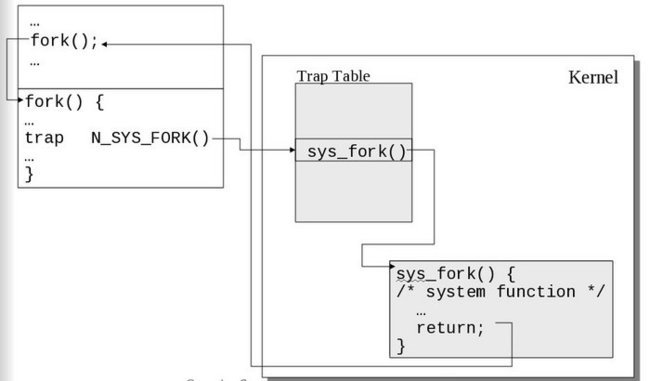
Interrupts are important because they give the user better control over the computer. Without interrupts, a user may have to wait for a given application to have a higher priority over the CPU to be ran. This ensures that the CPU will deal with the process immediately.

A key point towards understanding how operating systems work is to understand what the CPU does when an interrupt occurs. The hardware of the CPU does the exact same thing for each interrupt, which is what allows operating systems to take control away from the current running user process. The switching of running processes to execute code from the OS kernel is called a [context switch](http://faculty.salina.k-state.edu/tim/ossg/glossary.html#term-context-switch).

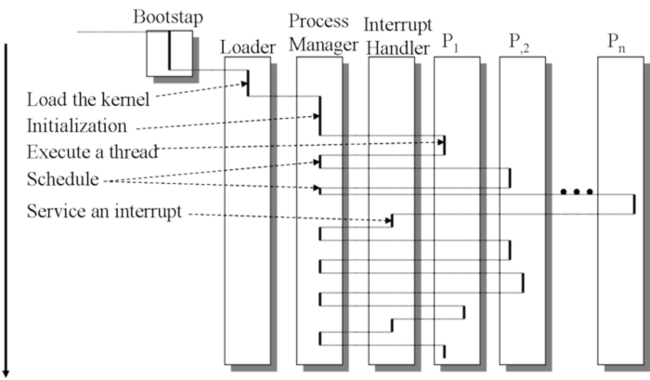
CPUs rely on the data contained in a couple registers to correctly handle interrupts. One register holds a pointer to the [process control block](http://faculty.salina.k-state.edu/tim/ossg/glossary.html#term-process-control-block) of the current running process. This register is set each time a process is loaded into memory. The other register holds a pointer to a table containing pointers to the instructions in the OS kernel for interrupt handlers and system calls. The value in this register and contents of the table are set when the operating system is initialized at boot time.

The CPU performs the following actions in response to an interrupt:

1. Using the pointer to the current process control block, the state and all register values for the process are saved for use when the process is later restarted.
2. The CPU mode bit is switched to *supervisory* mode.
3. Using the pointer to the interrupt handler table and the interrupt vector, the location of the kernel code to execute is determined. The interrupt vector is the IRQ for hardware interrupts (read from an interrupt controller register) and an argument to the interrupt assembly language instruction for software interrupts.
4. Processing is switched to the appropriate portion of the kernel.



The CPU uses a table and the interrupt vector to find OS the code to execute in response to interrupts. A software interrupt is shown here.

 As the computer runs, processing switches between user processes and the operating system as hardware and software interrupts are received.

**TOPIC 5:THREADS**

**A thread is a path of execution within a process. A process can contain multiple threads.**

A thread is also known as lightweight process. The idea is to achieve parallelism by dividing a process into multiple threads. For example, in a browser, multiple tabs can be different threads. MS Word uses multiple threads: one thread to format the text, another thread to process inputs, etc. More advantages of multithreading are discussed below

The primary difference is that threads within the same process run in a shared memory space, while processes run in separate memory spaces.  
Threads are not independent of one another like processes are, and as a result threads share with other threads their code section, data section, and OS resources (like open files and signals). But, like process, a thread has its own program counter (PC), register set, and stack space.

***Advantages of Thread***

1. Responsiveness: If the process is divided into multiple threads, if one thread completes its execution, then its output can be immediately returned.

2. Faster context switch: Context switch time between threads is lower compared to process context switch. Process context switching requires more overhead from the CPU.

3. Effective utilization of multiprocessor system: If we have multiple threads in a single process, then we can schedule multiple threads on multiple processor. This will make process execution faster.

4. Resource sharing: Resources like code, data, and files can be shared among all threads within a process.  
Note: stack and registers can’t be shared among the threads. Each thread has its own stack and registers.

5. Communication: Communication between multiple threads is easier, as the threads shares common address space. while in process we have to follow some specific communication technique for communication between two process.

6. Enhanced throughput of the system: If a process is divided into multiple threads, and each thread function is considered as one job, then the number of jobs completed per unit of time is increased, thus increasing the throughput of the system.  
**Types of Threads**  
There are two types of threads.  
User Level Thread  
Kernel Level Thread