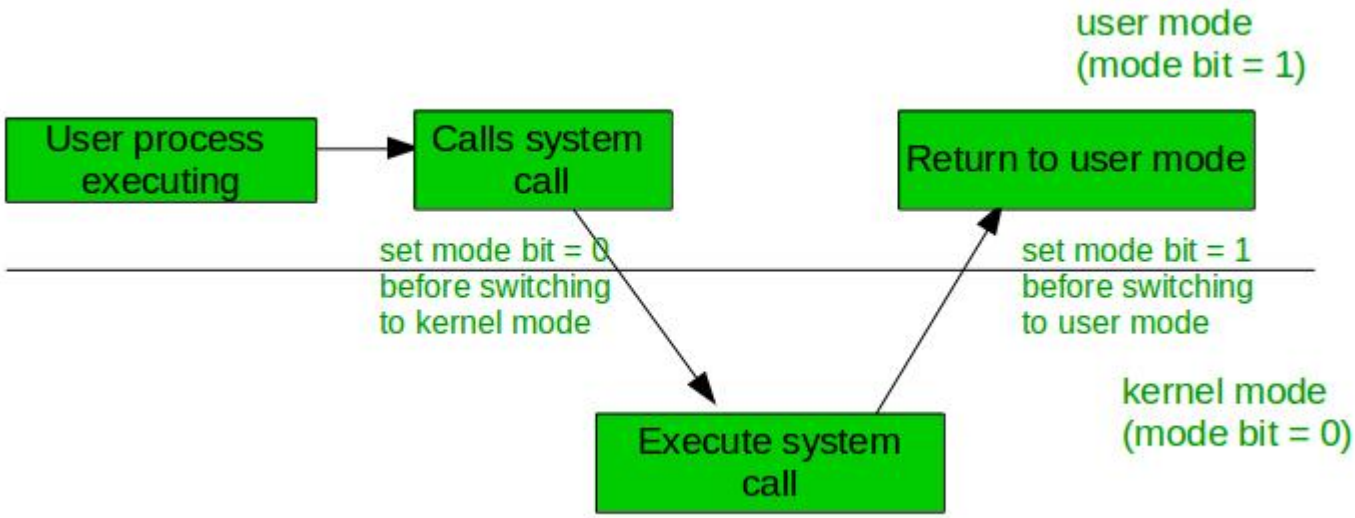
Operating Systems Organization

An error in one program can adversely affect many processes; it might modify data of another program, or also can affect the operating system. For example, if a process stuck in infinite loop then this infinite loop could affect correct operation of other processes. So to ensure the proper execution of the operating system, there are two modes of operation:

User mode –

When the computer system runs user applications like creating a text document or using any application program, then the system is in the user mode. When the user application requests for a service from the operating system or an interrupt occurs or system calls, then there will be a transition from user to kernel mode to fulfill the requests.

Given below image describes what happen when an interrupt occurs:



Kernel Mode –

When the system boots, hardware starts in kernel mode and when operating system is loaded, it start user application in user mode. To provide protection to the hardware, we have privileged instructions which execute only in kernel mode. If user attempt to run privileged instruction in user mode then it will treat instruction as illegal and traps to OS. Some of the privileged instructions are:

1. Handling Interrupts
2. To switch from user mode to kernel mode.
3. Input-Output management.

*****User mode and Kernel modes explanation**

- Because an operating system must hide the computer's hardware, and manage the hardware resources, it needs to prevent the application software from accessing the hardware directly. Without this sort of protection, the operating system would not be able to do its job.
- The computer's CPU provides two modes of operation which enforce this protection. The operating system runs in kernel mode, also known as supervisor mode or privileged mode. In kernel mode, the software has complete access to all of the computer's hardware, and can control the switching between the CPU modes. Interrupts are also received in the kernel mode software.
- The rest of the software runs in user mode. In this mode, direct access to the hardware is prohibited, and so is any arbitrary switching to kernel mode. Any attempts to violate these restrictions are reported to the

kernel mode software: in other words, to the operating system itself. Programs running in user mode are given an address space, visible only to themselves, which contains enough memory for them to do their job.

- By having two modes of operation which are enforced by the computer's own hardware, the operating system can force application programs to use the operating system's abstract services, instead of circumventing any resource allocations by direct hardware access.

****Kernel in Operating System**

Kernel is central component of an operating system that manages operations of computer and hardware. It basically manages operations of memory and CPU time. It is core component of an operating system. Kernel acts as a bridge between applications and data processing performed at hardware level using inter-process communication and system calls.
Kernel loads first into memory when an operating system is loaded and remains into memory until operating system is shut down again. It is responsible for various tasks such as disk management, task management, and memory management.

It decides which process should be allocated to processor to execute and which process should be kept in main memory to execute. It basically acts as an interface between user applications and hardware. The major aim of kernel is to manage communication between software i.e. user-level applications and hardware i.e., CPU and disk memory.

Objectives of Kernel:

- To establish communication between user level application and hardware.
- To decide state of incoming processes.
- To control disk management.
- To control memory management.
- To control task management.

There are mainly two types of kernels exist:

1. **Monolithic Kernel** –
It is one of types of kernel where all operating system services operate in kernel space. It has dependencies between systems components. It has huge lines of code which is complex.
Example :
Unix, Linux, Open VMS, XTS-400 etc.

- Advantage :
  It has good performance.
- Disadvantage :
  It has dependencies between system component and lines of code in millions.

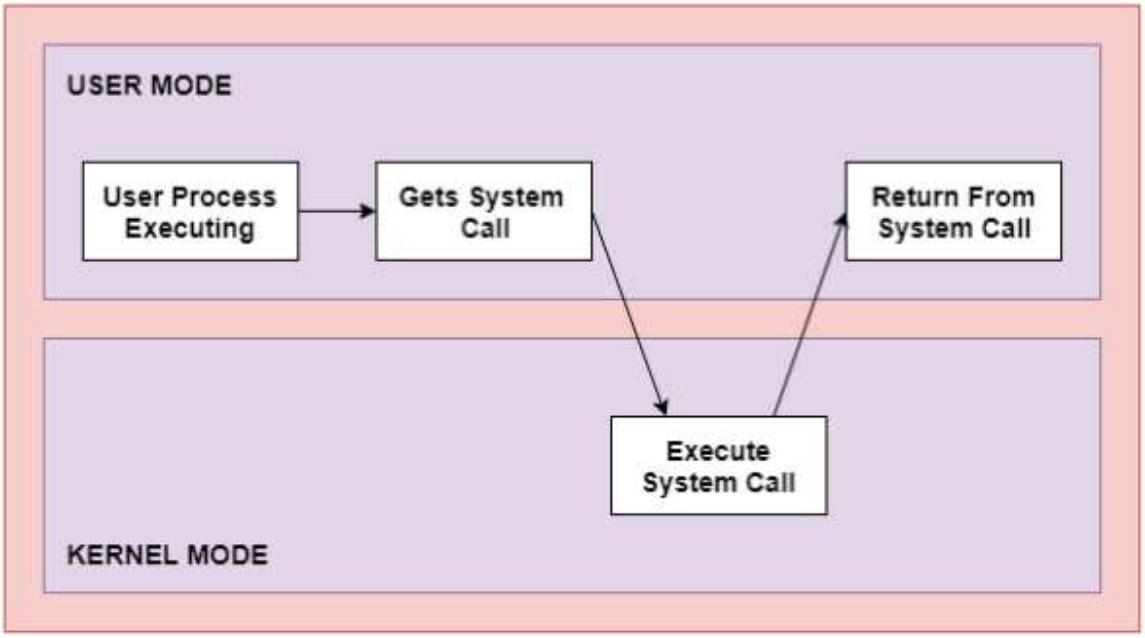
2. **Micro Kernel** –
It is kernel types which has minimalist approach. It has virtual memory and thread scheduling. It is more stable with less services in kernel space. It puts rest in user space.
Example :
Mach, L4, AmigaOS, Minix, K42 etc.

- Advantage :
  It is more stable.
- Disadvantage :
  There are lots of system calls and context switches.

***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.

Details of some of those system calls are as follows −

open()

The open() system call is used to provide access to a file in a file system. This system call allocates resources to the file and provides a handle that the process uses to refer to the file. A file can be opened by multiple processes at the same time or be restricted to one process. It all depends on the file organisation and file system.

read()

The read() system call is used to access data from a file that is stored in the file system. The file to read can be identified by its file descriptor and it should be opened using open() before it can be read. In general, the read() system calls takes three arguments i.e. the file descriptor, buffer which stores read data and number of bytes to be read from the file.

write()

The write() system calls writes the data from a user buffer into a device such as a file. This system call is one of the ways to output data from a program. In general, the write system calls takes three arguments i.e. file descriptor, pointer to the buffer where data is stored and number of bytes to write from the buffer.

****System Programs in Operating System**

System Programming can be defined as act of building Systems Software using System Programming Languages. According to Computer Hierarchy, one which comes at last is Hardware. Then it is Operating System, System Programs, and finally Application Programs. Program Development and Execution can be done conveniently in System Programs. Some of System Programs are simply user interfaces, others are complex. It traditionally lies between user interface and system calls.