Unit 2

Processes and Process Management

Process: can be defined as

- A program in execution
- An instance of a program running on a computer.
- The entity that can be assigned to and executed on a processor.
- A unit of activity characterized by the execution of a sequence of instructions, a current
- state, and an associated set of system resources.

When we write any **programming language code/program in computer in editor** when we execute this program then it becomes a process.

We can say active entity of a program is said to a process.

A program becomes a process when an executable file is loaded into memory

The status of the current activity of a process is represented by the value of the program counter and the contents of the processor’s registers.

The memory layout / components of a process:

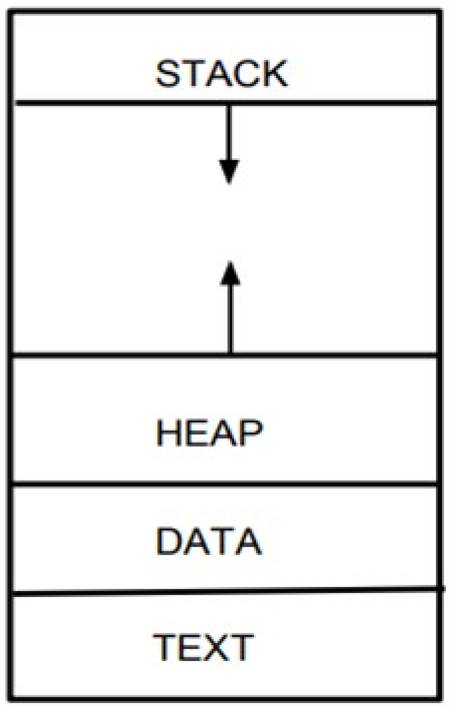
The memory layout / components of a process is typically divided into multiple sections,

- Text section— the executable code

• Data section—global variables

• Heap section—memory that is dynamically allocated during program run time

• Stack section— temporary data storage when invoking functions (such as function parameters, return addresses, and local variables).



Stack

Temporary data like method or function parameters, return address, and local variables are stored in the process stack. Each time a function is called, **an activation record** containing function parameters, local variables, and the return address is pushed onto the stack; when control is returned from the function, the activation record is popped from the stack

Heap

This is the memory that is dynamically allocated to a process during its execution.

Text

It has executable code.This comprises the contents present in the processor’s registers as well as the current activity reflected by the value of the program counter.

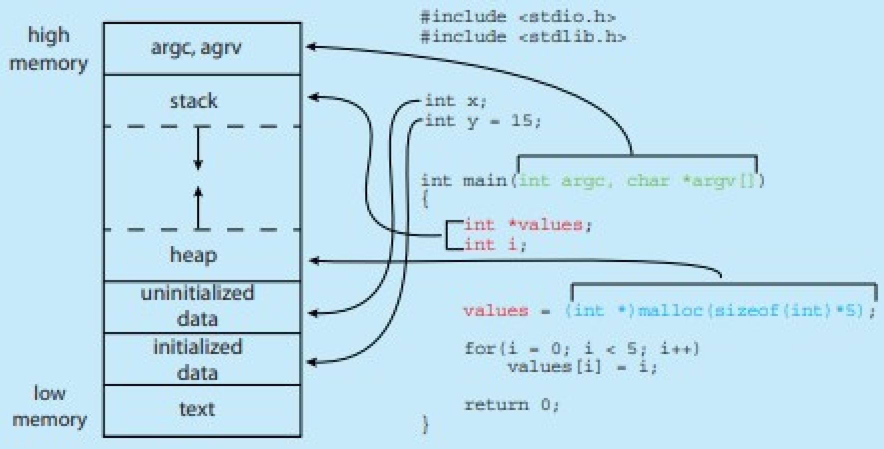
Data

The global as well as static variables are included in this section.

MEMORY LAYOUT OF A C PROGRAM

The figure shown below illustrates the layout of a C program in memory, highlighting how the different sections of a process relate to an actual C program. This figure is similar to the general concept of a process in memory as shown in Figure 3.1, with a few differences:

- The global data section is divided into different sections for (a) initialized data and (b) uninitialized data.
- A separate section is provided for the `argc` and `argv` parameters passed to the `main()` function.



```
#include <stdio.h>
#include <stdlib.h>

int x;
int y = 15;

int main(int argc, char *argv[])
{
   int *values;
   int i;

   values = (int *)malloc(sizeof(int)*5);

   for(i = 0; i < 5; i++)
      values[i] = i;

   return 0;
}
```

The GNU `size` command can be used to determine the size (in bytes) of some of these sections. Assuming the name of the executable file of the above C program is `memory`, the following is the output generated by entering the command `size memory`:

```
text    data    bss    dec    hex    filename
1158    284     8      1450   5aa    memory
```

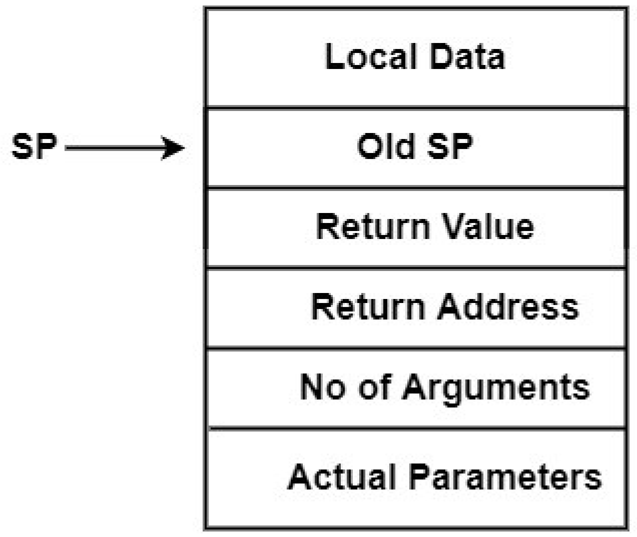
The `data` field refers to uninitialized data, and `bss` refers to initialized data. (`bss` is a historical term referring to ***block started by symbol***.) The `dec` and `hex` values are the sum of the three sections represented in decimal and hexadecimal, respectively.

Activation Record

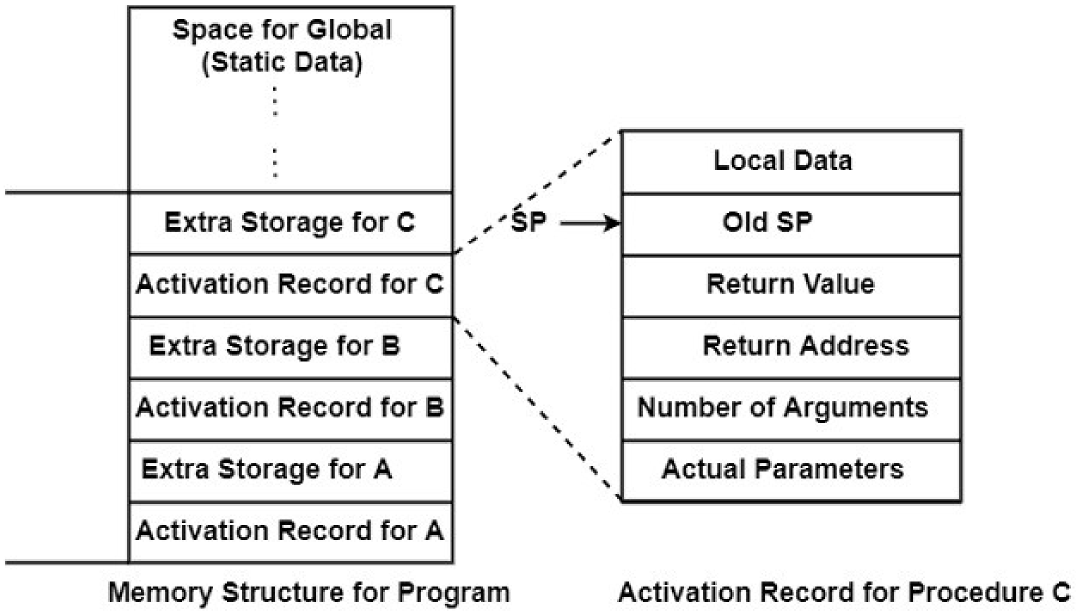
An Activation Record is a data structure that is activated/ created when a procedure/function is invoked, and it includes the following data about the function.

Activation Record consist of

- Actual Parameters
- Number of Arguments
- Return Address
- Return Value
- Old Stack Pointer (SP)
- Local Data in a function or procedure



Activation Record



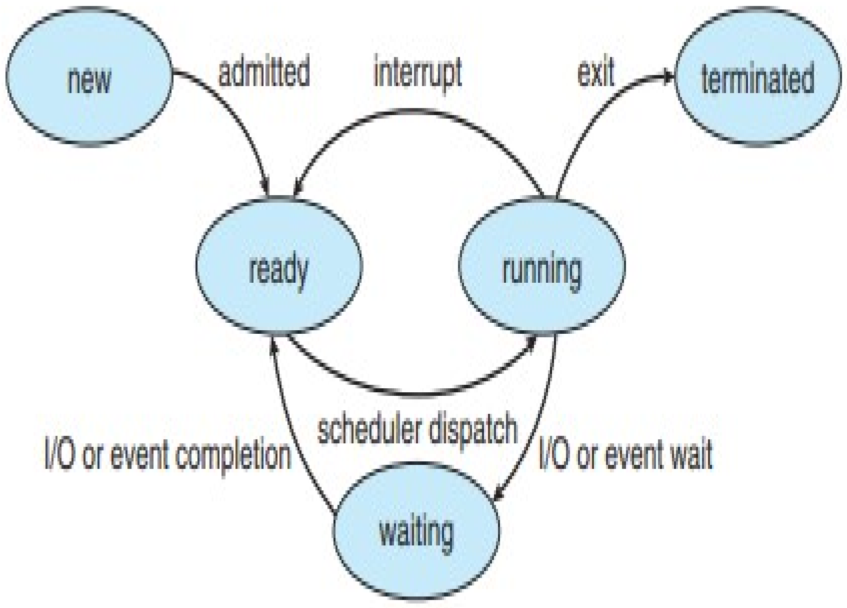
Stack Allocation Scheme

Process State

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

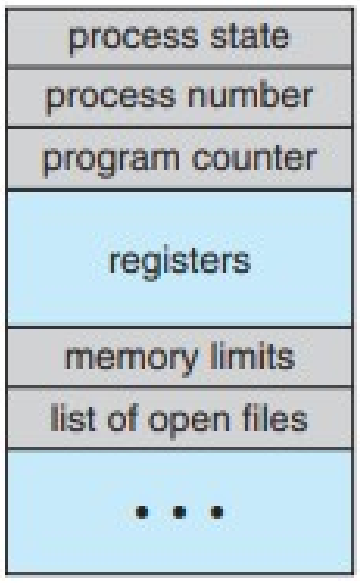
A process may be in one of the following states.

- **New:**The process is being created.
- **Ready:**The process is waiting to be assigned to a processor.
- **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).
- **Terminated :**The process has finished execution.



Process Control Block

- Each process is represented in the operating system by a process control block (PCB)—also called a task control block.
- It contains many pieces of information associated with a specific process.
- In brief, the PCB simply serves as the repository for all the data needed to start, or restart, a process, along with some accounting data.



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.

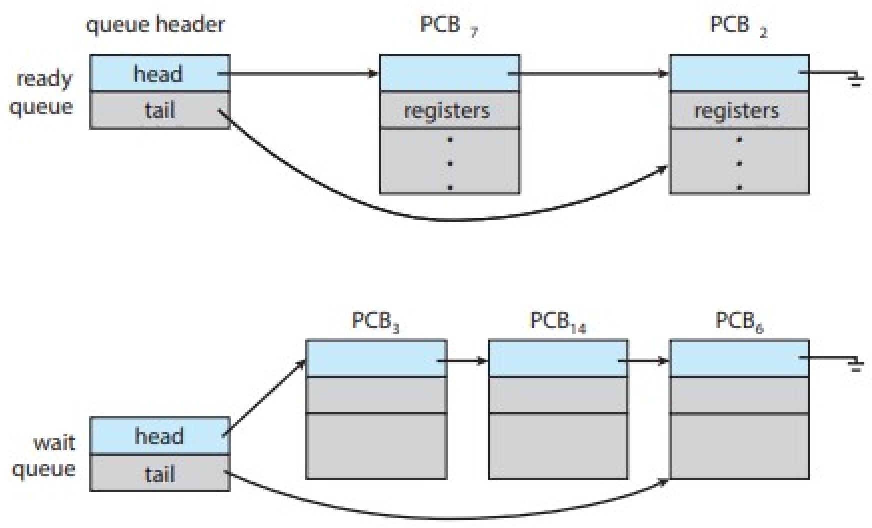
CPU registers: The registers vary in number and type, depending on the computer architecture. **They include accumulators, index registers, stack pointers, and general-purpose registers, plus any condition-code information.** Along with the program counter, this state information must be saved when an interrupt occurs, to allow the process to be continued correctly afterward when it is rescheduled to run.

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

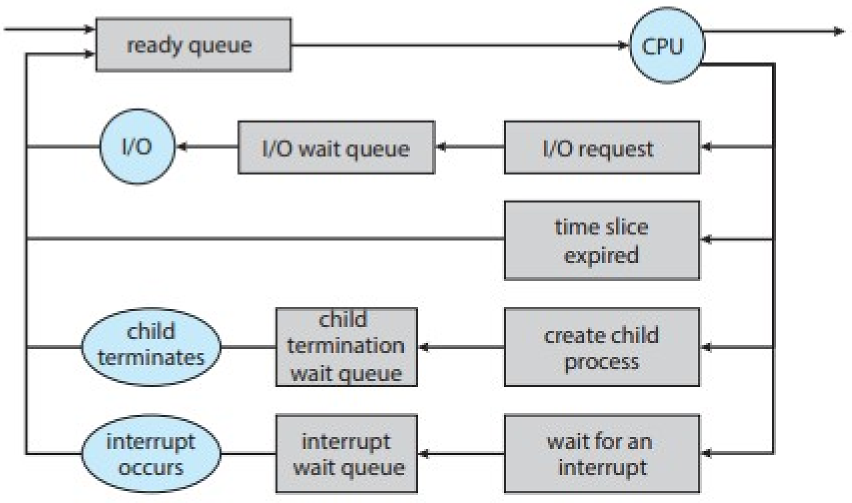
• **Memory-management information**:This information may include such items as the value of the base and limit registers and the page tables, or the segment tables, depending on the memory system used by the operating system.

• 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



Scheduling queues



• The process could issue an I/O request and then be placed in an I/O wait queue.

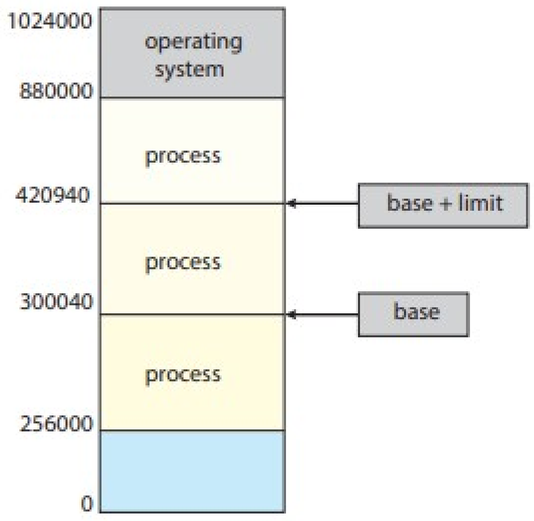
• The process could create a new child process and then be placed in a wait queue while it awaits the child's termination.

• The process could be removed forcibly from the core, as a result of an interrupt or having its time slice expire, and be put back in the ready queue.

An I/O-bound process is one that spends more of its time doing I/O than it spends doing computations.

A CPU-bound process, in contrast, generates I/O requests infrequently, using more of its time doing computations.

Base and limit registers



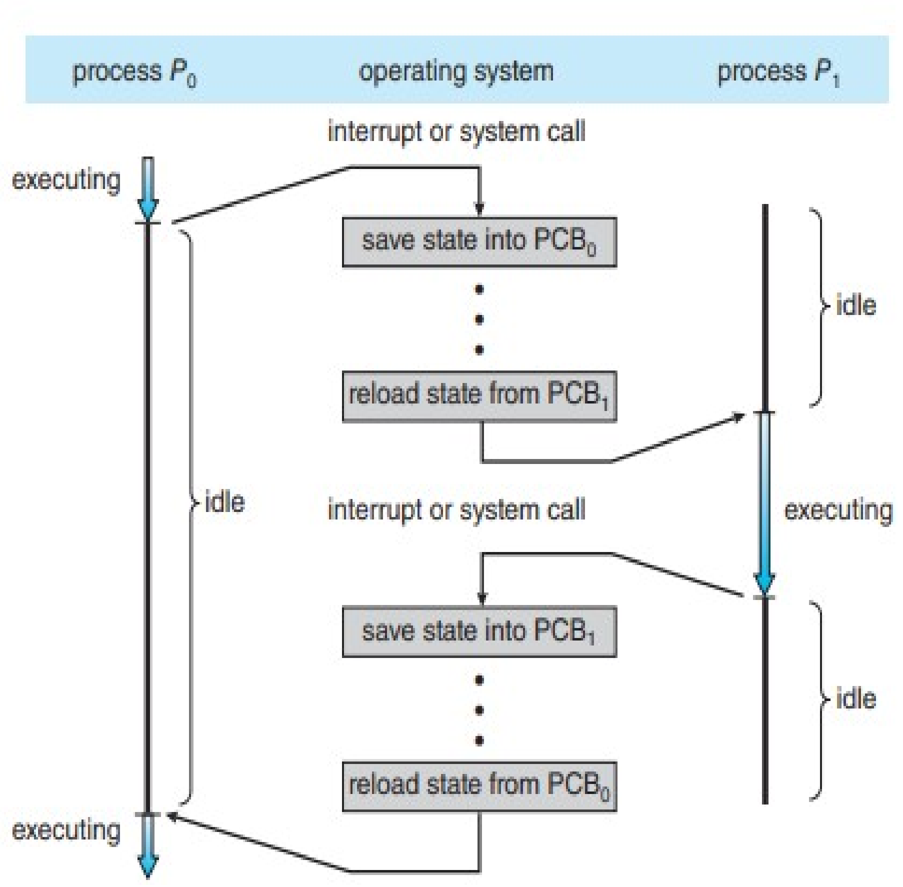
The base register holds the smallest legal physical memory address;

the limit register specifies the size of the range.

Context Switch

When an interrupt occurs, the system needs to save the current context of the process running on the CPU core so that it can restore that context when its processing is done, essentially suspending the process and then resuming it. The context is represented in the PCB of the process. It needs to perform a state save of the current state of the CPU core, be it in kernel or user mode, and then a state restore to resume operations.

Switching the CPU core to another process requires performing a state save of the current process and a state restore of a different process. This task is known as a context switch.



Thread:

A thread refers to an execution unit in the process that has its own programme counter, stack, as well as a set of registers.

A **thread** is the basic unit to which the operating system allocates processor time. A **thread** can execute any part of the **process** code.

Threads can’t exist outside any process.

Threads are popularly used to improve the application through **parallelism**.

Actually only one thread is executed at a time by the CPU, but the **CPU switches rapidly** between the threads to give an illusion that the threads are running parallelly.

The typical difference is that threads (of the same process) run in a shared memory space, while processes run in separate memory spaces.

Example:

If you start Ms Word, it is a process. In Ms Word, you type some thing and it gets automatically saved. Now, you would have observed editing and saving happens in parallel. These are threads.

If you start Paint, it is a process. In Paint it draw pictures by reading mouse movement. The program must give its full attention to the mouse input and draw at the same time. To do this two or more threads of the program will execute at the same time.

Process and Thread

In a computer or mobile game when you see objects like cars, trucks, etc, these are just threads that run in the game application. Consider another example where a web browser may have a thread to display images or videos while other threads to fetch the data from the internet. So, a single application may need to perform several similar tasks parallel. Thread is used to improve the application through running task simultaneously.

Example: Opening a new browser (say Chrome, etc) is an example of creating a process. At this point, a new process will start to execute. On the contrary, opening multiple tabs in the browser is an example of creating the thread.

When we write any **programming language code/program in computer in editor** when we execute this program then it becomes a process which the given task.

System Call

A system call is a function that a user program uses to ask the operating system for a particular service.

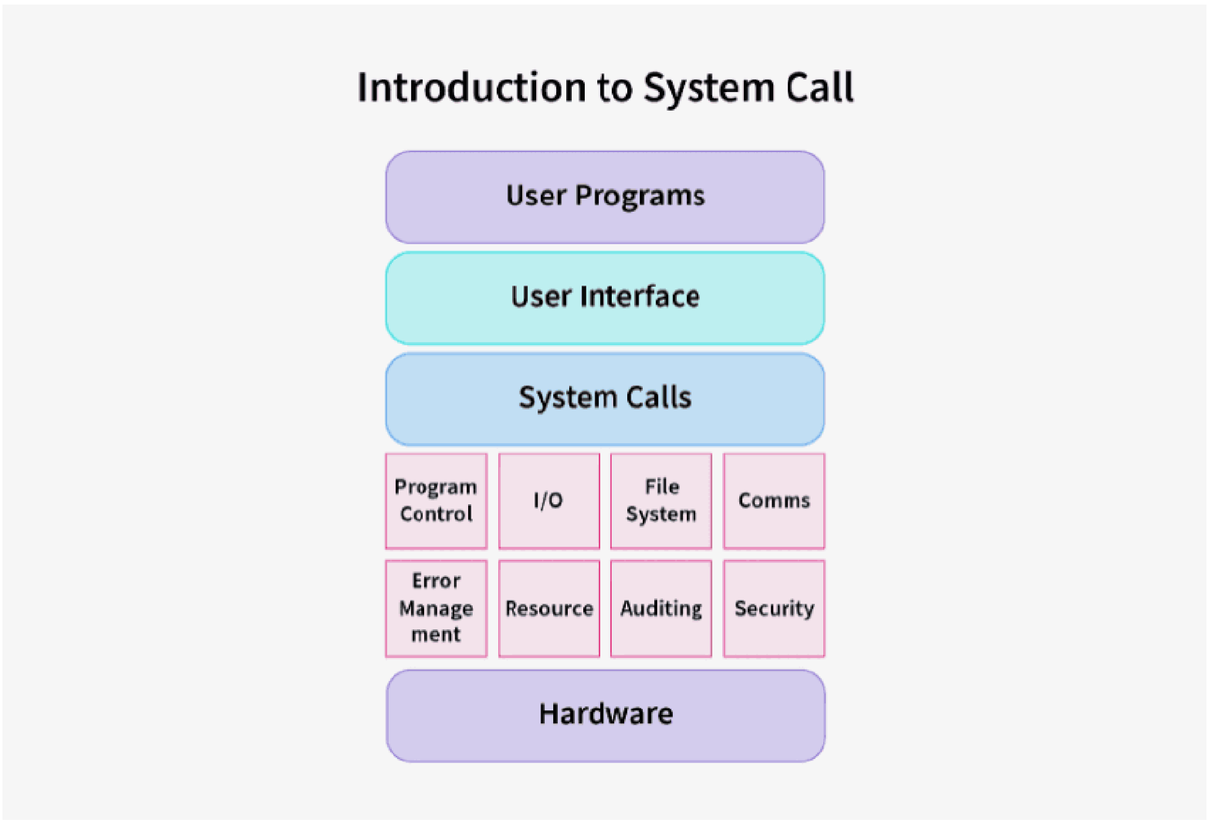
System calls serve as the interface between an operating system and a process.

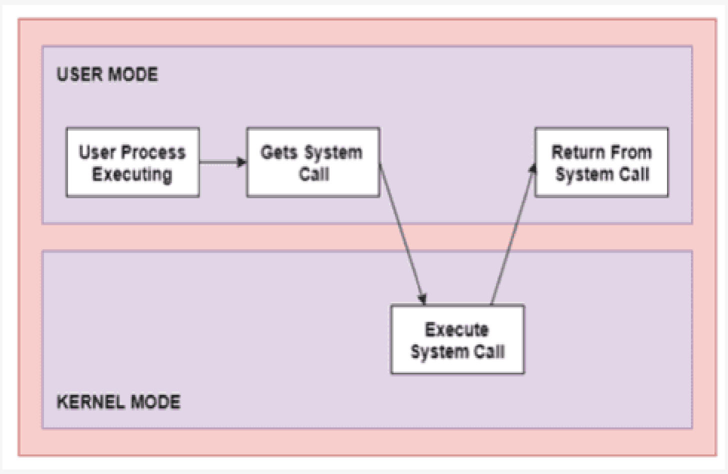
User programmers can communicate with the operating system to request its services using the interface that is created by a system call.

When a process in user mode needs access to a resource, system calls are typically generated. The resource is then requested from the kernel via a system call.

System calls are always carried out in the operating system's kernel mode.

System call uses Application Programming Interfaces to provide operating system services to user programmes (API).





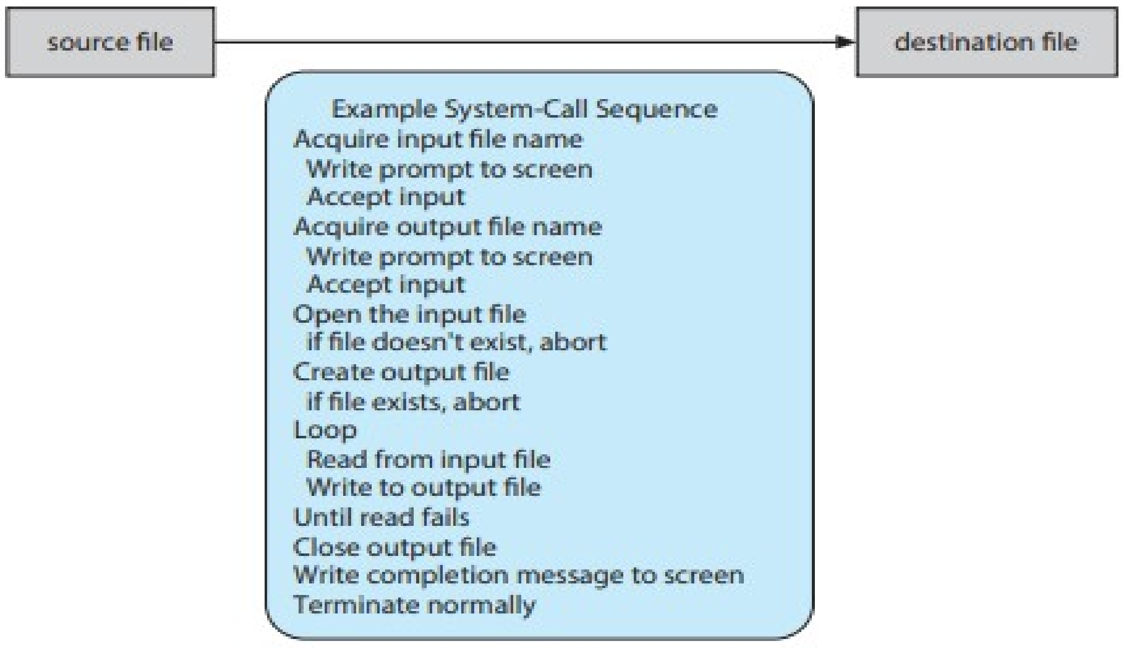
Working of System Call

The Applications operate in user space, a portion of memory. The operating system's kernel, which runs in kernel space, is reached through a system call.

An application must first request permission from the kernel before it can create a system call. It accomplishes this by sending an interrupt request, which pauses the running process and gives the kernel control.

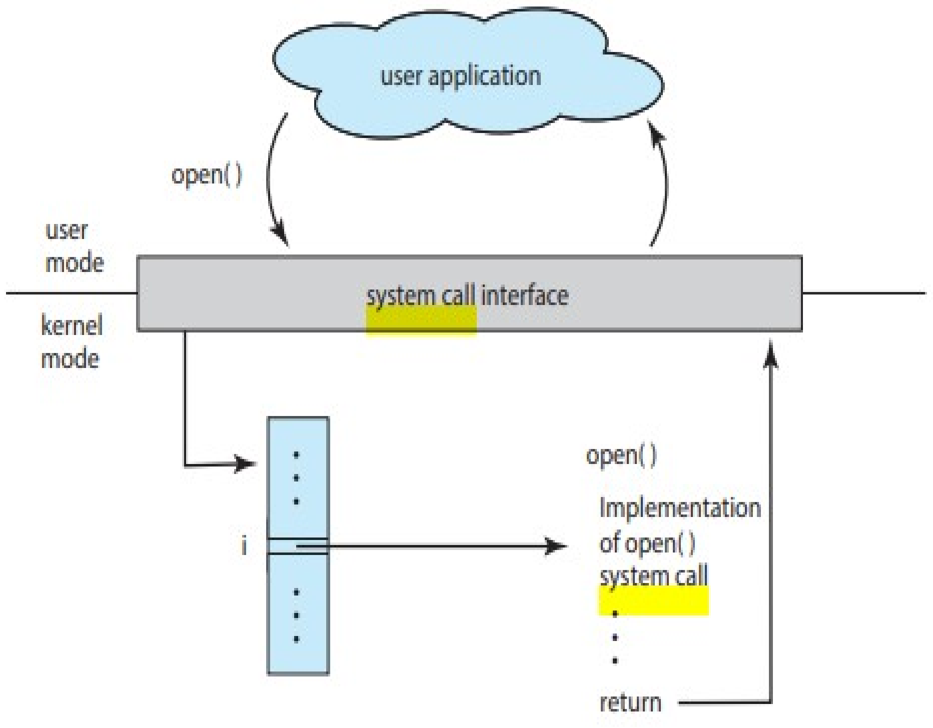
The kernel carries out the requested action, such as creating or deleting a file, if the request is approved. The output of the kernel is provided to the application as input. Once the input is received, the application continues the process. The kernel transfers data from kernel space to user space in memory after the operation is complete and returns the results to the application.

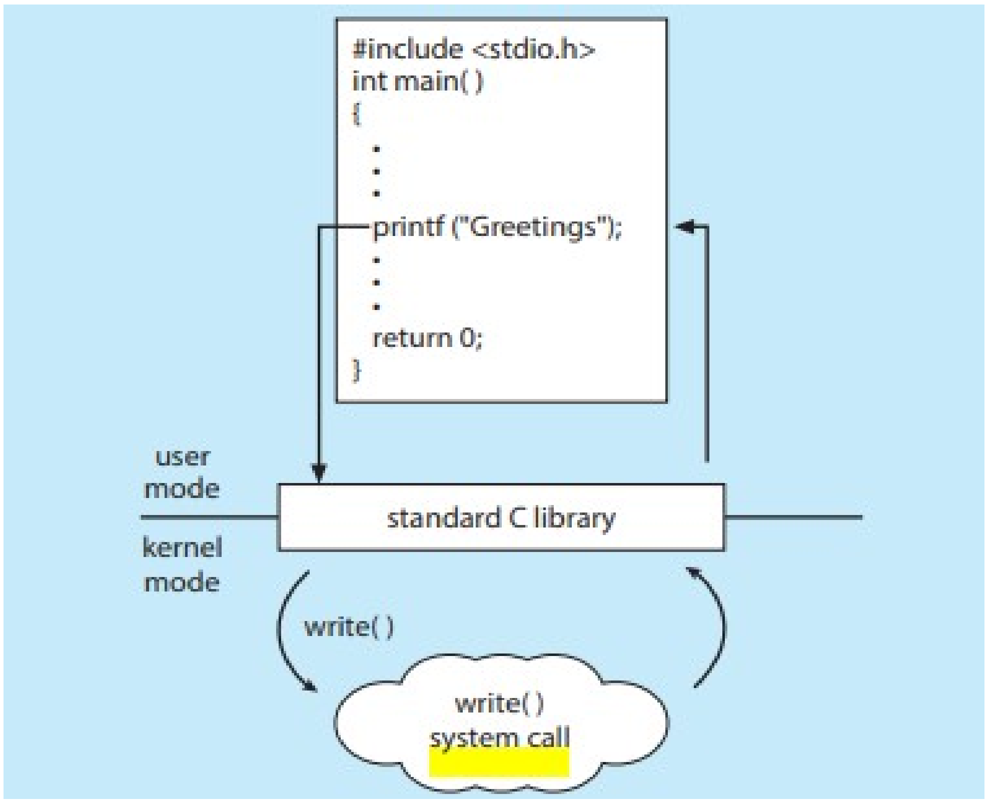
e.g. copy the content of one file to another file.



Example System-Call Sequence

```
Acquire input file name
  Write prompt to screen
  Accept input
Acquire output file name
  Write prompt to screen
  Accept input
Open the input file
  if file doesn't exist, abort
Create output file
  if file exists, abort
Loop
  Read from input file
  Write to output file
Until read fails
Close output file
Write completion message to screen
Terminate normally
```





Types of System Calls in OS

There are primarily 5 different types of system calls in the operating system:

1. **Process control**

◦ create process, terminate process
◦ load, execute
◦ get process attributes, set process attributes
◦ wait event, signal event
◦ allocate and free memory

1. Process Control

The following services are provided by Process Control System calls that are used to control a process.

- To forcefully abort the process, simply end it normally.
- Execute a process after loading it into the main memory.
- Terminate the current process before starting a new one.
- Wait for a process to complete running. Wait until a specific event happens, then announce it once it has.
- Allocate memory to a process, then release the memory if the process is terminated.

2. File management
◦ create file, delete file
◦ open, close
◦ read, write,
◦ get file attributes, set file attributes

2. File Management

The following services are provided by system call for file management:

- Making and erasing files
- Open the file, then close it.
- Write to a specific file, read from a specific file.
- To obtain a file's attribute and to change a file's attribute

3. **Device management**
◦ request device, release device
◦ read, write, reposition
◦ get device attributes, set device attributes
◦ logically attach or detach devices

3. Device Management

The following services are offered by a system call that controls I/O devices:

- Devices might be needed while a process is running. such as access to the system, I/O devices, main memory, etc.
- As a result, it can ask for a device and then release it once the task is complete.

- when a requested device is granted access by the process. It is capable of reading, writing, and repositioning operations.
- To detach a device from the processor that is currently executing a command, the call can be made.

4. Information Maintenance

The system calls for information maintenance call moves data from the user programme to the operating system. Considering this, the services offered by this type of system call are:

- Obtain the system's time or date. Set the system's time or date.
- Obtain system-related information. Configure the system data.
- Obtain the characteristics of a specific operating system process. Alternatively, of a specific file on the system or on any attached devices.

5. Communication

System call facilitates the system's network connection. The services that these system calls offer are:

- Open a fresh connection to send the data. After the transmission is finished,disconnect from the connection.
- On a particular connection, send a message. Obtain communication from a specific connection.
- Identify and connect a specific remote device to the network. Remove a specific remote computer or device from the network.

• **Information maintenance**
◦ get time or date, set time or date
◦ get system data, set system data
◦ get process, file, or device attributes
◦ set process, file, or device attributes

• **Communications**
◦ create, delete communication connection
◦ send, receive messages
◦ transfer status information
◦ attach or detach remote devices

• **Protection**
◦ get file permissions
◦ set file permissions

Types of System Calls	Windows	Linux

Process Control	CreateProcess()
	ExitProcess()
	WaitForSingleObject()
		fork()
		exit()
		wait()

File Management	CreateFile()
	ReadFile()
	WriteFile()
	CloseHandle()
		open()
		read()
		write()
		close()

Device Management	SetConsoleMode()
	ReadConsole()
	WriteConsole()
		ioctl()
		read()
		write()

Information Maintenance	GetCurrentProcessID()
	SetTimer()
	Sleep()
		getpid()
		alarm()
		sleep()

Communication	CreatePipe()
	CreateFileMapping()
	MapViewOfFile()
		pipe()
		shmget()
		mmap()

For Example:

writing a simple program to read data from one file and copy them to another file.