UNIT – 2

What is a Process?

A process is **a program in execution**. A **process** is an **'active' entity** instead of a **program**, which is considered a **'passive' entity**.

For example, when we write a program in C or C++ and compile it, the compiler creates binary code. The original code and binary code are both programs. When we actually run the binarycode, it becomes a process.

A **single program can create many processes when run multiple times**; for example, when we open a.exe or binary file multiple times, multiple processes are created.

What is Process Management?

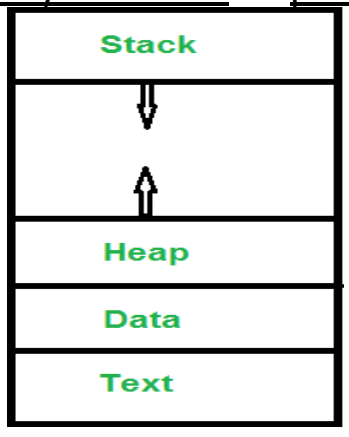
There may exist more than one process in the system which may require the same resource at the same time. The Operating system has to **keep track** of all the **completing processes, schedule them, dispatch them one after another**. But user should feel that he has the full control of the CPU. Therefore the operating system has to **manage all the processes and the resources in a convenient and efficient way**. This whole process is called **Process Management**.

The operating system is responsible for the following activities in connection with Process Management :-

1. Scheduling processes and threads on the CPUs.
2. Creating and deleting both user and system processes.
3. Suspending and resuming processes.
4. Providing mechanisms for process synchronization.
5. Providing mechanisms for process communication.

Explanation of Process

- **Text Section:** A Process, sometimes known as the Text Section, also includes the current activity represented by the value of the Program Counter.
- **Stack:** The stack contains temporary data, such as function parameters, returns addresses and local variables.
- **Data Section:** Contains the global variable.
- **Heap Section**: Dynamically memory allocated to process during its run time.



Attributes or Characteristics of a Process

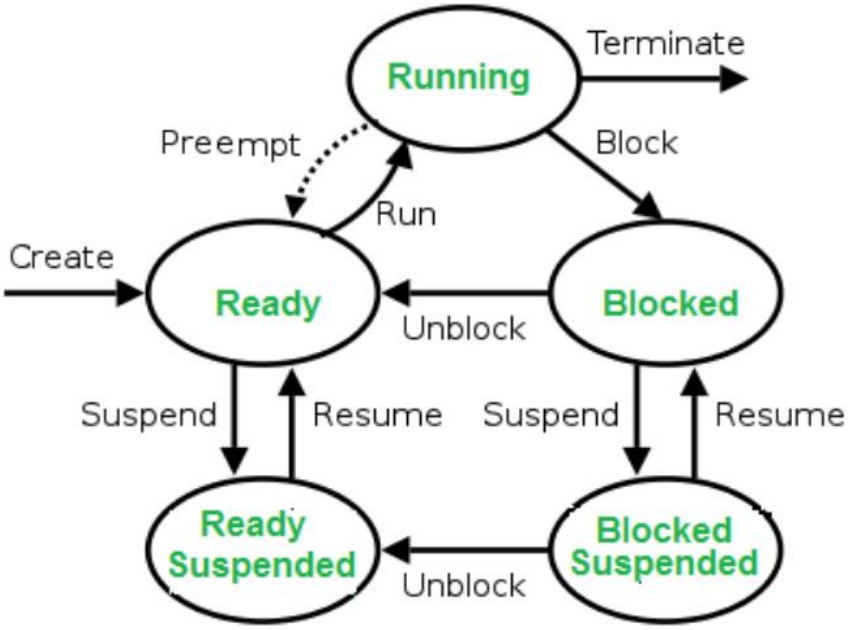
1. **Process Id:** A unique identifier assigned by the operating system.
2. **Process State:** Can be ready, running, etc.
3. **CPU registers:** Like the Program Counter (CPU registers must be saved and restored when a process is swapped in and out of CPU).
4. **Accounts information:** Amount of CPU used for process execution, time limits, execution ID etc.
5. **I/O status information:** For example, devices allocated to the process, open files, etc.
6. **CPU scheduling information:** For example, Priority (Different processes may have different priorities, for example a shorter process assigned high priority in the shortest job first scheduling).

All of the above attributes of a process are also known as the **context of the process**. **Every process has its own process control block(PCB)**, i.e. each process will have a unique PCB. All of the **above attributes** are part of the **PCB**.

States of Process

A process is in one of the following states:

1. **New:** Newly Created Process (or) being-created process.
2. **Ready:** After the creation process moves to the Ready state, i.e. the process is ready for execution.
3. **Running:** Currently running process in CPU (**only one process at a time can be under execution in a single processor**)
4. **Wait (or Block):** When a process requests I/O access.
5. **Complete (or Terminated):** The process completed its execution.
6. **Suspended Ready:** When the ready queue becomes full, some processes are moved to a suspended ready state.
7. **Suspended Block:** When the waiting queue becomes full.



Context Switching of Process

The process of saving the context of one process and loading the context of another process is known as **Context Switching**. In simple terms, it is like loading and unloading the process from the running state to the ready state.

When Does Context Switching Happen?

1. When a high-priority process comes to a ready state (i.e. with higher priority than the running process).
2. An Interrupt occurs.
3. User and kernel-mode switch
4. Preemptive CPU scheduling is used.

Process Schedulers in Operating System

The process scheduling is the activity of the **process manager** that handles the **removal of the running process from the CPU** and the **selection of another process on the basis of some scheduling algorithm**. Process scheduling is an essential part of a Multiprogramming operating systems.

There are three types of process scheduler :-

1. Long Term or job scheduler :

It brings the **new process to the 'Ready State'**. It **controls *Degree of Multi-programming***, i.e., number of process present in ready state at any point of time. It is important that the long-term scheduler makes a careful selection of both I/O and CPU-bound processes. **I/O bound processes** are those which spend much of their time in input and output operations while **CPU bound processes** are those which spend much of their time on CPU. The job scheduler increases efficiency by maintaining a balance between the two.

2. Short term or CPU scheduler :

It is responsible for **selecting one process from ready state for scheduling it on the running state**. Short-term scheduler **only selects the process to schedule, it doesn't load the process on running state**. Here, the scheduling algorithms are used and the CPU scheduler is responsible for ensuring there is **no starvation** owing to high burst time processes.

Dispatcher is responsible for **loading the process selected by Short-term scheduler on the CPU (Ready to Running State)**. Context switching is done by dispatcher only.

A **dispatcher** does the following:

1. Switching context.
2. Switching to user mode.
3. Jumping to the proper location in the newly loaded program.

3. Medium-term scheduler :

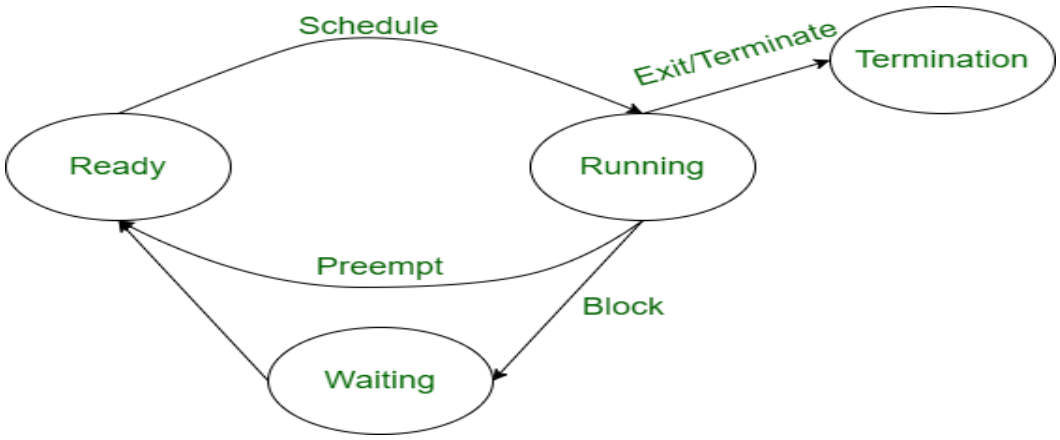
It is responsible for **suspending and resuming the process**. It mainly does **swapping** (moving processes from main memory to disk and vice versa). Swapping may be necessary to improve the process mix. It is helpful in maintaining a **perfect balance** between the **I/O-bound and the CPU-bound processes**. It **reduces** the **degree of multiprogramming**.

Operations on Processes

Following are the operations that are performed while execution of a process:

1. **Creation:** This is the **initial step** of process execution activity. Process creation means the **construction of a new process for the execution**. This might be **performed** by **system, user or old process itself**. There are several events that leads to the process creation. Some of such events are following:
   - When we **start the computer**, **system** creates **several background processes**.
   - A **user** may **request** to create a new process.
   - A **process** can **create a new process itself while executing**.
   - Batch system takes **initiation of a batch job**.
2. **Scheduling/Dispatching:** The **event or activity** in which the **state of the process is changed from ready to running**. It means the operating system puts the process from ready state into the running state. Dispatching is done by operating system when the **resources are free or the process has higher priority than the ongoing process**.
3. **Blocking:** When a **process invokes an input-output system call** the OS blocks the process and put in block mode. Block mode is basically a **mode where process waits for input-output**. Hence on the demand of process itself, **operating system blocks the process and dispatches another process to the processor**. Hence, in **process blocking operation**, the operating system puts the process in **'waiting' state**.
4. **Preemption:** When a **timeout occurs** that means the process hadn't been terminated in the **allotted time interval** and next process is ready to execute, then the **operating system preempts the process**. This operation is **only valid where CPU scheduling supports preemption**. Basically this happens in **priority scheduling** where on the incoming of high priority process the ongoing process is preempted. Hence, in process preemption operation, the operating system puts the process in **'ready' state**.
5. **Termination:** Process termination is the **activity of ending the process**. In other words, process termination is the **relaxation of computer resources taken by the process for the execution**. Like creation, in termination also there may be **several events** that may lead to the process termination. Some of them are:
   - Process **completes its execution fully** and it **indicates** to the **OS that it has finished**.

- **Operating system itself terminates** the process due to **service errors**.
- There may be **problem in hardware** that terminates the process.
- One process can be **terminated** by another process.



What is Cooperating Process?

Cooperating processes are those processes **that depend on other process or processes**. They **work together to achieve a common task** in an operating system. These processes **interact with each other** by **sharing the resources** such as **CPU**, **memory**, and **I/O devices to complete the task**.

Advantages of Cooperating Process in Operating System

1. Information Sharing

Cooperating processes can be used to **share information between various processes**. It could **involve having access to the same files**. A technique is necessary so that the processes may **access the files concurrently**.

2. Modularity

Modularity refers to the **division of complex tasks into smaller subtasks**. **Different cooperating processes can complete these smaller subtasks**. As a result, the **required tasks** are **completed more quickly and efficiently**.

3. Computation Speedup

Cooperating processes can be **used to accomplish subtasks of a single task simultaneously**. It **improves computation speed by allowing the task to be accomplished faster**. Although, it is only possible if the system contains several processing elements.

4. Convenience

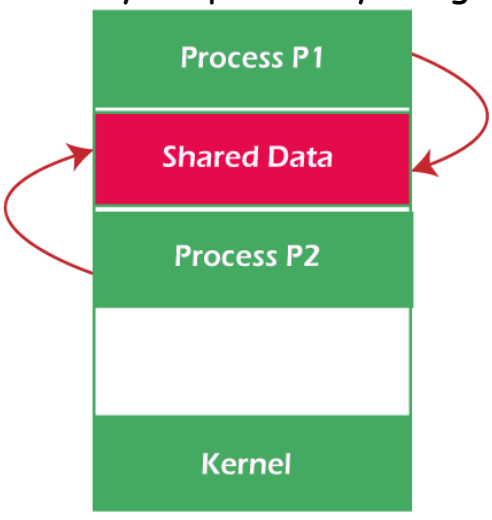
There are **multiple tasks that a user requires** to perform, such as **printing**, **compiling**, **editing**, etc. It is more **convenient** if these activities may be **managed through cooperating processes**.

Methods of Cooperating Process

1. Cooperation by sharing

The processes **may cooperate by sharing data**, including variables, memory, databases, etc. The **critical section provides data integrity**, and **writing is mutually exclusive to avoid inconsistent data.**

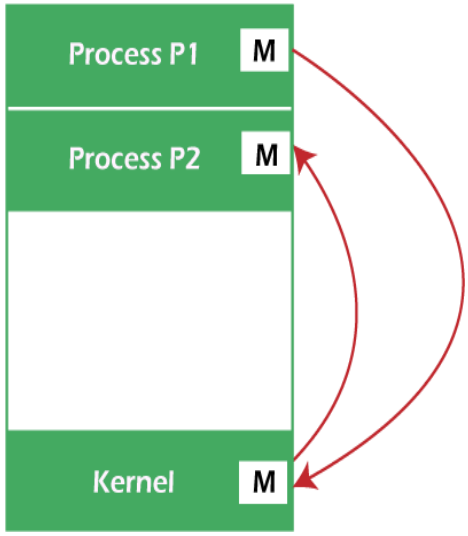
In this diagram, Process P1 and P2 may cooperate by using shared data like databases, files, variables, memory, etc.



2. Cooperation by Communication

The cooperating processes **may cooperate by using messages. If every process waits for a message from another process to execute a task, it may cause a deadlock.** If a process does not receive any messages, it may cause **starvation.**

In this diagram, Process P1 and P2 may cooperate by using messages to communicate.



Inter Process Communication (IPC)

In general, 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. Inter-process communication is **used for exchanging useful information between numerous threads in one or more processes (or programs)**.

Role of Synchronization in Inter Process Communication

It is one of the essential parts of inter process communication. Typically, this is provided by inter process communication control mechanisms, but sometimes it can also be controlled by communication processes.

These are the following methods that used to provide the synchronization:

1. **Mutual Exclusion**
2. **Semaphore**
3. **Barrier**
4. **Spinlock**

1. Mutual Exclusion:-

It is generally required that only one process thread can enter the critical section at a time. This also helps in synchronization and creates a stable state to avoid the race condition.

2.Semaphore:-

Semaphore is a type of variable that usually controls the access to the shared resources by several processes. Semaphore is further divided into two types which are as follows:

- Binary Semaphore
- Counting Semaphore

3.Barrier:-

A barrier typically not allows an individual process to proceed unless all the processes does not reach it. It is used by many parallel languages, and collective routines impose barriers.

4.Spinlock:-

Spinlock is a type of lock as its name implies. The processes are trying to acquire the spinlock waits or stays in a loop while checking that the lock is available or not. It is known as **busy waiting** because even though the process active, the process does not perform any functional operation (or task).

Approaches to Interprocess Communication

We will now discuss some different approaches to inter-process communication which are as follows:

These are a few different approaches for Inter- Process Communication:

1. **Pipes**
2. **Shared Memory**
3. **Message Queue**
4. **Direct Communication**
5. **Indirect communication**
6. **Message Passing**
7. **FIFO**

Pipe:-

The pipe is a type of data channel that is unidirectional in nature. It means that the data in this type of data channel can be moved in only a single direction at a time. Still, one can use two-channel of this type, so that he can able to send and receive data in two processes. Typically, it uses the standard methods for input and output. These pipes are used in all types of POSIX systems and in different versions of window operating systems as well.

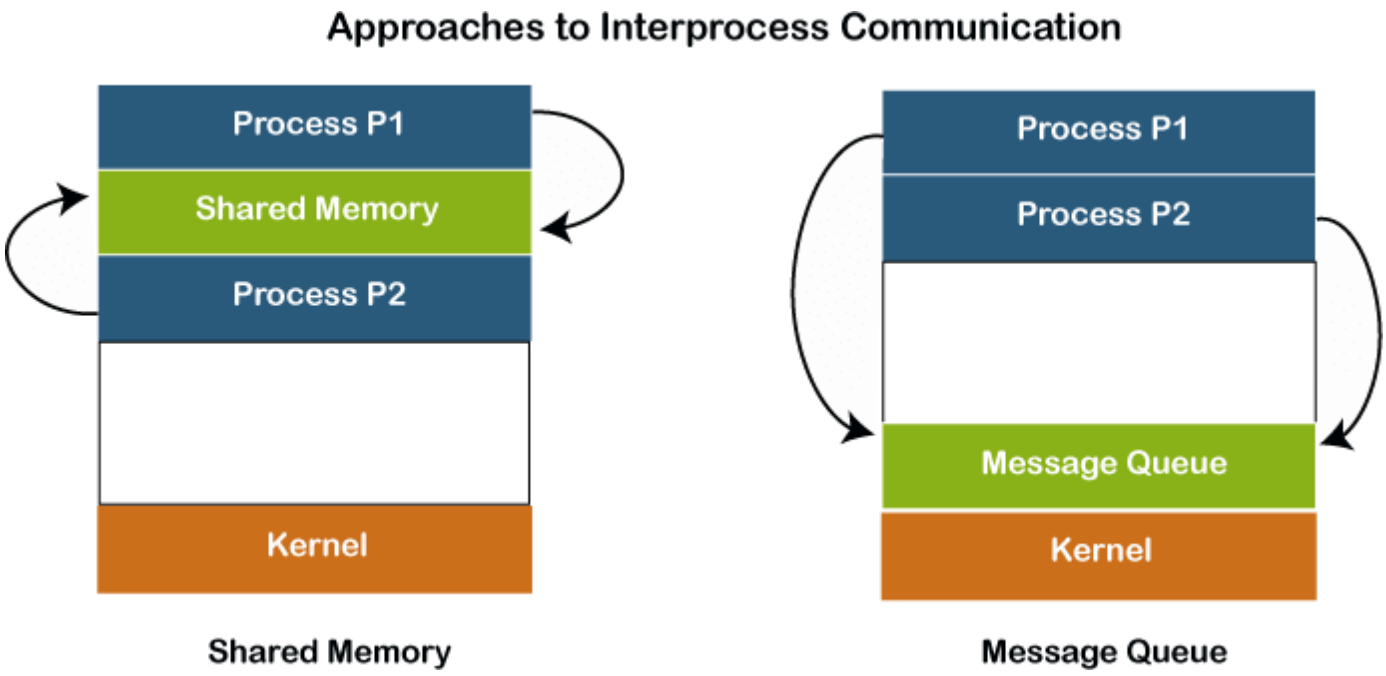
Shared Memory:-

It can be referred to as a type of memory that can be used or accessed by multiple processes simultaneously. It is primarily used so that the processes can communicate with each other. Therefore the shared memory is used by almost all POSIX and Windows operating systems as well.

Message Queue:-

In general, several different messages are allowed to read and write the data to the message queue. In the message queue, the messages are stored or stay in the queue unless their recipients retrieve them. In short, we can also say that the message queue is very helpful in inter-process communication and used by all operating systems.

To understand the concept of Message queue and Shared memory in more detail, let's take a look at its diagram given below:



Message Passing:-

It is a type of mechanism that allows processes to synchronize and communicate with each other. However, by using the message passing, the processes can communicate with each other without restoring the hared variables.

Usually, the inter-process communication mechanism provides two operations that are as follows:

- send (message)
- received (message)

Direct Communication:-

In this type of communication process, usually, a link is created or established between two communicating processes. However, in every pair of communicating processes, only one link can exist.

Indirect Communication:-

Indirect communication can only exist or be established when processes share a common mailbox, and each pair of these processes shares multiple communication links. These shared links can be unidirectional or bi-directional.

FIFO:-

It is a type of general communication between two unrelated processes. It can also be considered as full-duplex, which means that one process can communicate with another process and vice versa.

Some other different approaches

- **Socket:-**
  It acts as a type of endpoint for receiving or sending the data in a network. It is correctfor data sent between processes on the same computer or data sent between different computers on the same network. Hence, it used by several types of operating systems.
- **File:-**
  A file is a type of data record or a document stored on the disk and can be acquired on demand by the file server. Another most important thing is that several processes canaccess that file as required or needed.
- **Signal:-**
  As its name implies, they are a type of signal used in inter process communication in a minimal way. Typically, they are the massages of systems that are sent by one processto another. Therefore, they are not used for sending data but for remote commands between multiple processes.

Operating Systems Client/Server Communication

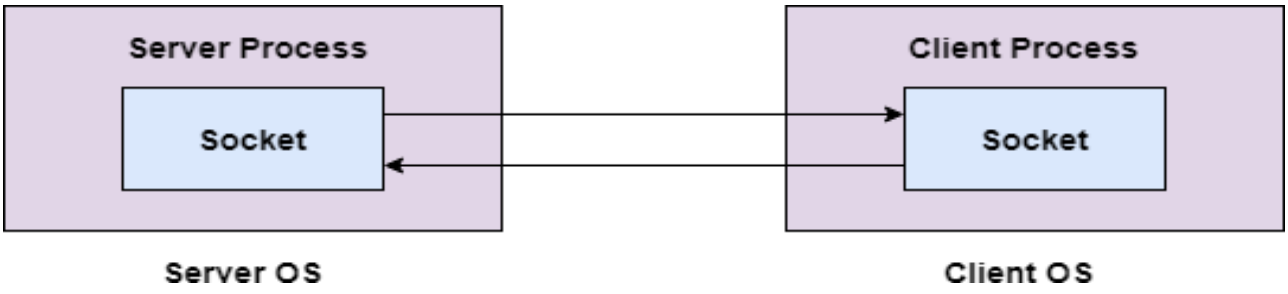
Client/Server communication **involves two components**, namely **a client and a server**. There are usually multiple clients in communication with a single server. The **clients send requests to the server and the server responds to the client requests**.

There are three main methods to client/server communication :-

1.Sockets

Sockets **facilitate communication between two processes on the same machine or different machines**. They are **used in a client/server framework and consist of the IP address and port number**. Many application protocols **use sockets for data connection and data transfer** between a client and a server.

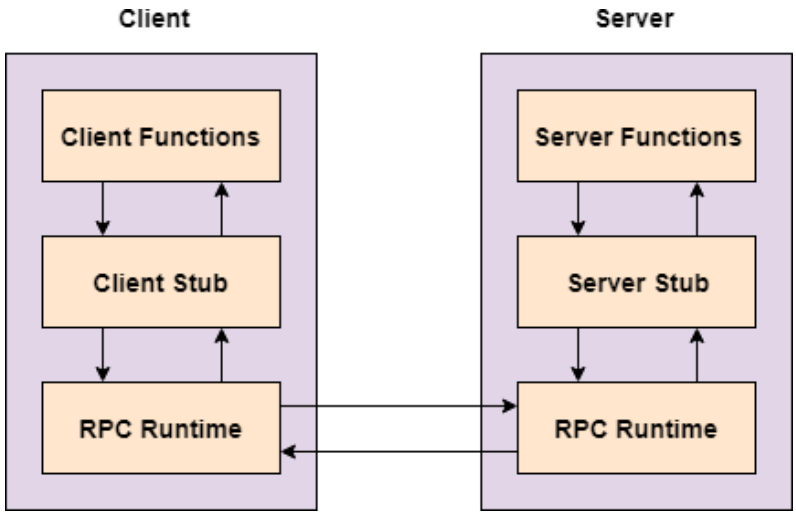
Socket communication is **quite low-level as sockets only transfer an unstructured byte stream across processes**. The **structure** on the byte stream is **imposed** by the **client and server applications**.



2.Remote Procedure Calls

These are interprocess communication techniques that are used for client-server based applications. A remote procedure call is also known as a **subroutine call or a function call**.

A client has a request that the RPC translates and sends to the server. This **request may be a procedure or a function call to a remote server**. When the **server receives the request**, it **sends the required response back to the client**.



3.Pipes

These are interprocess communication **methods** that **contain two end points**. **Data is entered from one end of the pipe by a process and consumed from the other end by the other process**.

The two different types of pipes are **ordinary pipes** and **named pipes**. ***Ordinary pipes*** only **allow one way communication**. For two way communication, two pipes are required. Ordinary pipes have **a parent child relationship between the processes as the pipes can only be accessed by processes that created or inherited them**.

Named pipes are **more powerful** than ordinary pipes and **allow two way communication**. These pipes **exist even after the processes using them have terminated**. They **need to be explicitly deleted** when not required anymore.

What is a Thread?

A thread is a **single sequence stream within a process**. Threads are also called **lightweight processes** as they **possess** some of the **properties of processes**. **A process can contain multiple threads but each thread belongs to exactly one process**. In an operating system that supports multithreading, the process can consist of many threads. But threads can be effective only if the CPU is more than 1 otherwise two threads have to context switch for that single CPU.

Process vs Thread?

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 **dependent on each another within a process** and they **share** their **code section, data section, and OS resources** (like open files and signals) with other threads. But, like process, a **thread** has **its own program counter** (PC), **register set**, and **stack space**.

Advantages of Thread over Process

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 so they **do not need to use inter-process communication**.

Note: stack and registers can't be shared among the threads. Each thread has its own stack and registers.

Components of Threads

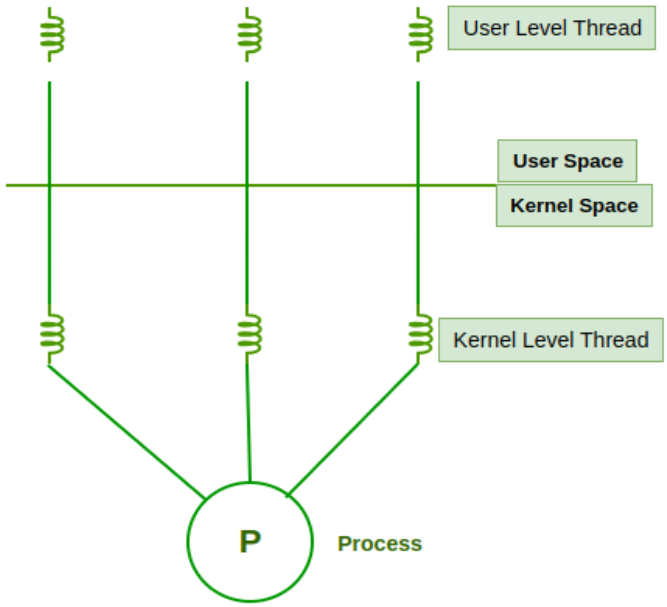
These are the basic components of the Operating System.

- Stack Space
- Register Set
- Program Counter

Types of Thread in Operating System

Threads are of two types. These are described below.

- User Level Thread
- Kernel Level Thread



1. User Level Threads

User Level Thread is **a type of thread that is not created using system calls**. The **kernel has no work in the management of user-level threads**. User-level threads can be easily implemented by the **user**. In case when user-level threads are single-handed processes, kernel-level thread manages them.

Advantages of User-Level Threads

- **Implementation** of the User-Level Thread is **easier** than Kernel Level Thread.
- **Context Switch Time** is **less** in User Level Thread.
- User-Level Thread is **more efficient** than Kernel-Level Thread.
- Because of the presence of **only Program Counter, Register Set, and Stack Space**, it has a **simple representation**.

Disadvantages of User-Level Threads

- There is a **lack of coordination between Thread and Kernel**.
- In **case** of a **page fault**, the **whole process can be blocked**.

2. Kernel Level Threads

A kernel Level Thread is **a type of thread that can recognize the Operating system easily**. Kernel Level Threads has **its own thread table where it keeps track of the system**. The operating System Kernel helps in **managing threads**. Kernel Threads have **longer context switching time**.

Advantages of Kernel-Level Threads

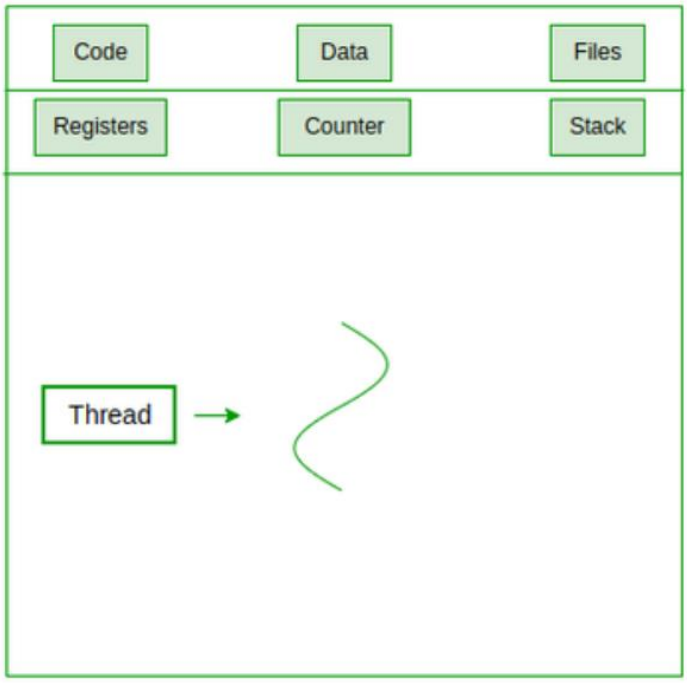
- It has **up-to-date information on all threads**.
- **Applications that block frequency** are to be **handled by the Kernel-Level Threads**.
- Whenever **any process requires more time to process**, Kernel-Level Thread **provides more time to it**.

Disadvantages of Kernel-Level threads

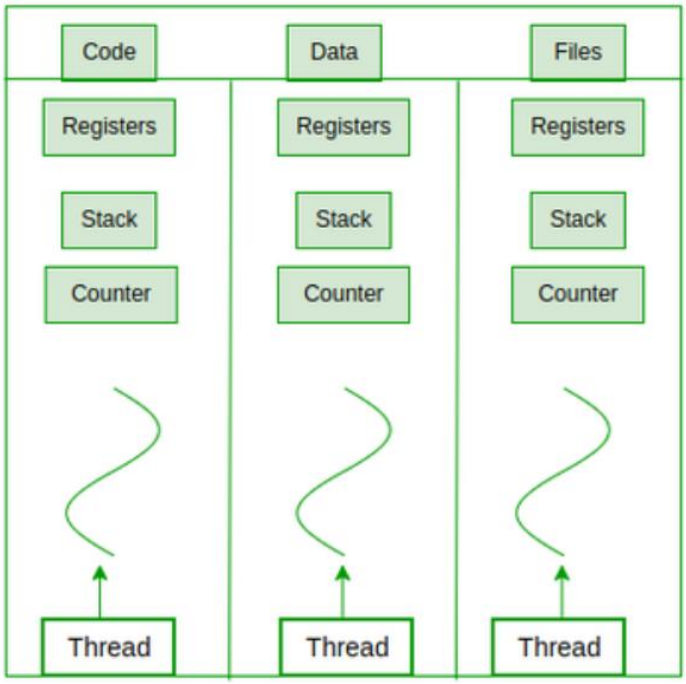
- Kernel-Level Thread is **slower** than User-Level Thread.
- **Implementation** of this type of thread is a **little more complex** than a user-level thread.

Why Multithreading?

A thread is a lightweight process. The idea is to **achieve parallelism by dividing a process into multiple threads**. Multithreading is a **technique** used in operating systems **to improve the performance and responsiveness of computer systems**.
Multithreading **allows multiple threads** (i.e., lightweight processes) **to share the same resources of a single process**, such as the **CPU**, **memory**, and **I/O devices**.
The **main drawback of single threading systems** is that **only one task can be performed at a time**, so to overcome the drawback of this single threading, there is **multithreading** that **allows multiple tasks to be performed**.



Single Threaded Process



Multi Threaded Process

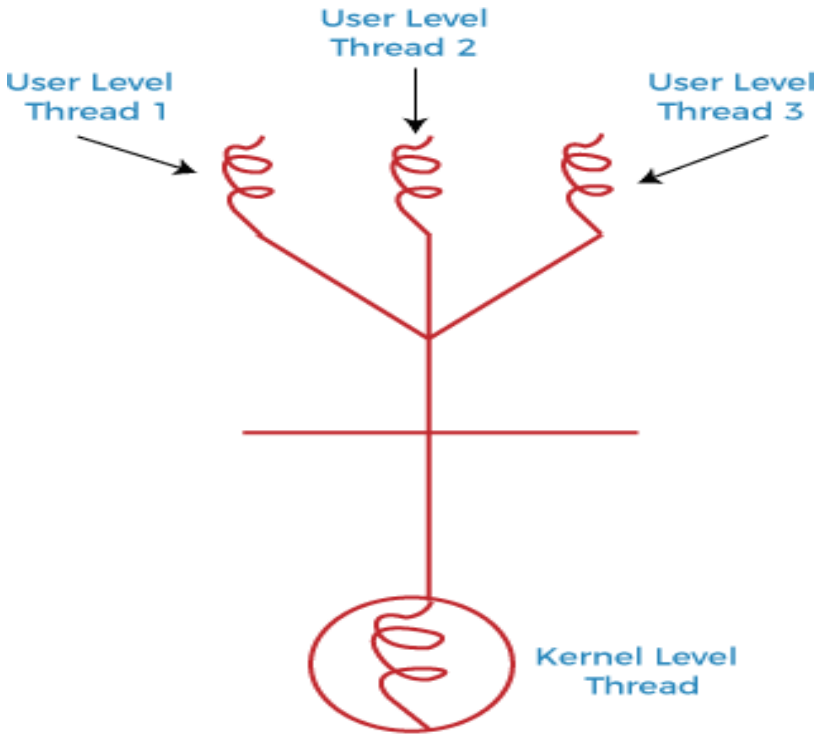
Multithreading Models in Operating system

Multi threading model are of three types :-

- Many to one multithreading model
- One to one multithreading model
- Many to Many multithreading models

1. ***Many to one multithreading model***

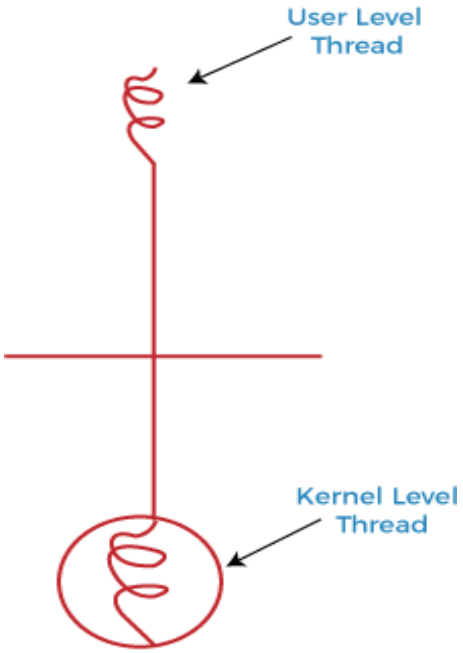
In this model, we have **multiple user threads mapped to one kernel thread**. This type of relationship **facilitates an effective context-switching environment**, **easily implemented** even on the simple kernel with no thread support.
As we have **only one kernel thread and only one user thread can access kernel at a time**, so **multiple threads are not able to access multiprocessor at the same time** that's why 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** so it is more efficient but **if blocking comes, this model blocks the whole system**.



Many-to-one model

2. ***One to one multithreading model***

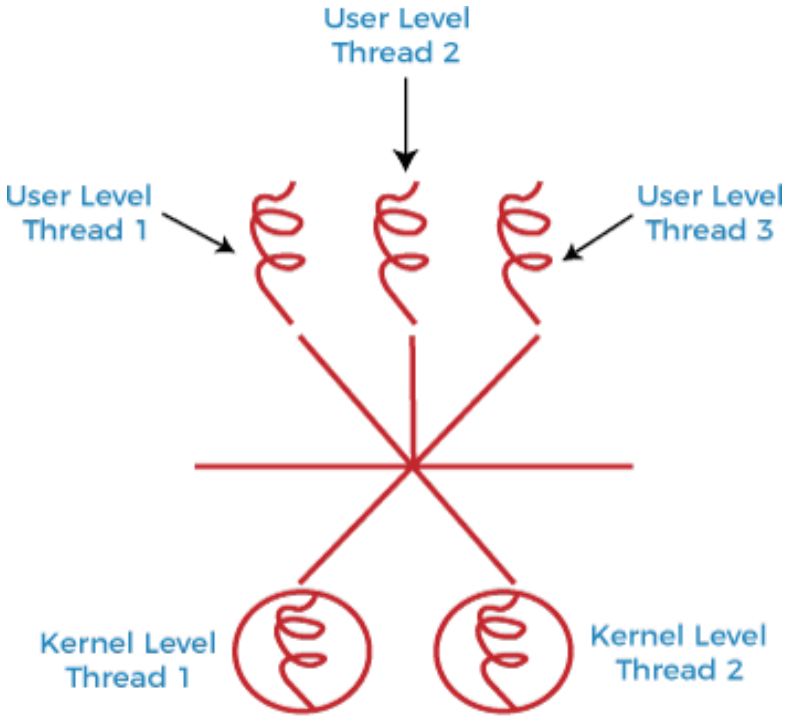
The one-to-one model **maps a single user-level thread to a single kernel-level thread**. This type of relationship **facilitates the running of multiple threads in parallel**. **Problem** with this model is that **creating a user thread requires the corresponding kernel thread causing an overhead**, which can **hinder the performance of the parent process**. 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**.



One-to-one model

3. Many to Many multithreading models

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 many to many model is a **combination of 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 like the other models**. It is the **best multi threading model**.



Many-to-Many model

DIFFERENCE BETWEEN PROCESS AND THREAD

S.no.	PROCESS	THREAD
1.	Process means **any program in execution**.	Thread means **a segment of a process**.
2.	The process takes **more time to terminate**.	The thread takes **less time to terminate**.
x3.	It takes **more time for creation**.	It takes **less time for creation**.
4.	It also takes **more time for context switching**.	It takes **less time for context switching**.
5.	The process is **less efficient** in terms of **communication**.	Thread is **more efficient** in terms of **communication**.
6.	**Multiprogramming** holds the concepts of **multi-process**.	We **don't need multi programs** in action for multiple threads because a **single process consists of multiple threads**.
7.	The process is **isolated**.	Threads **share memory**.
8.	The process is called the **heavyweight process**.	A Thread is **lightweight** as **each thread in a process shares code, data, and resources**.
9.	If **one process is blocked** then it **will not affect the execution of other processes**	If a **user-level thread is blocked**, then **all other user-level threads are blocked**.
10.	The process has its **own Process Control Block, Stack, and Address Space**.	Thread has **Parents' PCB**, its **own Thread Control Block, Stack** and **common Address space**.
11.	The process **does not share data** with each other.	Threads **share data** with each other.

Threading Issues in OS

1. fork() and exec() System Calls

The fork() and exec() are the system calls. The fork() call creates a duplicate process of the process that invokes fork(). The new duplicate process is called child process and process invoking the fork() is called the parent process. Both the parent process and the child process continue their execution from the instruction that is just after the fork().

Let us now discuss the issue with the fork() system call. Consider that a thread of the multithreaded program has invoked the fork(). So, the fork() would create a new duplicate process. Here the issue is whether the new duplicate process created by fork() will duplicate all the threads of the parent process or the duplicate process would be single-threaded.

Well, there are two versions of fork() in some of the UNIX systems. Either the fork() can duplicate all the threads of the parent process in the child process or the fork() would only duplicate that thread from parent process that has invoked it. Which version of fork() must be used totally depends upon the application.

Next system call i.e. exec() system call when invoked replaces the program along with all its threads with the program that is specified in the parameter to exec(). Typically the exec() system call is lined up after the fork() system call.

Here the issue is if the exec() system call is lined up just after the fork() system call then duplicating all the threads of parent process in the child process by fork() is useless. As the exec() system call will replace the entire process with the process provided to exec() in the parameter.

In such case, the version of fork() that duplicates only the thread that invoked the fork() would be appropriate.

2. Thread cancellation

Termination of the thread in the middle of its execution it is termed as 'thread cancellation'. Let us understand this with the help of an example. Consider that thereis a multithreaded program which has let its multiple threads to search through a database for some information. However, if one of the thread returns with the desired result the remaining threads will be cancelled.

Now a thread which we want to cancel is termed as target thread. Thread cancellation can be performed in two ways:

Asynchronous Cancellation: In asynchronous cancellation, a thread is employed to terminate the target thread instantly.

Deferred Cancellation: In deferred cancellation, the target thread is scheduled tocheck itself at regular interval whether it can terminate itself or not.

The issue related to the target threads are listed below:

- What if the resources had been allotted to the cancel target thread?
- What if the target thread is terminated when it was updating the data, it was sharingwith some other thread.

Here the asynchronous cancellation of the thread where a thread immediately cancelsthe target thread without checking whether it is holding any resources or not createstroublesome.

However, in deferred cancellation, the thread that indicates the target thread aboutthe cancellation, the target thread crosschecks its flag in order to confirm that it should it be cancelled immediately or not. The thread cancellation takes place where they can be cancelled safely such points are termed as **cancellation points** by Pthreads.

3. ***Signal Handling***

Signal handling is more convenient in the single-threaded program as the signal wouldbe directly forwarded to the process. But when it comes to multithreaded program, the issue arrives to which thread of the program the signal should be delivered.

Let's say the signal would be delivered to:

- All the threads of the process.
- To some specific threads in a process.
- To the thread to which it applies
- Or you can assign a thread to receive all the signals.

Well, how the signal would be delivered to the thread would be decided, depending upon the type of generated signal. The generated signal can be classified into two type's synchronous signal and asynchronous signal.

Synchronous signals are forwarded to the same process that leads to the generationof the signal. Asynchronous signals are generated by the event external to the running process thus the running process receives the signals asynchronously. So if the signal is synchronous it would be delivered to the specific thread causing the generation of the signal. If the signal is asynchronous it cannot be specified towhich thread of the multithreaded program it would be delivered. If the asynchronous signal is notifying to terminate the process the signal would be delivered to all the thread of the process.

The issue of an asynchronous signal is resolved up to some extent in most of the multithreaded UNIX system. Here the thread is allowed to specify which signal it can accept and which it cannot. However, the Window operating system does not support the concept of the signal instead it uses asynchronous procedure call (ACP) which is similar to the asynchronous signal of the UNIX system.

UNIX allow the thread to specify which signal it can accept and which it will notwhereas the ACP is forwarded to the specific thread.

4. ***Thread Pool***

When a user requests for a webpage to the server, the server creates a separate thread to service the request. Although the server also has some potential issues. Consider if we do not have a bound on the number of actives thread in a system andwould create a new thread for every new request then it would finally result in exhaustion of system resources.

We are also concerned about the time it will take to create a new thread. It must notbe that case that the time require to create a new thread is more than the time required by the thread to service the request and then getting discarded as it would result in wastage of CPU time.

The solution to this issue is the **thread pool**. The idea is to create a finite amount ofthreads when the process starts. This collection of threads is referred to as the thread pool. The threads stay in the thread pool and wait till they are assigned any request to be serviced.

Whenever the request arrives at the server, it invokes a thread from the pool andassigns it the request to be serviced. The thread completes its service and returnback to the pool and wait for the next request.

If the server receives a request and it does not find any thread in the thread pool it waits for some or the other thread to become free and return to the pool. This muchbetter than creating a new thread each time a request arrives and convenient for thesystem that cannot handle a large number of concurrent threads.

5. ***Thread Specific data***

We all are aware of the fact that the threads belonging to the same process share the data of that process. Here the issue is what if each particular thread of the process needs its own copy of data. So the specific data associated with the specificthread is referred to as **thread-specific data**.

Consider a transaction processing system, here we can process each transaction in adifferent thread. To determine each transaction uniquely we will associate a unique identifier with it. Which will help the system to identify each transaction uniquely.

As we are servicing each transaction in a separate thread. So we can use thread-specific data to associate each thread to a specific transaction and its unique id.

Thread libraries such as Win32, Pthreads and Java support to thread-specific data.So these are threading issues that occur in the multithreaded programming environment. We have also seen how these issues can be resolved.

UNIT – 5

Cryptography

Cryptography is technique of securing information and communications through use of codes so that only those person for whom the information is intended can understand it and process it. Cryptography plays a crucial role in securing data and communications within operating systems. Thus preventing unauthorized access to information. The prefix "crypt" means "hidden" and suffix "graphy" means "writing". In Cryptography the techniques which are use to protect information are obtained from mathematical concepts and a set of rule based calculations known as algorithms to convert messages in ways that make it hard to decode it. These algorithms are used for cryptographic key generation, digital signing, verification to protect data privacy, web browsing on internet and to protect confidential transactions such as credit card and debit card transactions.

Techniques of Cryptography

Encryption: The process where an ordinary plain text is converted to cipher text such that only the intended receiver of the text can decode it .
Decryption: The process of conversion of cipher text into plain text .

Features of Cryptography

1. **Confidentiality:** Information can only be accessed by the person for whom it is intended and no other person except him can access it.
2. **Integrity:** Information cannot be modified in storage or transition between sender and intended receiver without any addition to information being detected.
3. **Non-repudiation:** The creator/sender of information cannot deny his intention to send information at later stage.
4. **Authentication:** The identities of sender and receiver are confirmed. As well as destination/origin of information is confirmed.

Types/Methods Of Cryptography

1. **Symmetric Key Cryptography:** It is an encryption system where the sender and receiver of message use a single common key to encrypt and decrypt messages. Symmetric Key Systems are faster and simpler but the problem is that sender and receiver have to somehow exchange key in a secure manner.
   Example:- Encryption System(DES) and Advanced Encryption System(AES) and Triple DES (3DES).
2. **Hash Functions:** There is no usage of any key in this algorithm. A hash value with fixed length is calculated as per the plain text which makes it impossible for

contents of plain text to be recovered. Many operating systems use hash functions to encrypt passwords.
Example:- SHA-256 (Secure Hash Algorithm 256-bit), MD5 (Message Digest Algorithm 5).

3. **Asymmetric Key Cryptography:** Under this system a pair of keys is used to encrypt and decrypt information. A receiver's public key is used for encryption and a receiver's private key is used for decryption. Public key and Private Key are different. Even if the public key is known by everyone the intended receiver can only decode it because he alone know his private key. The most popular asymmetric key cryptography algorithm is RSA algorithm.
Example:- RSA (Rivest-Shamir-Adleman), Elliptic Curve Cryptography (ECC).

Applications Of Cryptography

1. **Computer passwords:** Cryptography is widely utilized in computer security, particularly when creating and maintaining passwords. When a user logs in, their password is hashed and compared to the hash that was previously stored. Passwords are hashed and encrypted before being stored. In this technique, the passwords are encrypted so that even if a hacker gains access to the password database, they cannot read the passwords.
2. **Digital Currencies:** To safeguard transactions and prevent fraud, digital currencies like Bitcoin also use cryptography. Complex algorithms and cryptographic keys are used to safeguard transactions, making it nearly hard to tamper with or forge the transactions.
3. **Secure web browsing:** Online browsing security is provided by the use of cryptography, which shields users from eavesdropping and man-in-the-middle assaults. Public key cryptography is used by the Secure Sockets Layer (SSL) and Transport Layer Security (TLS) protocols to encrypt data sent between the web server and the client, establishing a secure channel for communication.
4. **Electronic signatures:** Electronic signatures serve as the digital equivalent of a handwritten signature and are used to sign documents. Digital signatures are created using cryptography and can be validated using public key cryptography. In many nations, electronic signatures are enforceable by law, and their use is expanding quickly.
5. **Authentication:** Cryptography is used for authentication in many different situations, such as when accessing a bank account, logging into a computer, or using a secure network. Cryptographic methods are employed by authentication protocols to confirm the user's identity and confirm that they have the required access rights to the resource.
6. **Cryptocurrencies:** Cryptography is heavily used by cryptocurrencies like Bitcoin and Ethereum to safeguard transactions, thwart fraud, and maintain the network's

integrity. Complex algorithms and cryptographic keys are used to safeguard transactions, making it nearly hard to tamper with or forge the transactions.

7. **End-to-End Encryption:** End-to-end encryption is used to protect two-way communications like video conversations, instant messages, and email. Even if the message is encrypted, it assures that only the intended receivers can read the message. End-to-end encryption is widely used in communication apps like WhatsApp and Signal, and it provides a high level of security and privacy for users.

Advantages

1. **Access Control:** Cryptography can be used for access control to ensure that only parties with the proper permissions have access to a resource. Only those with the correct decryption key can access the resource thanks to encryption.
2. **Secure Communication:** For secure online communication, cryptography is crucial. It offers secure mechanisms for transmitting private information like passwords, bank account numbers, and other sensitive data over the internet.
3. **Protection against attacks:** Cryptography aids in the defence against various types of assaults, including replay and man-in-the-middle attacks. It offers strategies for spotting and stopping these assaults.
4. **Compliance with legal requirements:** Cryptography can assist firms in meeting a variety of legal requirements, including data protection and privacy legislation.

Access Matrix

Access Matrix is a **security model of protection state in computer system.** An access matrix **helps in the protection of operating system and is represented by a two-dimensional matrix.**

Access matrix is used to **define the rights of each process executing in the domain with respect to each object.** The **rows of matrix represent domains and columns represent objects. Each cell of matrix represents set of access rights which are given to the processes.**

Different types of rights:

There are different types of rights the files can have. The most common ones are:

1. **Read**- This is a right given to a process in a domain, which allows it to read the file.
2. **Write**- Process in domain can write into the file.
3. **Execute**- Process in domain can execute the file.
4. **Print**- Process in domain only has access to printer.

Sometimes, domains can have more than one right, i.e. combination of rights mentioned above.

Let us now understand how an access matrix works from the example given below.

Object / Domain	F_1	F_2	F_3	Laser Printer
D_1	read		read	
D_2				Print
D_3		read	exceute	
D_4	read / write		read / write	

Access Matirx

Observations of above matrix:

- There are **four domains** and **four objects**– three files(F1, F2, F3) and one printer.
- A process executing in D1 can read files F1 and F3.
- A process executing in domain D4 has same rights as D1 but it can also write on files.
- Printer can be accessed by only one process executing in domain D2.
- A process executing in domain D3 has the right to read file F2 and execute file F3.

Switch operation: When we switch a process from one domain to another, we execute a switch operation on an object(the domain). We can control domain switching by including domains among the objects of the access matrix. Processes should be able to switch from one domain (Di) to another domain (Dj) if and only if a switch right is given to access(i, j). This is explained using an example below:

	F1	F2	F3	Printer	D1	D2	D3	D4
D1	read		read			switch		
D2				print			switch	switch
D3		read	execute					
D4	read write		read write		switch			

According to the above matrix, a process executing in domain D2 can switch to domain D3 and D4. A process executing in domain D4 can switch to domain D1 and process executing in domain D1 can switch to domain D2.

There are various methods of implementing the access matrix in the operating system. These methods are as follows:

1. **Global Table**
2. **Access Lists for Objects**
3. **Capability Lists for Domains**
4. **Lock-Key Mechanism**

Access Lists for Objects

Every access matrix column may be used as a single object's access list. It is possible to delete the blank entries. For each object, the resulting list contains ordered pairs **<domain, rights-set>** that define all domains for that object and a nonempty set of access rights.

We may start by checking the default set and then find the access list. If the item is found, we enable the action; if it isn't, we verify the default set. If M is in the default set, we grant access. Access is denied if this is not the case, and an extraordinary scenario arises.

Global Table

It is the most basic access matrix implementation. A set of ordered triples **<domain, object, rights-set>** is maintained in a file. When an operation **M** has been performed on an object Oj within domain Di, the table is searched for a triple **<Di, Oj, Rk>**. The operation can proceed if this triple is located; otherwise, an exception (or error) condition has arrived. This implementation has various drawbacks. The table is generally large and cannot be stored in the main memory, so additional input and output are required.

Capability Lists for Domains

A domain's capability list is a collection of objects and the actions that can be done on them. A capacity is a name or address that is used to define an object. If you want to perform operation M on object **Oj,** the process runs operation M, specifying the capability for object **Oj.** The simple possession of the capability implies that access is allowed.

In most cases, capabilities are separated from other data in one of two ways. Every object has a tag to indicate its type as capability data. Alternatively, a program's address space can be divided into two portions. The programs may access one portion, including the program's normal instructions and data. The other portion is a capability list that is only accessed by the operating system.

Lock-Key Mechanism

It is a compromise between the access lists and the capability lists. Each object has a list of locks, which are special bit patterns. On the other hand, each domain has a set of keys that are special bit patterns. A domain-based process could only access an object if a domain has a key that satisfies one of the locks on the object. The process is not allowed to modify its keys.