Chapter 2: Process Management

Contents:

3.1 Process: Process concept, Process States, process control block.

Process scheduling: Scheduling Queues, Schedulers, context switch

IPC Inter process communication: Introduction, Shared Memory System, Message passing system

Thread: concept, Benefits, Users and kernel threads, multithreading models: Many to one, One to one, Many to Many

Execute process Commands: ps, wait, sleep, kill, and exit

3.1 Process:

3.1.1 Process Concept

- A process is basically a program in execution.
- The execution of a process must progress in a sequential fashion.
- An operating system executes a variety of programs that run as a process.
- No parallel execution of instructions of a single process
- A process is defined as an entity which represents the basic unit of work to be implemented in the system. To put it in simple terms, we write our computer programs in a text file and when we execute this program, it becomes a process which performs all the tasks mentioned in the program.
- When a program is loaded into the memory and it becomes a process, it can be divided into four sections ─ stack, heap, text and data.
- The following image shows a simplified layout of a process inside main memory.

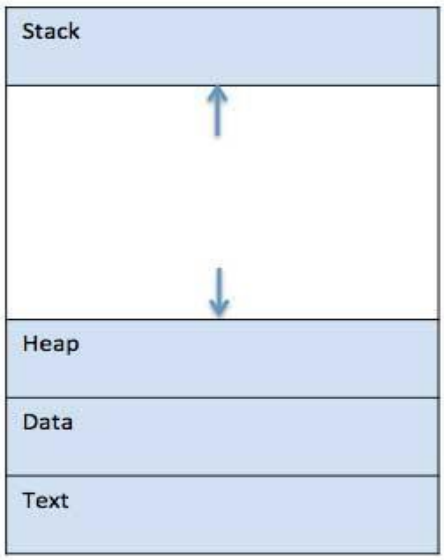


Fig.Process in Memory

- Multiple parts of process
    - The program code, also called **text section**
    - Current activity including **program counter**, processor registers
    - **Data section** containing global variables
    - **Heap** containing memory dynamically allocated during run time
    - **Text:** This includes the current activity represented by the value of Program Counter and the contents of the processor's registers.
    - **Stack**: The process Stack contains the temporary data such as method/function parameters, return address and local variables.
- Program is **passive** entity stored on disk (**executable file**); process is **active entity**
- Program becomes process when an executable file is loaded into memory
- Execution of program started via GUI mouse clicks, command line entry of its name, etc.
- One program can be several processes
- Consider multiple users executing the same program

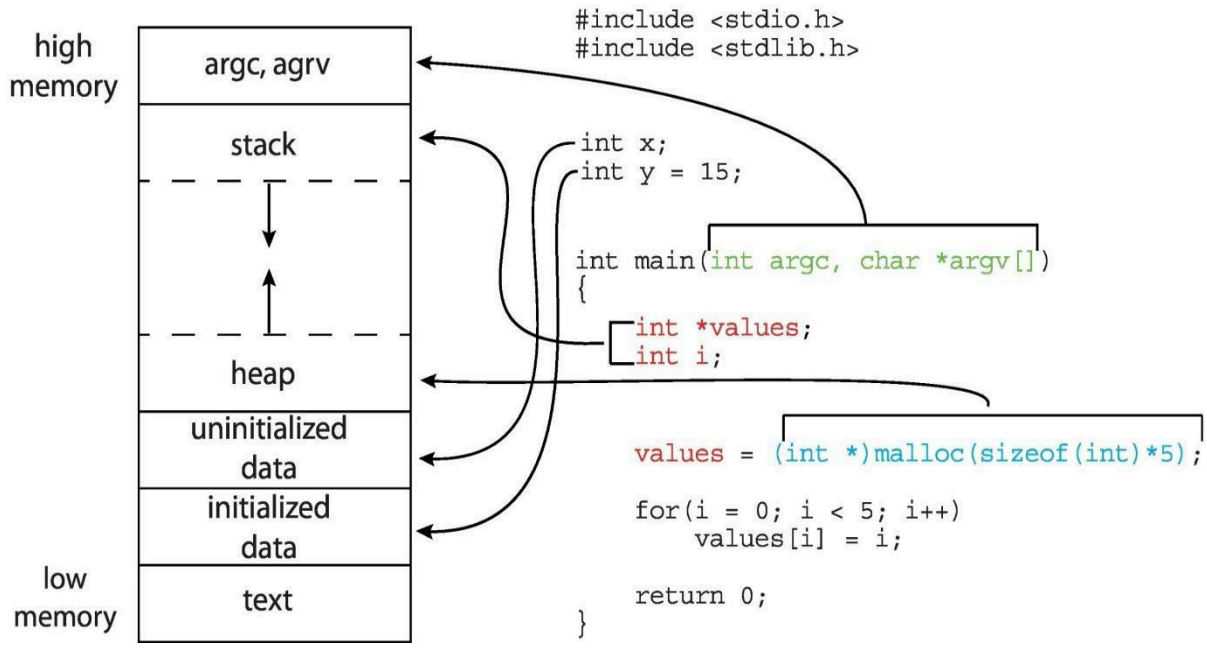


Fig : Memory Layout of a C Program

3.1.2 Process states (Process Life Cycle):

When a process executes, it passes through different states. As a process executes, it changes **state**

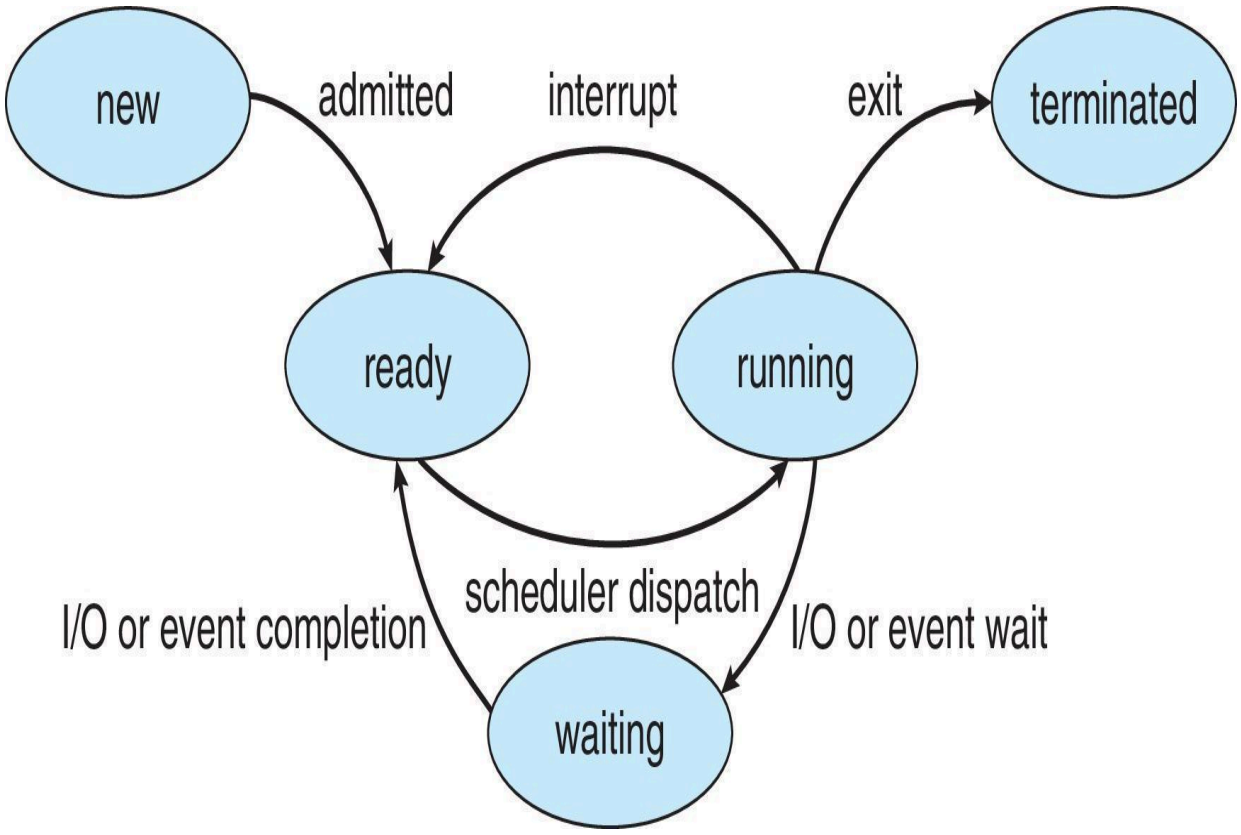


Fig. :- Process states

New: This is the initial state when a process is first created.

Ready: The process is waiting to be assigned to a processor. Ready processes are waiting to have the processor allocated to them by the operating system so that they can run. Process may come into this state after new state or while running it by but interrupted by the scheduler to assign CPU to some other process.

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

Waiting: Process moves into the waiting state if it needs to wait for a resource, such as waiting for user input, or waiting for a file to become available.

Terminated: Once the process finishes its execution, or it is terminated by the operating system, it is moved to the terminated state where it waits to be removed from main memory.

3.1.3 Process Control Block (PCB):

- A Process Control Block is a data structure maintained by the Operating System for every process.
- PCB contain Information associated with each process. It is also called **task control block**

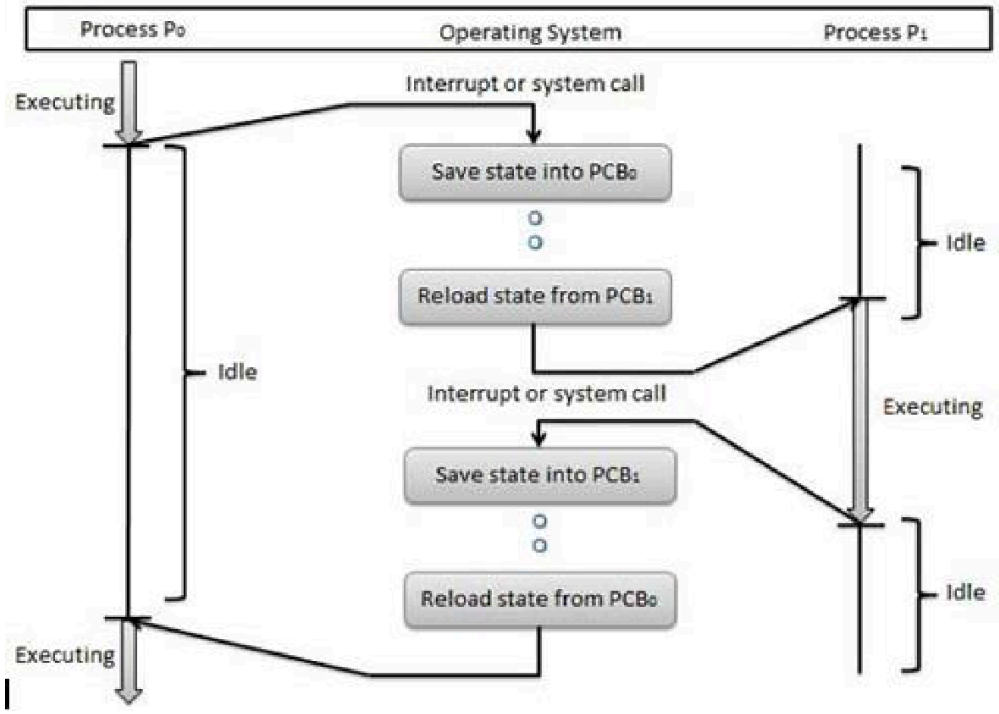
- The PCB is identified by an integer process ID (PID).
- A PCB keeps all the information needed to keep track of a process as listed below

1	**Process State** The current state of the process i.e., whether it is ready, running, waiting, or whatever.
2	**Process privileges** This is required to allow/disallow access to system resources.
3	**Process ID** Unique identification for each of the process in the operating system.
4	**Pointer** A pointer to parent process.
5	**Program Counter**

	Program Counter is a pointer to the address of the next instruction to be executed for this process.
6	**CPU registers** Various CPU registers where process need to be stored for execution for running state.
7	**CPU Scheduling Information** Process priority and other scheduling information which is required to schedule the process.
8	**Memory management information** This includes the information of page table, memory limits, Segment table depending on memory used by the operating system.
9	**Accounting information** This includes the amount of CPU used for process execution, time limits, execution ID etc.
10	**IO status information** This includes a list of I/O devices allocated to the process.

Context switching:

A CPU switch from process to process is referred as context switch. A context switch is a mechanism that store and restore the state or context of a CPU in Process Control block so that a process execution can be resumed from the same point at a later time. When the scheduler switches the CPU from one process to another process, the context switch saves the contents of all process registers for the process being removed from the CPU, in its process control block. Context switch includes two operations such as state save and state restore. State save operation stores the current information of running process into its PCB. State restore operation restores the information of process to be executed from its PCB. Switching the CPU from one process to another process requires performing state save operation for the currently executing process (blocked) and a state restore operation for the process ready for execution. This task is known as context switch.



Process scheduling:

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 a particular strategy.

Process scheduling is an essential part of a Multiprogramming operating systems. Such operating systems allow more than one process to be loaded into the executable memory at a time and the loaded process shares the CPU using time multiplexing.

Process Scheduling Queues

The OS maintains all PCBs in Process Scheduling Queues. The OS maintains a separate queue for each of the process states and PCBs of all processes in the same execution state are placed in the same queue. When the state of a process is changed, its PCB is unlinked from its current queue and moved to its new state queue.

The Operating System maintains the following important process scheduling queues −

- **Job queue** − This queue keeps all the processes in the system.
- **Ready queue** − This queue keeps a set of all processes residing in main memory, ready and waiting to execute. A new process is always put in this queue.
- **Device queues** − The processes which are blocked due to unavailability of an I/O device constitute this queue.

Schedulers:

Schedulers are special system software which handle process scheduling in various ways. Their main task is to select the jobs to be submitted into the system and to decide which process to run. Schedulers are of three types −

- Long-Term Scheduler
- Short-Term Scheduler
- Medium-Term Scheduler

Long Term Scheduler

It is also called a **job scheduler**. A long-term scheduler determines which programs are admitted to the system for processing. It selects processes from the queue and loads them into memory for execution. Process loads into the memory for CPU scheduling.

The primary objective of the job scheduler is to provide a balanced mix of jobs, such as I/O bound and processor bound. It also controls the degree of multiprogramming. If the degree of multiprogramming is stable, then the average rate of process creation must be equal to the average departure rate of processes leaving the system.

On some systems, the long-term scheduler may not be available or minimal. Time-sharing operating systems have no long term scheduler. When a process changes the state from new to ready, then there is use of long-term scheduler.

Short Term Scheduler

It is also called as **CPU scheduler**. Its main objective is to increase system performance in accordance with the chosen set of criteria. It is the change of ready state to running state of the process. CPU scheduler selects a process among the processes that are ready to execute and allocates CPU to one of them.

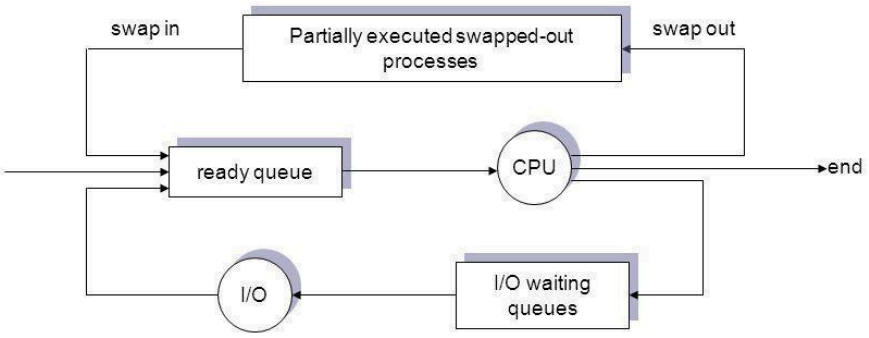
Short-term schedulers, also known as dispatchers, make the decision of which process to execute next. Short-term schedulers are faster than long-term schedulers.

Medium Term Scheduler

Medium-term scheduling is a part of **swapping**. It removes the processes from the memory. It reduces the degree of multiprogramming. The medium-term scheduler is in-charge of handling the swapped out-processes.

A running process may become suspended if it makes an I/O request. A suspended processes cannot make any progress towards completion. In this condition, to remove the process from memory and make space for other processes, the suspended process is moved to the secondary storage. This process is called **swapping**, and the process is said to be swapped out or rolled out. Swapping may be necessary to improve the process mix.

Medium-term scheduling model



Comparison among Schedulers:

S.N.	Long-Term Scheduler	Short-Term Scheduler	Medium-Term Scheduler
1	It is a job scheduler	It is a CPU scheduler	It is a process swapping scheduler.
2	Speed is lesser than short term scheduler	Speed is fastest among other two	Speed is in between both short and long term scheduler.
3	It controls the degree of multiprogramming	It provides lesser control over degree of multiprogramming	It reduces the degree of multiprogramming.
4	It is almost absent or minimal in time sharing system	It is also minimal in time sharing system	It is a part of Time sharing systems.

5	It selects processes from pool and loads them into memory for execution	It selects those processes which are ready to execute	It can re-introduce the process into memory and execution can be continued.

IPC (Inter process communication)

A process can be of two types:

- Independent process.
- Co-operating process.

An independent process is not affected by the execution of other processes while a co-operating process can be affected by other executing processes. Though one can think that those processes, which are running independently, will execute very efficiently but in practical, there are many situations when co-operative nature can be utilised for increasing computational speed, convenience and modularity. Inter process communication (IPC) is a mechanism which allows processes to communicate each other and synchronize their actions. The communication between these processes can be seen as a method of co-operation between them.

Advantages of process cooperation

- Advantages of process cooperation
- Information sharing
- Computation speed-up
- Modularity
- Convenience

Processes can communicate with each other using these two ways:

1. Shared Memory
2. Message passing

The Figure below shows a basic structure of communication between processes via shared memory method and via message passing.

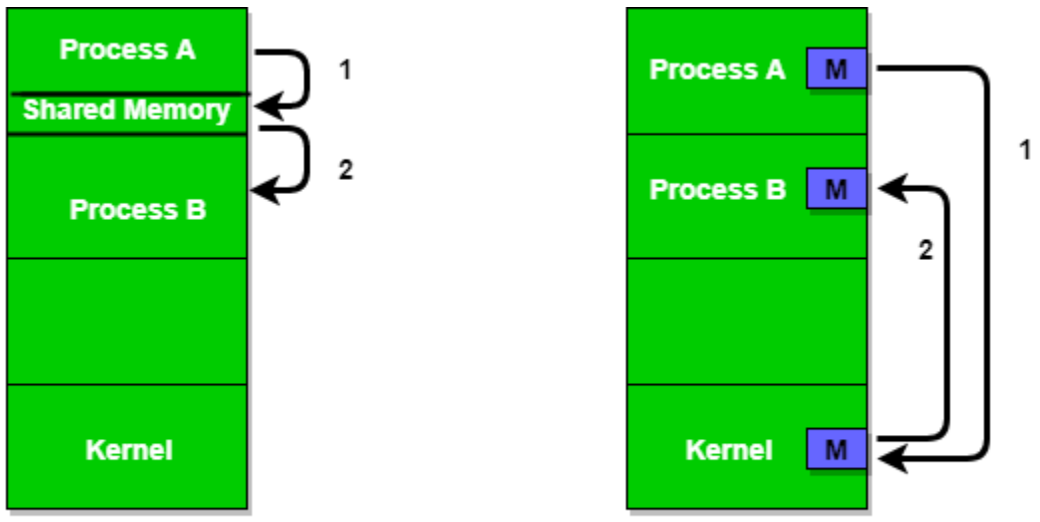


Figure 1 - Shared Memory and Message Passing

1)Shared memory: In this model, a region of the memory residing in an address space of a process creates a shared memory segment which can be accessed by all processes who want to communicate with each other. All the processes using the shared memory segment should attach to the address space of the shared memory. All the processes can exchange information by reading and/or writing data in shared memory segment. The form of data and location are determined by these processes who want to communicate with each other. These processes are not under the control of the operating system. The processes are also responsible for ensuring that they are not writing to the same location simultaneously. After establishing shared memory segment, all accesses to the shared memory segment are treated as routine memory access and without assistance of kernel.

2)Message Passing: In this model, communication takes place by exchanging messages between cooperating processes. It allows processes to communicate and synchronize their action without sharing the same address space. It is particularly useful in a distributed environment when communication process may reside on a different computer connected by a network. Communication requires sending and receiving messages through the kernel. The processes that want to communicate with each other must have a communication link between them. Between each pair of processes exactly one communication link exist.

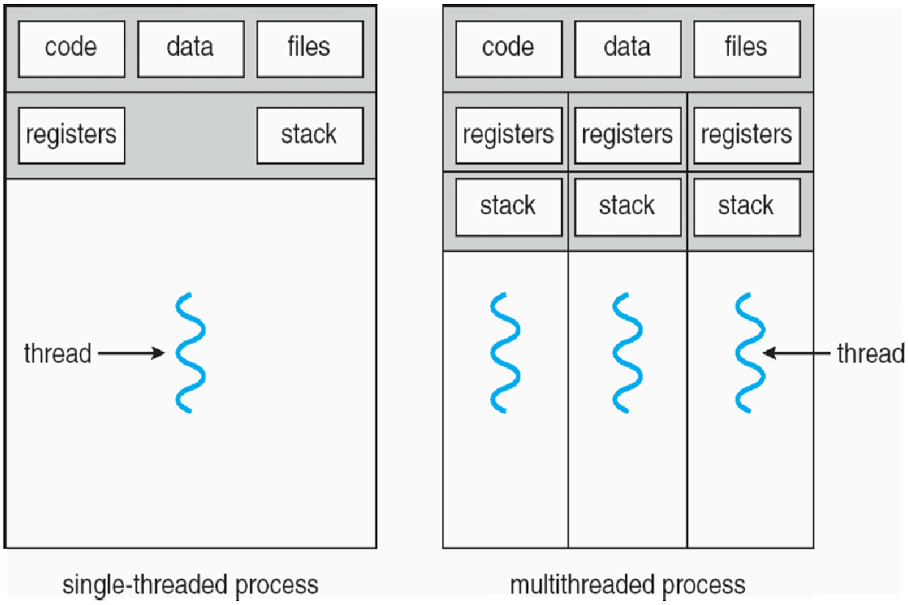
Threads:

What is Thread?

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

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

A thread is also called a **lightweight process**. Each thread belongs to exactly one process and no thread can exist outside a process.



Advantages of Thread

- Responsiveness
- Resource Sharing
- Economy
- Scalability

Difference between Process and Thread

S.N.	Process	Thread
1	Process is heavy weight or resource intensive.	Thread is light weight, taking lesser resources than a process.
2	Process switching needs interaction with operating system.	Thread switching does not need to interact with operating system.
3	In multiple processing environments, each process executes the same code but has its own memory and file resources.	All threads can share same set of open files, child processes.
4	If one process is blocked, then no other process can execute until the first process is unblocked.	While one thread is blocked and waiting, a second thread in the same task can run.
5	Multiple processes without using threads use more resources.	Multiple threaded processes use fewer resources.
6	In multiple processes each process operates independently of the others.	One thread can read, write or change another thread's data.

Types of threads:

Threads are implemented in following two ways –

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

User-Level Threads:

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

Advantages

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

Disadvantages

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

Kernel Level Threads

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

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

Advantages

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

Disadvantages

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

Difference between User-Level & Kernel-Level Thread

S.N.	User-Level Threads	Kernel-Level Thread
1	User-level threads are faster to create and manage.	Kernel-level threads are slower to create and manage.

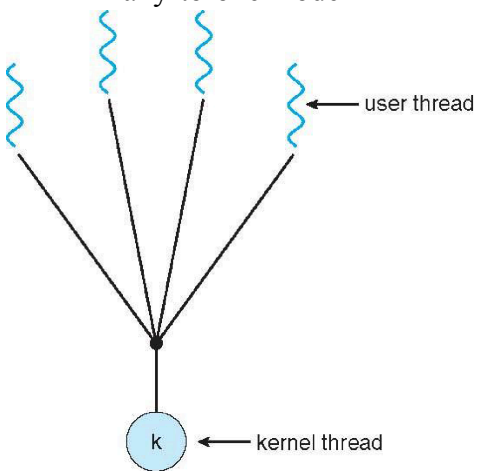
2	Implementation is by a thread library at the user level.	Operating system supports creation of Kernel threads.
3	User-level thread is generic and can run on any operating system.	Kernel-level thread is specific to the operating system.
4	Multi-threaded applications cannot take advantage of multiprocessing.	Kernel routines themselves can be multithreaded.

Multithreading Models

- Many to one model
- Many to many model
- One to one model

Many to One Model:

- This model maps many user level threads to one kernel level thread.
- If user level thread generates blocking system call then it blocks an entire process.
- At a time only one user level thread can access kernel level thread i.e multiple threads can't execute in parallel.
- Thread management is done by Thread libraries.
- Example: - Green threads – a thread library available for Solaris use
- many-to-one model



Advantages:-

- It is an efficient model as threads are managed by thread library in user
- space.
- Portable: Because user level threads packages are implemented
- entirely with standard Unix and POSIX library calls, they are often quite
- portable.
- One kernel level thread controls multiple user level threads.

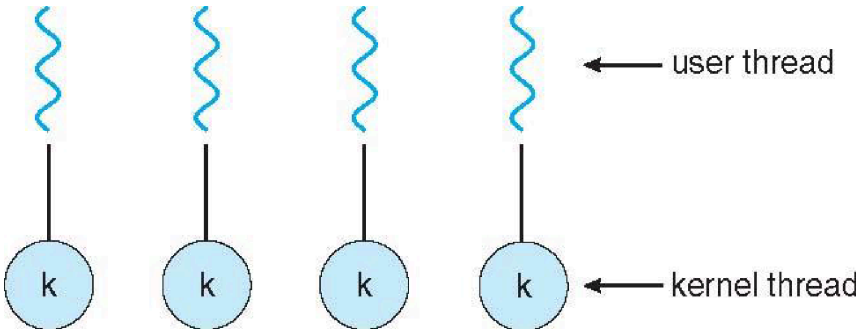
- Easy to do with few system dependencies.

Disadvantages:

- One block call from kernel level thread blocks all user level threads.
- Cannot take advantage of multiprocessing.

One to One Model:

- The one to one model maps each user thread to a single kernel thread.
- It provides more concurrency than the many to one model by allowing another thread to run when a thread makes a blocking system call.
- It also allows multiple threads to run in parallel on multiprocessors.
- Whenever user level thread is created, it compulsorily creates corresponding kernel level thread.
- This model is used in Linux & Windows version like 95,97,XP, NT.



Advantages:

- It allows multiple threads to run in parallel on multiprocessors.
- More concurrency
- Less complication in processing

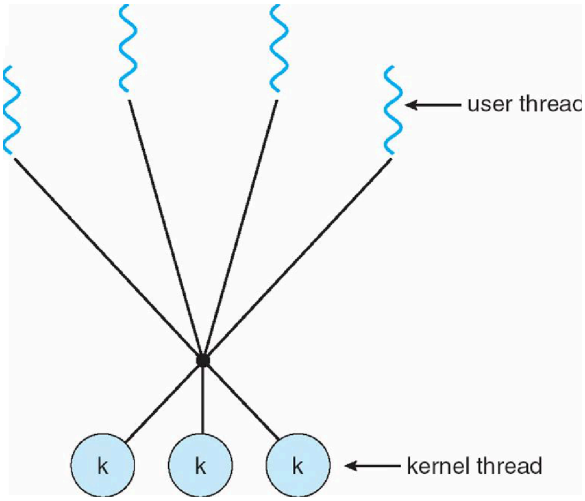
Disadvantages:

- Creating a user thread requires creating the corresponding kernel thread.
- Creating kernel thread may affect the performance of an application.
- It reduces performance of the system.
- Kernel thread is overhead.

Many to Many Model

The many-to-many model multiplexes any number of user threads onto an equal or smaller number of kernel threads.

The following diagram shows the many-to-many threading model where 6 user level threads are multiplexing with 6 kernel level threads. In this model, developers can create as many user threads as necessary and the corresponding Kernel threads can run in parallel on a multiprocessor machine. This model provides the best accuracy on concurrency and when a thread performs a blocking system call, the kernel can schedule another thread for execution.



Thread Scheduling:

Every thread has a thread priority assigned to it. Threads created within the common language runtime are initially assigned the priority of ThreadPriority.Normal. Threads created outside the runtime retain the priority they had before they entered the managed environment. You can get or set the priority of any thread with the Thread.Priority property.

Threads are scheduled for execution based on their priority. Even though threads are executing within the runtime, all threads are assigned processor time slices by the operating system. The details of the scheduling algorithm used to determine the order in which threads are executed varies with each operating system.

Approaches to Thread Scheduling:

- Load balancing
- Dedicated Processor Assignment
- Dynamic Scheduling