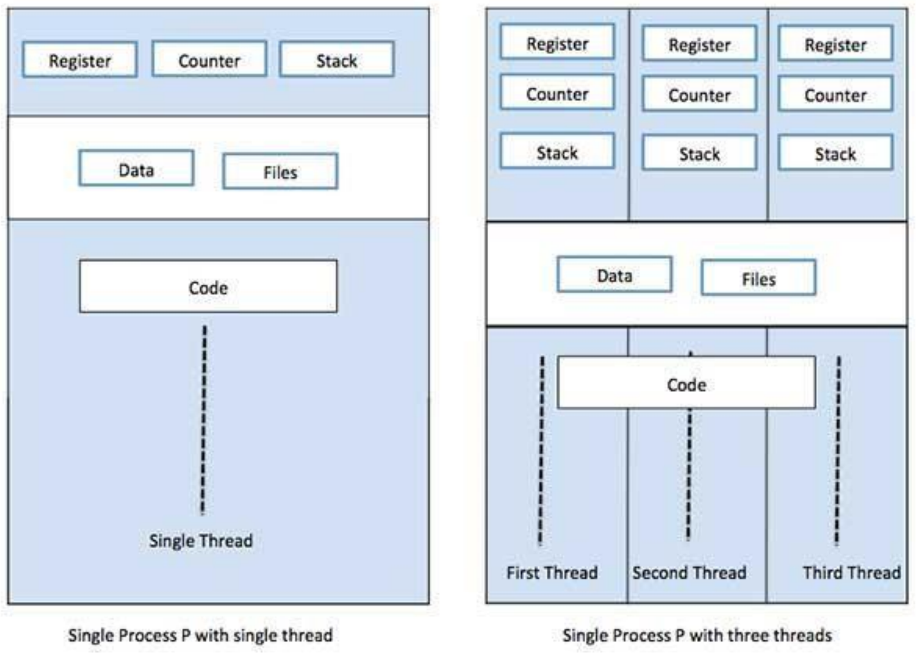
IV THREADS

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**. Threads provide a way to improve application performance through parallelism. Threads represent a software approach to improving performance of operating system by reducing the overhead thread is equivalent to a classical process.

Each thread belongs to exactly one process and no thread can exist outside a process. Each thread represents a separate flow of control. Threads have been successfully used in implementing network servers and web server. They also provide a suitable foundation for parallel execution of applications on shared memory multiprocessors. The following figure shows the working of a single-threaded and a multithreaded process.



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 operatingsystem.	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 andfile resources.	All threads can share same set ofopen 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 thesame task can run.
5	Multiple processes without using threads use moreresources.	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.

Advantages of Thread

- Threads minimize the context switching time.
- Use of threads provides concurrency within a process.
- Efficient communication.
- It is more economical to create and context switch threads.
- Threads allow utilization of multiprocessor architectures to a greater scale and efficiency.

Types of Thread

Threads are implemented in following two ways −

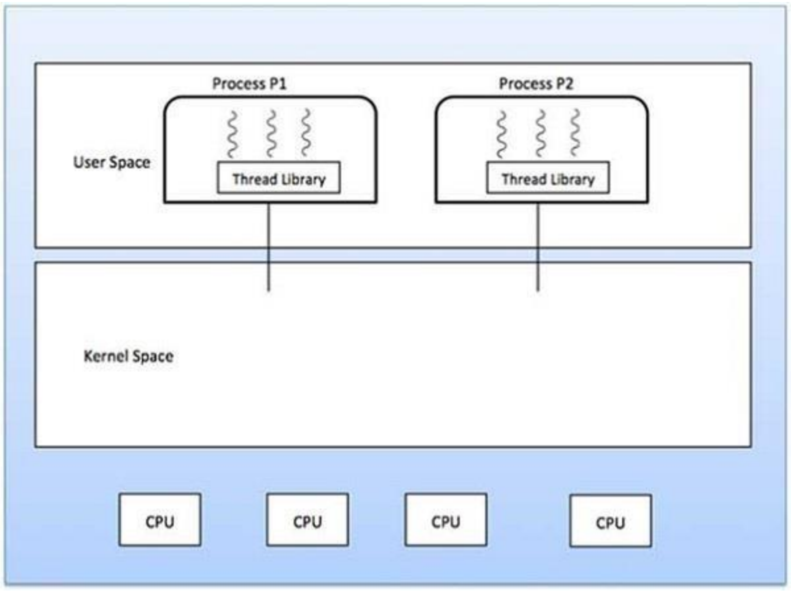
- **User Level Threads** − User managed threads.
- **Kernel Level Threads** − Operating System managed threads acting on kernel, anoperating system core.

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 individual's 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 multipleprocesses.
- If one thread in a process is blocked, the Kernel can schedule another thread of the sameprocess.
- 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 modeswitch to the Kernel.

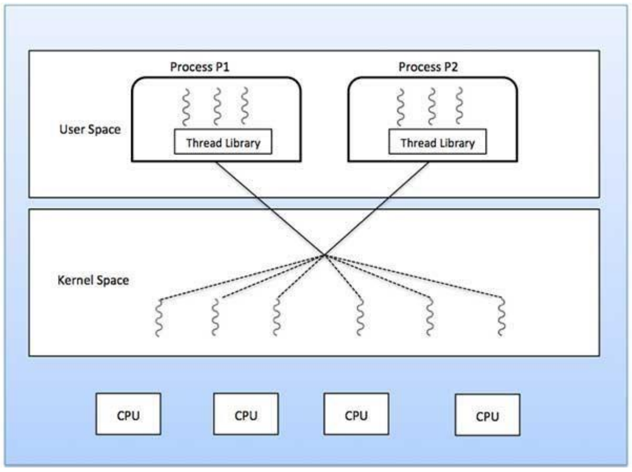
Multithreading Models

Some operating system provides a combined user level thread and Kernel level thread facility. Solaris is a good example of this combined approach. In a combined system, multiple threads within the same application can run in parallel on multiple processors and a blocking system call need not block the entire process. Multithreading models are three types

- Many to many relationship.
- Many to one relationship
- One to one relationship.

Many to Many Models

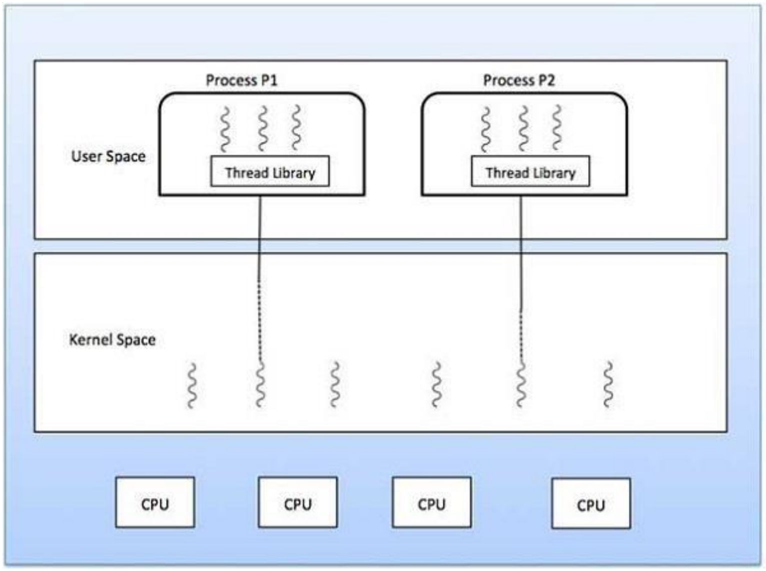
The many-to-many model multiplexes any number of user threads onto an equal or smaller number of kernel threads. The following diagram shows the many-to-many threading model where 6 user level threads are multiplexing with 6 kernel level threads. In this model, developers can create as many user threads 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.



Many to One Model

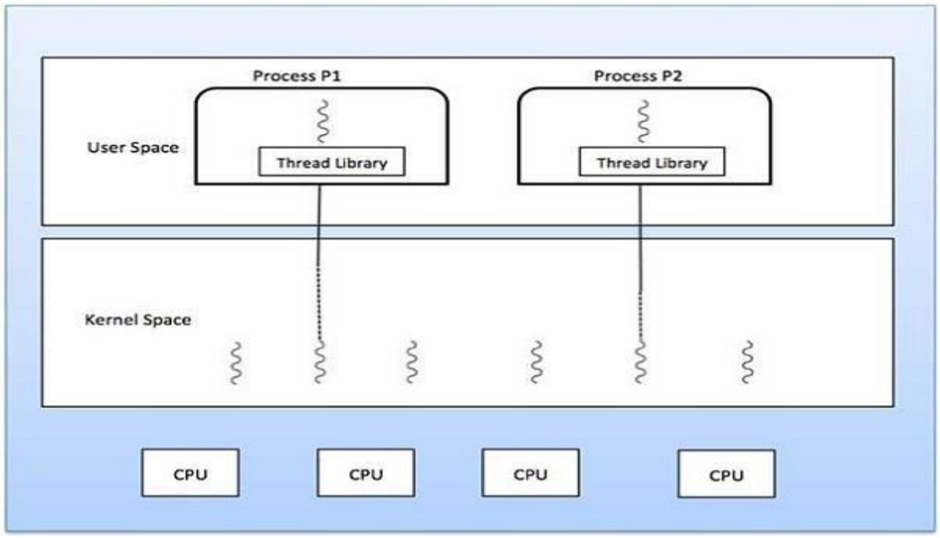
Many-to-one model maps many user level threads to one Kernel-level thread. Thread management is done in user space by the thread library. When thread makes a blocking system call, the entire process will be blocked. Only one thread can access the Kernel at a time, so multiple threads are unable to run in parallel on multiprocessors.

If the user-level thread libraries are implemented in the operating system in such a way that the system does not support them, then the Kernel threads use the many-to-one relationship modes.



One to One Model

There is one-to-one relationship of user-level thread to the kernel-level thread. This model provides more concurrency than the many-to-one model. It also allows another thread to run whena thread makes a blocking system call. It supports multiple threads to execute in parallel on microprocessors. Disadvantage of this model is that creating user thread requires the corresponding Kernel thread. OS/2, windows NT and windows 2000 use one to one relationship model.



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 tocreate and manage.
2	Implementation is by a thread library at the user level.	Operating system supports creationof Kernel threads.
3	User-level thread is generic and can run on anyoperating system.	Kernel-level thread is specific to theoperating system.
4	Multi-threaded applications cannot take advantage ofmultiprocessing.	Kernel routines themselves can bemultithreaded.

THREADING ISSUES:

These issued are as follows −

The fork() and exec() system calls

The fork() is used to create a duplicate process. The meaning of the fork() and exec() systemcalls change in a multithreaded program.

If one thread in a program which calls fork(), does the new process duplicate all threads, or is the new process single-threaded? If we take, some UNIX systems have chosen to have two versions of fork(), one that duplicates all threads and another that duplicates only the thread that invoked the fork() system call.

If a thread calls the exec() system call, the program specified in the parameter to exec() will replace the entire process which includes all threads.

Signal Handling

Generally, signal is used in UNIX systems to notify a process that a particular event has occurred.A signal received either synchronously or asynchronously, based on the source of and the reason for the event being signalled.

All signals, whether synchronous or asynchronous, follow the same pattern as given below

- A signal is generated by the occurrence of a particular event.
- The signal is delivered to a process.
- Once delivered, the signal must be handled.

Cancellation

Thread cancellation is the task of terminating a thread before it has completed.

For example − If multiple database threads are concurrently searching through a database andone thread returns the result the remaining threads might be cancelled.

A target thread is a thread that is to be cancelled, cancellation of target thread may occur in two different scenarios −

- **Asynchronous cancellation** − One thread immediately terminates the target thread.
- **Deferred cancellation** − The target thread periodically checks whether it should terminate,allowing it an opportunity to terminate itself in an ordinary fashion.

Thread polls

Multithreading in a web server, whenever the server receives a request it creates a separatethread to service the request.

Some of the problems that arise in creating a thread are as follows −

- The amount of time required to create the thread prior to serving the request together withthe fact that this thread will be discarded once it has completed its work.
- If all concurrent requests are allowed to be serviced in a new thread, there is no bound onthe number of threads concurrently active in the system.
- Unlimited thread could exhaust system resources like CPU time or memory.

A thread pool is to create a number of threads at process start-up and place them into a pool,where they sit and wait for work.