**1.About Thread**

A thread is a single sequence stream within in a process. An operating system that has thread facility, the basic unit of CPU utilization is a thread. A thread has or consists of a program counter (PC), a register set, and a stack space. Threads are not independent of one other like processes as a result threads shares with other threads their code section, data section, OS resources  also known as task, such as open files and signals.

Following are some reasons why we use threads in designing operating systems.

1. A process with multiple threads make a great server for example printer server.
2. Because threads can share common data, they do not need to use interprocess communication.
3. Because of the very nature, threads can take advantage of multiprocessors.

**Types:**

There are two types of thread

1.User level Threads

2.Kernel level Threads

**User level Threads**

Threads are visible only from within the process, where they share all process resources like address space, open files, and so on. The following state is unique to each thread.

* Thread ID
* Register state (including PC and stack pointer)
* Stack
* Signal mask
* Priority
* Thread-private storage

**Example**: User-thread libraries include POSIX Pthreads, Mach C-threads, and Solaris 2 UI-threads.

##### **Advantages of User Level Threads**

User level threads are managed by a user level library however, they still require a kernel system call to operate. It does not mean that the kernel knows anything about thread management. Not at all, It only takes care of the execution part. The lack of cooperation between user level threads and the kernel is a known disadvantage. In this case, the kernel may not favor a process that has many threads. User level threads are typically fast. Creating threads, switching between threads and synchronizing threads only needs a procedure call. They are a good choice for non blocking tasks otherwise the entire process will block if any of the threads blocks.

**Kernel level Threads**

A Kernel thread, sometimes called a light weight process, is created and scheduled by the kernel. Supporting thread at kernel level means that operating system is multithreaded. Operating system kernel maintains thread abstractions, synchronization and scheduling. It allows the resource to share.Kernel threads are supported directly by operating system.Kernel threads maintain context information for the process as a whole and for individual thread within process.

**Example**: Windows NT, Windows 2000, Solaris 2, BeOS, and Tru64 UNIX (formerly Digital UNIX)-support kernel threads.

##### **Advantages of Kernel Level Thread**

Kernel level threads are managed by the OS, therefore, thread operations (ex. Scheduling) are implemented in the kernel code. This means kernel level threads may favor thread heavy processes. Moreover, they can also utilize multiprocessor systems by splitting threads on different processors or cores. They are a good choice for processes that block frequently. If one thread blocks it does not cause the entire process to block. Kernel level threads have disadvantages as well. They are slower than user level threads due to the management overhead. Kernel level context switch involves more steps than just saving some registers.

**Threads are scheduled for the implementation or execution on CPU**.

There are four states of a thread:

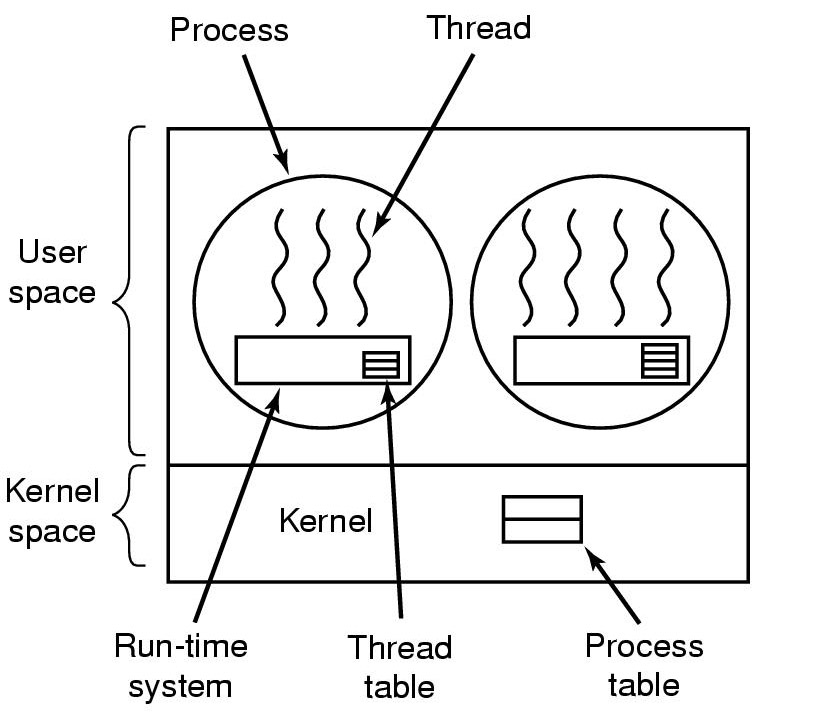
1. Running
2. Blocked
3. Read
4. Terminated

**There are two ways of implementing a thread package:**

1. In user space
2. In kernel

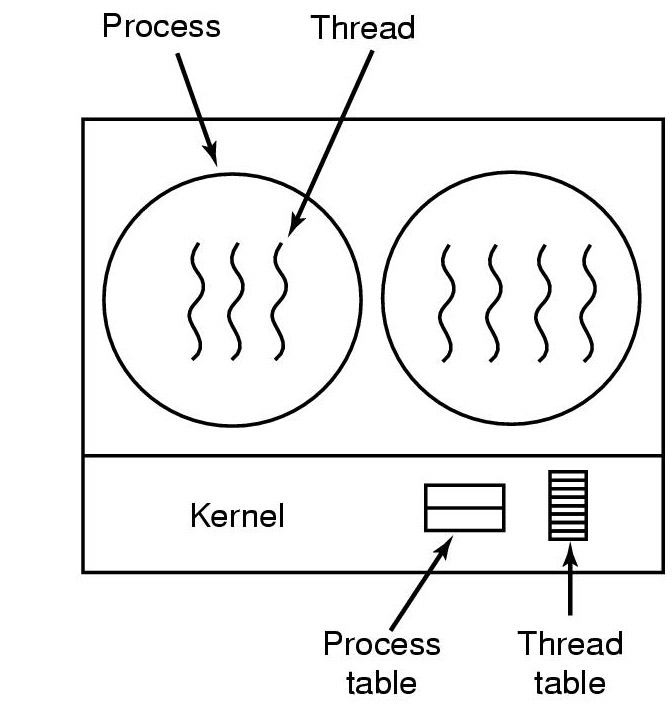
**In user Space**

Thread management done by user-level threads library The kernel is not aware of the existence of threads .All thread management is done by the application by using a thread library Thread switching does not require kernel mode privileges .Scheduling is application specific.

****

**In Kernel**

All thread management is done by kernel .No thread library but an API to the kernel thread facility .Kernel maintains context information for the process and the threads .Switching between threads requires the kernel .Scheduling on a thread basis

****