**2.4 Threads**

**2.4.1 Definitions of Threads**

* A thread is also known as light weight process, is a sub-division of work within the process.
* A thread is a basic unit of CPU utilization; it comprises a thread ID, a program counter, a register set, and a stack. It shares with other threads belonging to the same process its code section, data section, and other operating system resources, such as open files and signals.
* Thread simply enables us to split up a program into logically separate pieces and have the pieces run independently of one another until they need to communicate.

**2.4.2 Types of threaded process**



Fig a: Single threaded process Fig b: Multithreaded process

***In figure a:*** Three process each with one thread(single threaded) organization is used when three processes are unrelated.

***In figure b:*** One process with three threads(Multi threaded) would be appropriate when thethree threads are actually part of the same job and are actively and closely cooperating with each other. Each thread maintains its own stack.

**Multithreading:**

Many software package that run on modern desktop PCs are multi-threaded. An application is

implemented as a separate process with several threads of control. A web browser might have one thread to display images or text while other thread retrieves data from the network. A word-processor may have a thread for displaying graphics, another thread for reading the character entered by user through the keyboard, and a third thread for performing spelling and grammar checking in the background.

**2.4.3 Why Multithreading?**

In certain situations, a single application may be required to perform several similar tasks such as a web server accepts client requests for web pages, images, sound, graphics etc. A busy web server may have several clients concurrently accessing it. So if the web server runs on traditional single threaded process, it would be able to service only one client at a time. The amount of time that the client might have to wait for its request to be serviced is enormous.

One solution of this problem can be thought by creation of new process. When the server receives a new request, it creates a separate process to service that request. But this method is heavy weight. In fact this process creation method was common before threads become popular. Process creation is time consuming and resource intensive. If new process performs the same task as the existing process, why incur all that overhead? It is generally more efficient for one process that contains multiple threads to serve the same purpose. This approach would multithreaded the web server process. The server would create a separate thread that would listen for client’s requests. When a request is made, rather than creating another process, it will create a separate thread to service the request.

**2.4.4 Benefits of Multi-thread**

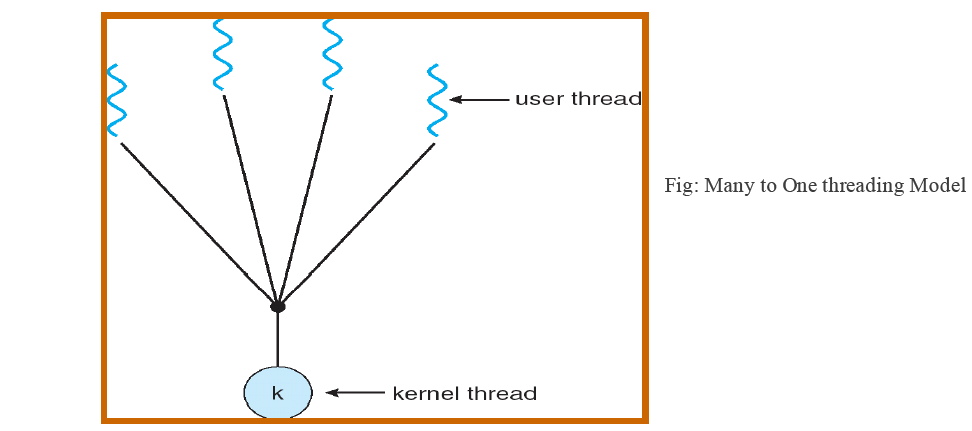
1. **Responsiveness:** Multithreaded interactive application continues to run even if part of it is blocked or performing a lengthy operation, thereby increasing the responsiveness to the user.
2. **Resource Sharing:** By default, threads share the memory and the resources of the process to which they belong. It allows an application to have several different threads of activity within the same address space.
3. **Economy:** Allocating memory and resources for process creation is costly. Since thread shares the resources of the process to which they belong, it is more economical to create and context switch threads. It is more time consuming to create and manage process than threads.
4. **Utilization of multiprocessor architecture:** The benefits of multi threading can be greatly increased in multiprocessor architecture, where threads may be running in parallel on different processors. Multithreading on a multi-CPU increases concurrency.

**2.4.6 Multi-Threading model**

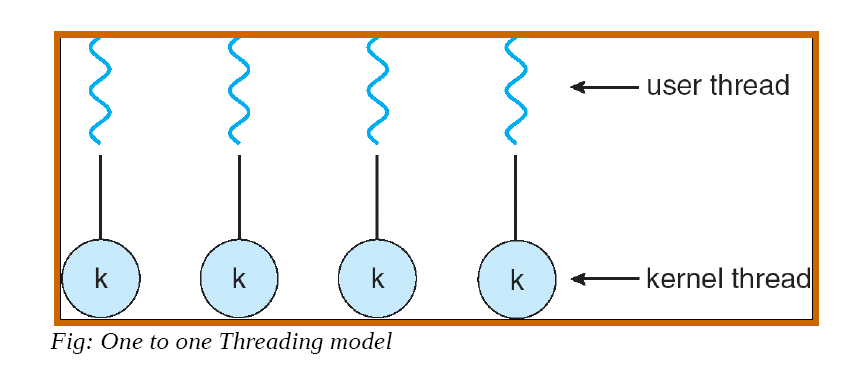
User threads are supported above the kernel and are managed without the kernel support whereas kernel threads are supported and are managed directly by the operating system. Virtually all operating-system includes kernel threads. Ultimately there must exists a relationship between user threads and kernel threads. We have three models for it.

**1. Many-to-one model**

It maps many user level threads to one kernel thread. Thread management is done by the thread library in user space so it is efficient; but the entire process will block if a thread makes a blocking system call. Also only one thread can access the kernel at a time; multiple threads are unable to run in parallel on multiprocessors. Green Threads - a thread library available for Solaris use this model.



**2. One-to-one Model:** maps each user thread to a kernel thread. It provides more concurrency than many to one model by allowing another thread to run when a thread makes a blocking system call. The only drawback to this model is that creating a user thread requires creating the corresponding kernel thread. Linux along with families of Windows operating system use this model.



**3. Many-to-many Model:** multiplexes many user level threads to a smaller or equal number of kernel threads. The number of kernel thread may be specific to either a particular application or a particular machine. Many-to-many model allows the users to create as many threads as he wishes but the true concurrency is not gained because the kernel can schedule only one thread at a time.

