Multi-Threading Model:

User thread are supported above the kernel and are managed without the kernel support whereas kernel threads are supported and are manged 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

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.

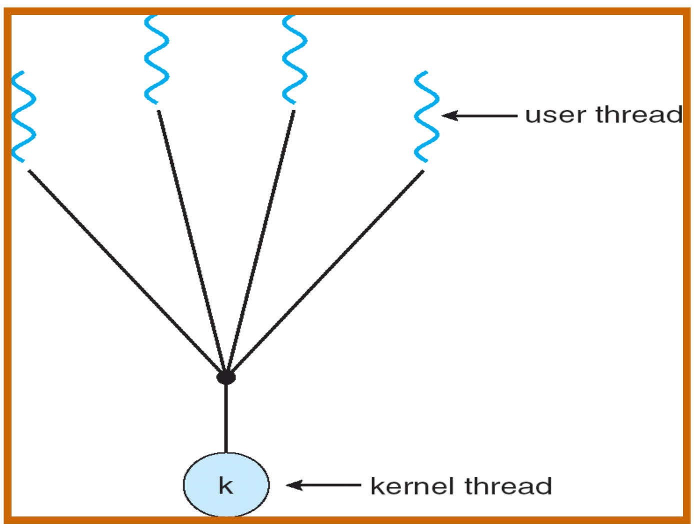

Fig: Many to One threading Model

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.

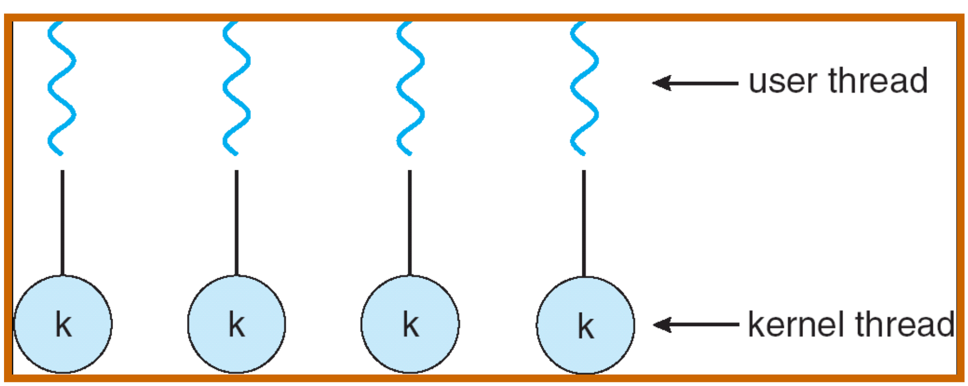

Fig: One to one Threading model

3. Many-to-many Model: multiplexes many user level thread 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.

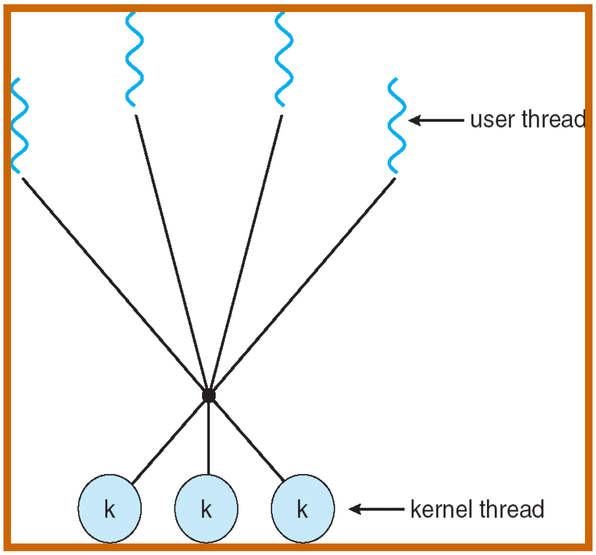

Fig:Many to Many

Interprocess Communication:

Processes frequently needs to communicate with each other. For example in a shell pipeline, the output of the first process must be passed to the second process and so on down the line. Thus there is a need for communication between the process, preferably in a well-structured way not using the interrupts.

IPC enables one application to control another application, and for several applications to share the same data without interfering with one another. Inter-process communication (IPC) is a set of techniques for the exchange of data among multiple threads in one or more processes. Processes may be running on one or more computers connected by a network. IPC techniques are divided into methods for **message passing, synchronization, shared memory, and remote procedure calls (RPC).**

co-operating Process: A process is independent if it can't affect or be affected by another process. A process is co-operating if it can affects other or be affected by the other process. Any process that shares data with other process is called co-operating process. There are many reasons for providing an environment for process co-operation.

1.Information sharing: Several users may be interested to access the same piece of information(for instance a shared file). We must allow concurrent access to such information.

2.Computation Speedup: Breakup tasks into sub-tasks.

3.Modularity: construct a system in a modular fashion.

4.convenience:

co-operating process requires IPC. There are two fundamental ways of IPC.
a. Shared Memory
b. Message Passing

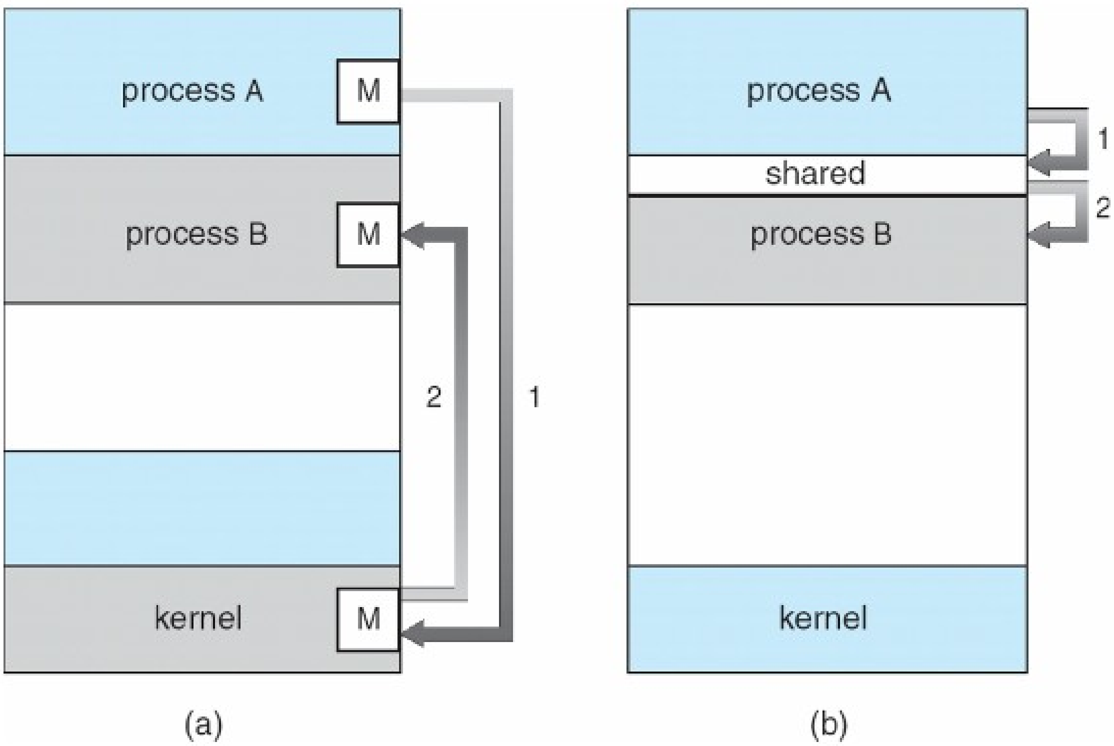

Fig: Communication Model a. Message Passing b. Shared Memory

Source : http://dayaramb.files.wordpress.com/2012/02/operating-system-pu.pdf