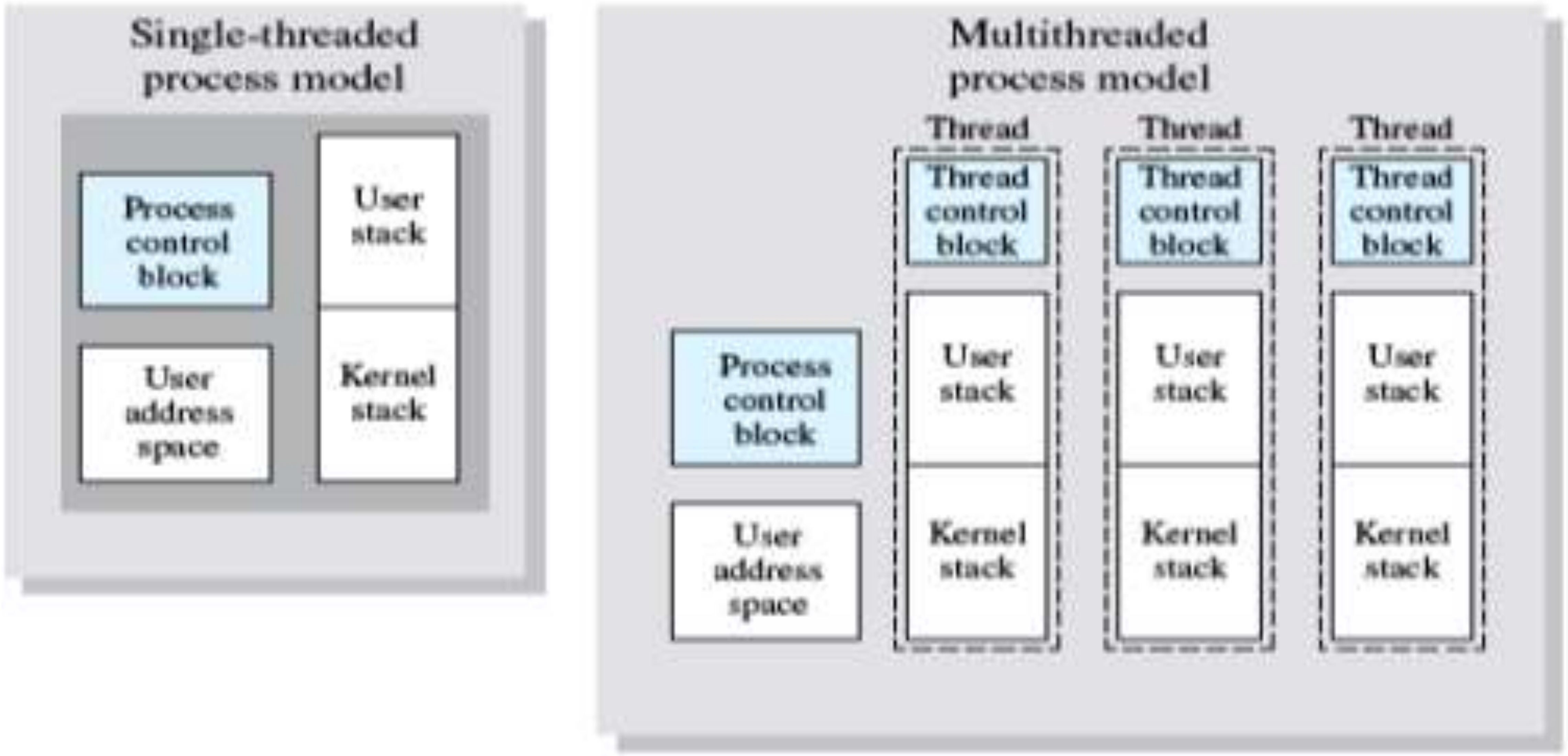
Process And Concurrency Management

Module 3.2

Thread

- A thread, sometimes called a lightweight process (LWP), 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.
- A traditional (or heavyweight) process has a single thread of control. If the process has multiple threads of control, it can do more than one task at a time.

Single-threaded and multithreaded Process Model



Benefits- Multithreaded Process Model

Responsiveness: Multithreading an interactive application may allow a program to continue running even if part of it is blocked or is performing a lengthy operation, thereby increasing responsiveness to the user. For instance, a multithreaded web browser could still allow user interaction in one thread while an image is being loaded in another thread.

Resource sharing: By default, threads share the memory and the resources of the process to which they belong. The benefit of code sharing is that it allows an application to have several different threads of activity all within the same address space.

Utilization of multiprocessor architectures: The benefits of multithreading can be greatly increased in a multiprocessor architecture, where each thread may be running in parallel on a different processor.

The OS supports the threads that can provided in following two levels:

- User-Level Threads
- Kernel-Level Threads

User-Level Threads

- User-level threads implement in user-level libraries, rather than via systems calls, so thread switching does not need to call operating system and to cause interrupt to the kernel.
- In fact, the kernel knows nothing about user-level threads and manages them as if they were single-threaded processes.

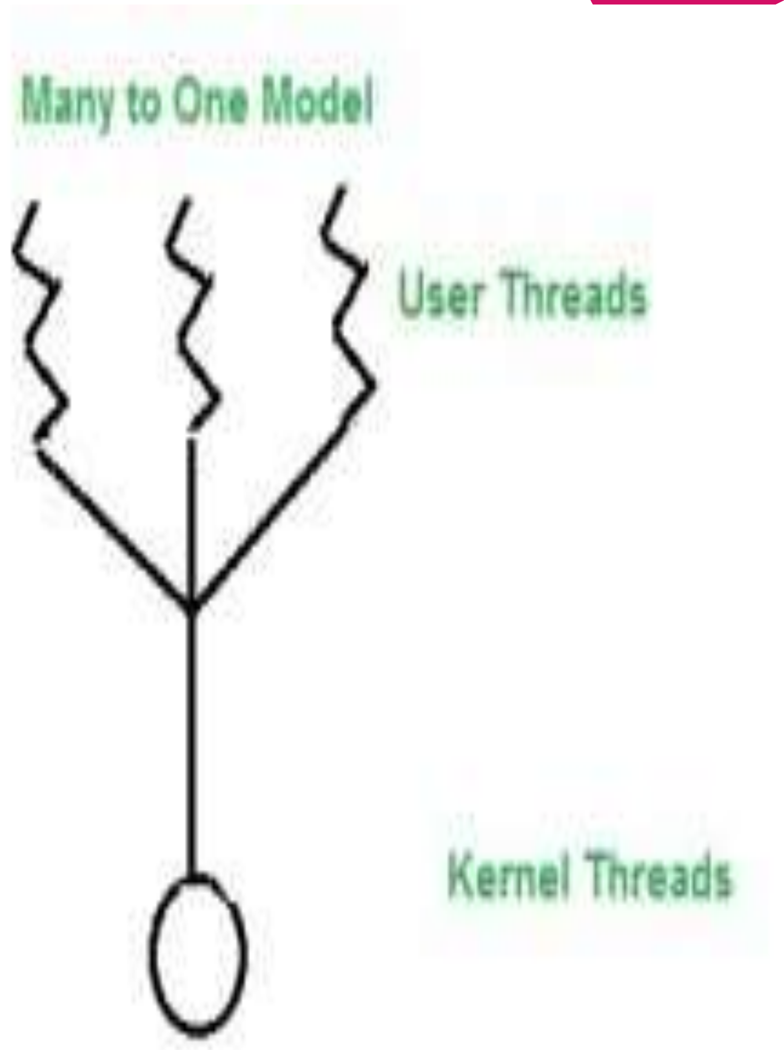
Kernel-Level Threads

- In this method, the kernel knows about and manages the threads.
- Instead of thread table in each process, the kernel has a thread table that keeps track of all threads in the system.
- Operating Systems kernel provides system call to create and manage threads.

Multithreading Models

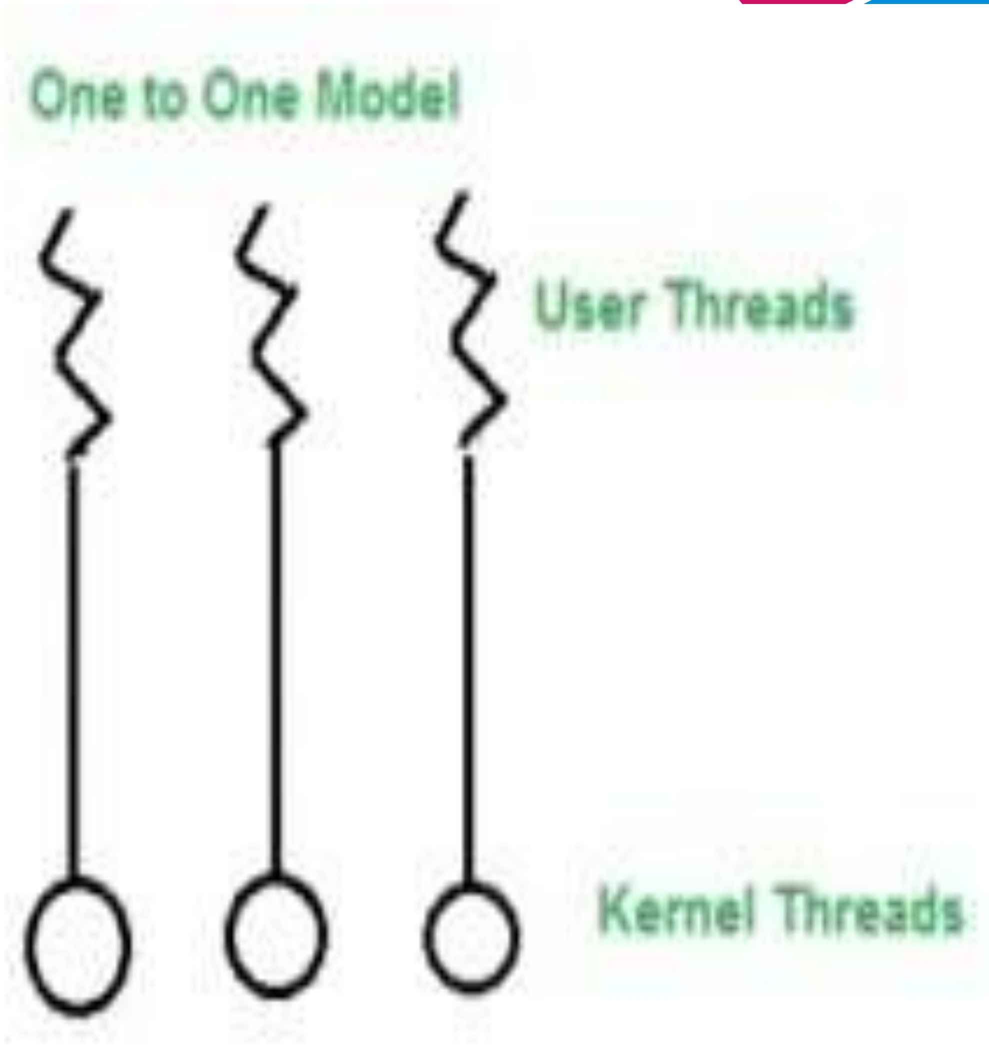
Many-to-One Model

- The many-to-one model maps many user-level threads to one kernel thread.
- Thread management is done in user space, so it is efficient, but the entire process will block if a thread makes a blocking system call.
- Also, because only one thread can access the kernel at a time, multiple threads are unable to run in parallel on multiprocessors.



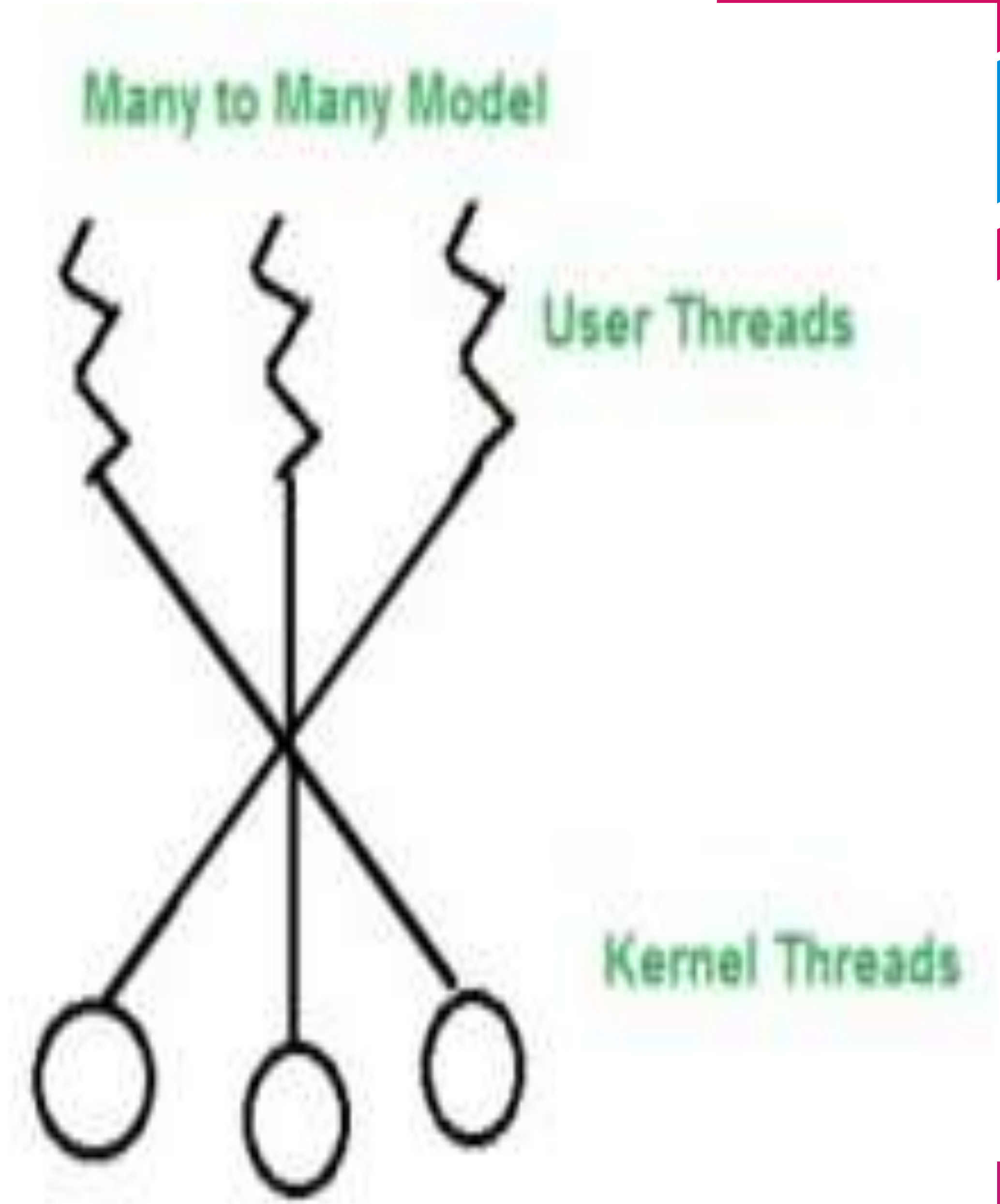
One-to-one Model

- The one-to-one model maps each user thread to a 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.
- The only drawback to this model is that creating a user thread requires creating the corresponding kernel thread.



Many-to-Many Model

- In this model, we have multiple user threads multiplex to same or lesser number of kernel level threads.
- Number of kernel level threads are specific to the machine, advantage of this model is if a user thread is blocked we can schedule others user thread to other kernel thread.
- Thus, System doesn't block if a particular thread is blocked.



Process	Vs Thread
1.Process cannot share the same memory area(address space)	1.Threads can share memory and files.
2.It takes more time to create a process	2.It takes less time to create a thread.
3.It takes more time to complete the execution and terminate.	3.Less time to terminate.
4.Execution is very slow.	4.Execution is very fast.
5.It takes more time to switch between two processes.	5.It takes less time to switch between two threads.
6.System calls are required to communicate each other	6.System calls are not required.
7.It requires more resources to execute.	7.Requires fewer resources.
8.Implementing the communication between processes is bit more difficult.	8.Communication between two threads are very easy to implement because threads share the memory