**MULTITHREADING MODELS**

In an [operating system](https://www.javatpoint.com/os-tutorial), threads are divided into 2 types

1)user threads

2)Kernel threads.

**User threads:**

Threads that are implemented at the user level are termed as user threads. The thread management of user threads is done by the thread library present at the user space. User threads are above the kernel and without kernel support. The kernel does not have any information about the user-level threads. These are the threads that application programmers use in their programs.

**Kernel threads:**

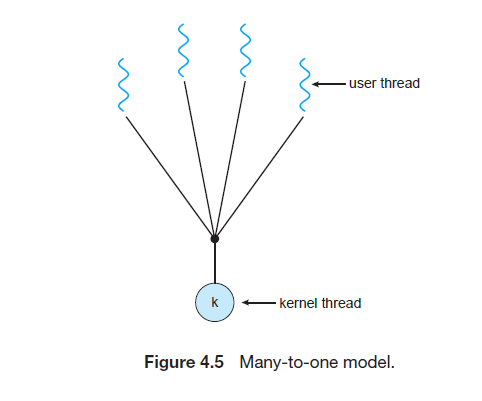
The threads implemented at the kernel level are termed as kernel threads. Kernel threads supported and managed directly by operating system itself. The thread management is at the kernel level and the kernel threads cannot be managed by the code in the application level.

Ultimately, a relationship must exist between user threads and kernel threads in order to execute user level threads on CPU. The user threads must be mapped to kernel threads, by one of the following strategies:

1. Many-to-One Model
2. One-to-One Model
3. Many-to-Many Model

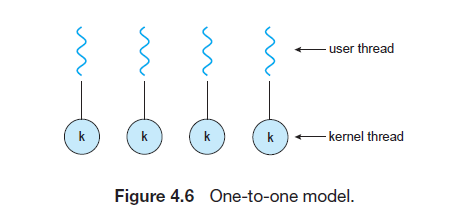
**1)Many-to-One Model:**

The 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, because 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—uses this model, as does GNU Portable Threads.

****

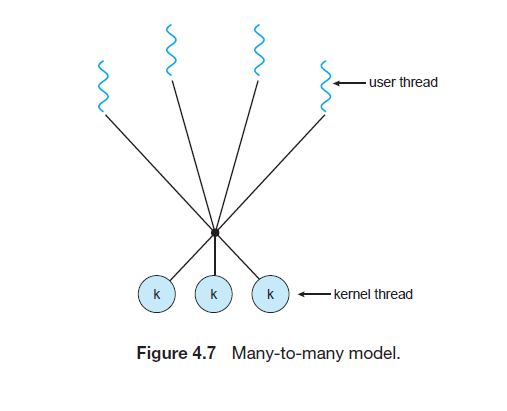
**2)One-to-One Model:**

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. Because the overhead of creating kernel threads can burden the performance of an application, most implementations of this model restrict the number of threads supported by the system. Linux, along with the family of Windows operating systems—including Windows 95, 98, NT, 2000, and XP— implement the one-to-one model.



**3)Many-to-Many Model:**

The many-to-many model multiplexes many user-level threads to a smaller or equal number of kernel threads. The number of kernel threads may be specific to either a particular application or a particular machine.



Whereas the many-to-one model allows the developer to create as many user threads as she wishes, true concurrency is not gained because the kernel can schedule only one thread at a time. The one-to-one model allows for greater concurrency, but the developer has to be careful not to create too many threads within an application (and in some instances may be limited in the number of threads she can create). The many-to-many model suffers from neither of these shortcomings: Developers can create as many user threads as necessary, and the corresponding kernel threads can run in parallel on a multiprocessor. Also, when a thread performs a blocking system call, the kernel can schedule another thread for execution.

One variation on the many-to-many model still multiplexes many user-level threads to a smaller or equal number of kernel threads but also allows a user-level thread to be bound to a kernel thread. This variation, sometimes referred to as the two-level model. The Solaris operating system supported the two-level model in versions older than Solaris 9. However, beginning with Solaris 9, this system uses the one-to-one model.

