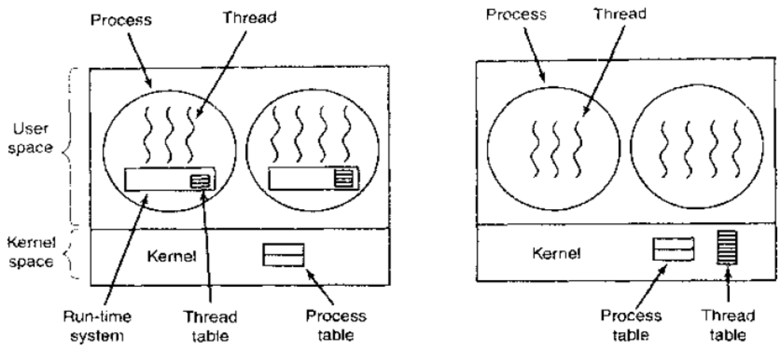
Threading Models

1- User-Level Threads

The first method is to put the threads package entirely in user space, the kernel knows nothing about them, so User level threads perform threading operations in the user space, meaning that threads are created by runtime libraries that cannot execute privileged instructions or access kernel primitives directly.



(a) (b)

Figure 4: (a) A user-level threads package, (b) A threads package managed by the kernel.

2- Kernel-Level Threads

Kernel-level threads attempt to address the limitations of user-level threads by mapping each thread to its own execution context.

3- Combining User- level Threads and kernel-level Threads

Also called hybrid Threads, various ways have been investigated to try to combine the advantages of user-level threads with kernel-level threads. One way is use kernel-level threads and then multiplex user-level threads onto some or all of the kernel threads, as shown in figure bellow

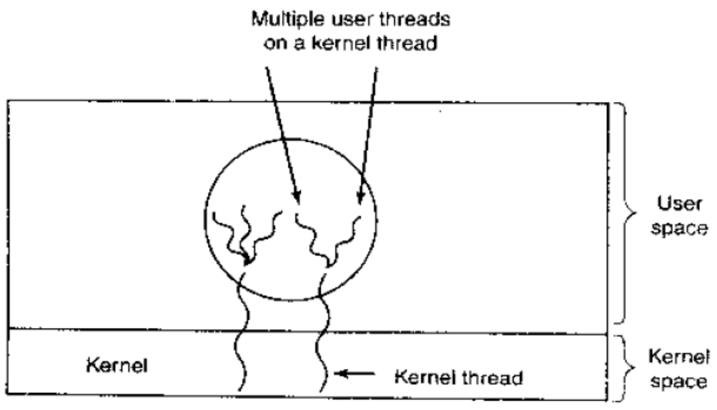


Figure 5: Multiplexing user-level threads onto kernel-level threads.

Comparison among three models

a-user-level threads

1- threads are created by runtime libraries that cannot execute privileged instructions or access kernel primitives directly.

2- The operating system treats each multithreaded process as a single execution context.

3- The user-level thread implementations are also called many-to-one thread mappings because the operating system maps all threads in a multithreaded process to a single execution context.

4- When a process employs user-level threads, user-level libraries perform scheduling and dispatching operations on the process's threads.

5- User level threads do not require the operating system support threads.

6- User level threads are more portable because of not relying on a particular operating system's threading API.

7- user level threads consume less resource than Kernel level threads

8- user-level threads don't require that the operating system manage all threads in the system.

9- User level thread performance varies depending on the system and on process behavior and in multiprocessor User level threads do not scale well to multiprocessor systems, so can result in suboptimal performance in multiprocessor systems.

b-kernel-level threads

1- threads are can execute privileged instructions or access kernel primitives directly.

2- The operating system creates a kernel thread that executes the user thread's instructions.

3- Kernel-level threads are often described as one-to-one threads mapping

4- mapping require operating system to provide each user thread with a kernel thread that the operating system can dispatch.

5- Kernel level threads require the operating system support threads.

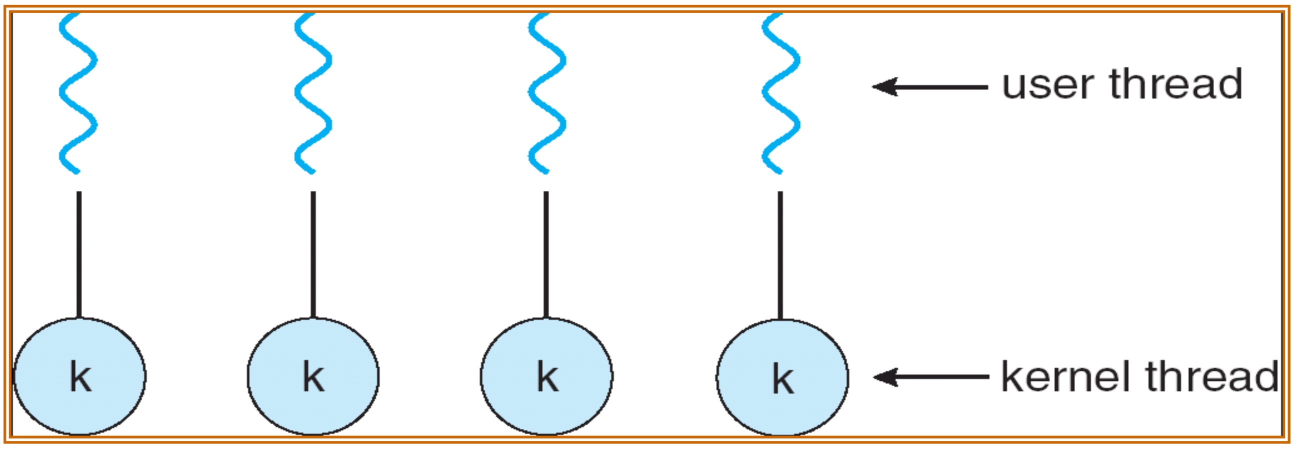
6- software that employs kernel-level threads is often less portable than software employs user-level threads

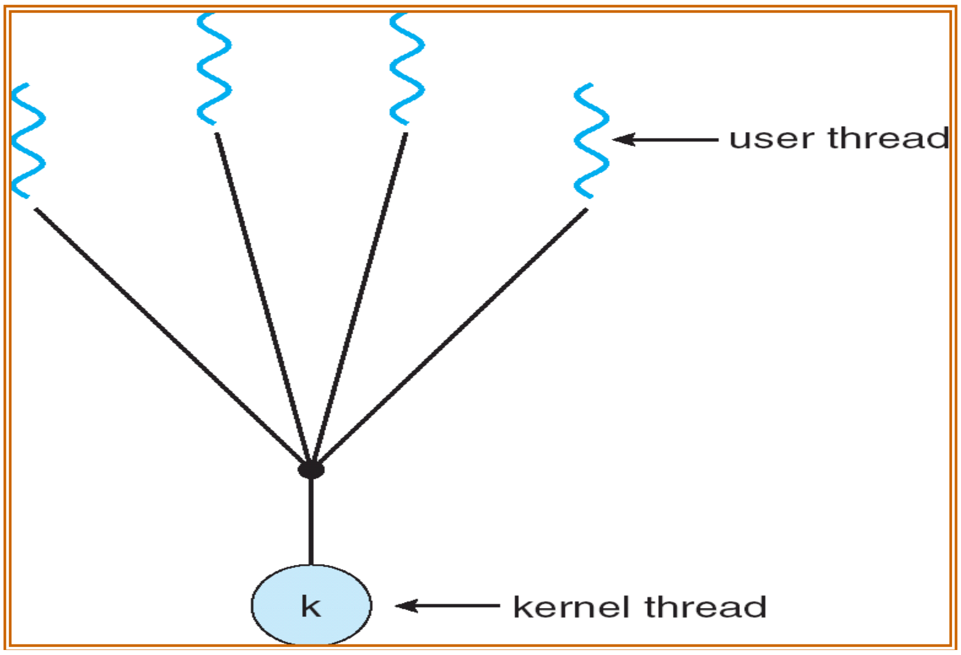
7- Kernel level threads consume more resource than user level threads

8- Kernel-level threads require that the operating system manage all threads in the system.

9- kernel can manage each thread individually, meaning that the operating system
can dispatch a process's ready threads even if one of its threads is blocked
(improve performance).

c-hybrid threads
1- threads are can execute privileged instructions or access kernel primitives directly.
2- The operating system creates a kernel thread that executes the user thread's instructions.
3- The combination is known as the many-to-many thread mapping as its name,this implementation maps many user-level-threads to a set of kernel-level threads. Some refer to this technique as (m-to-n threads mapping)
4-Hybrid threads require the operating system support threads.
5- applications can improve performance by customizing the threading library's scheduling algorithm.


user thread
k
k
k
k
kernel thread


user thread
k
kernel thread