Operating System Concepts 9th Edition Chapter 4

Part 3 – Multithreading Models

Key words:

* Kernel and user threads
* Relationship between those two
* Many-to-one model
* One-to-one model
* Many-to-many model

Questions:

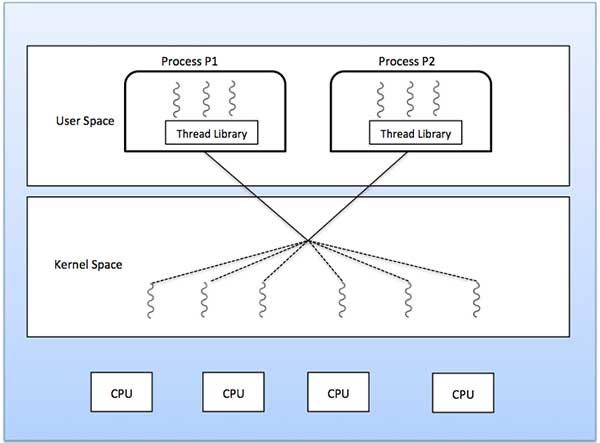
* What is kernel and user threads?
* What is this part about? (relationships)
* What is the different models and who use them?

# Kernel and User threads

There are two types of threads one is at the user level called user thread and the other at the kernel level called kernel thread.

User threads does not require any kernel privileges and is created by the thread library, they are also fully managed and scheduled by the library.

The kernel thread however are directly supported and managed by the OS and has therefore no thread management as the user thread has [1]. They are used to provide applications with privileged services e.g. system calls (write, close, wait etc.). The kernel use kernel threads to know what is running on the system, how much resource is in need and scheduling them [3].



**F\qigure 1.0** kernel and user thread visualization

In order to run user threads parallel on different CPUs (which it can’t by itself) it is in use of help from the kernel threads since they can, as they use the kernel scheduler. This means that a relationship between those two (kernel and user threads) must be established [2]. The relationship must consider the pros and cons of the two such as kernel is a slower to create but can run simultaneously and user is fast to create and manage but as said before can’t take advantage of multiprocessing [1].

One could think of a version of the philosopher’s dilemma where kernel threads could be forks and user threads are the different philosophers who must eat but also think and cannot do the same at once. The forks are very expensive so we want the smallest number of forks as possible, but as the philosophers are hungry must therefore eat, we cannot have too few forks. A good balance must be made in order that no philosopher will starve but will also have time to think.

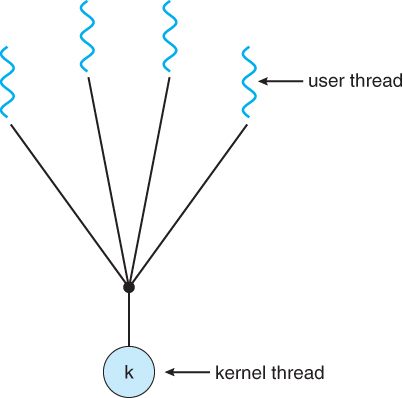
Different operating systems follow different models on how to make this relationship, chapter 4.3 acknowledge these three (common) models: the many-to-one model, the one-to-one model and the many-to-many model [4] (also called many/one-to-many/one models).

## Many to One Model

This model maps many user level threads to one kernel level thread where the management is done in user space by the thread library [4 p.169]. Only one thread can access the kernel at once making it unable to run multiple threads parallel on multicore systems. If one thread makes a block the entire process will result in a block as well.

This means that only one schedulable entity (user level thread) is known to the operating system [5].

Early years of java threads on the Solaries system used this model and a few system still does. See figure 2.0



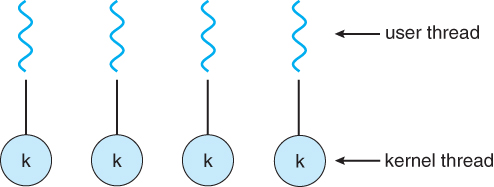
**Figure 2.0** many-to-one model

## One to One Model

The one-to-one model is one of the earliest implementation of true multithreading [5]. Each user-level thread is mapped to a corresponding kernel thread making this model a heavyweight process. This allows threads to access the kernel when another makes a blocking call.

The drawback of this implementation is that it is heavyweight, when a new thread is created a kernel thread is also created and as mentioned before this takes more time resulting in a slower performance of the system.

Because of the performance is crucial most implementation of this model restrict the number of threads. Both Linux and the Windows family implements the one-to-one model [4 p.170]. See figure 3.0

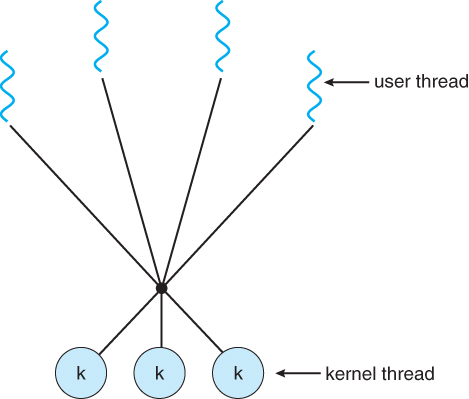


**Figure 3.0** one-to-one model

## Many to Many Model

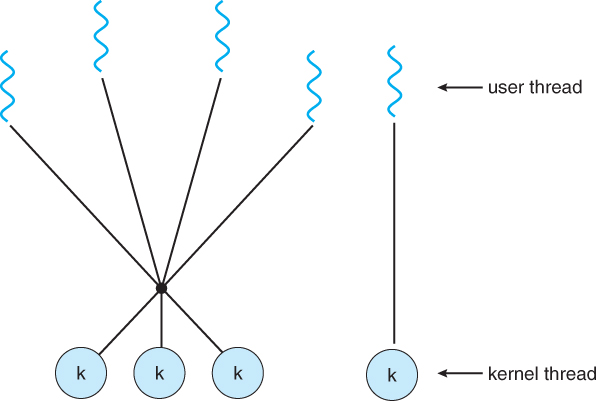
The many-to-many model maps many user-level threads to a smaller or equal number of kernel threads. This model avoids many limitations that the one-to-one model has, while the creation of threads can be as many as the developer wants the threads can only access the kernel at once. But with many to many it can still create as lots of threads and when a thread creates a blocking call it can schedule a new kernel allowing grater concurrency resulting that threads can run parallel on a multiprocessor [4 p.170-171].

This implementation results in a reduced weight and cost of each thread making it more lightweight as it optimizes the kernel-level and user-level thread correspondent that the one-to-one does not. See figure 4.0



**Figure 4.0** many-to-many model

One variation of the many-to-many model is called two-level model where it extends the many-to-many as a user-level thread can be corresponded with a kernel-level thread [4 p.171]. The Solaries operating system (above version 9) supports the two-level model. See figure 4.1.



**Figure 4.1** two-level model

# Reference

[1] Tutorialspoint. (2016). Operating System – Multi-Threading [online]. Available: <https://www.tutorialspoint.com/operating_system/os_multi_threading.htm>

[2] Pankaj Gautam. (February 13, 2010). What is the difference between user level threads and kernel level threads? [online]. Available: <http://www.answers.com/Q/What_is_the_difference_between_user_level_threads_and_kernel_level_threads>

[3] samoz. (July 24 2009). Operating System – Relationship between a kernel and a user thread [online]. Available: <http://stackoverflow.com/questions/1178785/relationship-between-a-kernel-and-a-user-thread>

[4] (actual book)

[5] <https://docs.oracle.com/cd/E19455-01/806-3461/6jck06gqk/index.html>

Images

Figure 1.0 <https://www.tutorialspoint.com/operating_system/os_multi_threading.htm>

Figure 2.0, 3.0, 4.0, 4.1 <https://www.cs.uic.edu/~jbell/CourseNotes/OperatingSystems/4_Threads.html>