

Fundamentally, then, data parallelism involves the distribution of data across multiple cores and task parallelism on the distribution of tasks across multiple cores. In practice, however, few applications strictly follow either data or task parallelism. In most instances, applications use a hybrid of these two strategies.

4.3 Multithreading Models

Our discussion so far has treated threads in a generic sense. However, support for threads may be provided either at the user level, for **user threads**, or by the kernel, for **kernel threads**. User threads are supported above the kernel and are managed without kernel support, whereas kernel threads are supported and managed directly by the operating system. Virtually all contemporary operating systems—including Windows, Linux, Mac OS X, and Solaris—support kernel threads.

Ultimately, a relationship must exist between user threads and kernel threads. In this section, we look at three common ways of establishing such a relationship: the many-to-one model, the one-to-one model, and the many-to-many model.

4.3.1 Many-to-One Model

The many-to-one model (Figure 4.5) maps many user-level threads to one kernel thread. Thread management is done by the thread library in user space, so it is efficient (we discuss thread libraries in Section 4.4). However, 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 multicore systems. **Green threads**—a thread library available for Solaris systems and adopted in early versions of Java—used the many-to-one model. However, very few systems continue to use the model because of its inability to take advantage of multiple processing cores.

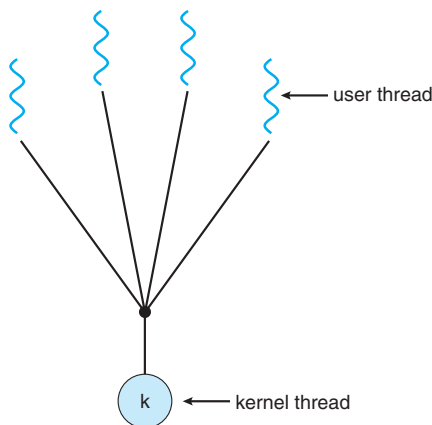


Figure 4.5 Many-to-one model.

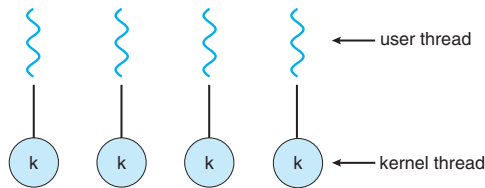


Figure 4.6 One-to-one model.

4.3.2 One-to-One Model

The one-to-one model (Figure 4.6) 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, implement the one-to-one model.

4.3.3 Many-to-Many Model

The many-to-many model (Figure 4.7) 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 (an application may be allocated more kernel threads on a multiprocessor than on a single processor).

Let's consider the effect of this design on concurrency. Whereas the many-to-one model allows the developer to create as many user threads as she wishes, it does not result in true concurrency, because the kernel can schedule only one thread at a time. The one-to-one model allows 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

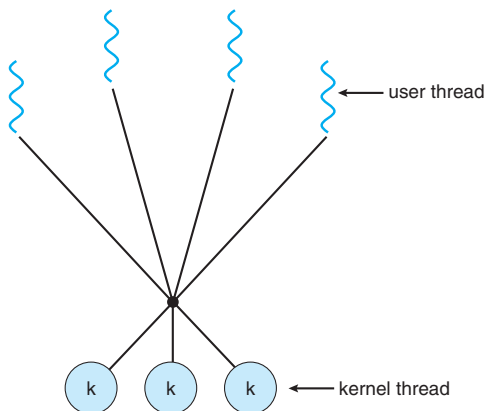


Figure 4.7 Many-to-many model.

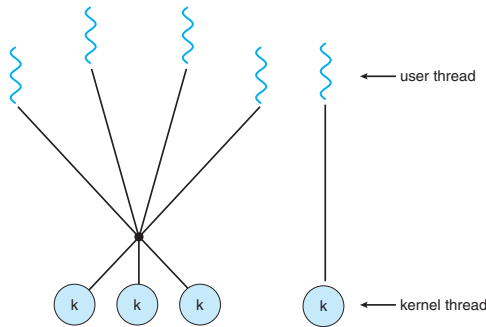


Figure 4.8 Two-level model.

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 is sometimes referred to as the **two-level model** (Figure 4.8). 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.

4.4 Thread Libraries

A **thread library** provides the programmer with an API for creating and managing threads. There are two primary ways of implementing a thread library. The first approach is to provide a library entirely in user space with no kernel support. All code and data structures for the library exist in user space. This means that invoking a function in the library results in a local function call in user space and not a system call.

The second approach is to implement a kernel-level library supported directly by the operating system. In this case, code and data structures for the library exist in kernel space. Invoking a function in the API for the library typically results in a system call to the kernel.

Three main thread libraries are in use today: POSIX Pthreads, Windows, and Java. Pthreads, the threads extension of the POSIX standard, may be provided as either a user-level or a kernel-level library. The Windows thread library is a kernel-level library available on Windows systems. The Java thread API allows threads to be created and managed directly in Java programs. However, because in most instances the JVM is running on top of a host operating system, the Java thread API is generally implemented using a thread library available on the host system. This means that on Windows systems, Java threads are typically implemented using the Windows API; UNIX and Linux systems often use Pthreads.