# Chapter 4 – Threads and Concurrency

Virtually all modern operating systems allow a process to contain multiple threads of control. In this chapter we explore the concepts associated with multithreaded computer systems.

## *Contents*



## *Chapter Objectives*

- *Identify the basic components of a thread, and contrast threads and processes*
- *Describe the benefits and challenges of designing multithreaded applications*
- *Describe how the Windows and Linux operating systems represent threads*
- *Design multithreaded applications using the Pthreads, Java, and Windows threading APIs*

## 4.1 Overview

A **thread** is a basic unit of CPU utilization; it comprises a thread ID, a program counter (PC), a register set, and a stack. It shares with other threads belonging to the same process its code section, data section, and other OS resources such as open files. A traditional process has a single thread of control. If a process has multiple threads of control, it can perform more than one task at a time.

### 4.1.1 Motivation

Most software applications that run on modern computers and mobile devices are multithreaded. An application typically is implemented as a process having several threads of control. For example, a web browser may have one thread to display images or text and another thread to retrieve data from the

network. Process creation is a heavy-weight task while thread creation is comparatively light-weight. It can simplify the code and increase efficiency. Most OS kernels are generally multithreaded.

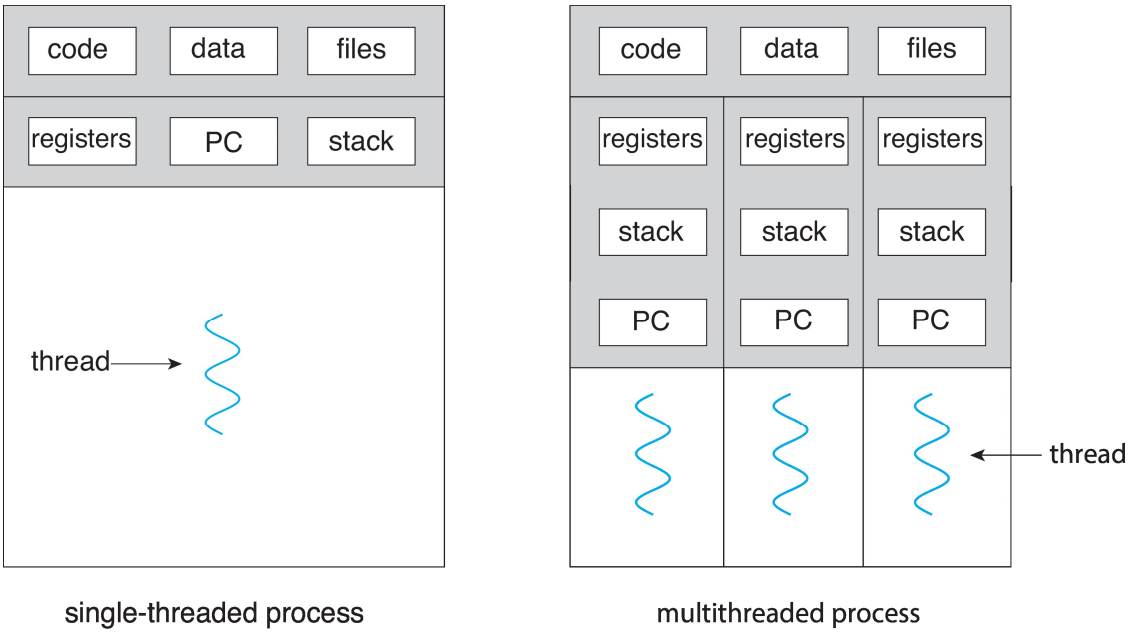


***Figure 1*** *Single threaded vs. multithreaded processes*

### 4.1.2 Benefits

The benefits of multithreaded programming can be broken down into **four** major categories:

- **Responsiveness** – Multithreading an interactive application allows a program to continue running even if part of the process is blocked or performing a lengthy operation, which increases responsiveness to the user. This is especially important in designing user interfaces.
- **Resource Sharing** – Processes can share resources only by mechanisms such as shared memory or message passing. These need to be implemented explicitly. In contrast, multithreading allows an application to have several different threads of activity within the same address space. Hence, threads can share the memory and resources of the process to which they belong.
- **Economy** – Allocating memory and resources for a process is costly. It is more economical to create and context switch among threads. Usually thread creation takes less time and memory than process creation, also thread switching is faster than context switching between processes.
- **Scalability** – The benefits of multithreading is great in multiprocessor architectures, as threads can run in parallel on different processing cores.

## 4.2 Multicore Programming

Multicore systems have multiple cores on a single processing chip where each core appears as a separate CPU to the operating system. Multithreaded programming provides a mechanism for more efficient use of these multiple computing cores and improved concurrency.

On a system with single computing core, concurrency means that the execution of the threads are interleaved over time. On a system with multiple cores, concurrency means that some threads can run

in parallel. A concurrent system supports more than one task by allowing all the tasks to make progress. In contrast, a parallel system can perform more than one task simultaneously. It is possible to have concurrency without parallelism.

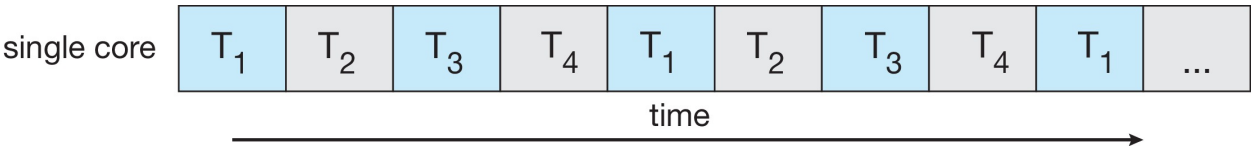


*Figure 2* Concurrent execution on a single-core system

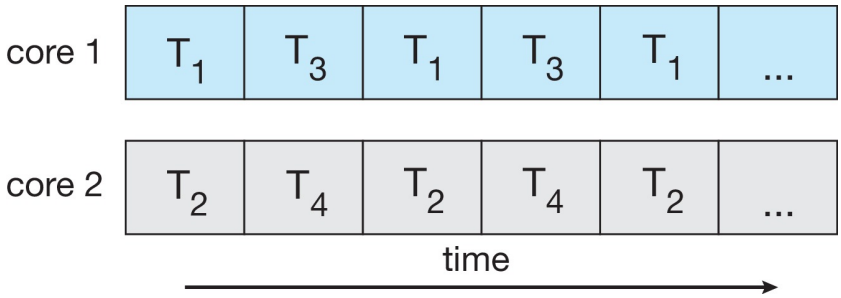


*Figure 3* Parallel execution on a multicore system

### 4.2.1 Programming Challenges

Multicore systems requires designers to write scheduling algorithms that can use multiple processor cores to allow parallel execution. The challenges of multicore programming include:

- **Identifying tasks** – Examining applications to find areas that can be divided into separate (and possibly independent) concurrent tasks.
- **Balance** – While identifying tasks that can run in parallel, programmers must also ensure that the tasks perform equal work of equal value.
- **Data splitting** – As an application is divided into separate tasks, the data accessed and manipulated by them also need to be divided to run on separate cores.
- **Data dependency** – The data accessed by the tasks must be examined for dependencies between two or more tasks. Programmers must ensure proper synchronization of threads that shares data.
- **Testing and debugging** – When a program runs in parallel on multiple cores, many different execution paths are possible. Testing and debugging such concurrent programs is more difficult than single threaded applications.

### Amdahl's Law

It is a formula that identifies performance gains from adding additional cores to an application that has both serial and parallel components. If S is the portion of the application that must be performed serially on a system with N processing cores, the formula is:

$$speedup \leq \frac{1}{S + \frac{(1-S)}{N}}$$

An application 75% parallel and 25% serial, run on a system with 2 processing cores, will speedup 1.6 times. As N approaches infinity, speedup approaches 1 / S. For example, if 50% of an application is to be performed serially, the maximum speedup is 2, regardless of the number of processors core added. This is the fundamental principle behind Amdahl's law: Serial portion of an application has disproportionate effect on performance gained by adding additional cores.

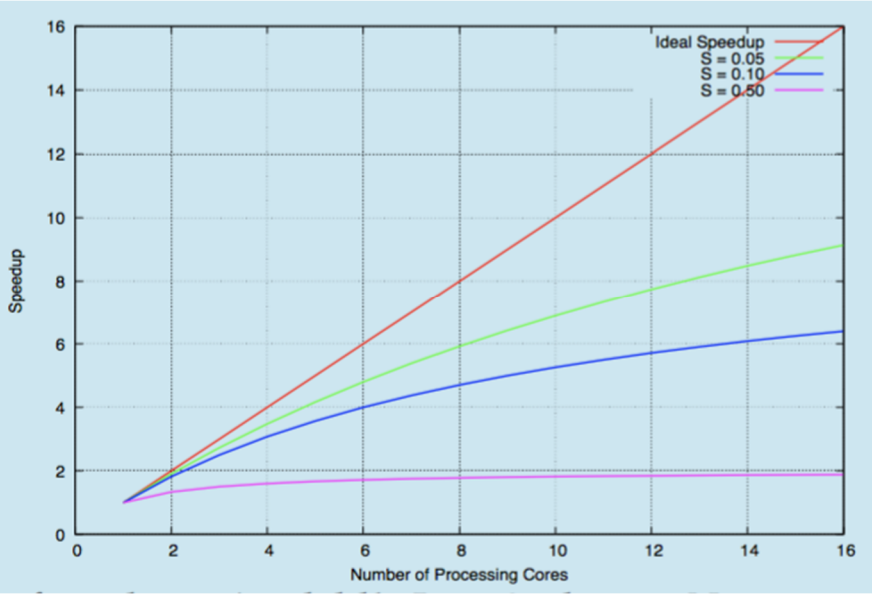


***Figure 4*** *Illustration of Amdahl's law*

## 4.2.2 Types of Parallelism

There are two types of parallelism, not mutually exclusive and an application may use both strategies:

- Data parallelism – It focuses on distributing subsets of the same data across multiple cores and performing the same operation on each.
- Task parallelism – It involves distributing threads across cores with each thread performing a unique operation.

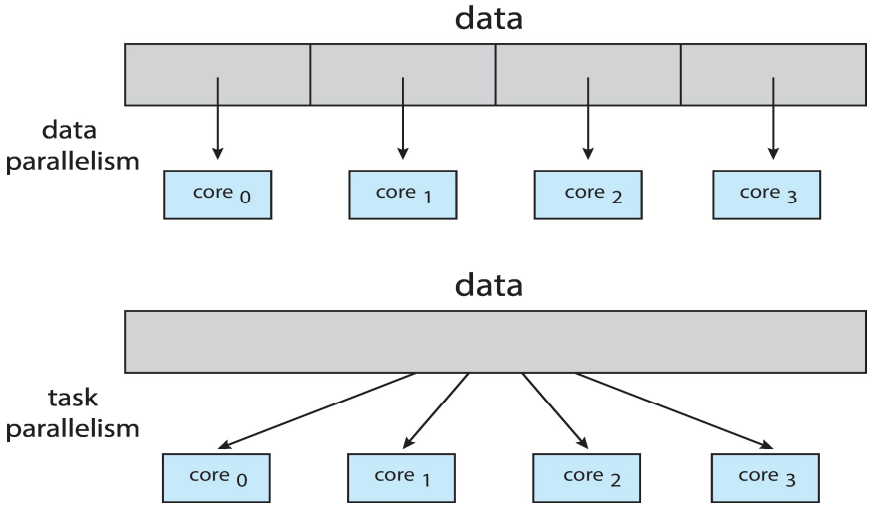


***Figure 5*** *Data and task parallelism*

## 4.3 Multithreading Models

Supports for threads can be provided either at user level (for user threads) or at kernel level (for kernel threads). User threads are managed without kernel support, whereas kernel threads are supported and managed directly by the OS. Virtually all contemporary OS (including Windows, Linux and macOS) support kernel threads.

In this section we look at **three** common ways of establishing connection between user threads and kernel threads.

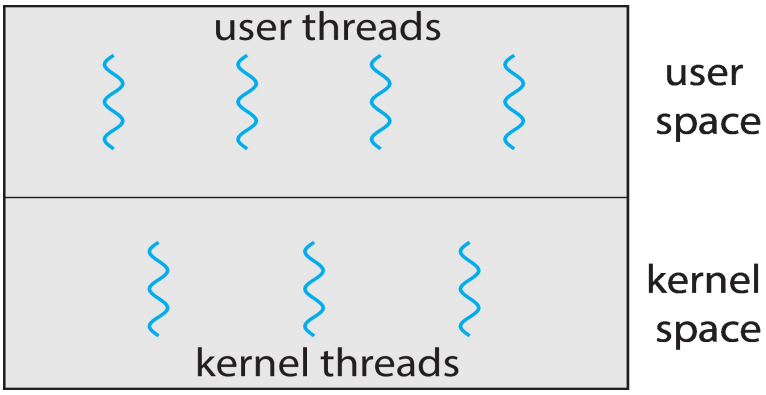


***Figure 6*** *User and kernel threads*

### 4.3.1 Many-to-One Model

This model maps many user-level threads to one kernel thread. Thread management is done by the thread library in user space. Because only one thread can access the kernel at a time, multiple threads are unable to run in parallel on multicore systems. So very few systems continue to use the model.

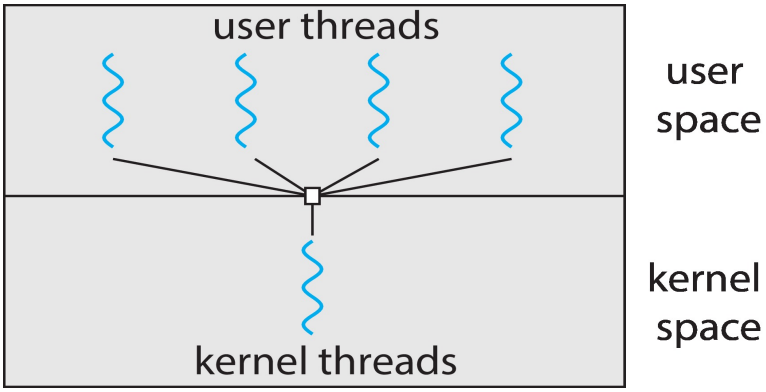


***Figure 7*** *Many-to-one model*

### 4.3.2 One-to-One Model

This model maps each user thread to a kernel thread. It provides more concurrency than the many-to-one model. It also allows multiple threads to run in parallel in multiprocessors. As each user thread requires a corresponding kernel thread, it downgrades the performance.

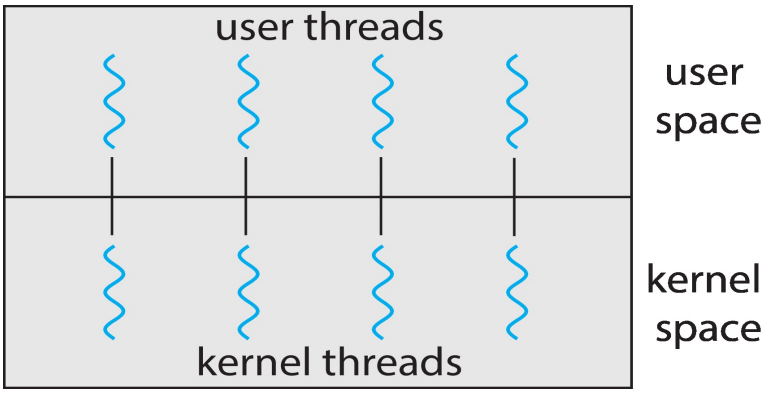


***Figure 8*** *One-to-one model*

### 4.3.3 Many-to-Many Model

This model allows many user level threads to be mapped to many kernel (smaller or equal number) threads. This allows the developers to create as many threads as needed, and the corresponding kernel threads can run in parallel on a multiprocessor.

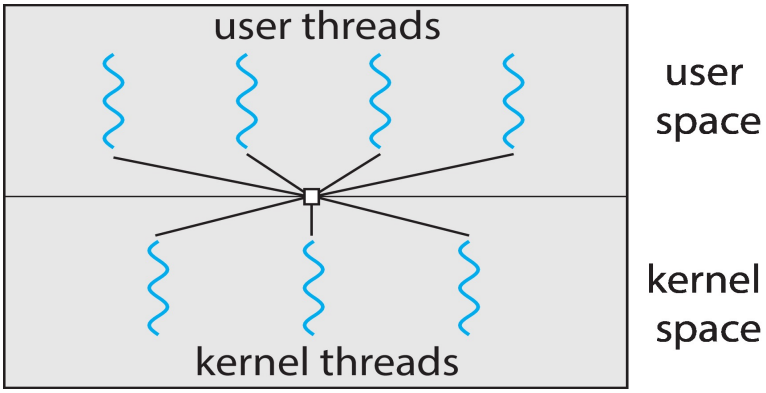


***Figure 9*** *Many-to-many model*

Despite the flexibility, implementation of this model is hard. In practice most OS use one-to one model.

## 4.4 Thread Libraries

A thread library provides the programmer with an API for creating and managing threads. **Three** main libraries are in use today:

- **POSIX Pthreads** – provided as either a user-level or a kernel-level library, common in UNIX OS.
- **Windows** – a kernel level library available in Windows systems.
- **Java** – allows threads to be created and managed directly in Java programs.