**Chapter 4 Thread and Concurrency**

# 4.1 Overview

## Motivation

A thread represents a basic unit of CPU utilization, and threads belonging to the same process share many of the process resources, including code and data.

## Benefit

There are four primary benefts to multithreaded applications:

- Responsiveness

- Resource Sharing

- Economy

- Scalability

# 4.2 Multicore Programming

Concurrency exists when multiple threads are making progress, whereas parallelism exists when multiple threads are making progress simultaneously. On a system with a single CPU, only concurrency is possible; parallelism requires a multicore system that provides multiple CPUs.

## Programming Challenges:

There are several challenges in designing multithreaded applications.

They include dividing and balancing the work, dividing the data between the different threads, and identifying any data dependencies.

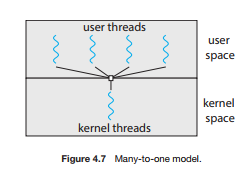
Finally, multithreaded programs are especially challenging to test and debug.

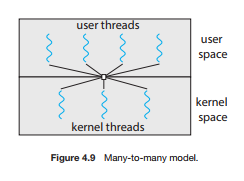
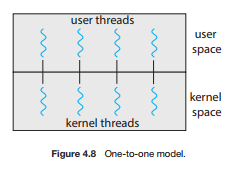
## Type of Parallelism

* **Data parallelism** distributes subsets of the same data across different computing cores and performs the same operation on each core.
* **Task parallelism** distributes not data but tasks across multiple cores. Each task is running a unique operation.

# 4.3 Multithreading Models

User applications create user-level threads, which must ultimately be mapped to kernel threads to execute on a CPU.

- The ***many-to-one*** model maps many user-level threads to one kernel thread.

******- Other approaches include the ***one-to-one*** and ***many-to-man*** ***y*** models.

# 4.4 Thread Libraries

A thread library provides an API for creating and managing threads.

Three common thread libraries include Windows, Pthreads, and Java threading.

Windows is for the Windows system only, while Pthreads is available for POSIX-compatible systems such as UNIX, Linux, and macOS. Java threads will run on any system that supports a Java virtual machine.

# 4.5 Implicit Threading

Implicit threading involves identifying tasks—not threads—and allowing languages or API frameworks to create and manage threads. There are several approaches to implicit threading, including thread pools, fork-join frameworks, and Grand Central Dispatch. Implicit threading is becoming an increasingly common technique for programmers to use in developing concurrent and parallel applications.

# 4.6 Threading Issues

Threads may be terminated using either asynchronous or deferred cancellation. Asynchronous cancellation stops a thread immediately, even if it is in the middle of performing an update. Deferred cancellation informs a thread that it should terminate but allows the thread to terminate in an orderly fashion. In most circumstances, deferred cancellation is preferred to asynchronous termination.