# Chapter 4: Threads

Chapter 4: Threads

- Overview
- Multicore Programming
- Multithreading Models
- Thread Libraries
- Implicit Threading
- Threading Issues
- Operating System Examples
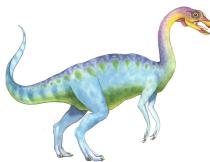
Objectives

- To introduce the notion of a thread—a fundamental unit of CPU utilization that forms the basis of multithreaded computer systems

- To discuss the APIs for the Pthreads, Windows, and Java thread libraries

- To explore several strategies that provide implicit threading

- To examine issues related to multithreaded programming

- To cover operating system support for threads in Windows and Linux

Motivation
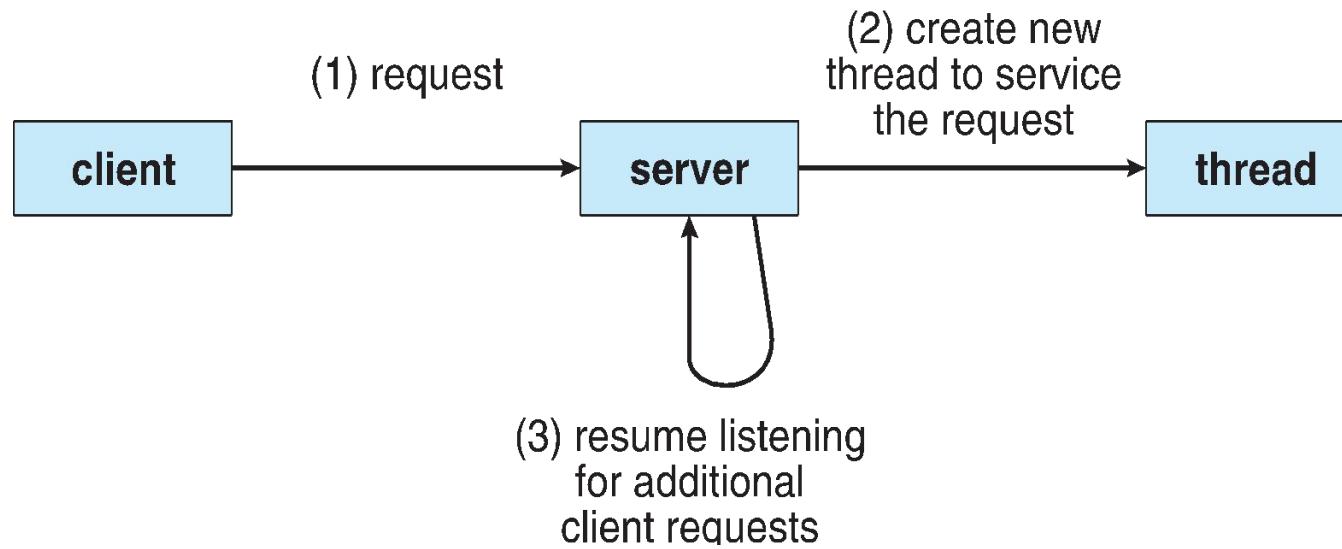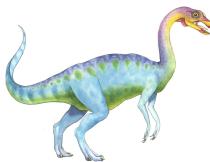
- Most modern applications are multithreaded

- Threads run within application

- Multiple tasks with the application can be implemented by separate threads

  - Update display

  - Fetch data

  - Spell checking

  - Answer a network request

- Process creation is heavy-weight while thread creation is light-weight

- Can simplify code, increase efficiency

- Kernels are generally multithreaded

Multithreaded Server Architecture

Benefits

- **Responsiveness –** may allow continued execution if part of process is blocked, especially important for user interfaces

- **Resource Sharing –** threads share resources of process, easier than shared memory or message passing

- **Economy –** cheaper than process creation, thread switching lower overhead than context switching

- **Scalability –** process can take advantage of multiprocessor architectures

Multicore Programming

- **Multicore** or **multiprocessor** systems putting pressure on programmers, challenges include:

  - **Dividing activities**

  - **Balance**

  - **Data splitting**

  - **Data dependency**

  - **Testing and debugging**

- *Parallelism* implies a system can perform more than one task simultaneously

- *Concurrency* supports more than one task making progress

  - Single processor / core, scheduler providing concurrency

Multicore Programming (Cont.)

- Types of parallelism

  - **Data parallelism** – distributes subsets of the same data across multiple cores, same operation on each

  - **Task parallelism** – distributing threads across cores, each thread performing unique operation

- As # of threads grows, so does architectural support for threading

  - CPUs have cores as well as *hardware threads*

 - Consider Oracle SPARC T4 with 8 cores, and 8 hardware threads per core
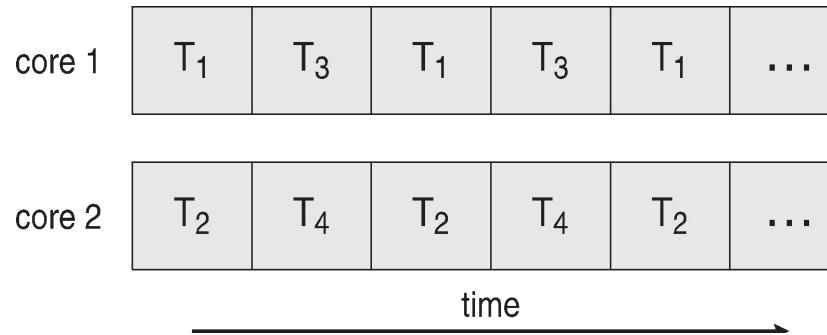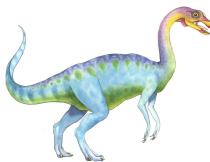
Concurrency vs. Parallelism

- **Concurrent execution on single-core system:**

| single core | T₁ | T₂ | T₃ | T₄ | T₁ | T₂ | T₃ | T₄ | T₁ | ... |

time →
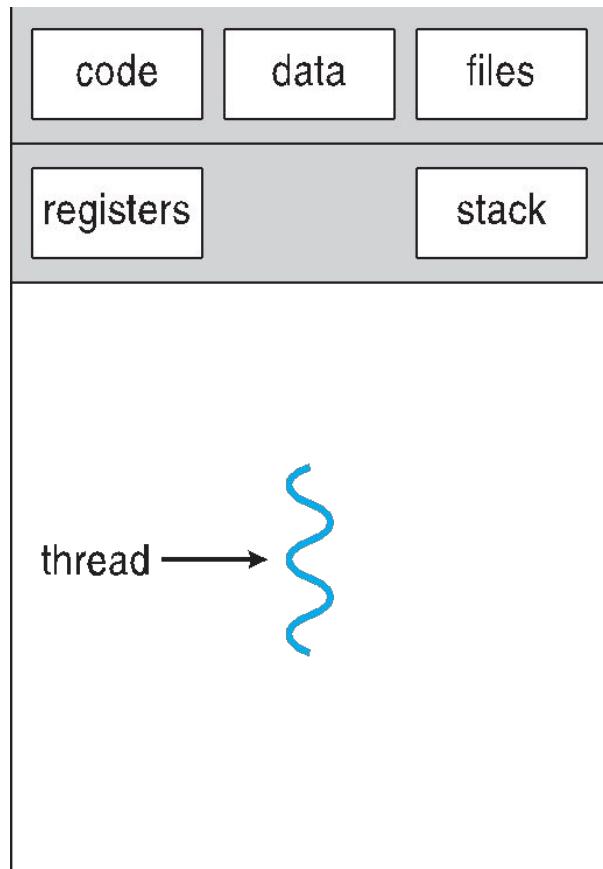
- **Parallelism on a multi-core system:**

| core 1 | T₁ | T₃ | T₁ | T₃ | T₁ | ... |
| core 2 | T₂ | T₄ | T₂ | T₄ | T₂ | ... |

time →
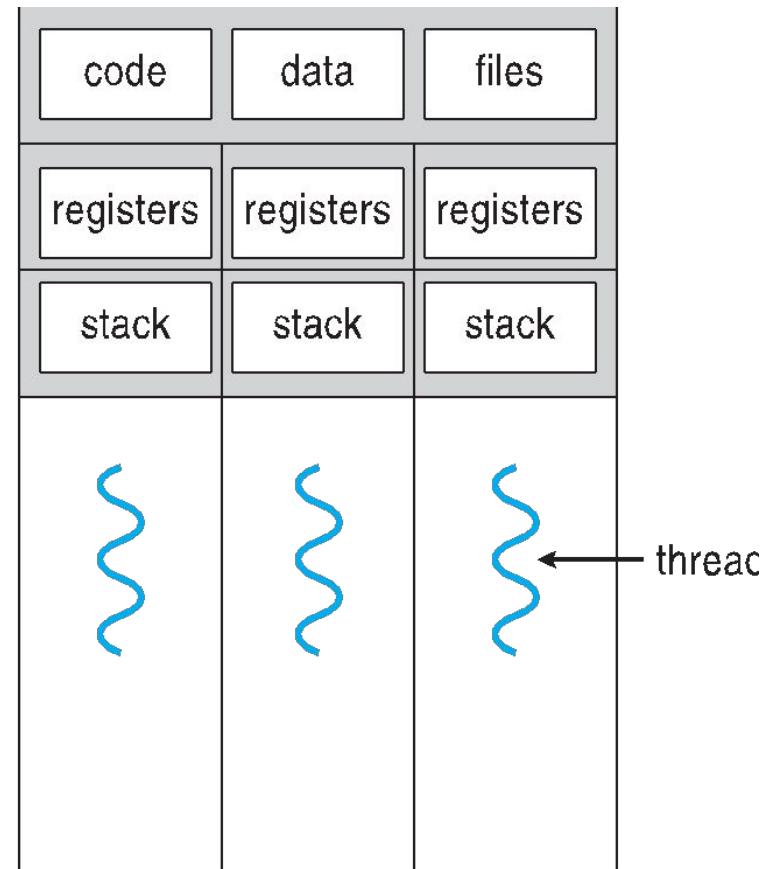
Single and Multithreaded Processes
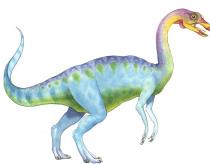
single-threaded process

multithreaded process

Amdahl's Law

- Identifies performance gains from adding additional cores to an application that has both serial and parallel components

- S is serial portion

- N processing cores

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

- That is, if application is 75% parallel / 25% serial, moving from 1 to 2 cores results in speedup of 1.6 times

- As N approaches infinity, speedup approaches $1 / S$

  Serial portion of an application has disproportionate effect on performance gained by adding additional cores

- But does the law take into account contemporary multicore systems?
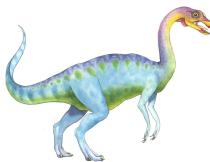
User Threads and Kernel Threads

- **User threads** - management done by user-level threads library

- Three primary thread libraries:
  - POSIX **Pthreads**
  - Windows threads
  - Java threads

- **Kernel threads** - Supported by the Kernel

- Examples – virtually all general purpose operating systems, including:
  - Windows
  - Solaris
  - Linux
  - Tru64 UNIX
  - Mac OS X

Multithreading Models
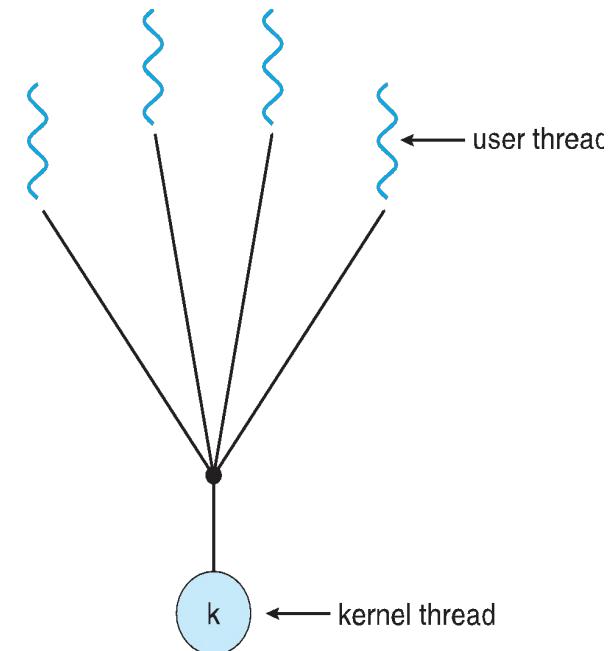
- Many-to-One

- One-to-One

- Many-to-Many

Many-to-One
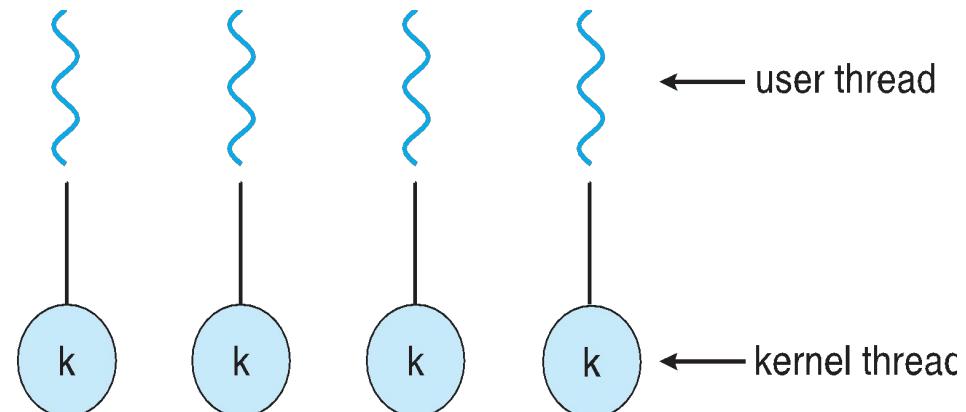
- Many user-level threads mapped to single kernel thread

- One thread blocking causes all to block

- Multiple threads may not run in parallel on muticore system because only one may be in kernel at a time

- Few systems currently use this model

- Examples:

  - **Solaris Green Threads**
  - **GNU Portable Threads**

One-to-One
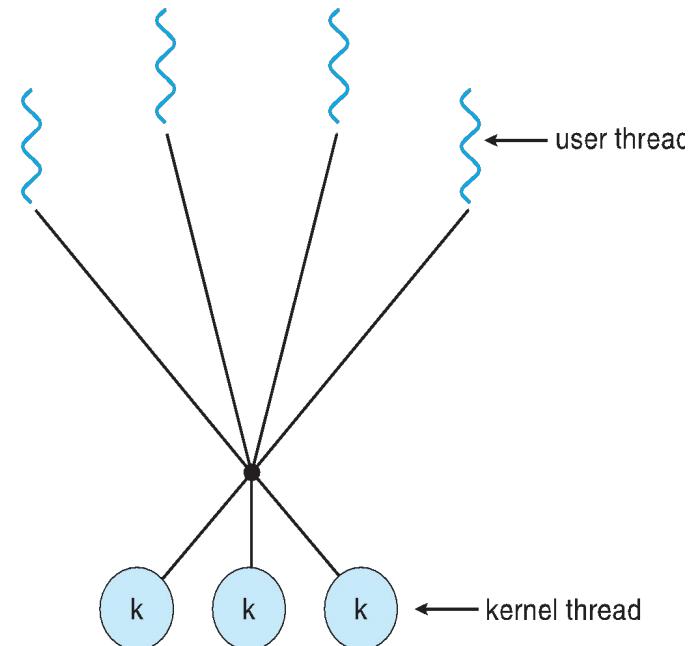
- Each user-level thread maps to kernel thread

- Creating a user-level thread creates a kernel thread

- More concurrency than many-to-one

- Number of threads per process sometimes restricted due to overhead

- Examples
  - Windows
  - Linux
 - Solaris 9 and later

Many-to-Many Model
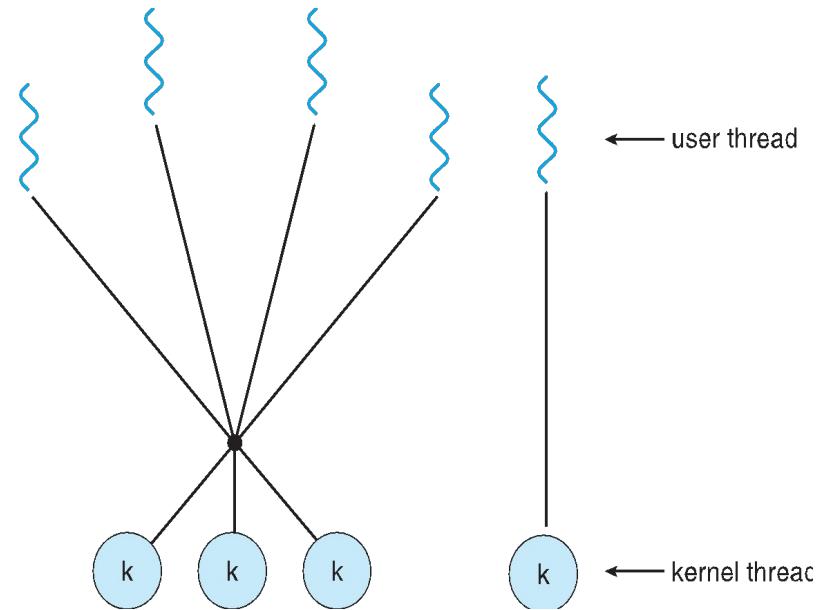
- Allows many user level threads to be mapped to many kernel threads

- Allows the operating system to create a sufficient number of kernel threads

- Solaris prior to version 9

- Windows with the *ThreadFiber* package

Two-level Model

- Similar to M:M, except that it allows a user thread to be **bound** to kernel thread

- Examples
  - IRIX
  - HP-UX
  - Tru64 UNIX
  - Solaris 8 and earlier

End of Chapter 4