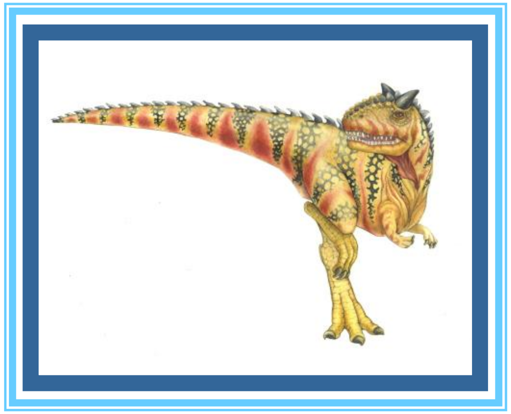
Chapter 4: Threads

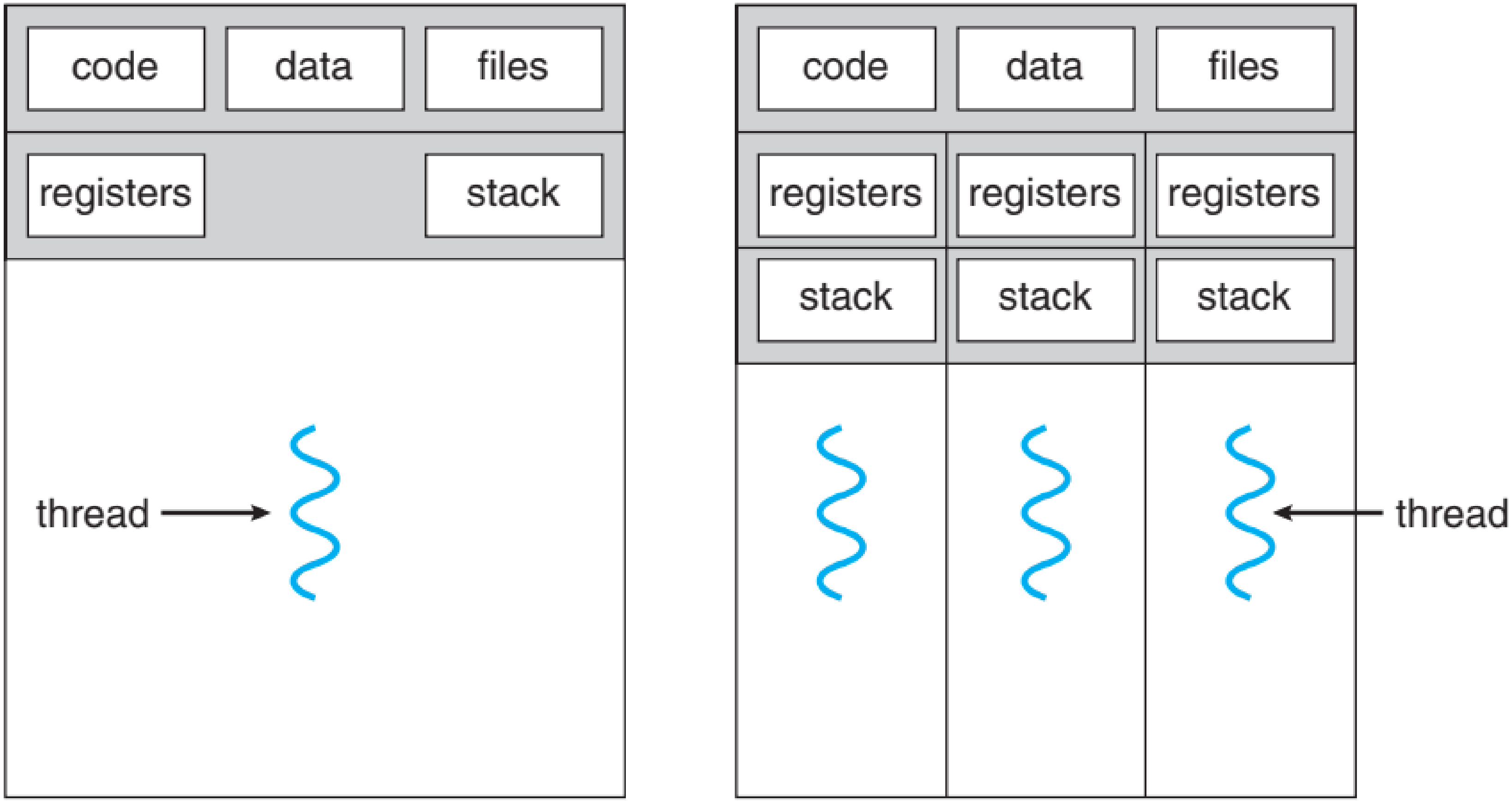


Motivation

- A thread is a basic unit of CPU utilization
  - Thread ID
  - Program counter
  - Register set
  - Stack
- A thread shares with other threads belonging to the same process
  - Code section
  - Data section
  - Operating-system resources, such as open files

A thread is a lightweight process

Thread



single-threaded process

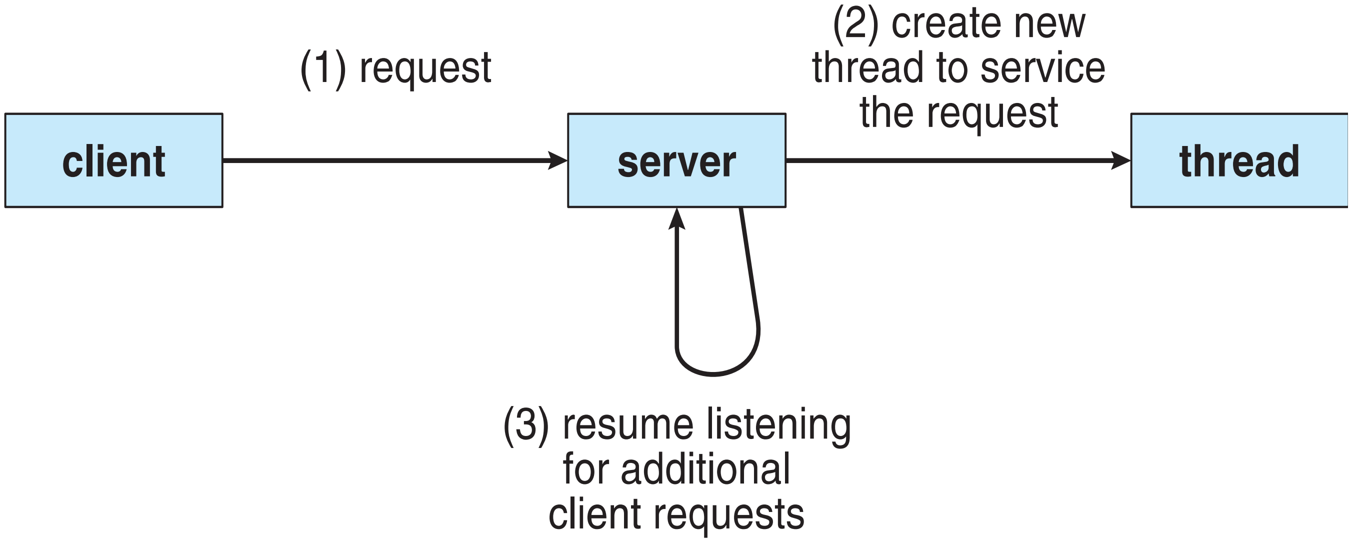
multithreaded process

Thread

- 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
- Significantly improve efficiency
  - Several CPU-intensive tasks in parallel across the multiple computing cores
- Kernels are generally multithreaded

Multithreaded Server Architecture

- A single application may be required to perform several similar tasks
- Example: A web server accepts client requests for web pages


(1) request
(2) create new thread to service the request
client
server
thread
(3) resume listening for additional client requests

Benefits

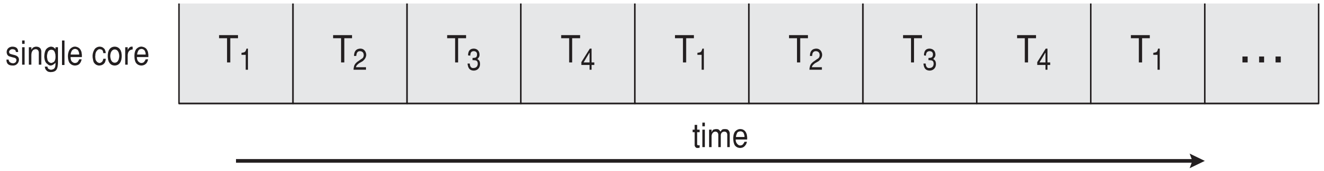
- **Responsiveness –** may allow continued execution if part of process is blocked, especially important for user interfaces
- **Resource Sharing –** threads share resources of process, easier than shared memory or message passing
- **Economy –** cheaper than process creation, thread switching lower 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

Concurrency vs. Parallelism

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

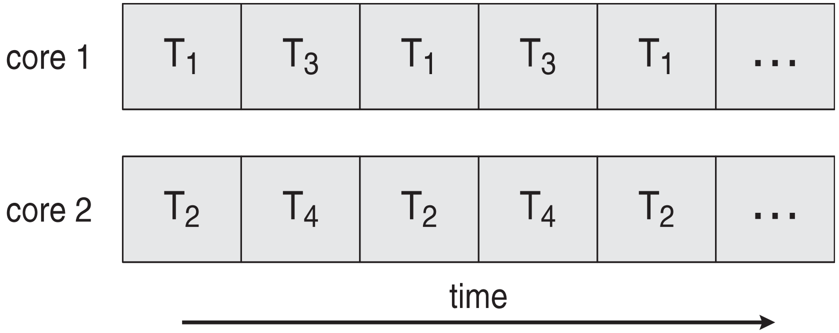


single core

T_1 T_2 T_3 T_4 T_1 T_2 T_3 T_4 T_1 . . .

time

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



core 1

T_1 T_3 T_1 T_3 T_1 . . .

core 2

T_2 T_4 T_2 T_4 T_2 . . .

time

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***

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

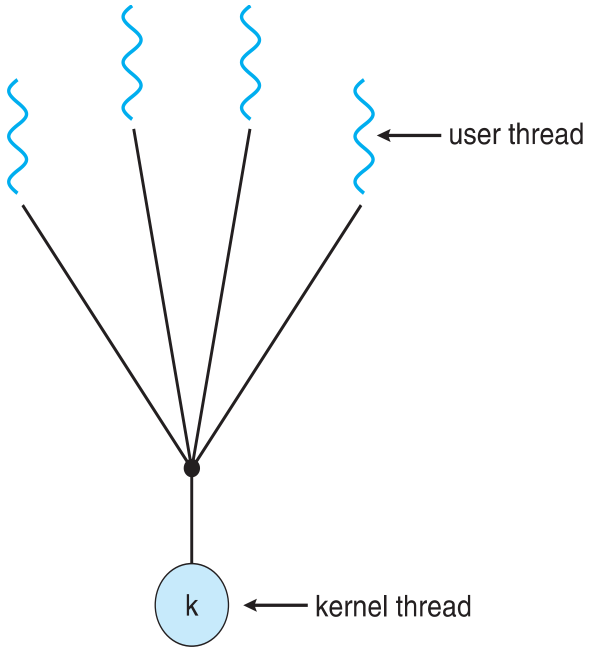
User Threads and Kernel Threads

- **User threads** - management done by user-level threads library
- **Kernel threads** - Supported by the Kernel
- Examples – virtually all general-purpose operating systems, including:
  - Windows
  - Solaris
  - Linux
  - Tru64 UNIX
  - Mac OS X

MULTITHREADING MODELS

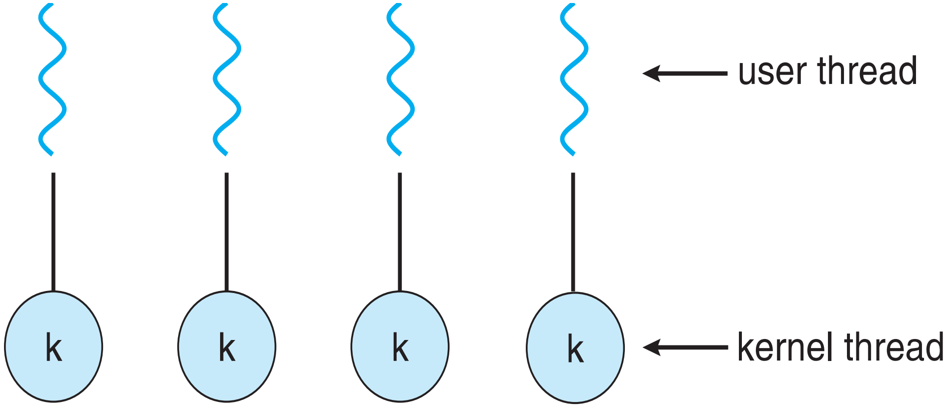
Many-to-One

- 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



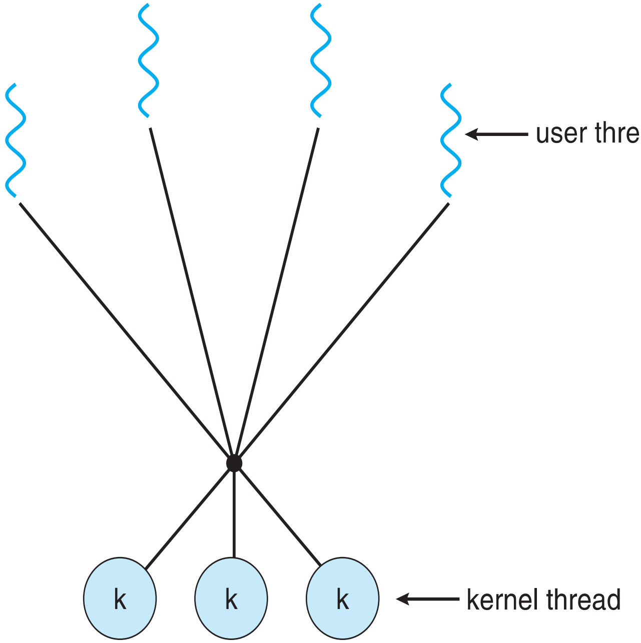
One-to-One

- 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



Threading Issues

Semantics of fork() and exec()

- Does **`fork()`** duplicate only the calling thread or all threads?
  - Some UNIXes have two versions of fork
- **`exec()`** usually works as normal – replace the running process including all threads

End of Chapter 4