Dr. Hakim Mellah
Department of Computer Science & Software Engineering
Concordia University, Montreal, Canada

Lecture 4: Threads & Concurrency
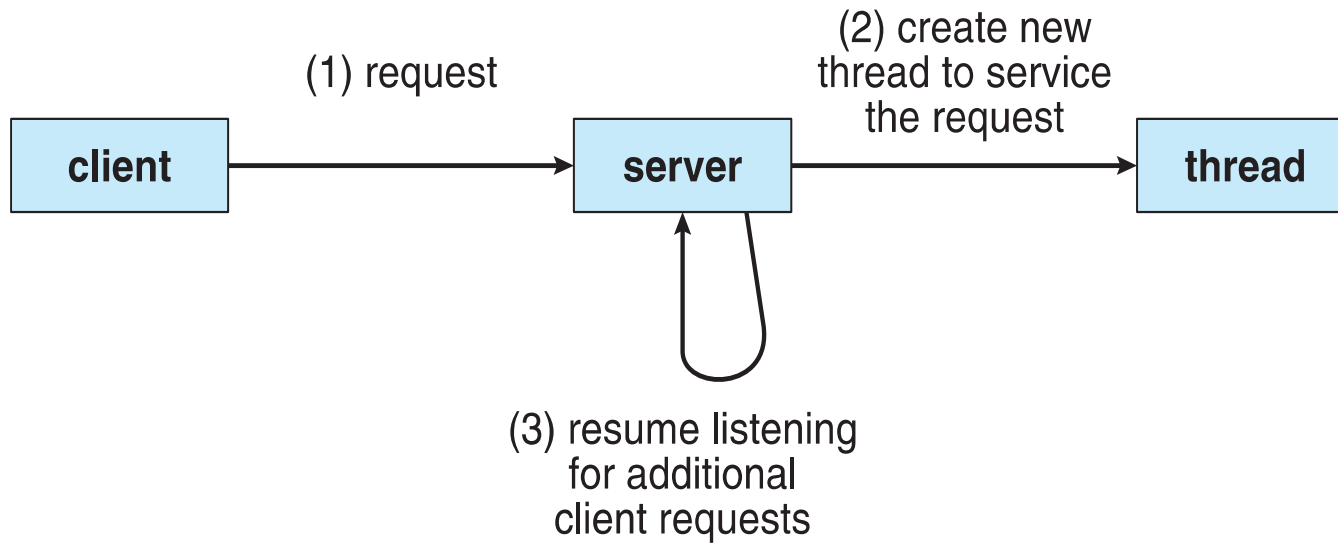
COMP 346: Operating Systems

Lecture 4: Threads

- Overview

- Multicore Programming

- Multithreading Models

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

(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
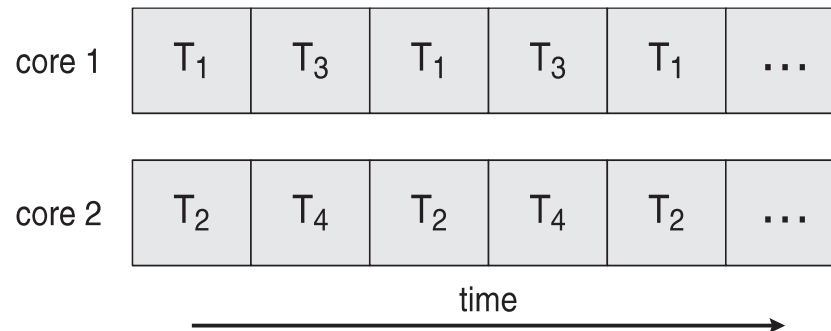
Multicore Programming

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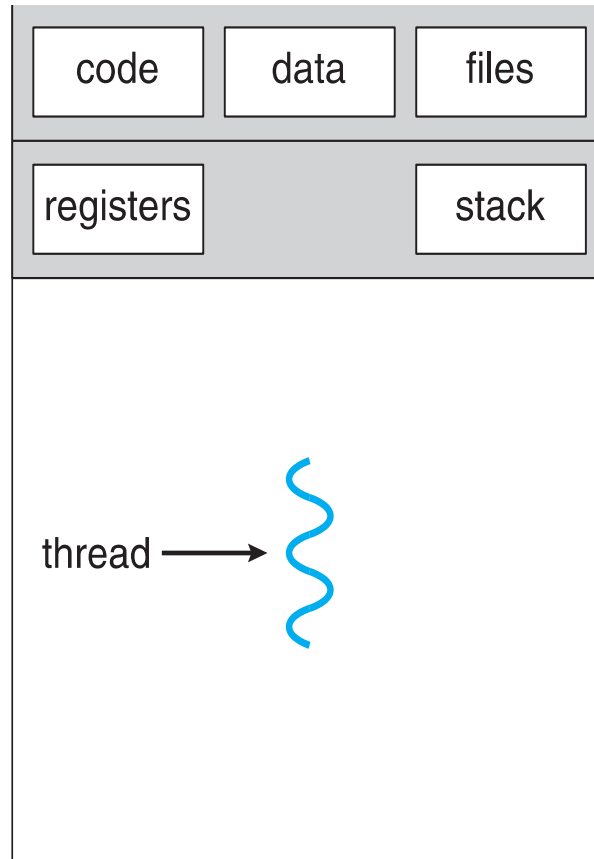
☐ **Concurrent execution on single-core system:**

single core	T_1	T_2	T_3	T_4	T_1	T_2	T_3	T_4	T_1	...

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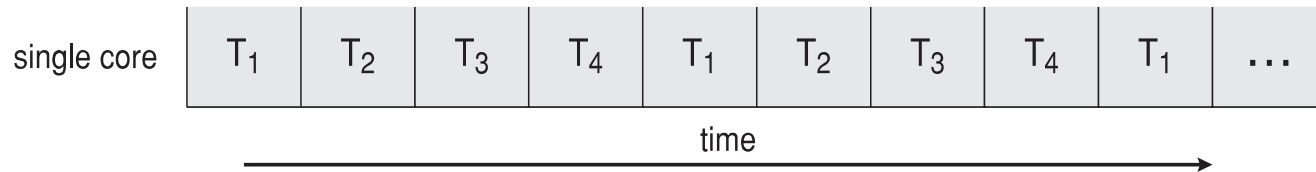
☐ **Parallelism on a multi-core system:**

core 1	T_1	T_3	T_1	T_3	T_1	...

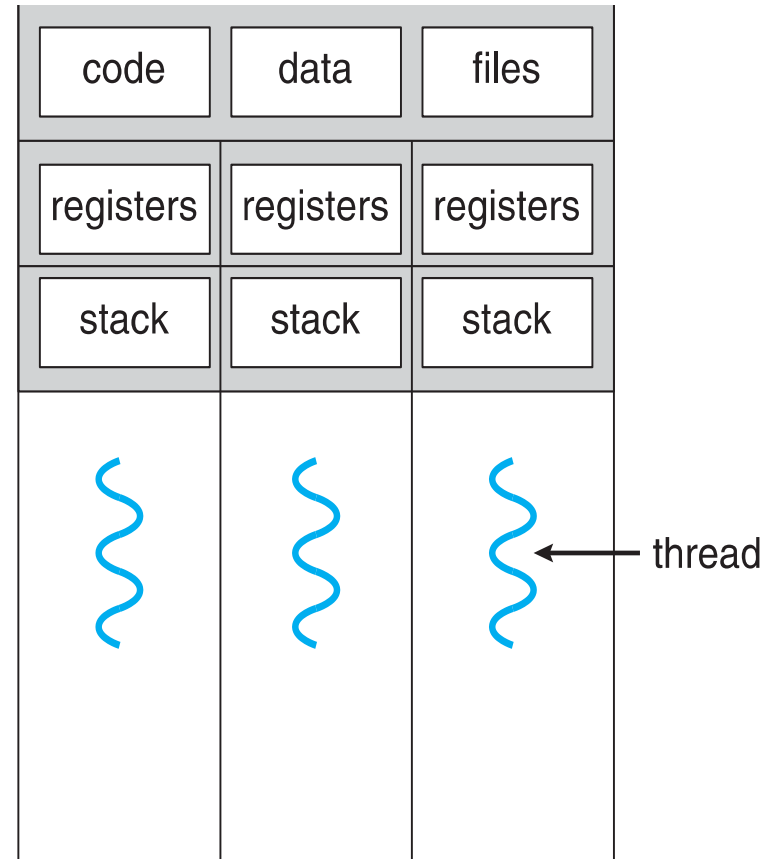| core 2	T_2	T_4	T_2	T_4	T_2	...

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

Single and Multithreaded Processes

code	data	files

registers		stack

thread ➝ 〜

single-threaded process

code	data	files

registers	registers	registers
stack	stack	stack

〜 〜 〜 ⟵ thread

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

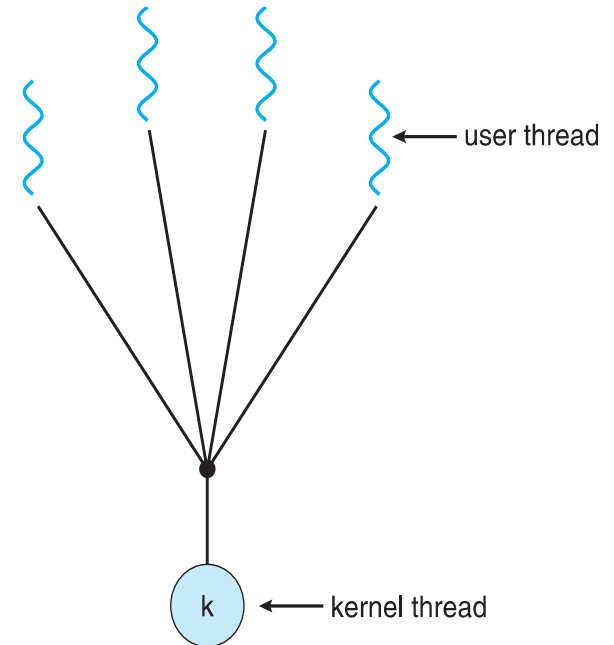# User Threads and Kernel Threads

- **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**
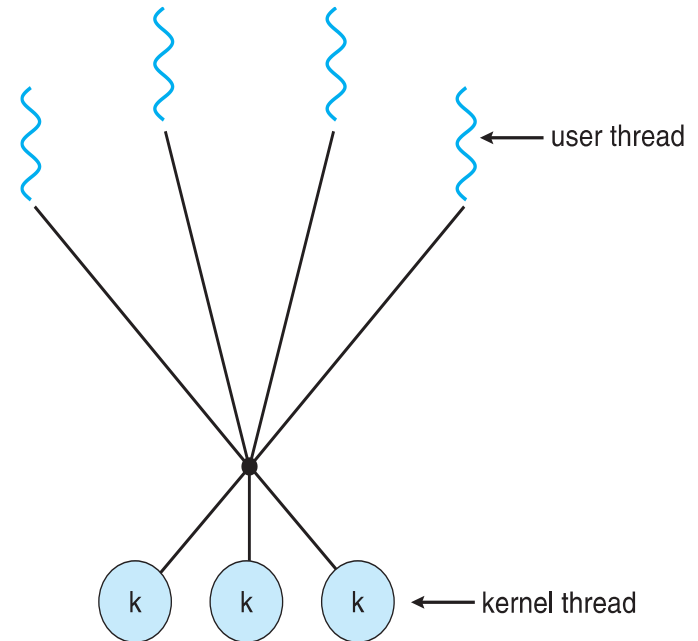

user thread

kernel thread

k

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

← user thread
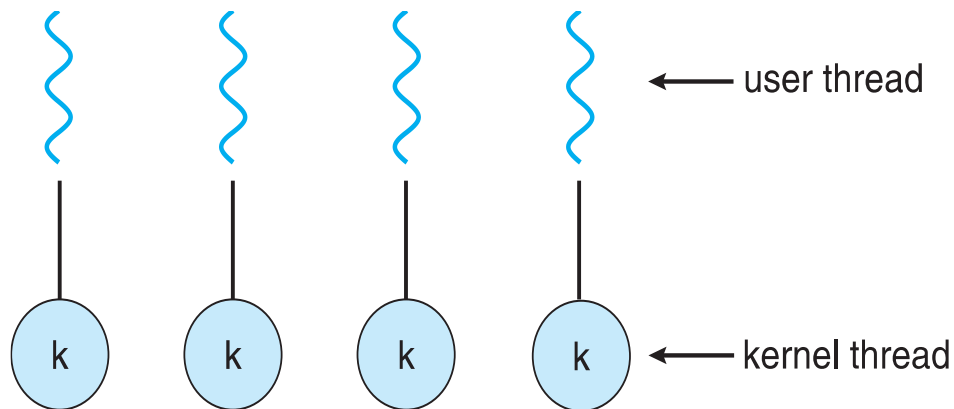
k   k   k   k   ← kernel thread

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

user thread

k k k ← kernel thread

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

user thread ←

kernel thread ←