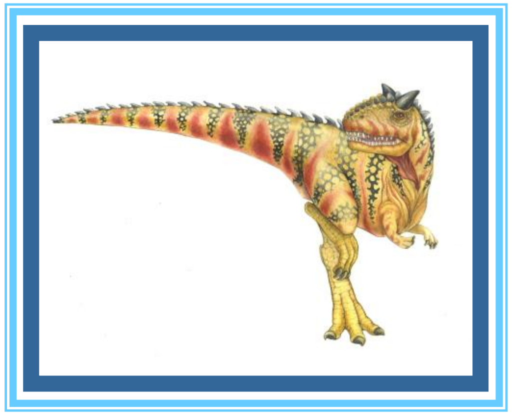
Operating Systems

Chapter 4: Threads



Lecturer: Dalya Samer

Chapter4 Outlines

- Introduction
- Multicore Programming
- Multithreading Models
- Benefits of threads
- Thread Libraries

Introduction (1/6)

- A **thread** is a basic unit of CPU utilization; it comprises a thread ID, a program counter, a register set, and a stack.

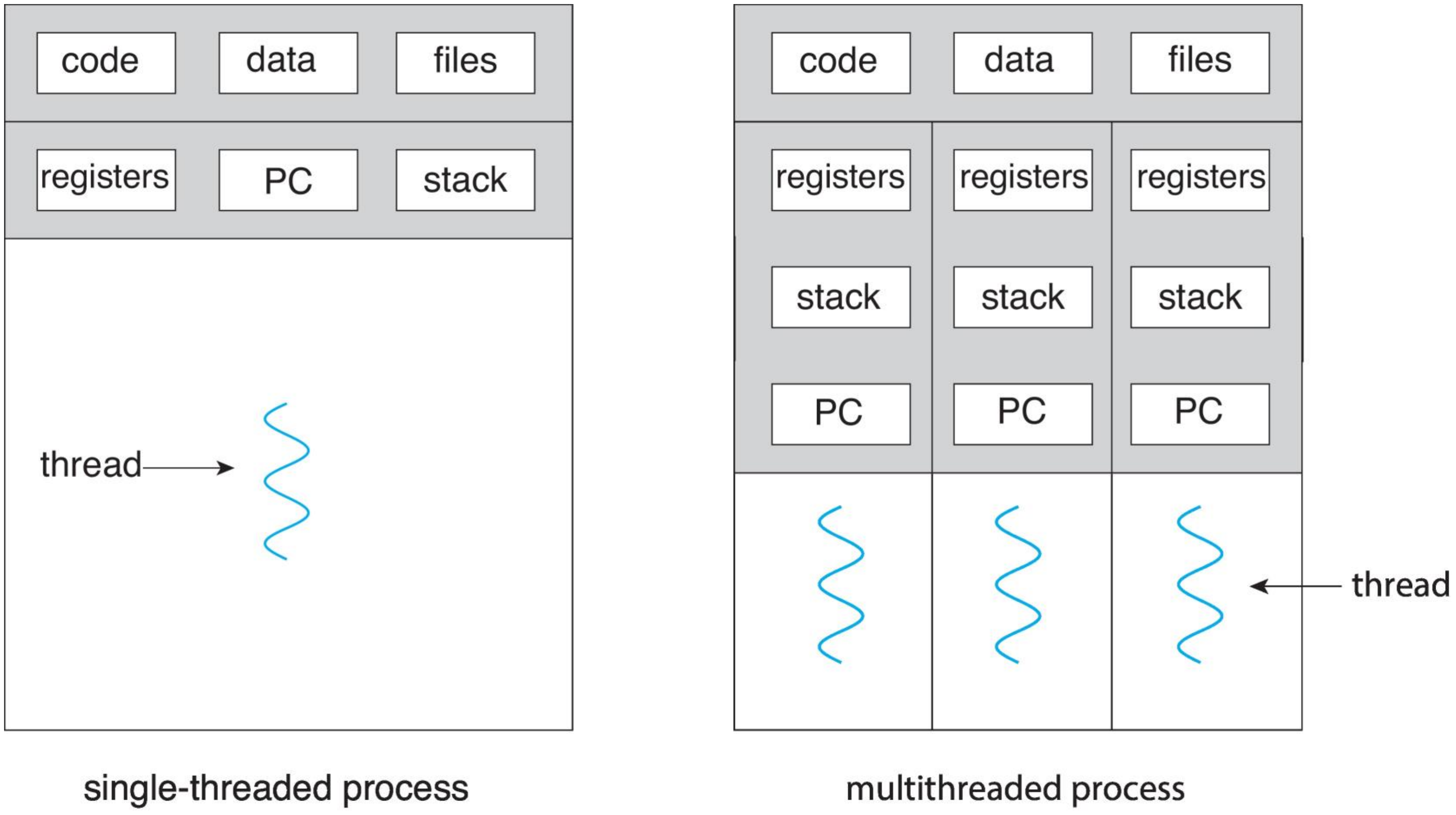
- It shares with other threads belonging to the same process its **code section**, data **section**, and other operating-system **resources**, such as open files and signals.

Introduction (2/6)

- Let's say, for example, a program is not capable of drawing pictures while reading keystrokes. The program must give its full attention to the keyboard input lacking the ability to handle more than one event at a time.

- The ideal solution to this problem is the seamless execution of two or more sections of a program at the same time. Threads allows us to do this.

Introduction (3/6)



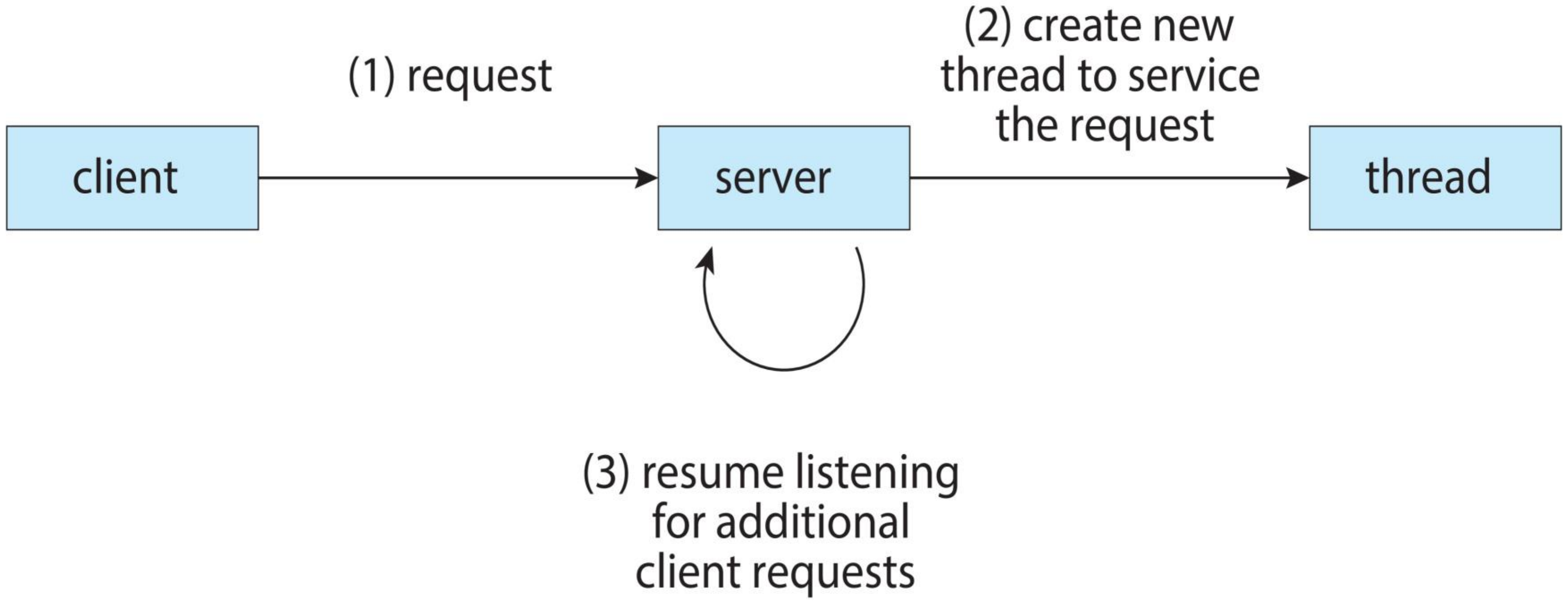
Introduction (4/6)

- Most software applications that run on modern computers are **multithreaded**.

- An application typically is implemented as a separate process with several threads of control.
  - ➢ A web browser might have one thread display images or text while another thread retrieves data from the network, for example.
  - ➢ A word processor may have a thread for displaying graphics, another thread for responding to keystrokes from the user, and a third thread for performing spelling and grammar checking in the background.

Introduction (5/6)

•**Multithreaded Server Architecture**

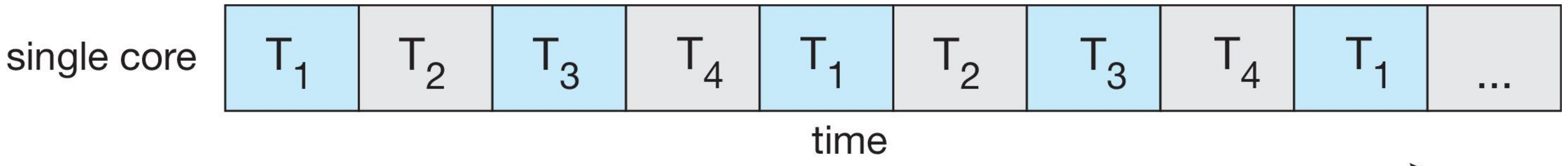

client
(1) request
server
(2) create new thread to service the request
thread
(3) resume listening for additional client requests

Introduction (6/6)

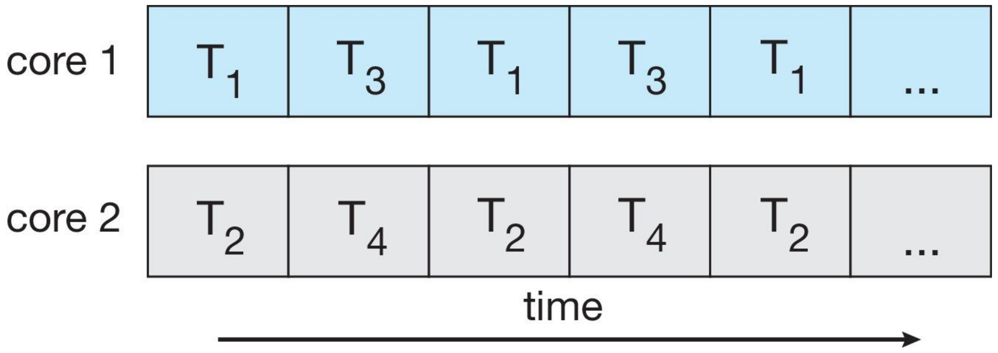
- Finally, most operating-system kernels are now multithreaded. Several threads operate in the kernel, and each thread performs a specific task, such as managing devices, managing memory, or interrupt handling.

Multicore Programming

- Multithreaded programming provides a mechanism for more efficient use of these multicore or multiprocessor systems.



Concurrent execution on single-core system

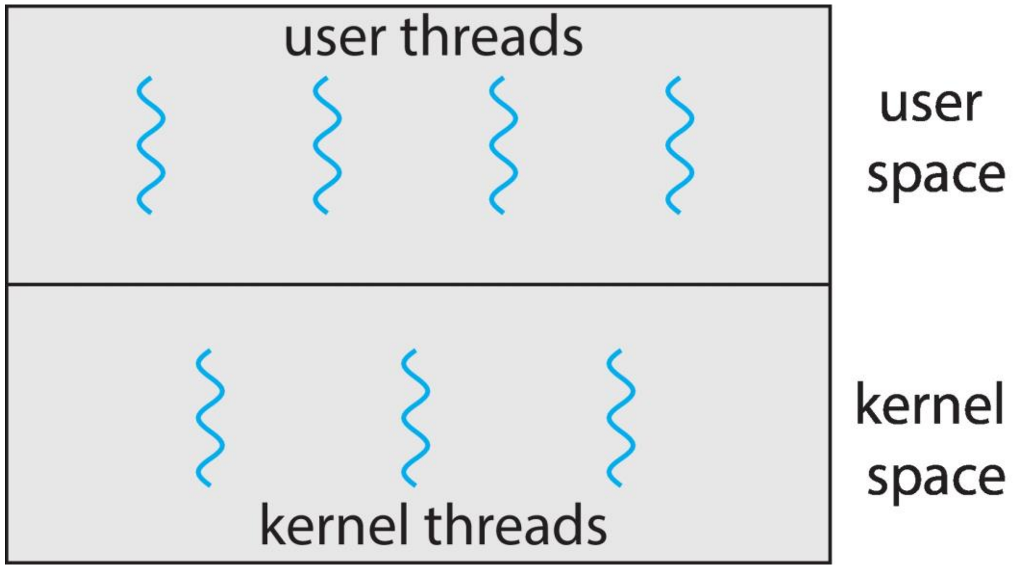

core 1
T_1 T_3 T_1 T_3 T_1 ...
core 2
T_2 T_4 T_2 T_4 T_2 ...
time

Parallelism on a multi-core system

Multithreading Models (1/6)

- Support for threads may be provided either at the user level, for **user threads**, or by the kernel, for **kernel threads**.
- User threads are supported above the kernel and are managed without kernel support, whereas kernel threads are supported and managed directly by the operating system.

User and Kernel Threads



Multithreading Models (2/6)

- Ultimately, a relationship must exist between user threads and kernel threads.
- We look at **three** common models:

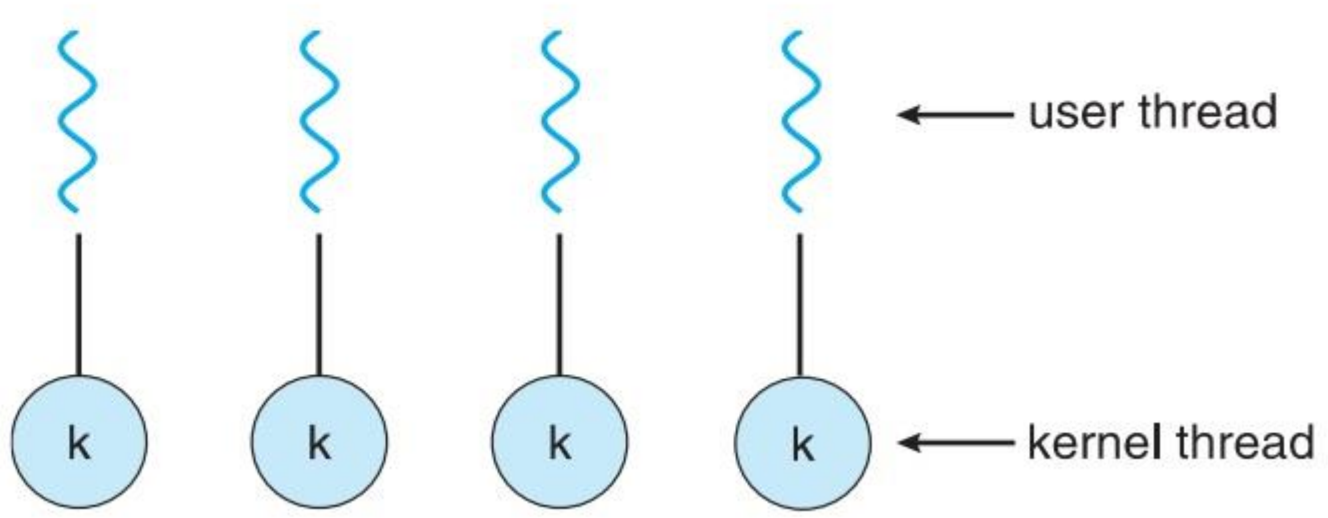
➢ the one-to-one model

➢ the many-to-one model

➢ the many-to-many model.

Multithreading Models (3/6)

- **One-to-One model (1/3)**



Multithreading Models (3/6)

- **One-to-One model (2/3)**
  - Each user-level thread maps to kernel thread
  - Creating a user-level thread creates a kernel thread
  - More concurrency than many-to-one
  - Examples
    - Windows
    - Linux

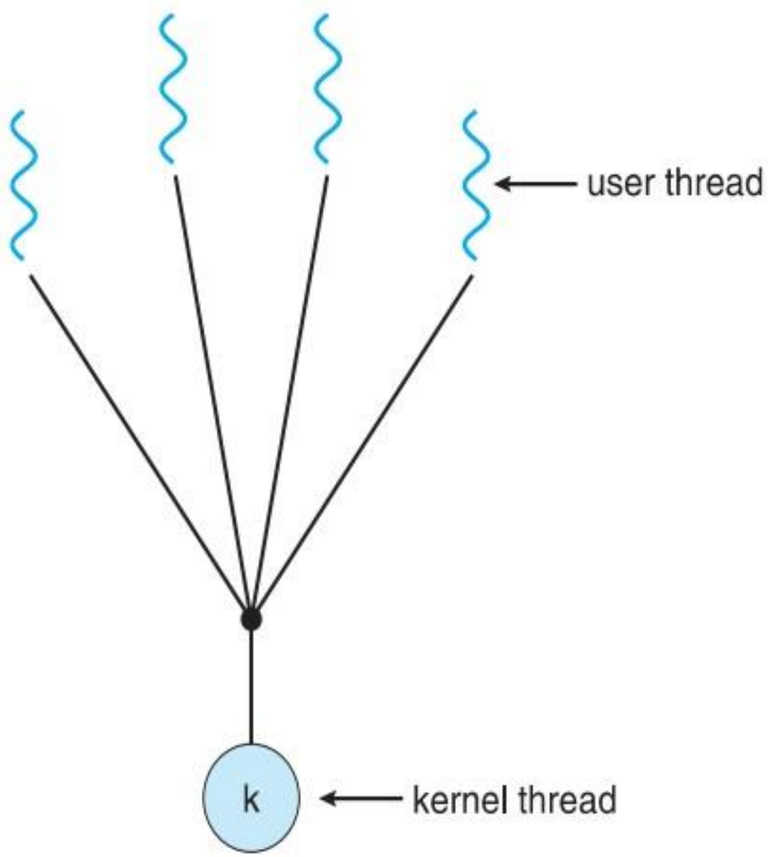
Multithreading Models (3/6)

- **One-to-One model (3/3)**

➢The only drawback to this model is that creating a user thread requires creating the corresponding kernel thread. Because the overhead of creating kernel threads can burden the performance of an application, most implementations of this model **restrict** the number of threads supported **by the system**. Linux, along with the family of Windows operating systems, implement the one-to-one model.

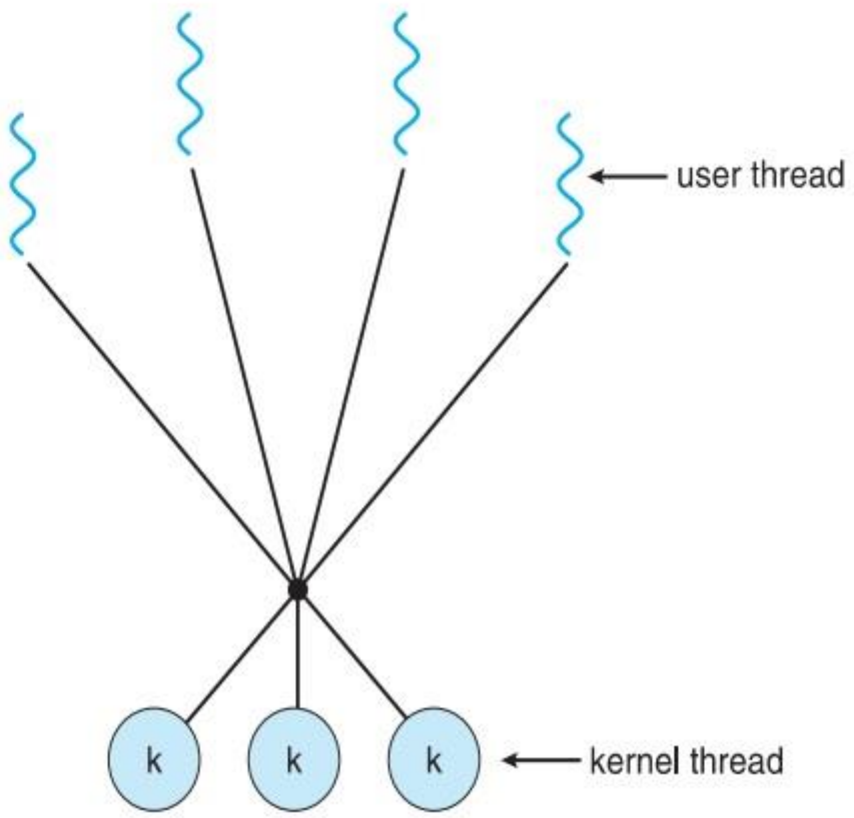
Multithreading Models (4/6)

- **Many-to-One model**



Multithreading Models (5/6)

- **Many-to-Many Model (1/2)**



Multithreading Models (5/6)

•**Many-to-Many Model (2/2)**

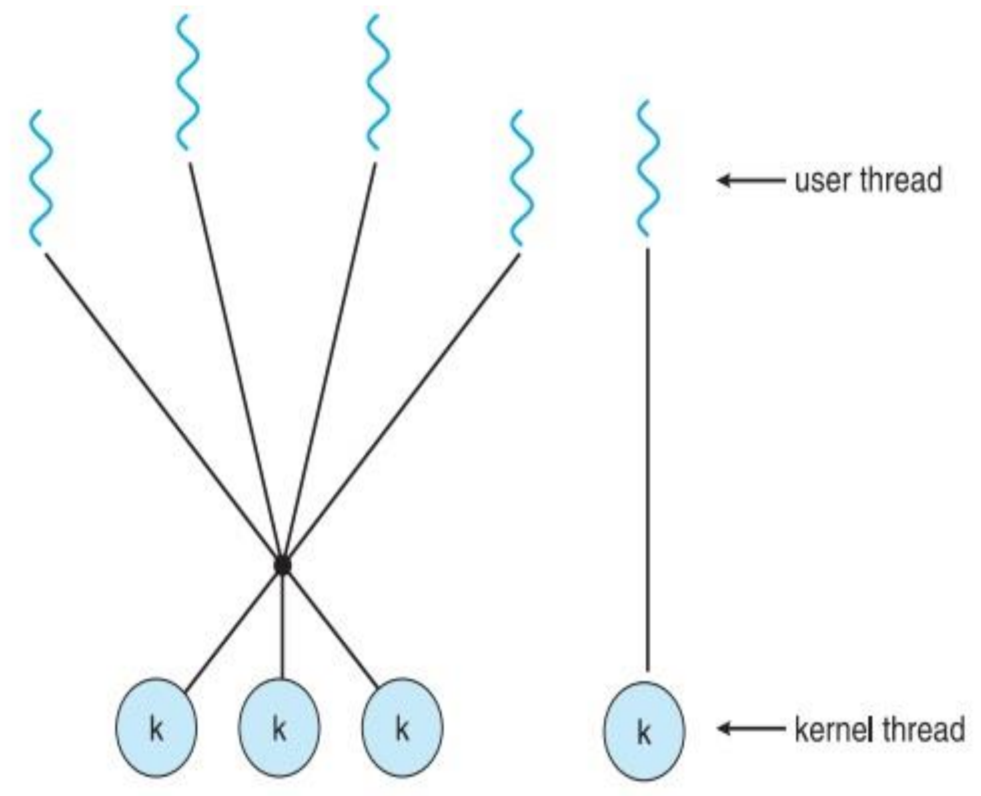
➢Allows many user level threads to be mapped to many kernel threads.

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

➢Allows the developer to create as many user threads as she wishes, it does not result in true concurrency, because the kernel can schedule only one thread at a time.

Multithreading Models (6/6)

- **Two-level Model**
  - Similar to Many to Many, except that it allows a user thread to be **bound** to kernel thread



Benefits of threads

- **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 multicore architectures.

Thread Libraries

- **Thread library** provides programmer with API for creating and managing threads.

- **User Threads** - management done by user-level threads library
- Three primary thread libraries are in use today:
  - POSIX **Pthreads**
  - Windows threads
  - Java threads

Pthreads Example

```
#include <pthread.h>
#include <stdio.h>

#include <stdlib.h>

int sum; /* this data is shared by the thread(s) */
void *runner(void *param); /* threads call this function */

int main(int argc, char *argv[])
{
  pthread_t tid; /* the thread identifier */
  pthread_attr_t attr; /* set of thread attributes */

  /* set the default attributes of the thread */
  pthread_attr_init(&attr);
  /* create the thread */
  pthread_create(&tid, &attr, runner, argv[1]);
  /* wait for the thread to exit */
  pthread_join(tid,NULL);

  printf("sum = %d\n",sum);
}
```

Pthreads Example (Cont.)

```
/* The thread will execute in this function */
void *runner(void *param)
{
   int i, upper = atoi(param);
   sum = 0;

   for (i = 1; i <= upper; i++)
      sum += i;

   pthread_exit(0);
}
```

Windows Multithreaded C Program

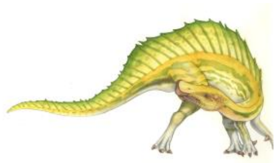
```
#include <windows.h>
#include <stdio.h>
DWORD Sum; /* data is shared by the thread(s) */

/* The thread will execute in this function */
DWORD WINAPI Summation(LPVOID Param)
{
   DWORD Upper = *(DWORD*)Param;
   for (DWORD i = 1; i <= Upper; i++)
      Sum += i;
   return 0;
}
```

Java Threads

- Java threads are managed by the JVM
- Typically implemented using the threads model provided by underlying OS
- Java threads may be created by:
  - Extending Thread class
  - Implementing the Runnable interface

```
public interface Runnable
{
    public abstract void run();
}
```

  - Standard practice is to implement Runnable interface



Java Threads

Implementing Runnable interface:

```
class Task implements Runnable
{
   public void run() {
      System.out.println("I am a thread.");
   }
}
```

Creating a thread:

```
Thread worker = new Thread(new Task());
worker.start();
```

Waiting on a thread:

```
try {
   worker.join();
}
catch (InterruptedException ie) { }
```

End of Chapter 4