Threads

Chapter 4

Motivation

- Most software applications that run on modern computers are **multithreaded**.
- **An application** typically is implemented as a **separate process** with several **threads of control.**
  - ***Web browser:*** A thread for display images or text while another thread retrieves data from the network.
  - ***Word processor:*** 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.

Motivation

- 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

Motivation

- In certain situations, a **single application** may be required to perform several similar tasks.
  - *For example, **a web server** accepts **client requests** for web pages, images, sound, and so forth.*
  - *A busy web server may have several (perhaps thousands of) clients concurrently accessing it.*
  - *If the web server ran as a **traditional single-threaded** process, it would be able to **service only one client at a time,** and a client might have to wait a very long time for its request to be serviced.*

Motivation (*Solutions*)

1. Have the server run as a single process that accepts requests. When the server receives a request, it creates a separate process to service that request. *But process creation is time consuming and resource intensive.*
2. If the new process will perform the same tasks as the existing process, why incur all that overhead?

   So, it is generally **more efficient** to use *one process that contains multiple threads.*

"Make the ***web-server process is multithreaded***, server will create a separate thread that listens for client requests. When a request is made, rather than creating another process, the server creates a new thread to service the request and resume listening for additional requests"

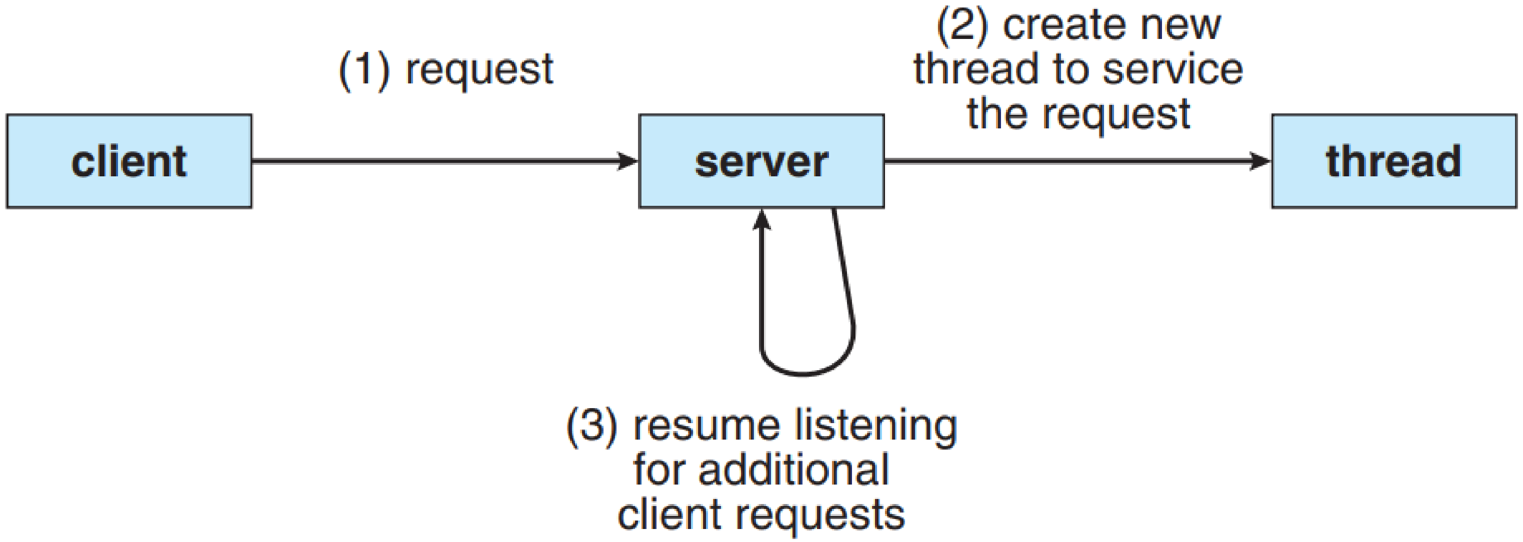


Figure 4.2 Multithreaded server architecture.

What about Operating System Kernel?

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.

For example, Solaris has a set of threads in the kernel specifically for interrupt handling; Linux uses a kernel thread for managing the amount of free memory in the system

Process: What we know, not again!

- Process is a program in execution
- Independent entity
- State at any point of time:
  - New, ready, run, terminated, wait, suspend ready and suspend block
- Every process has a Process Control Block
- OS allocates each process necessary to the process, not shared
- Heavyweight: Resource heavy

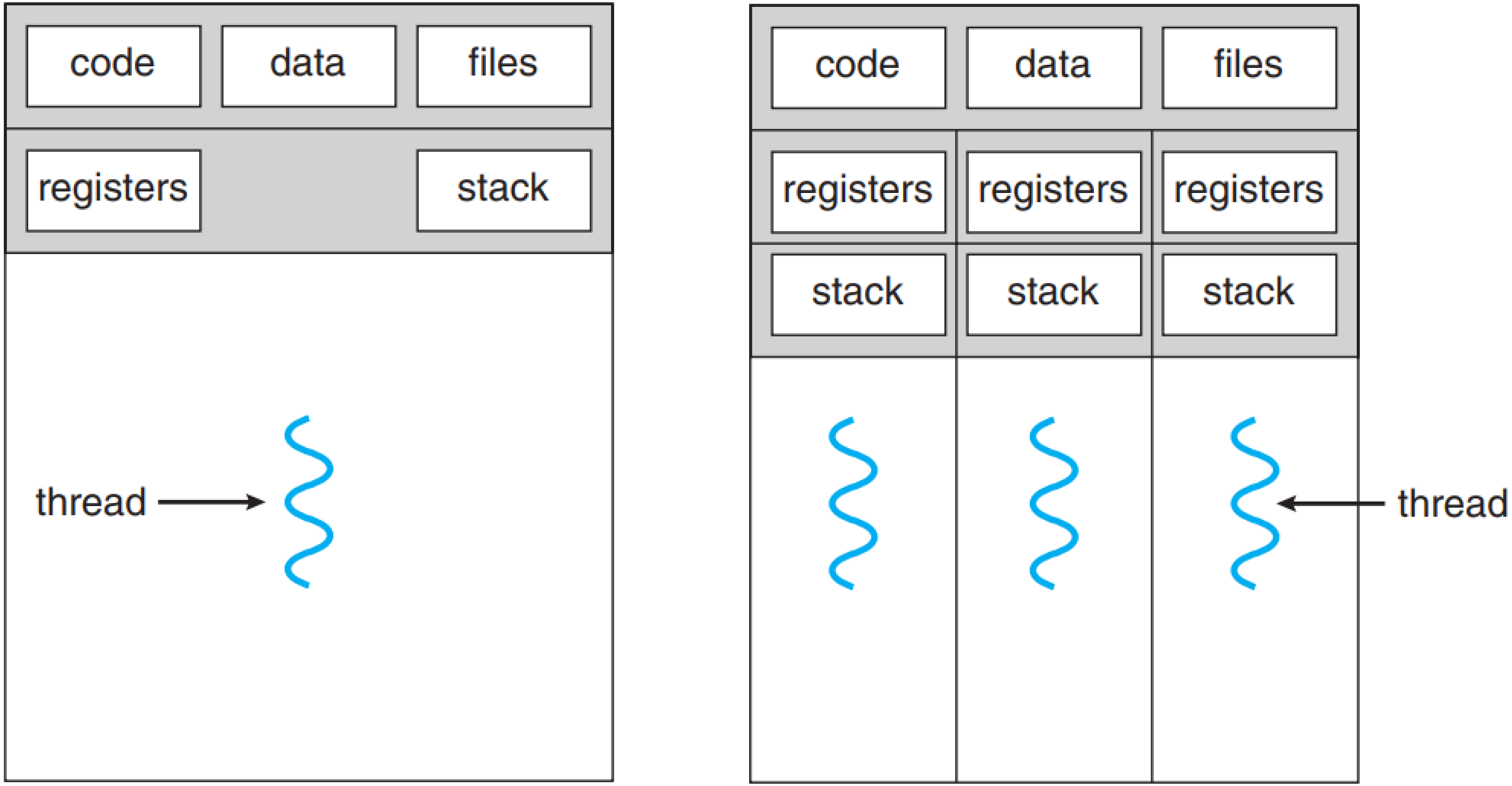
Thread: Overview

- **Thread:** A basic unit of CPU utilization | part of a process
- Comprises: Thread ID, program code, register set and stack
- 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.
- A traditional (or heavyweight) process has a single thread of control.
- If a process has multiple threads of control, it can perform more than one task at a time
- ***Quasi parallel execution***: Sequence but context switch is very fast

Process vs Thread: What we should know!

- Process is a program in execution | Thread is part of process
- Independent entity | Thread is dependent on process
- State at any point of time:
  - New, ready, block
- Process Control Block | Thread Control Block
- OS allocates each process necessary to the process, not shared | Thread shares resources among the various threads of the same process
- Heavyweight | Lightweight process: no need separate memory space
- Creation Time: More | Less
- Context Switch: More | Less

Single threaded and Multithreaded Processes

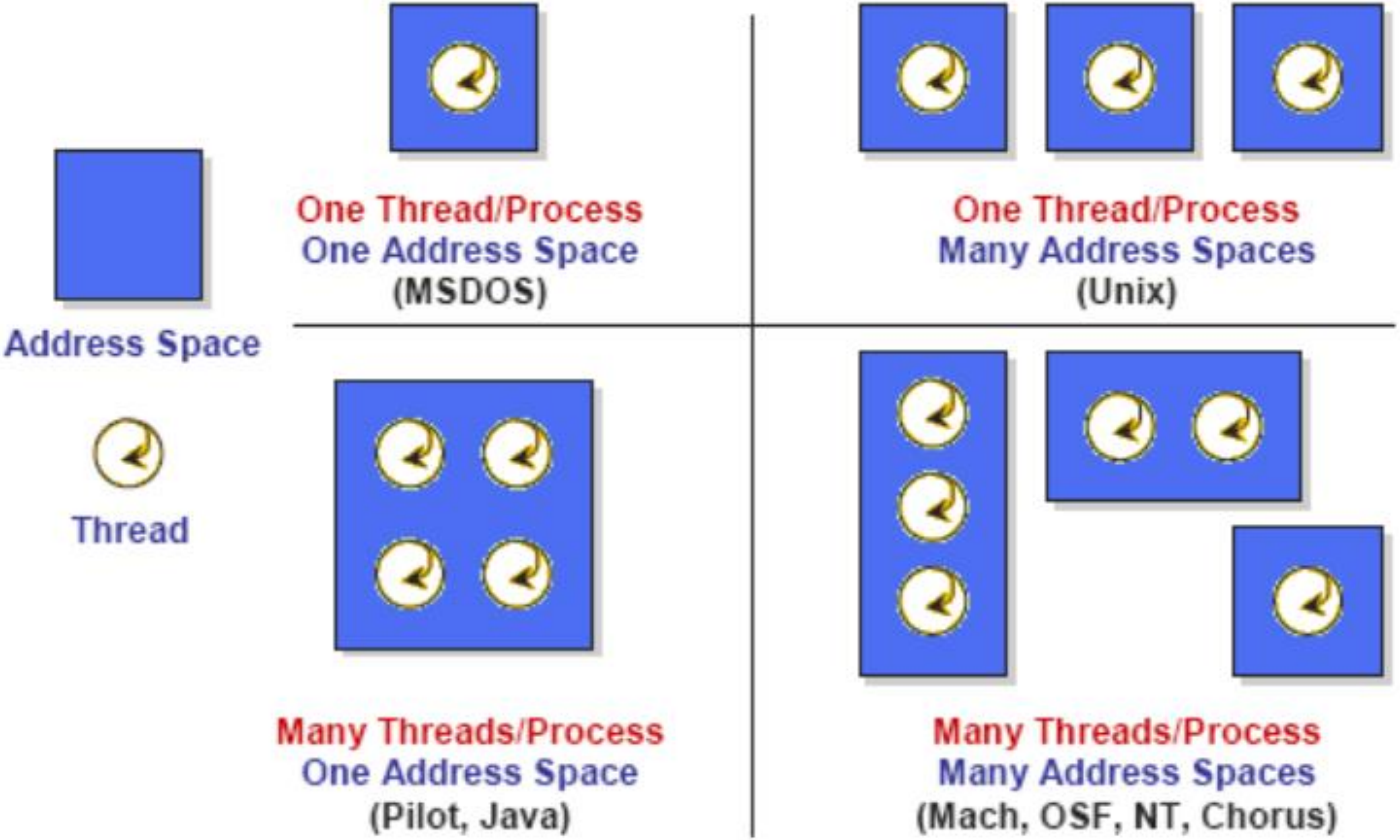


single-threaded process

multithreaded process

Benefits of Multithreaded programming:

1. **Responsiveness:** may allow continued execution if part of process is blocked, especially important for user interfaces
2. **Resource Sharing:** threads share resources of process, easier than shared memory or message passing
3. **Economy:** cheaper than process creation, thread switching lower overhead than context switching
4. **Scalability:** process can take advantage of multiprocessor architectures


Address Space
Thread
One Thread/Process
One Address Space
(MSDOS)
One Thread/Process
Many Address Spaces
(Unix)
Many Threads/Process
One Address Space
(Pilot, Java)
Many Threads/Process
Many Address Spaces
(Mach, OSF, NT, Chorus)

User Threads and Kernel Threads

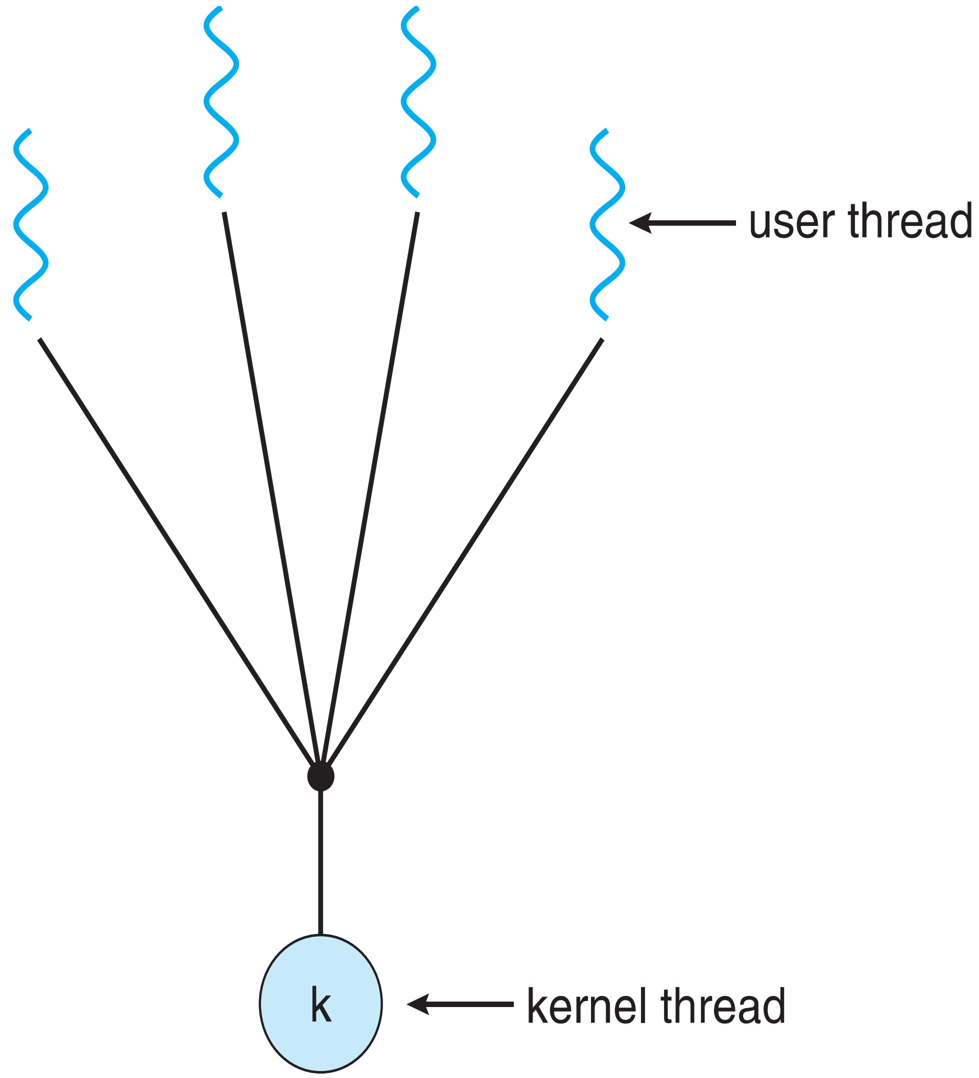
- **User threads:** Management done by user-level threads library
- Three primary thread libraries:
  - *POSIX Pthreads*
  - *Windows threads*
  - *Java threads*
- **Kernel threads:** Supported by the Kernel
- Examples – virtually all general-purpose operating systems, including:
  - *Windows*
  - *Solaris*
  - *Linux*
  - *Tru64 UNIX*
  - *Mac OS X*

Multithreading Models

- Many-to-One
- One-to-One
- Many-to-Many

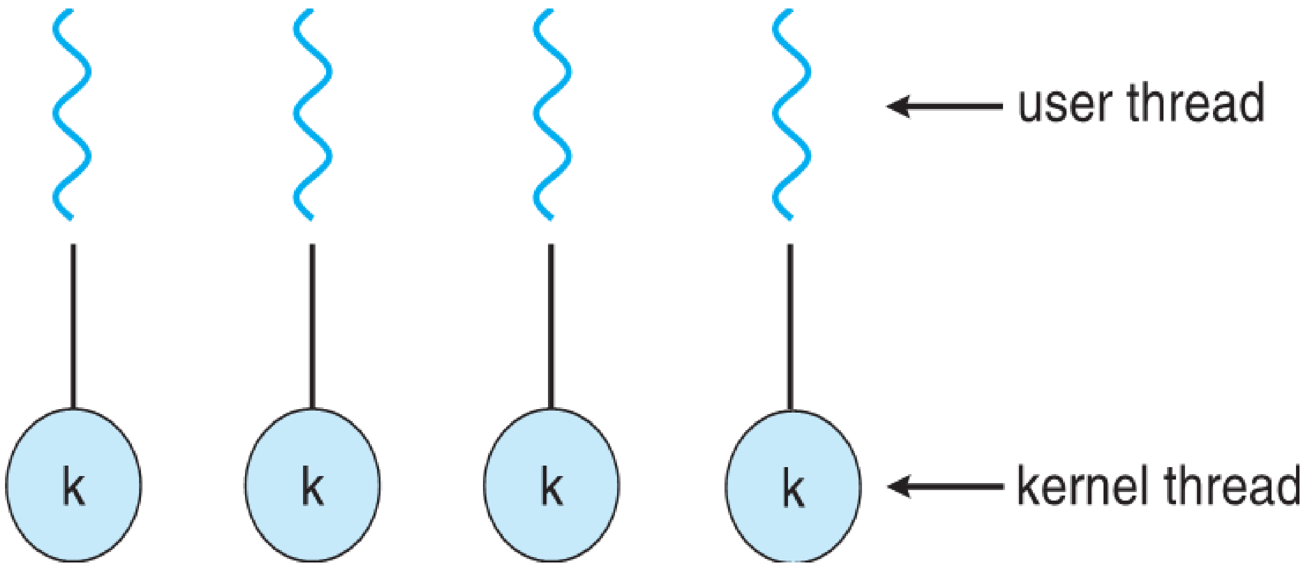
Multithreading Models: Many to one

- 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*



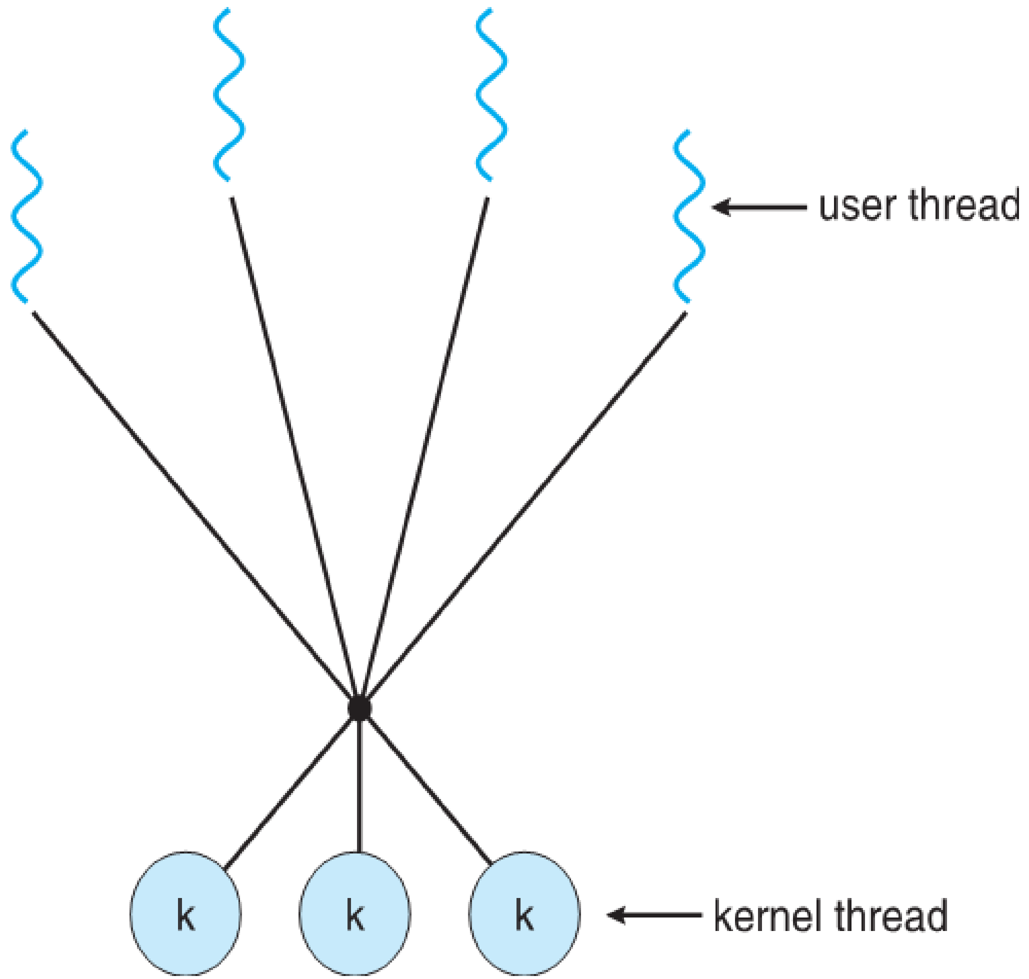
Multithreading Models: One to one

- 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



Multithreading Models: Many to Many

- 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 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

Thread Libraries

- **Thread library** provides programmer with API for creating and managing threads
- Two primary ways of implementing
  - Library entirely in user space
  - Kernel-level library supported by the OS

Pthreads

- May be provided either as user-level or kernel-level
- A POSIX standard (IEEE 1003.1c) API for thread creation and synchronization
- Specification, not implementation
- API specifies behavior of the thread library, implementation is up to development of the library
- Common in UNIX operating systems (Solaris, Linux, Mac OS X)

Pthreads Example

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

   if (argc != 2) {
      fprintf(stderr,"usage: a.out <integer value>\n");
      return -1;
   }
   if (atoi(argv[1]) < 0) {
      fprintf(stderr,"%d must be >= 0\n",atoi(argv[1]));
      return -1;
   }
```

Pthreads Example (Cont.)

```
    /* get the default attributes */
    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);
}

/* The thread will begin control 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);
}
```

Pthreads Code for Joining 10 Threads

```
#define NUM_THREADS 10

/* an array of threads to be joined upon */
pthread_t workers[NUM_THREADS];

for (int i = 0; i < NUM_THREADS; i++)
   pthread_join(workers[i], NULL);
```

Windows Multithreaded C Program

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

/* the thread runs in this separate function */
DWORD WINAPI Summation(LPVOID Param)
{
   DWORD Upper = *(DWORD*)Param;
   for (DWORD i = 0; i <= Upper; i++)
      Sum += i;
   return 0;
}

int main(int argc, char *argv[])
{
   DWORD ThreadId;
   HANDLE ThreadHandle;
   int Param;

   if (argc != 2) {
      fprintf(stderr,"An integer parameter is required\n");
      return -1;
   }
   Param = atoi(argv[1]);
   if (Param < 0) {
      fprintf(stderr,"An integer >= 0 is required\n");
      return -1;
   }
```

Windows Multithreaded C Program (Cont.)

```
/* create the thread */
ThreadHandle = CreateThread(
   NULL, /* default security attributes */
   0, /* default stack size */
   Summation, /* thread function */
   &Param, /* parameter to thread function */
   0, /* default creation flags */
   &ThreadId); /* returns the thread identifier */

if (ThreadHandle != NULL) {
    /* now wait for the thread to finish */
   WaitForSingleObject(ThreadHandle,INFINITE);

   /* close the thread handle */
   CloseHandle(ThreadHandle);

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

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:

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

  - Extending Thread class
  - Implementing the Runnable interface

Java Multithreaded Program

```
class Sum
{
   private int sum;

   public int getSum() {
     return sum;
   }

   public void setSum(int sum) {
     this.sum = sum;
   }
}

class Summation implements Runnable
{
   private int upper;
   private Sum sumValue;

   public Summation(int upper, Sum sumValue) {
     this.upper = upper;
     this.sumValue = sumValue;
   }

   public void run() {
     int sum = 0;
     for (int i = 0; i <= upper; i++)
        sum += i;
     sumValue.setSum(sum);
   }
}
```

Java Multithreaded Program (Cont.)

```
public class Driver
{
  public static void main(String[] args) {
    if (args.length > 0) {
     if (Integer.parseInt(args[0]) < 0)
      System.err.println(args[0] + " must be >= 0.");
     else {
      Sum sumObject = new Sum();
      int upper = Integer.parseInt(args[0]);
      Thread thrd = new Thread(new Summation(upper, sumObject));
      thrd.start();
      try {
         thrd.join();
         System.out.println
                 ("The sum of "+upper+" is "+sumObject.getSum());
     } catch (InterruptedException ie) { }
     }
    }
    else
     System.err.println("Usage: Summation <integer value>"); }
}
```

End of Chapter 4