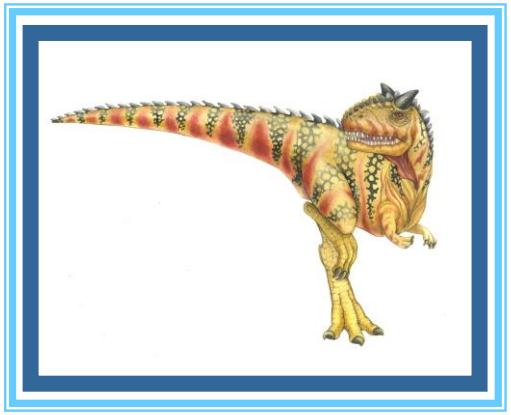
Chapter 4: Threads



Chapter 4: Threads

- Overview
- Multithreading Models
- Thread Libraries
- Threading Issues

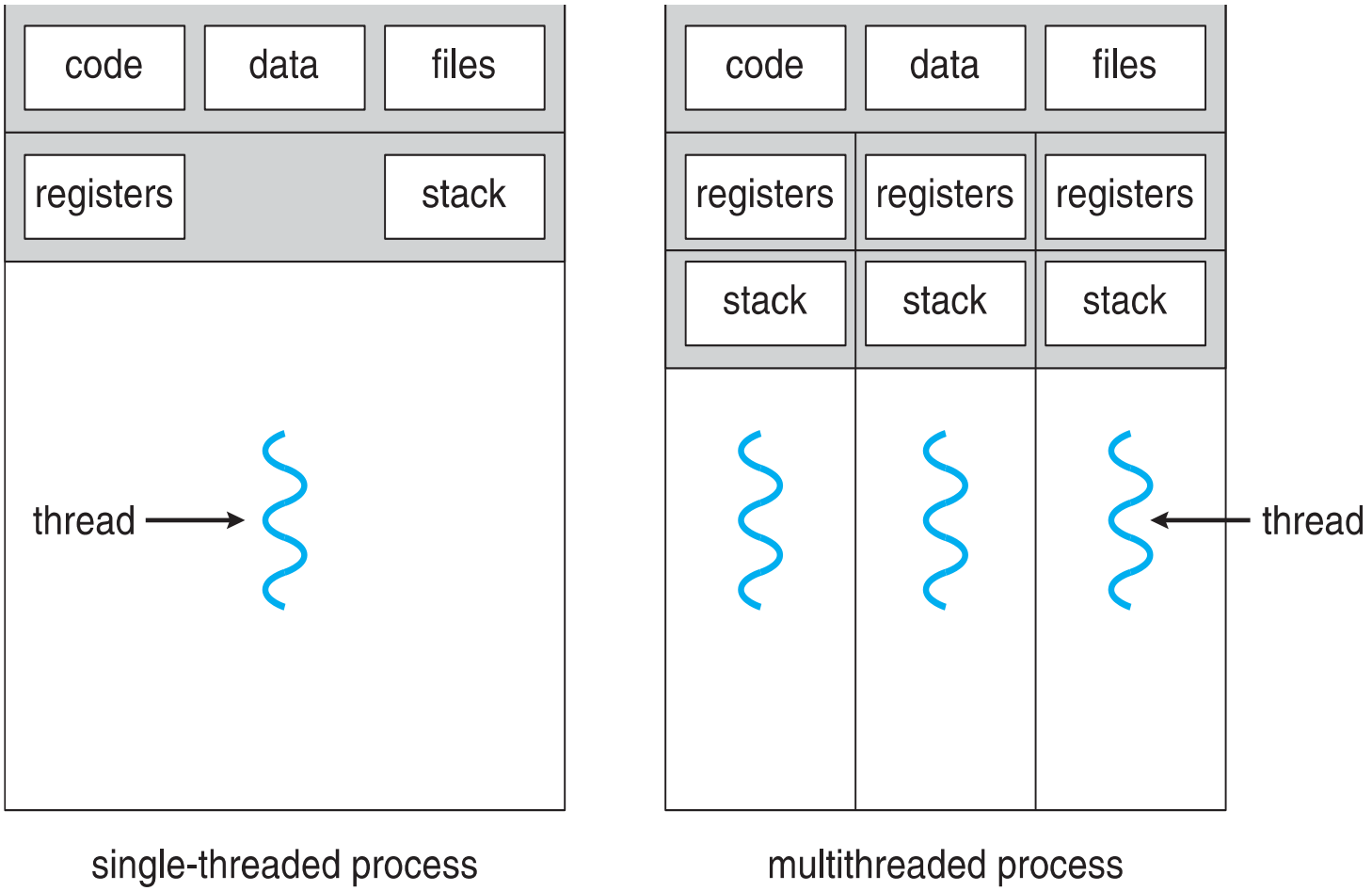
Objectives

- 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 examine issues related to multithreaded programming

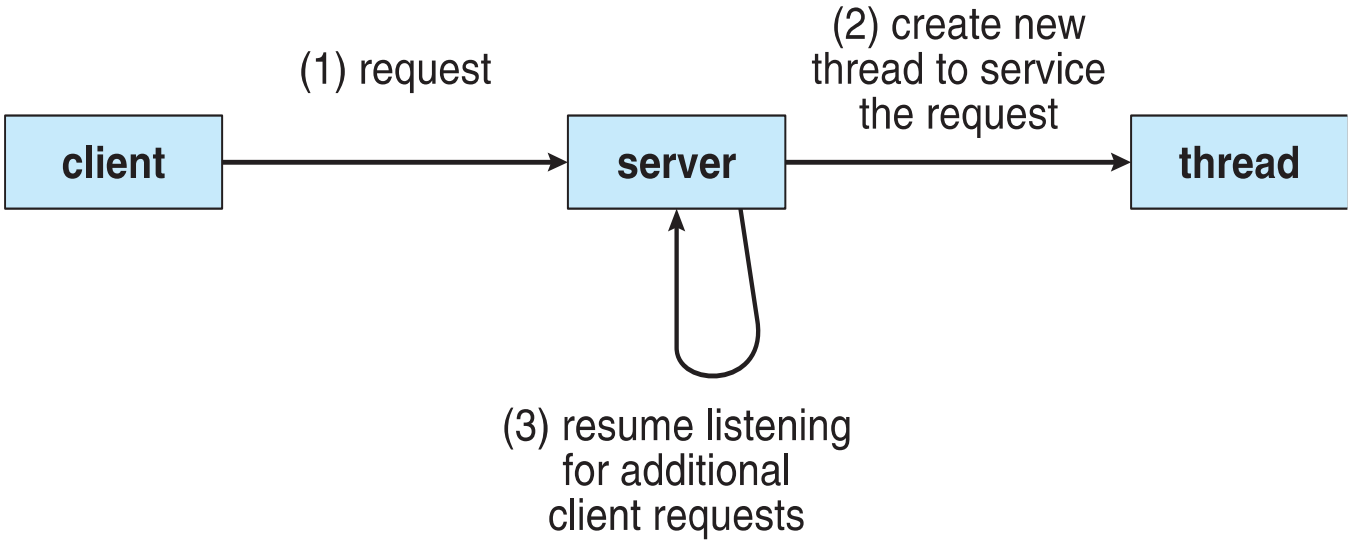
Motivation

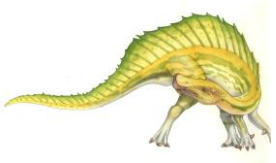
- Most modern applications are **multithreaded**
- Threads run within application
- Multiple tasks with the application can be implemented by separate threads
  - Update display
  - Fetch data
  - Spell checking
  - Answer a network request
- Process creation is **heavy-weight** while thread creation is **light-weight**
- Can simplify code, increase efficiency
- Kernels are generally multithreaded with each thread managing specific task (managing device, memory, interrupt etc)

Single and Multithreaded Processes



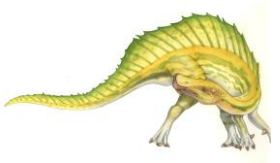
Multithreaded Server Architecture





Benefits: Multithreaded

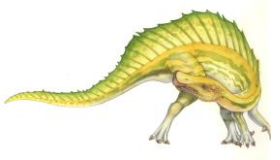
- **Responsiveness –** may allow continued execution if part of process is blocked, especially important for multi-threaded interactive applications. Ex: user interfaces
- **Resource Sharing –** threads share resources of the process it belong to by default, easier than shared memory or message passing.
- **Economy –** Allocating resources for process creation is costly. Thread creation and switching is more economical.
- **Scalability –** Multithreaded process can take advantage of multiprocessor architectures, where threads can run in parallel on different processing cores.



User Threads and Kernel Threads

Threads can be supported at two levels

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

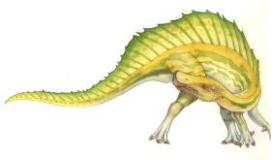


Multithreading Models

Relation(mapping) between user and kernel threads:

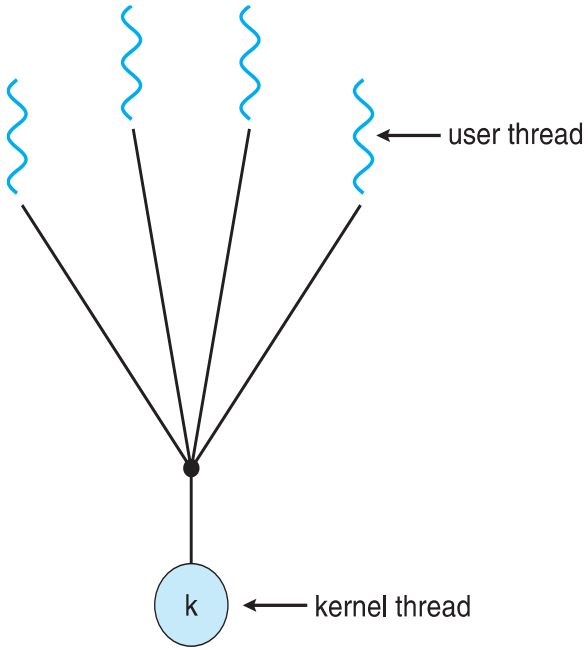
- Many-to-One
- One-to-One
- Many-to-Many
- Use of these models:
  - User level thread cannot achieve true parallelism on their own
  - Kernel level thread support parallelism and allows threads to run simultaneously on multiple CPU cores.

Many modern systems use threading models



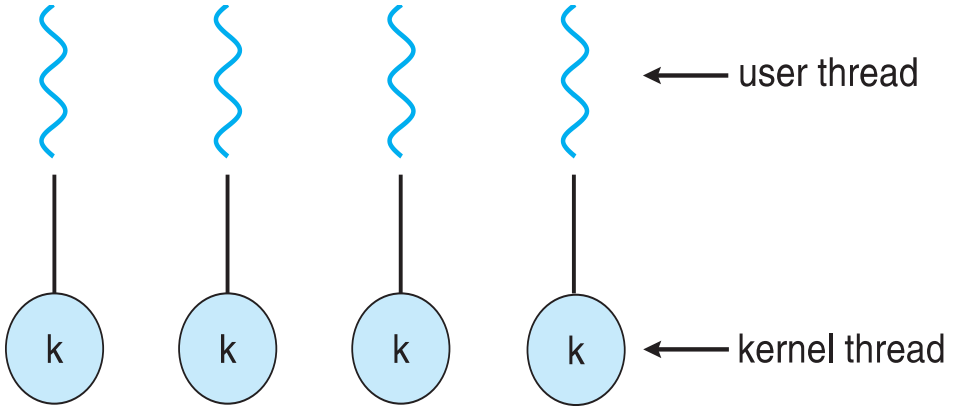
Many-to-One

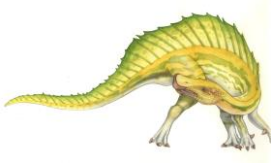
- Many user-level threads mapped to single kernel thread
- One thread blocking causes all to block
- Multiple threads may not run in parallel on muticore system because only one may be in kernel at a time
- Few systems currently use this model
- Examples:
  - **Solaris Green Threads**
  - **GNU Portable Threads**



One-to-One

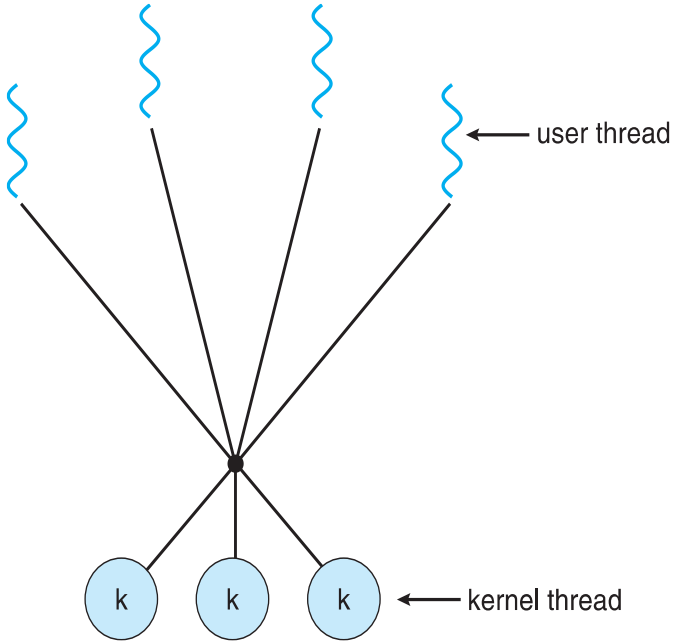
- Each user-level thread maps to kernel thread
- Creating a user-level thread creates a kernel thread
- More concurrency than many-to-one
- Number of threads per process sometimes restricted due to overhead
- Examples
  - Windows
  - Linux
  - Solaris 9 and later

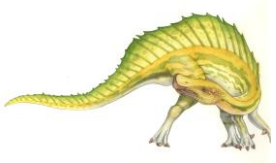




Many-to-Many Model

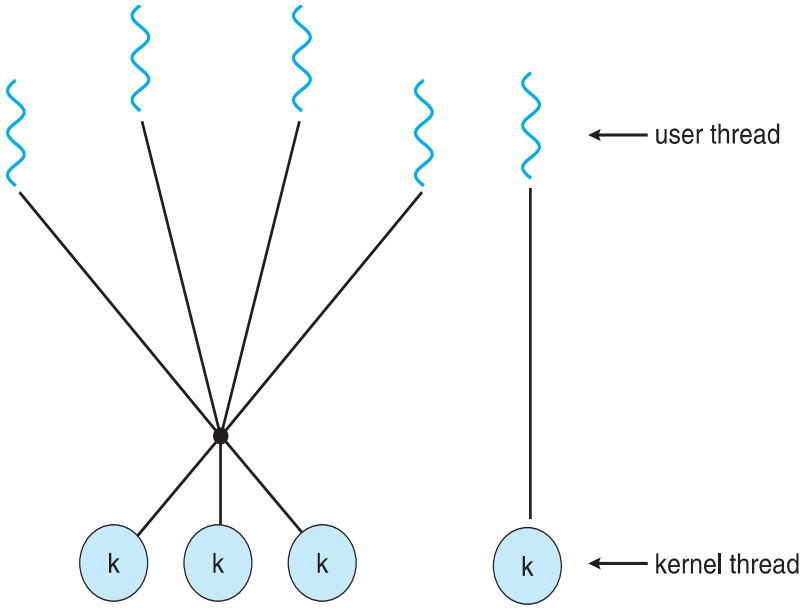
- Allows many user level threads to be mapped to many kernel threads
- Allows the operating system to create a sufficient number of kernel threads
- Solaris prior to version 9
- Windows with the *ThreadFiber* package

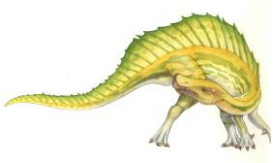




Two-level Model

- 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





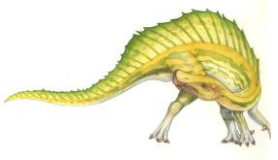
Thread Libraries

- **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

- Three main thread libraries are in use today: POSIX Pthreads, Windows, and Java.
- POSIX (Portable Operating System Interface) is **a set of standard operating system interfaces based on the Unix operating system**.
- The Windows thread library is a kernel-level library available on Windows systems.
- The Java thread API allows threads to be created and managed directly in Java programs.

Pthreads

- May be provided either as user-level or kernel-level
- Its a POSIX standard (IEEE 1003.1c) API for thread creation and synchronization
- The Portable Operating System Interface is a family of standards specified by the IEEE Computer Society for maintaining compatibility between operating systems.
- *This is a **Specification**, not **implementation.** OS can implement their way*
- 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)
- **Windows** does not support this



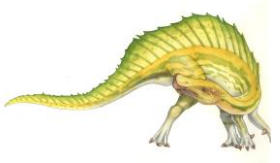
Pthreads Example

```
#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.)

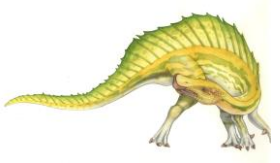
```
   /* get the default attributes */
   pthread_attr_init(&attr);
   /* create the thread */
   pthread_create(&tid,&attr,runner,argv[1]);
   /* wait for the thread to exit */
   pthread_join(tid,NULL);

   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

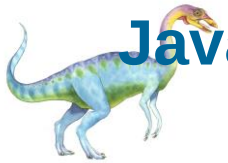
```
#define NUM_THREADS 10

/* 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);
```

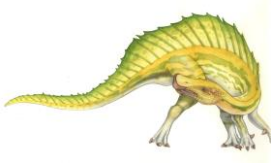
Java program for the summation of a non-negative integers

```
public class Summation implements Runnable {
    private int upper;
    private int sum;
    public Summation(int upper) {
        this.upper = upper;
    }
    public int getSum() {
        return sum;
    }
    public void run() {
        sum = 0;
        for (int i = 0; i <= upper; i++) {
            sum += i;
        }
    }
```



Java program for the summation of a non-negative integers

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



End of Chapter 4