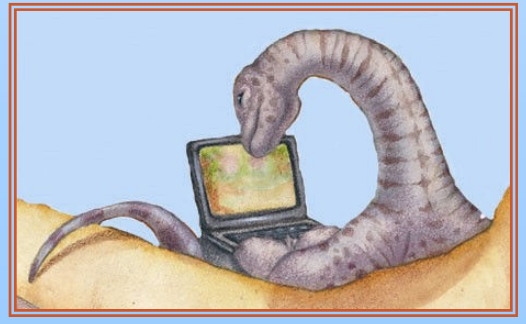
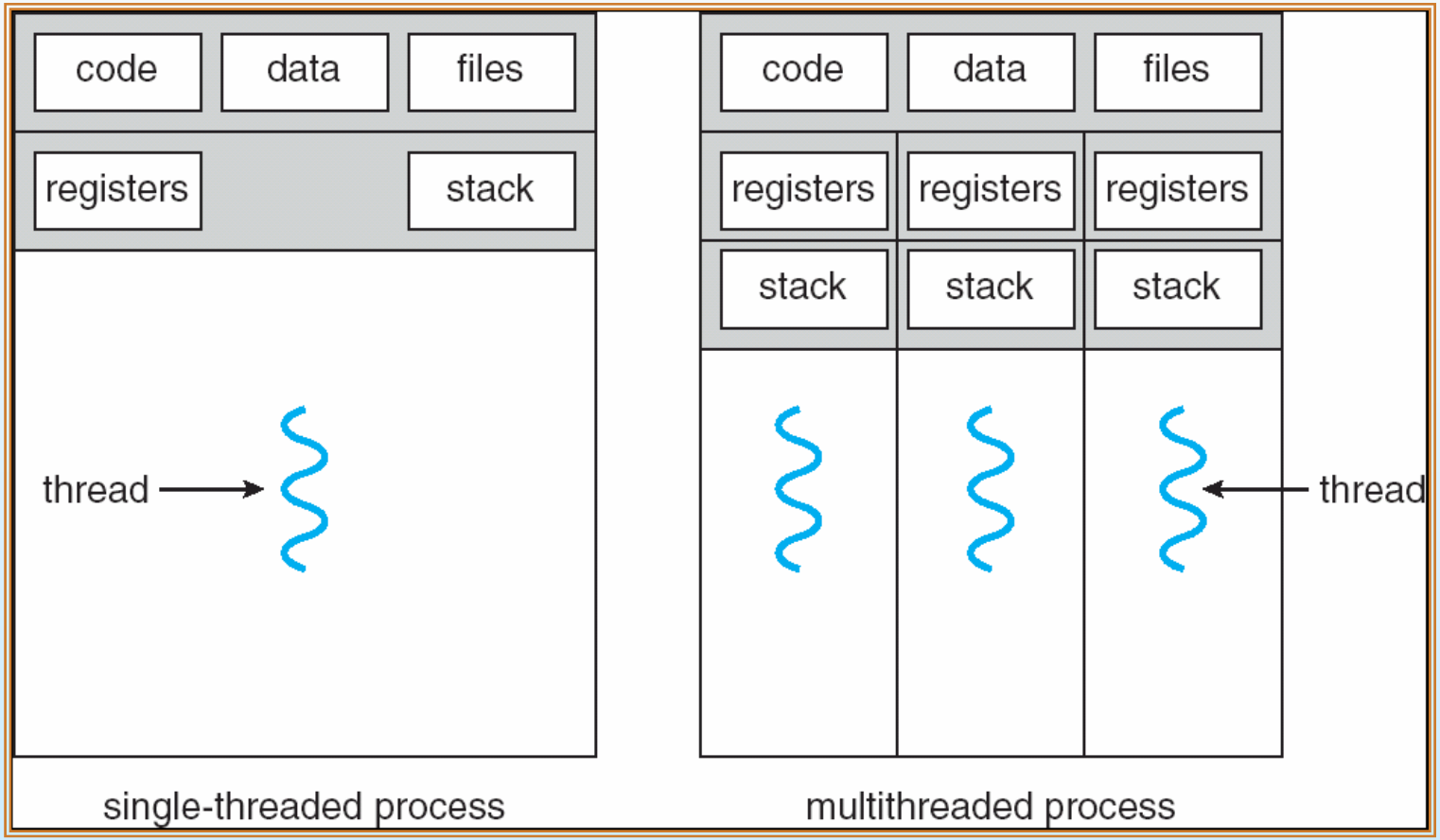
Chapter 4: Threads



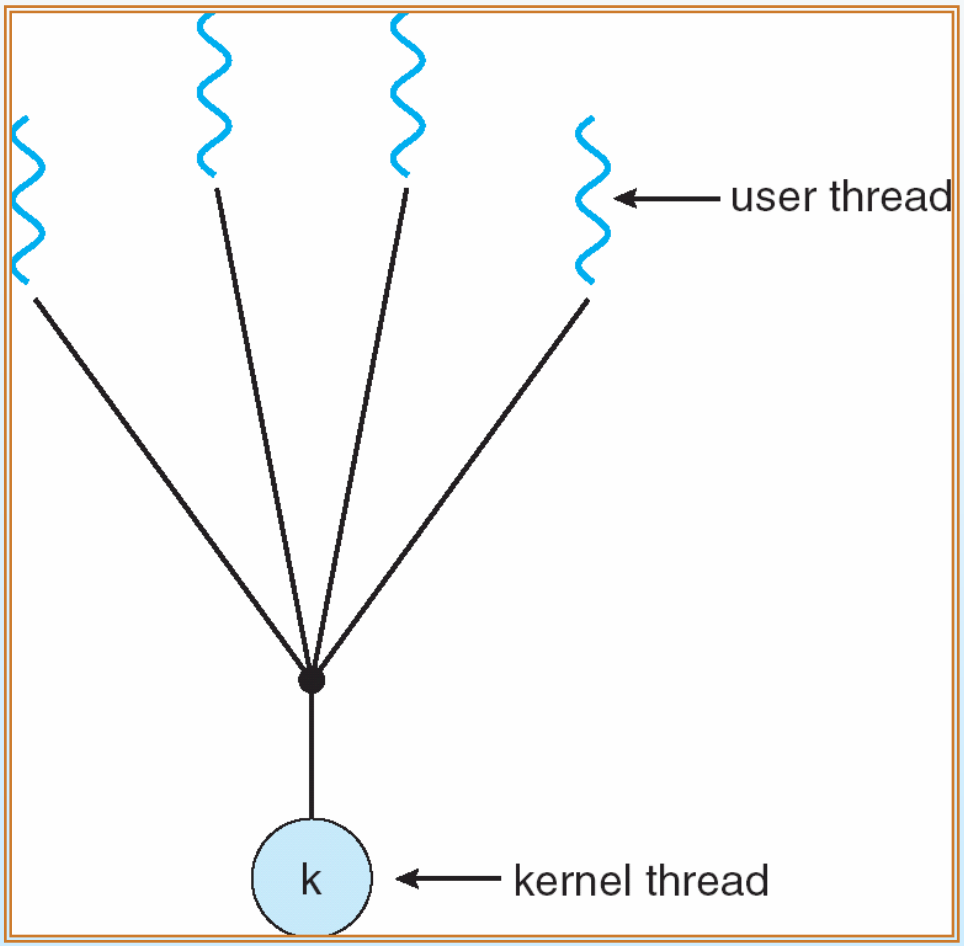
Single and Multithreaded Processes



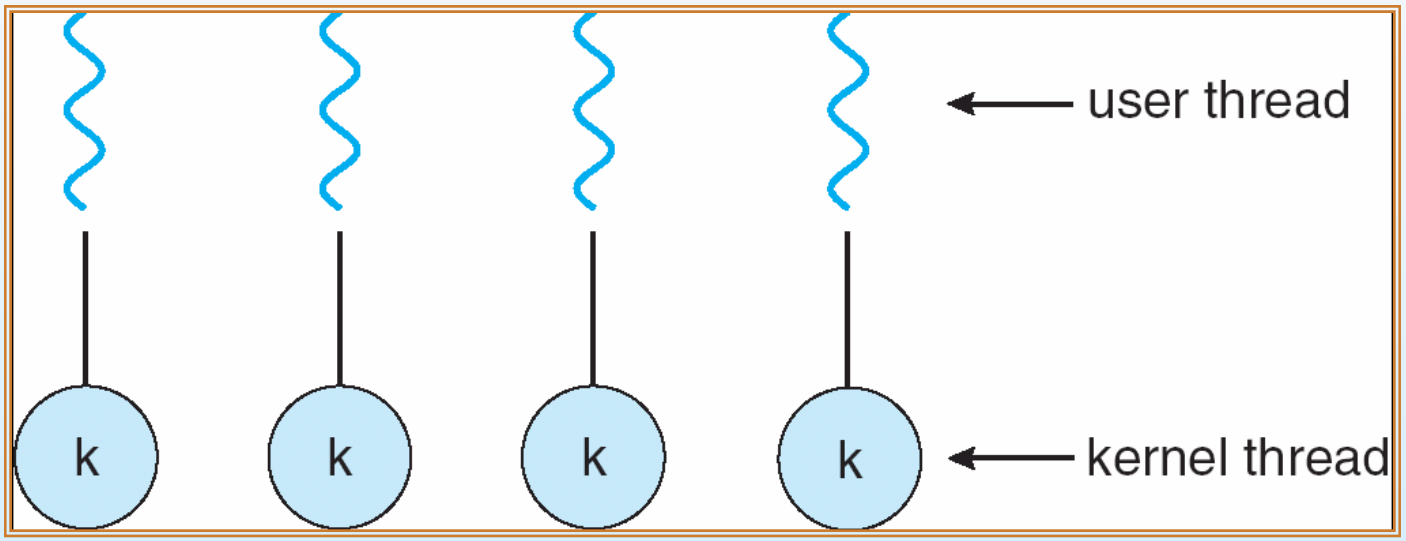
Benefits

- Responsiveness
- Resource Sharing
- Economy
- Utilization of MP Architectures

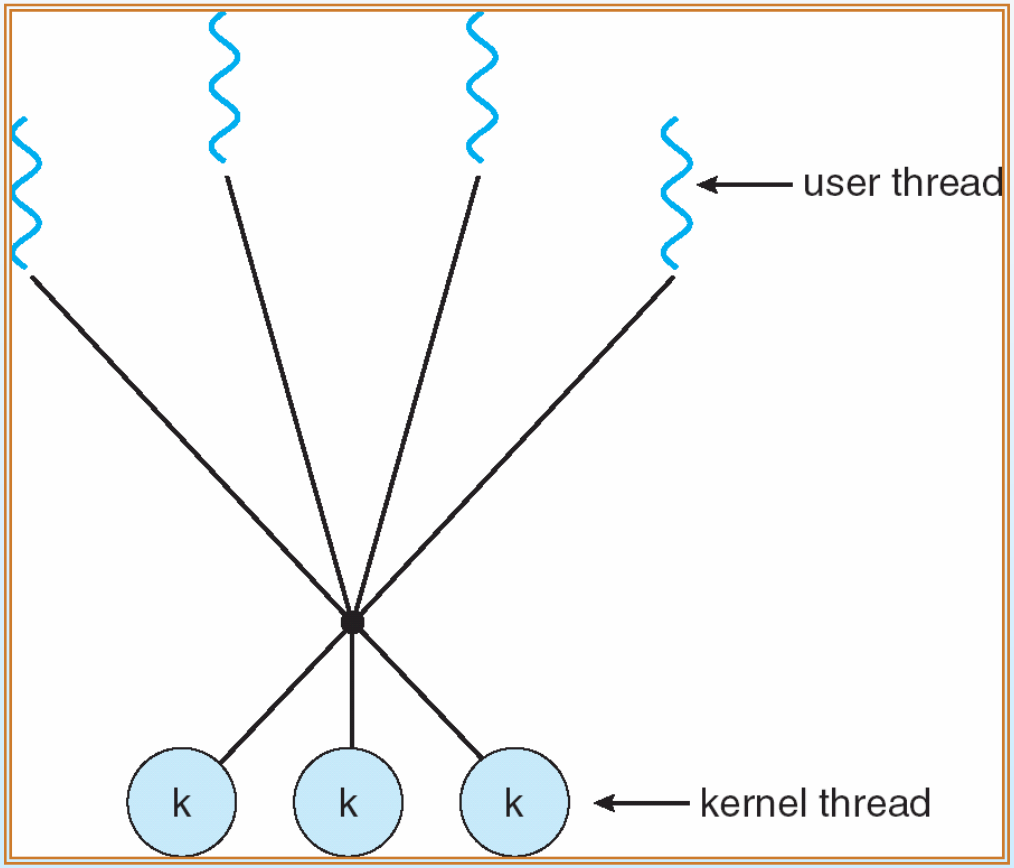
Many-to-One Model



One-to-one Model



Many-to-Many Model



Multithread C program with PThreads

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

int sum;                                        /* shared by threads */
void * runner( void *param);                    /* the thread */

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

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

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

void *runner( void *param )
{
   int i, upper = atoi( param );
   sum = 0;

   for ( i = 0; i < upper; i++ )
     sum += i;

   pthread_exit( 0 );
}
```

Threading Issues

- Semantics of **fork()** and **exec()** system calls
- Thread cancellation
- Signal handling
- Thread pools
- Thread specific data
- Scheduler activations

Semantics of fork() and exec()

- Does **fork()** duplicate only the calling thread or all threads?

Thread Cancellation

- Terminating a thread before it has finished
- Two general approaches:
  - **Asynchronous cancellation** terminates the target thread immediately
  - **Deferred cancellation** allows the target thread to periodically check if it should be cancelled

Signal Handling

- Signals are used in UNIX systems to notify a process that a particular event has occurred
- A **signal handler** is used to process signals
  1. Signal is generated by particular event
  2. Signal is delivered to a process
  3. Signal is handled
- Options:
  - Deliver the signal to the thread to which the signal applies
  - Deliver the signal to every thread in the process
  - Deliver the signal to certain threads in the process
  - Assign a specific threa to receive all signals for the process

Thread Pools

- Create a number of threads in a pool where they await work
- Advantages:
  - Usually slightly faster to service a request with an existing thread than create a new thread
  - Allows the number of threads in the application(s) to be bound to the size of the pool

Thread Specific Data

- Allows each thread to have its own copy of data
- Useful when you do not have control over the thread creation process (i.e., when using a thread pool)

Scheduler Activations

- Both M:M and Two-level models require communication to maintain the appropriate number of kernel threads allocated to the application
- Scheduler activations provide **upcalls** - a communication mechanism from the kernel to the thread library
- This communication allows an application to maintain the correct number kernel threads

Pthreads

- A POSIX standard (IEEE 1003.1c) API for thread creation and synchronization
- 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 XP Threads

- Implements the one-to-one mapping
- Each thread contains
  - A thread id
  - Register set
  - Separate user and kernel stacks
  - Private data storage area
- The register set, stacks, and private storage area are known as the **context** of the threads
- The primary data structures of a thread include:
  - ETHREAD (executive thread block)
  - KTHREAD (kernel thread block)
  - TEB (thread environment block)

Linux Threads

- Linux refers to them as *tasks* rather than *threads*
- Thread creation is done through **clone()** system call
- **clone()** allows a child task to share the address space of the parent task (process)

Java Threads

- Java threads are managed by the JVM
- Java threads may be created by:
  - Extending Thread class
  - Implementing the Runnable interface