Threads

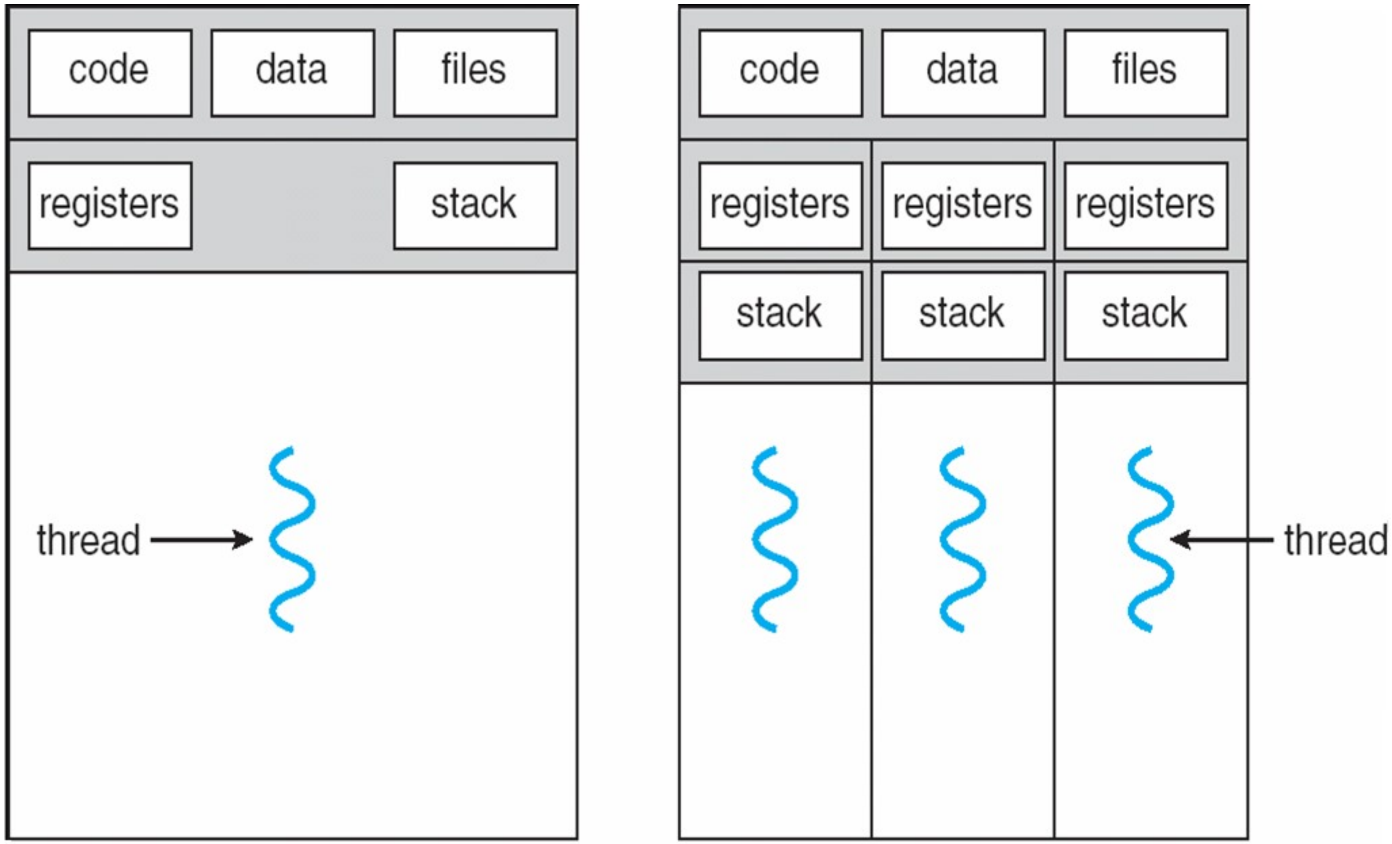
Fan Wu
Department of Computer Science and Engineering
Shanghai Jiao Tong University

Spring 2020

What is a thread?

- A thread is a basic unit of CPU utilization
  - contains a thread ID, a program counter, a register set, and a stack
  - shares with other threads belonging to the same process
    - code section
    - data section
    - other operating-system resources, such as open files

Single and Multithreaded Processes



single-threaded process

multithreaded process

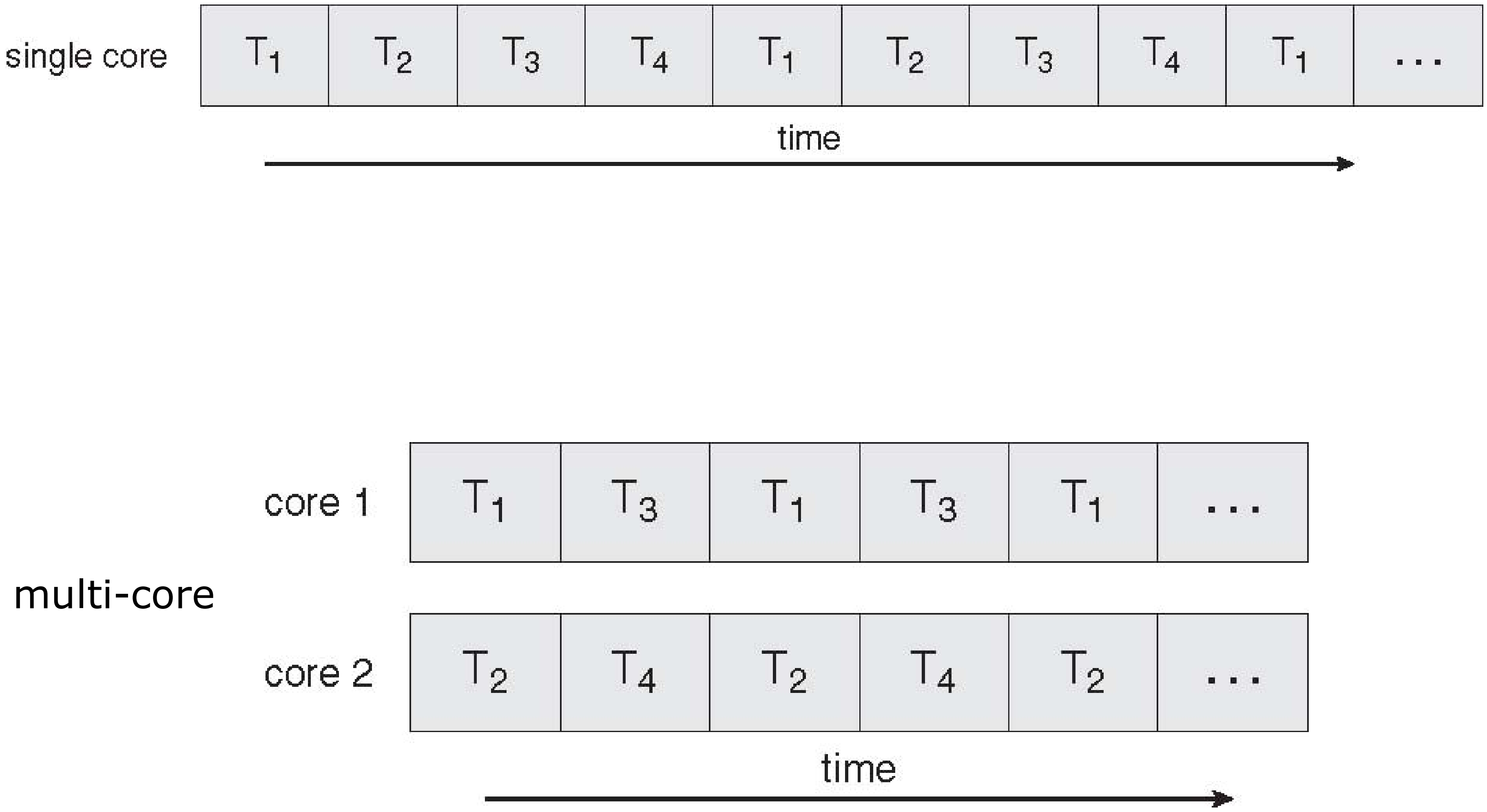
Motivation

- Threads run within application
- Multiple tasks with the application can be implemented by separating threads
  - Update display
  - Fetch data
  - Spell checking
  - Answer a network request
- Process creation is heavy-weight while thread creation is light-weight
- Increase efficiency of C-S applications
- Kernels are generally multithreaded

Benefits

- **Responsiveness**
  - A program continues running even if part of it is blocked or is performing a lengthy operation
- **Resource Sharing**
  - Threads share the memory and the resources of the process to which they belong
  - IPC techniques are not needed
- **Economy**
  - Creating a thread is much faster than creating a process
- **Scalability**
  - Multithreading on a multi-CPU machine increases concurrency

Parallel Execution on a Multi-core System



Drawbacks

- Make the programming more complicated
- Make the debugging harder
- Possible error when threads concurrently access the shared resources
- Poorly divided jobs can cause even worse system performance
- ......

Process vs. Thread

Process

1. independent
2. carries considerably more state information
3. has separate address space
4. interact only through IPC
5. context switching is relatively slow

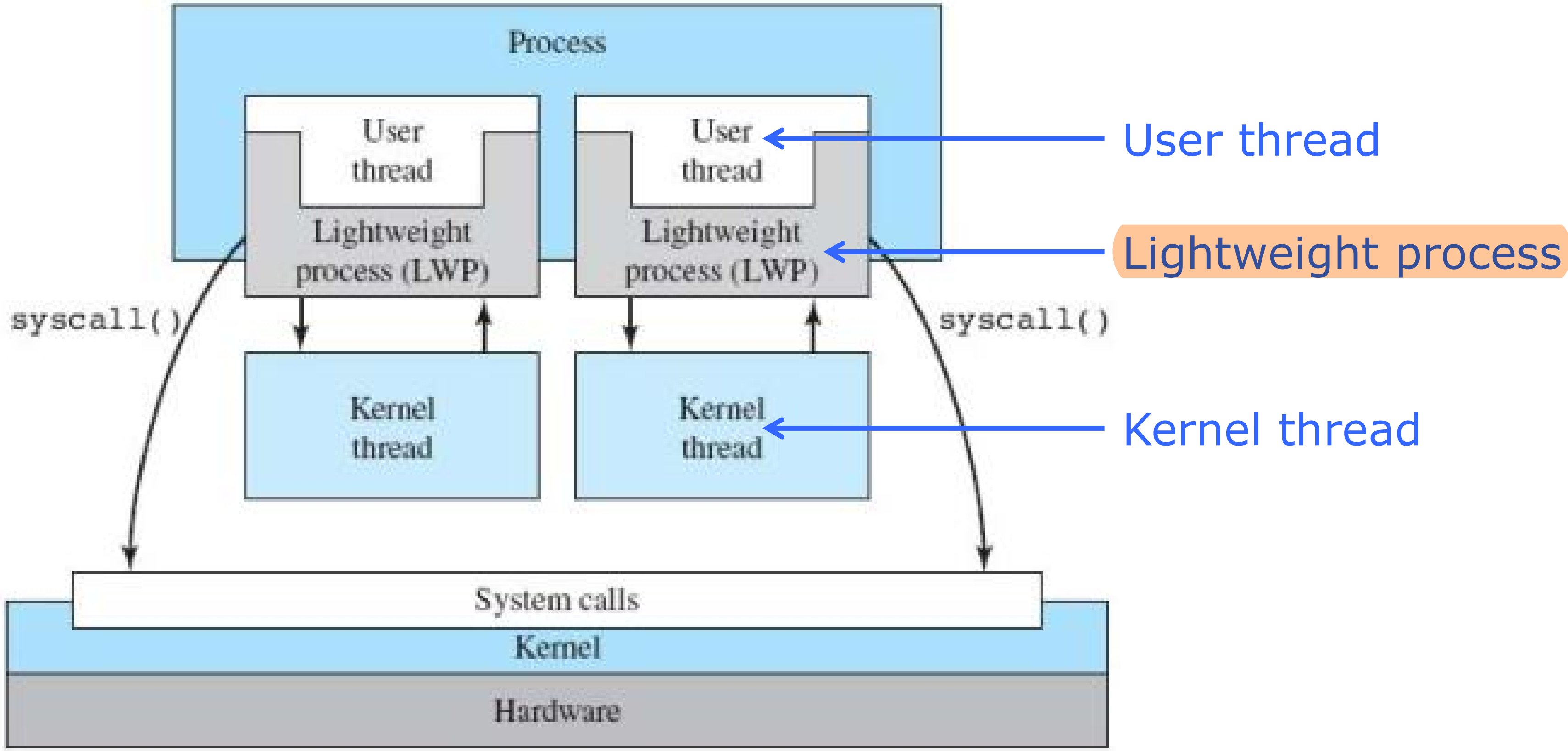
Thread

1. exists as subsets of a process
2. shares process state as well as memory and other resources
3. shares process's address space
4. more ways to communicate
5. context switching in the same process is typically faster

Supports for Threads

- Kernel Threads
  - Supported by the operating system kernel
  - Examples
    - Windows XP/2000, Solaris, Linux, Tru64 UNIX, Mac OS X
- User Threads
  - Thread management done by user-level threads library
  - Three primary thread libraries:
    - POSIX **Pthreads**
    - Win32 threads
    - Java threads

Thread Model

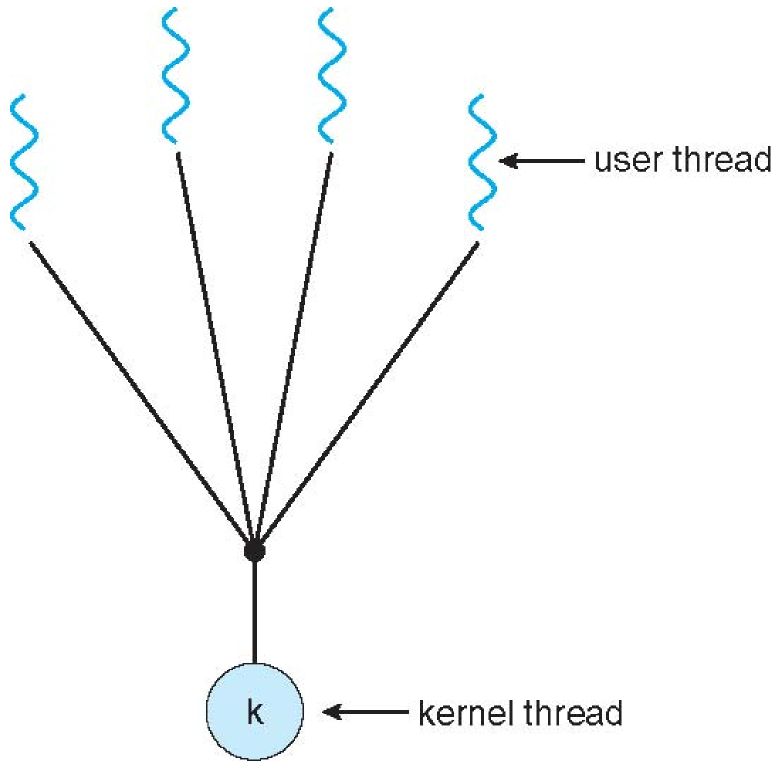


Multithreading Models

- Four common connections between user threads and kernel threads
  - Many-to-One
  - One-to-One
  - Many-to-Many
  - Two-Level Model

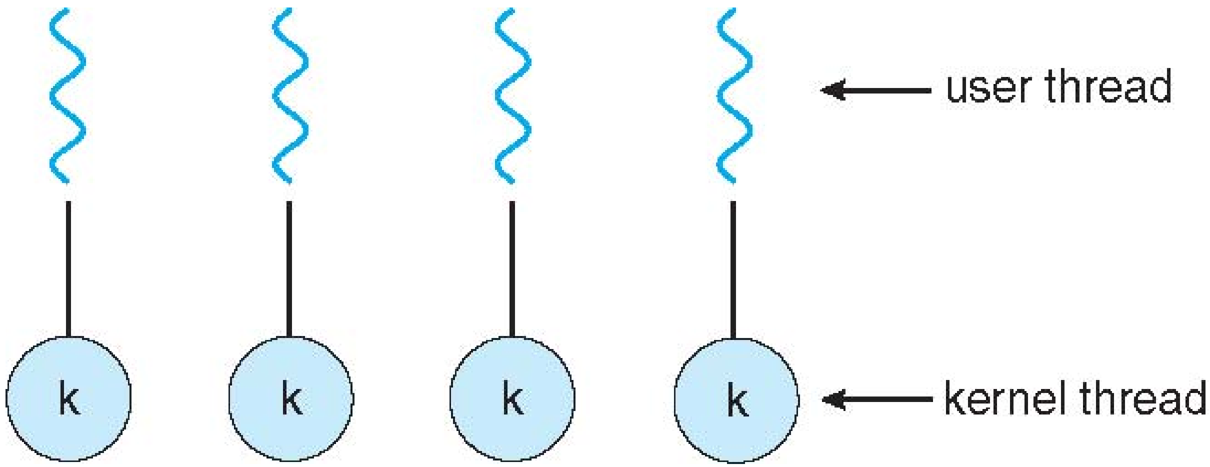
Many-to-One Model

- Many user-level threads are mapped to a single kernel thread
- Strength
  - Multiple threads are hidden by user-level thread library
- Weaknesses
  - The entire process will block if a thread makes a blocking system call
  - Multiple threads are unable to run in parallel on multiprocessors
- Examples:
  - **Solaris Green Threads**
  - **GNU Portable Threads**



One-to-One

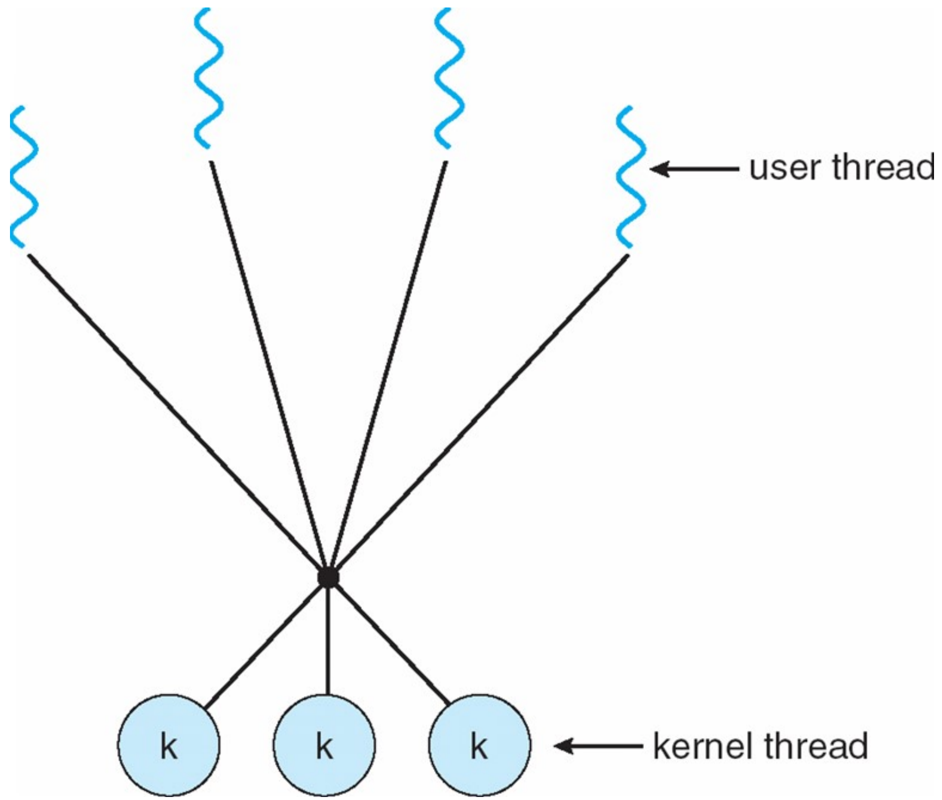
- Each user-level thread is mapped to a kernel thread
- Strength
  - More concurrency
- Weakness
  - Creating a user thread requires creating the corresponding kernel thread, which incurs overhead
- Examples
  - Windows NT/XP/2000
  - Linux
  - Solaris 9 and later



Many-to-Many Model

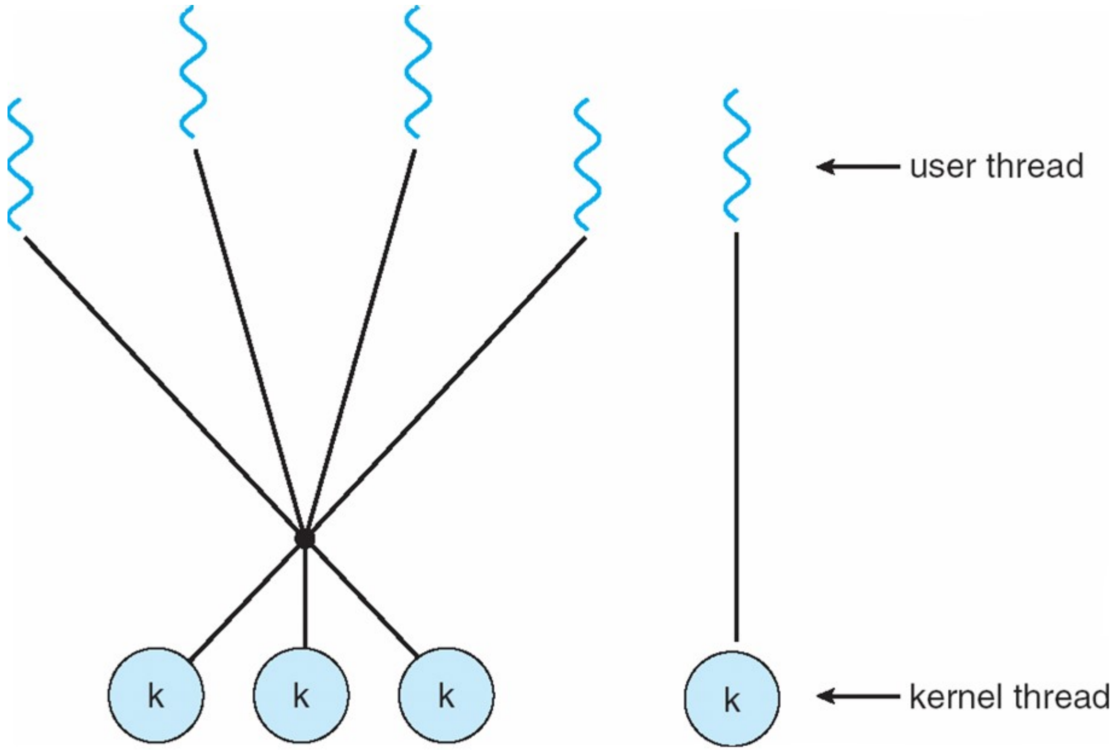
- Allows many user level threads to be mapped to many kernel threads
  - The operating system creates a sufficient number of kernel threads
- Examples
  - Windows NT/2000 with

    the *ThreadFiber* package



Two-Level Model

- Similar to Many-to-Many, except that it allows a user thread to be **bound** to a 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 implementation
  - User-level threads library
    - All codes and data structures for the library exist in user space
    - Invoking a function in the library results in a local function call in user space
  - Kernel-level threads library supported by the OS
    - Code and data structures for the library exist in kernel space
    - Invoking a function in the library results in a system call to the kernel
- Three primary thread libraries:
  - POSIX **Pthreads**, Win32 threads, Java threads

Pthreads

- Is provided either in user-level or kernel-level
- 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)

Example Using Pthreads

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

int sum;                /* this data is shared by the thread(s) */

/* 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) ;
}
```

Example Using Pthreads (Cont.)

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

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

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

Threading Issues

- **Semantics** of **fork()** and **exec()** system calls
  - Does **fork()** duplicate only the calling thread or all threads?
  - **exec()** will replace the entire process with the program specified in the parameter
- **Thread cancellation** of **target thread**
  - 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.

Threading Issues (Cont.)

- **Signal** handling
  - Signals are used in UNIX systems to notify a process that a particular event has occurred.
  - **Synchronous** and **asynchronous**
  - 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
  - **Delivery** 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 thread to receive all signals for the process

Threading Issues (Cont.)

- **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**
  - Create Facility needed for data private to thread
  - 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

Operating System Examples

- Linux Thread
- Windows XP Threads

Linux Threads

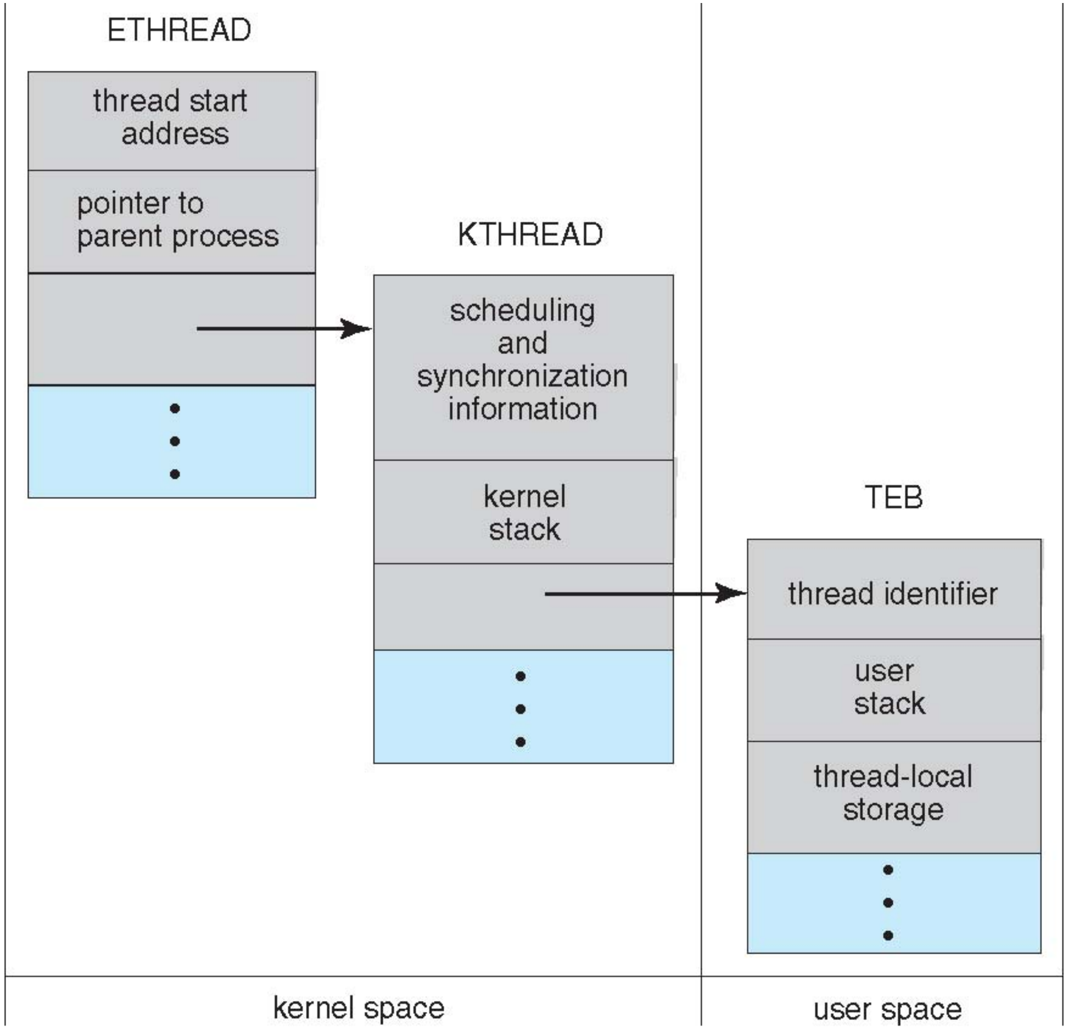
- fork() and clone() system calls
- clone() takes options to determine sharing on process create
- struct task_struct points to process data structures (shared or unique)

flag	meaning
CLONE_FS	File-system information is shared.
CLONE_VM	The same memory space is shared.
CLONE_SIGHAND	Signal handlers are shared.
CLONE_FILES	The set of open files is shared.

Windows XP Threads

- Implements the one-to-one mapping, kernel-level
- 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)

Windows XP Threads Data Structures



Pop-quiz

```
int value = 0;
void *runner(void *param) {
    value = 5;
    pthread_exit ( 0) ;
}

int main(int argc, char *argv[])
{
    int pid;
    pthread_t tid;
    pthread_attr_t attr;
    pid = fork();
```

```
    if (pid == 0) {
        pthread_attr_init (&attr) ;
        pthread_create(&tid, &attr, runner, NULL);
        pthread_join(tid,NULL) ;
        printf("Child: value = %d", value);
    }
    else if (pid > 0) {
        wait (NULL) ;
        printf("Parent: value = %d", value);
    }
}
```

child: 5 , parent: 0

What are the outputs from the above program?

Homework

- Reading:
  - Chapter 4