# Lecture 4 /
# Chapter 4 (Threads)
# Multicore Programming
# Multithreading Models
# Thread Libraries
# Implicit Threading
# Threading Issues

# OS Process Review





Process Concept

Process Scheduling

Operations on Processes

Interprocess Communication

# Threads



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

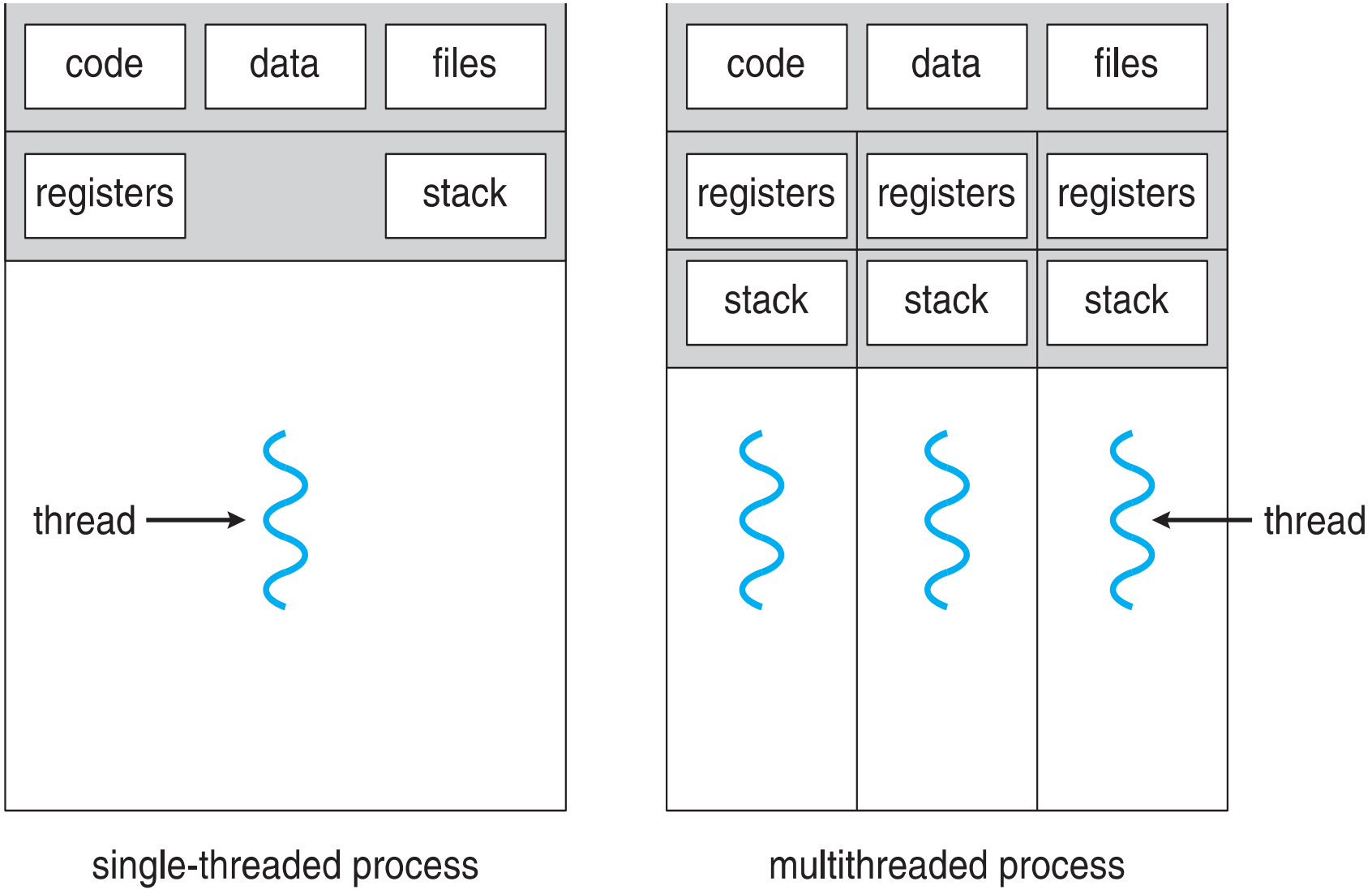
It **shares** with other threads belonging to the same process its code section, data section, and other operating-system resources, such as open files and signals.

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.

# Single and Multithreaded Processes





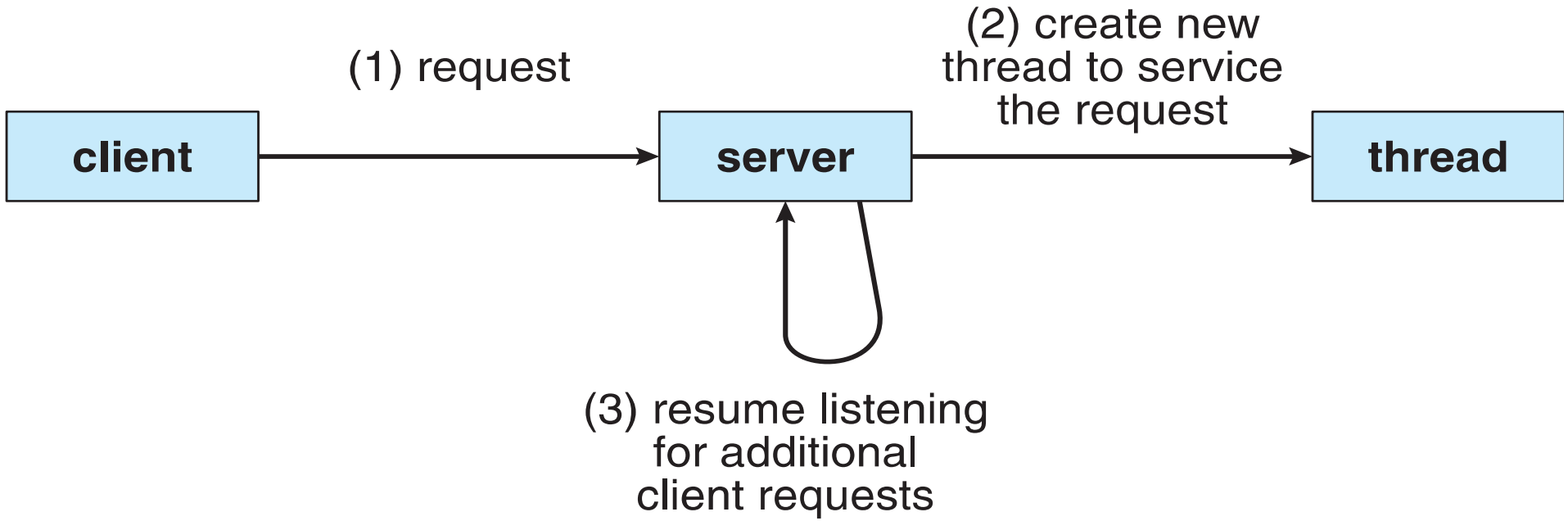
# Threads Motivation



- 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

# Multithreaded Server Architecture






If the web-server process is multithreaded, the 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.

# Threads Benefits



- **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 multiprocessor architectures

# Multicore Programming





**Multicore** or **multiprocessor** systems putting pressure on programmers, challenges include:

- **Dividing activities**
- **Balance**
- **Data splitting**
- **Data dependency**
- **Testing and debugging**

***Parallelism*** implies a system can perform more than one task simultaneously

***Concurrency*** supports more than one task making progress

- Single processor / core, scheduler providing concurrency

# Multicore Programming





- Types of parallelism
  - **Data parallelism** – distributes subsets of the same data across multiple cores, same operation on each
  - **Task parallelism** – distributing threads across cores, each thread performing unique operation

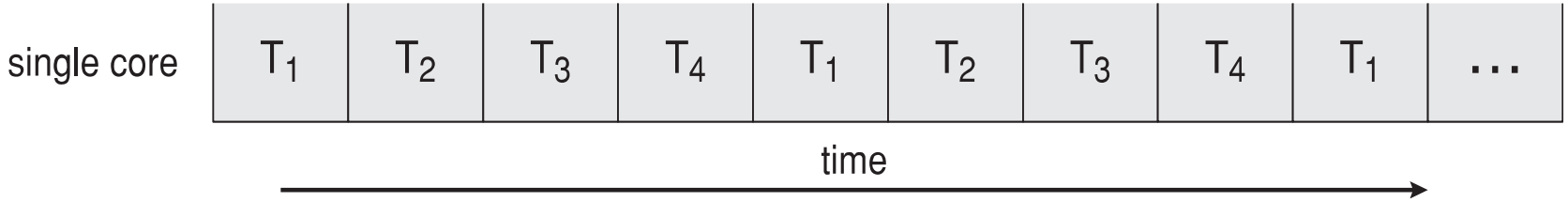
- As # of threads grows, so does architectural support for threading
  - CPUs have cores as well as ***hardware threads***
  - Consider Oracle SPARC T4 with 8 cores, and 8 hardware threads per core

# Concurrency vs. Parallelism

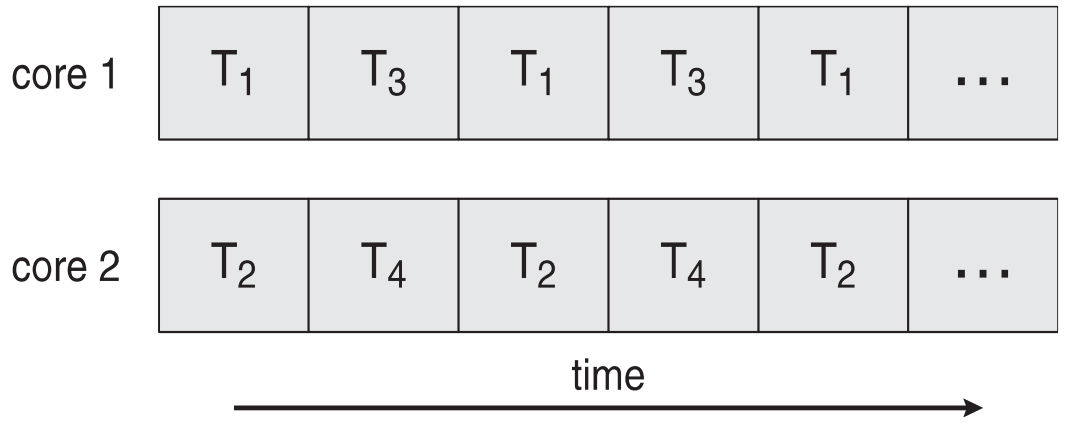




- **Concurrent execution on single-core system:**




- **Parallelism on a multi-core system:**



# 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



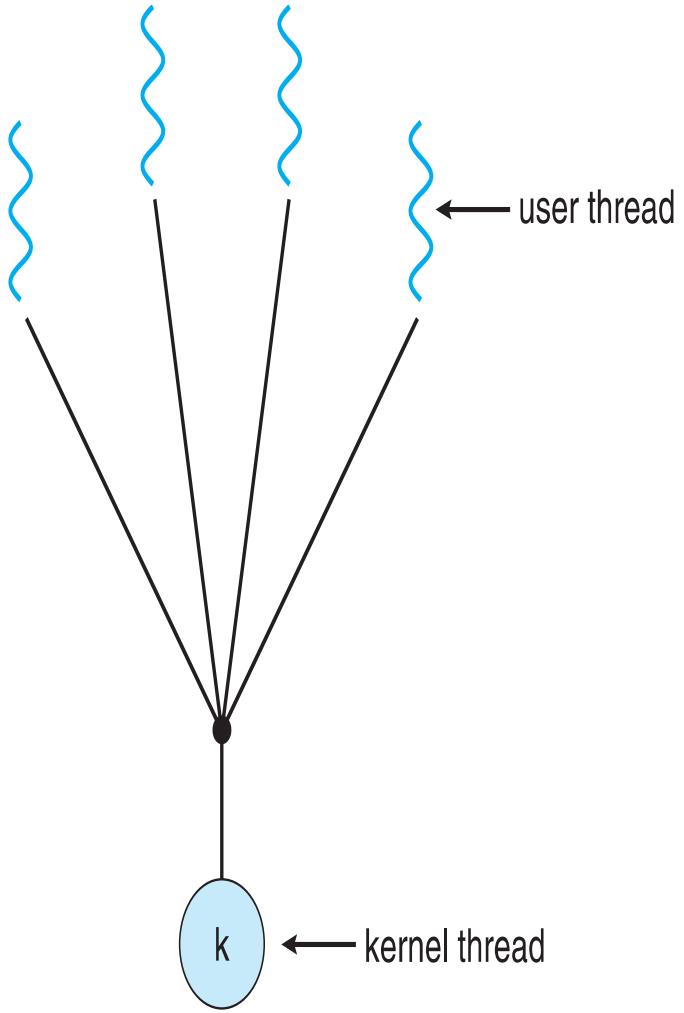


- Many-to-One
- One-to-One
- Many-to-Many

# Many-to-One



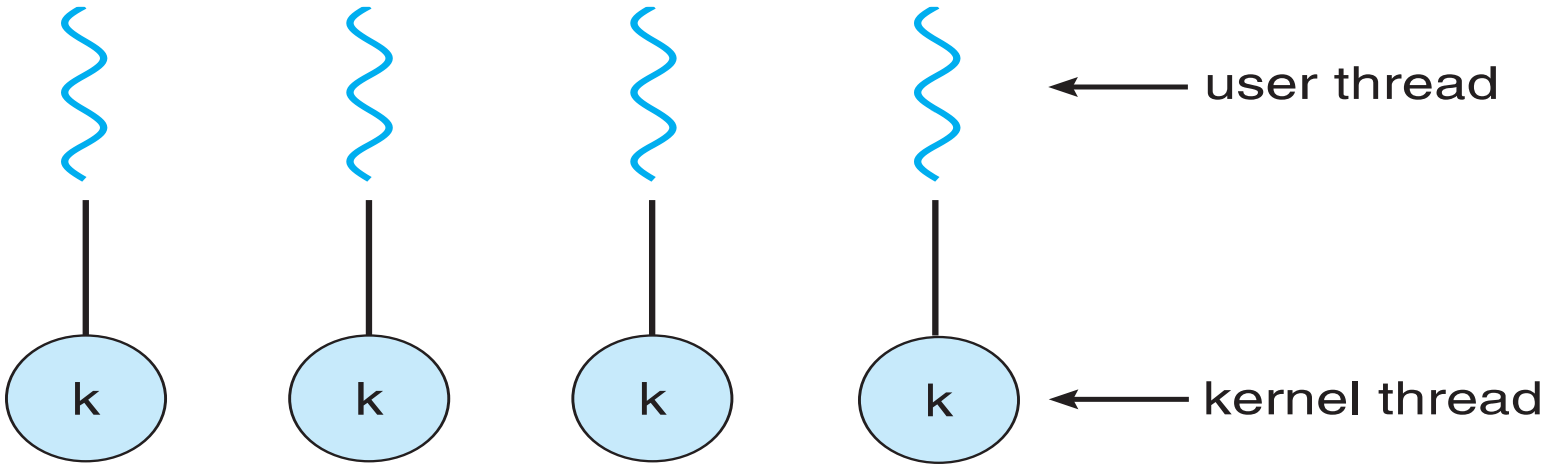
- Many user-level threads mapped to single kernel thread
- One thread blocking causes all to block
- Multiple threads may not run in parallel on muticore system because only one may be in kernel at a time
- Few systems currently use this model
- Examples:
  - **Solaris Green Threads**
  - **GNU Portable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



# One-to-One



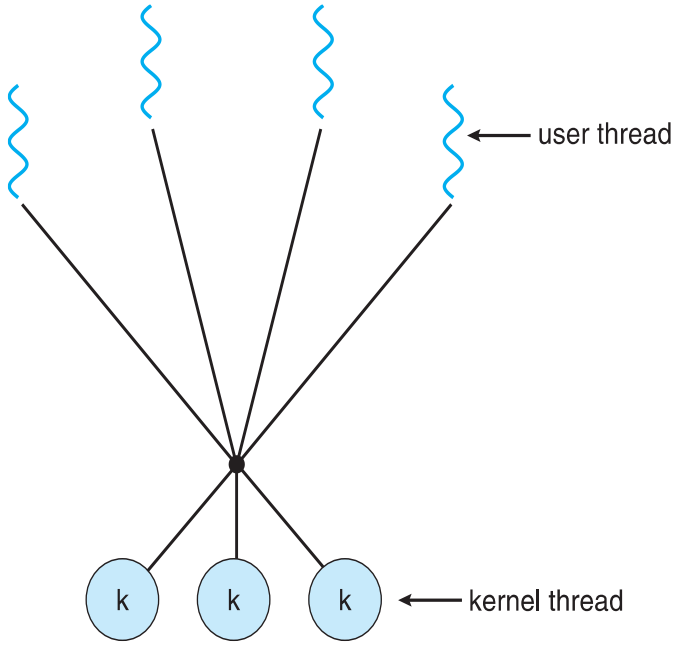
- Each user-level thread maps to kernel thread
- Creating a user-level thread creates a kernel thread
- More concurrency than many-to-one
- Number of threads per process sometimes restricted due to overhead
- Examples
  - Windows
  - Linux
  - Solaris 9 and later



# Many-to-Many Model



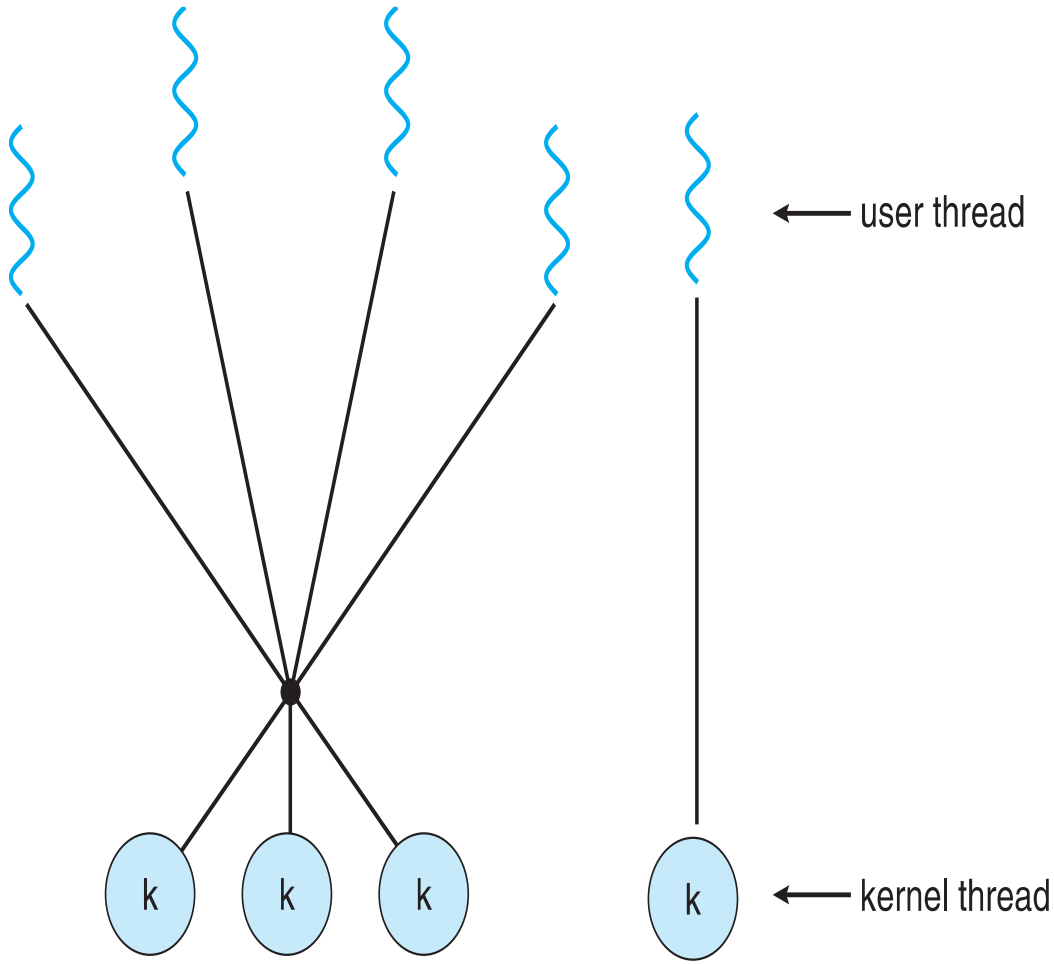
- Allows many user level threads to be mapped to many kernel threads
- Allows the operating system to create a sufficient number of kernel threads
- Solaris prior to version 9
- Windows with the *ThreadFiber* package



# 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

# Multiple Threads



- General strategies for creating Multiple Threads:
  - Asynchonous
  - Synchronous

- Synchronous Threading Libraries
  - Pthreads
  - Windows Threads
  - Java Threads

# Implicit Threading



- Growing in popularity as numbers of threads increase, program correctness more difficult with explicit threads
- Creation and management of threads done by compilers and run-time libraries rather than programmers
- Three methods explored
  - Thread Pools
  - OpenMP
  - Grand Central Dispatch
- Other methods include Microsoft Threading Building Blocks (TBB), **java.util.concurrent** package

# 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
  - Separating task to be performed from mechanics of creating task allows different strategies for running task
    - i.e.Tasks could be scheduled to run periodically
- Windows API supports thread pools:

# Threading Issues



- Semantics of **fork()** and **exec()** system calls
- Signal handling
  - Synchronous and asynchronous
- Thread cancellation of target thread
  - Asynchronous or deferred
- Thread-local storage
- Scheduler Activations

# OS Kernel Assignment





**Submission dead line 2018 23:55**

**Next Thursday No exceptions would be accepted**



# Chapter 6
# CPU scheduling