

CHAPTER 4: MULTITHREADED PROGRAMMING

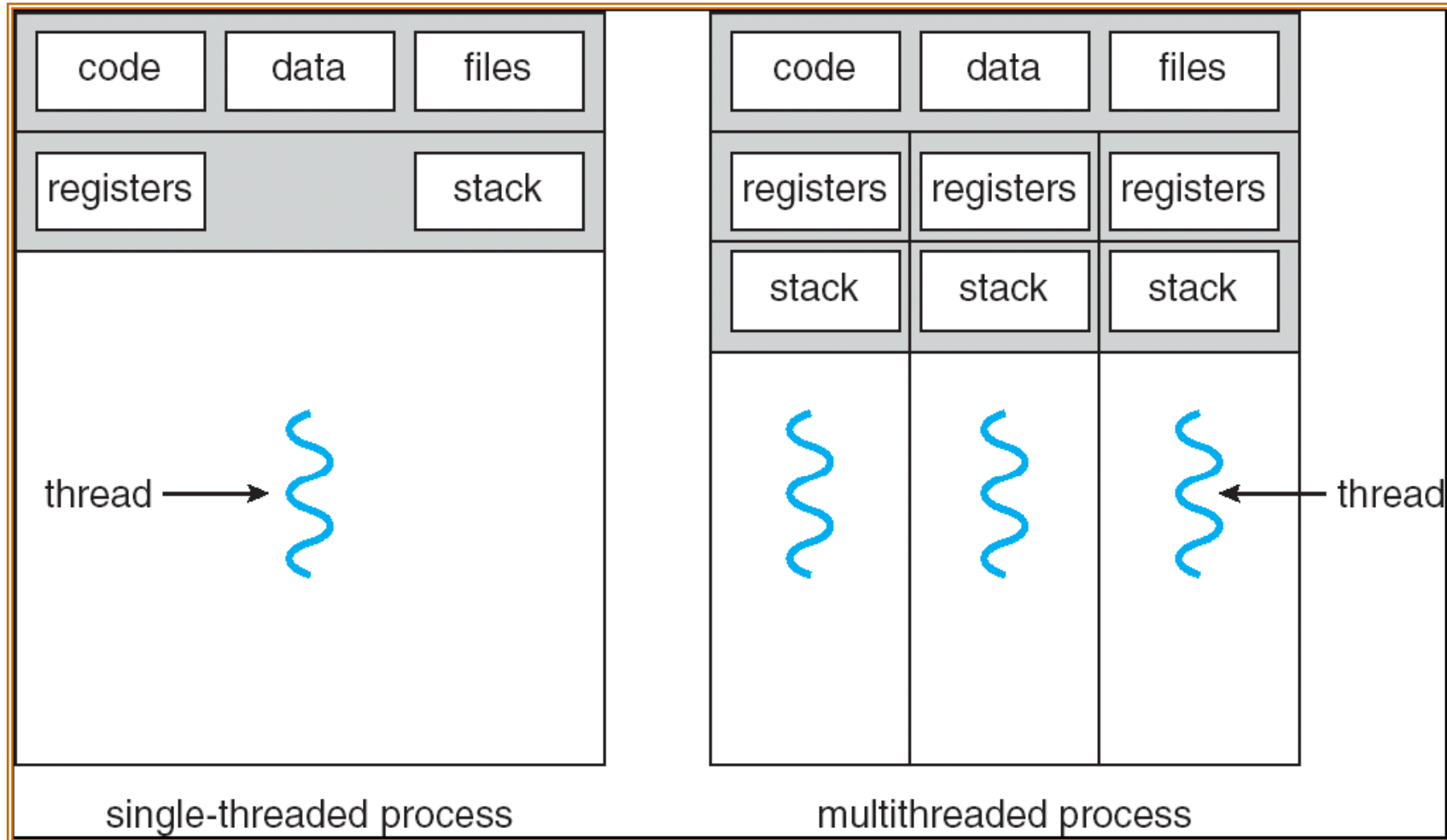


Chapter 4: Multithreaded Programming



- Overview
- Multithreading Models
- Thread Libraries
- Threading Issues
- Operating-System Examples

Single and Multithreaded Processes



Benefits



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

User Threads



- Thread management done by user-level threads library
- Three primary thread libraries:
 - POSIX Pthreads
 - Win32 threads
 - Java threads

Kernel Threads



- Supported by the Kernel
- Examples
 - ▣ Windows XP/2000
 - ▣ Solaris
 - ▣ Linux
 - ▣ Tru64 UNIX
 - ▣ Mac OS X

Multithreading Models



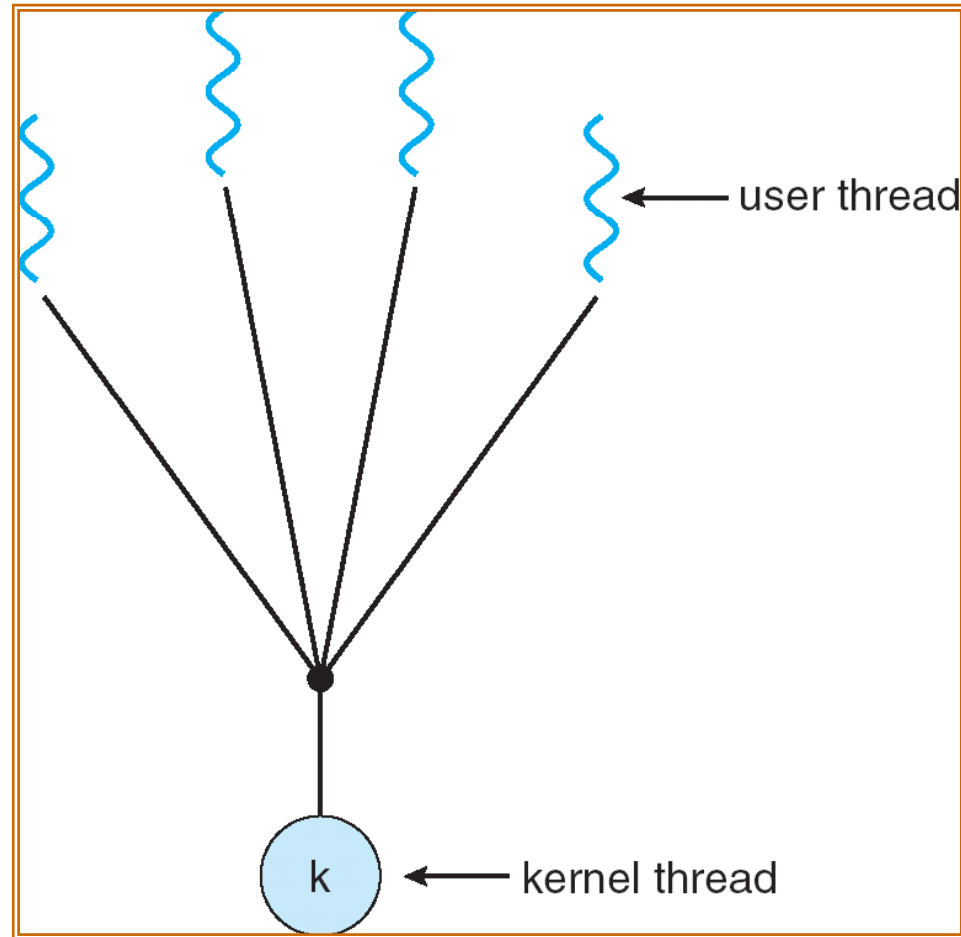
- Many-to-One
- One-to-One
- Many-to-Many

Many-to-One



- Many user-level threads mapped to single kernel thread
- Examples:
 - ▣ Solaris Green Threads
 - ▣ GNU Portable Threads

Many-to-One Model

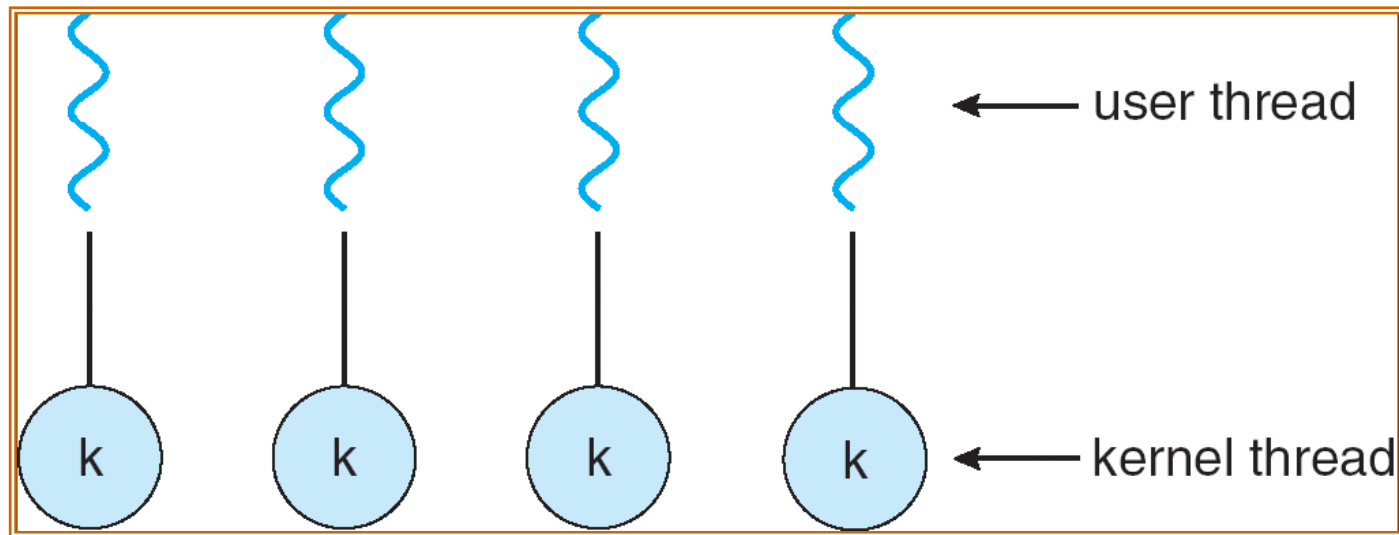


One-to-One



- Each user-level thread maps to kernel thread
- Examples
 - ▣ Windows NT/XP/2000
 - ▣ Linux
 - ▣ Solaris 9 and later

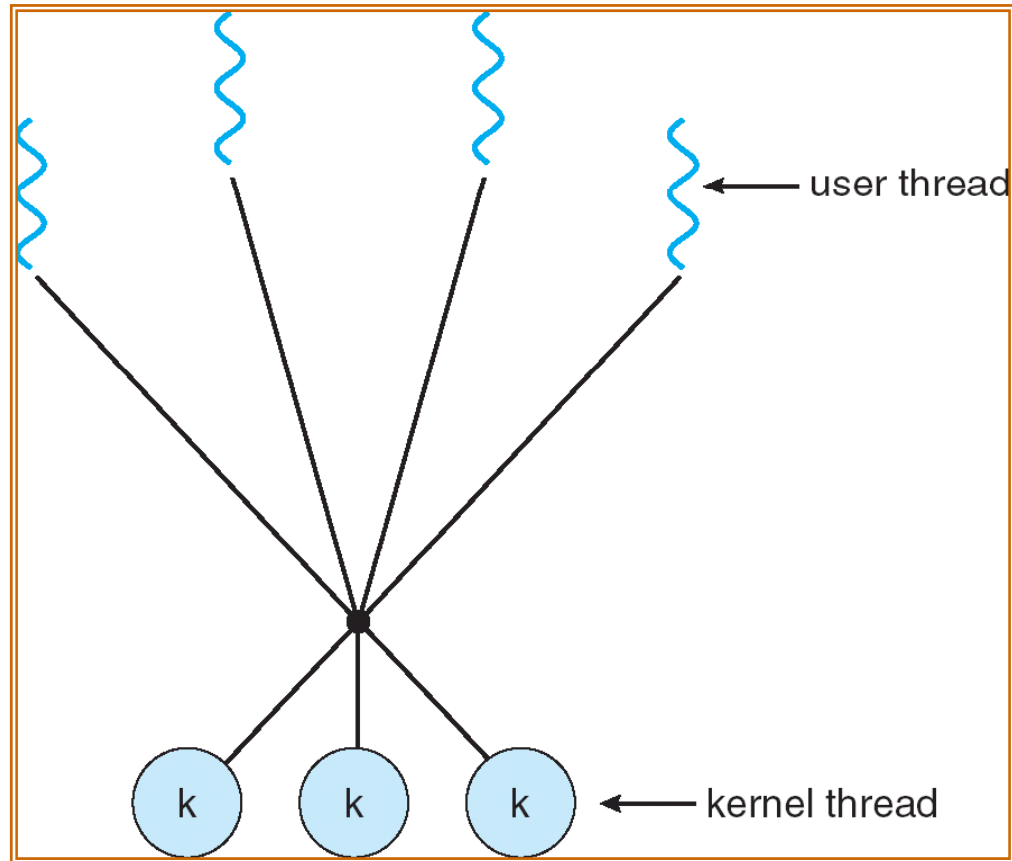
One-to-one Model



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 NT/2000 with the *ThreadFiber* package

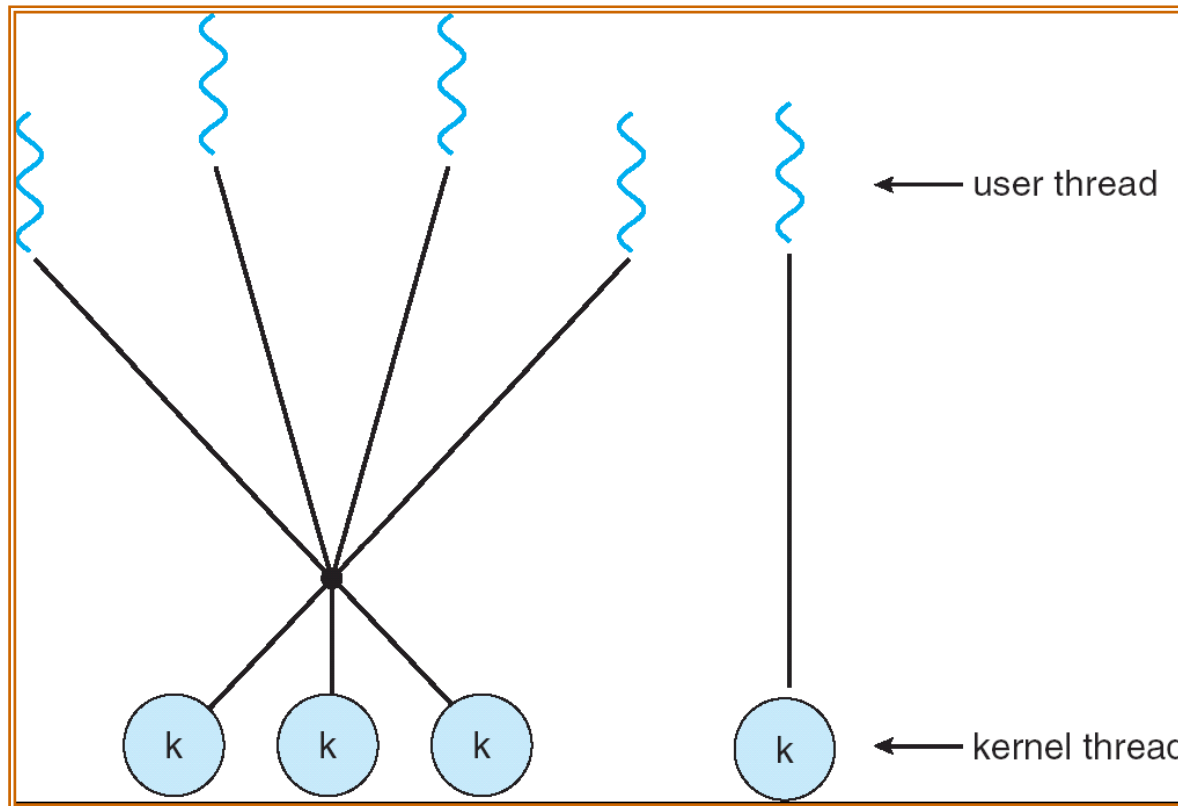
Many-to-Many Model



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

Two-level Model



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 thread to receive all signals for the process

Thread pools



- The scenario of a web server.
- A separate thread to serve a request.
 - ▣ Thread (Created -> discarded) : Request (start and finish)?
 - ▣ Unlimited requests -> unlimited threads?

- Thread pools
 - ▣ Threads sit and wait for work.
 - ▣ Faster to response a request.
 - ▣ The number of threads can be dynamically adjusted.

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

Operating-system Example



- Explore how threads are implemented in Windows XP, Linux, Solaris systems.

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.
- User-level thread library
- Common in UNIX operating systems.
- `pthread_create()`, `pthread_exit()`, `pthread_join()`

Solaris 2 threads

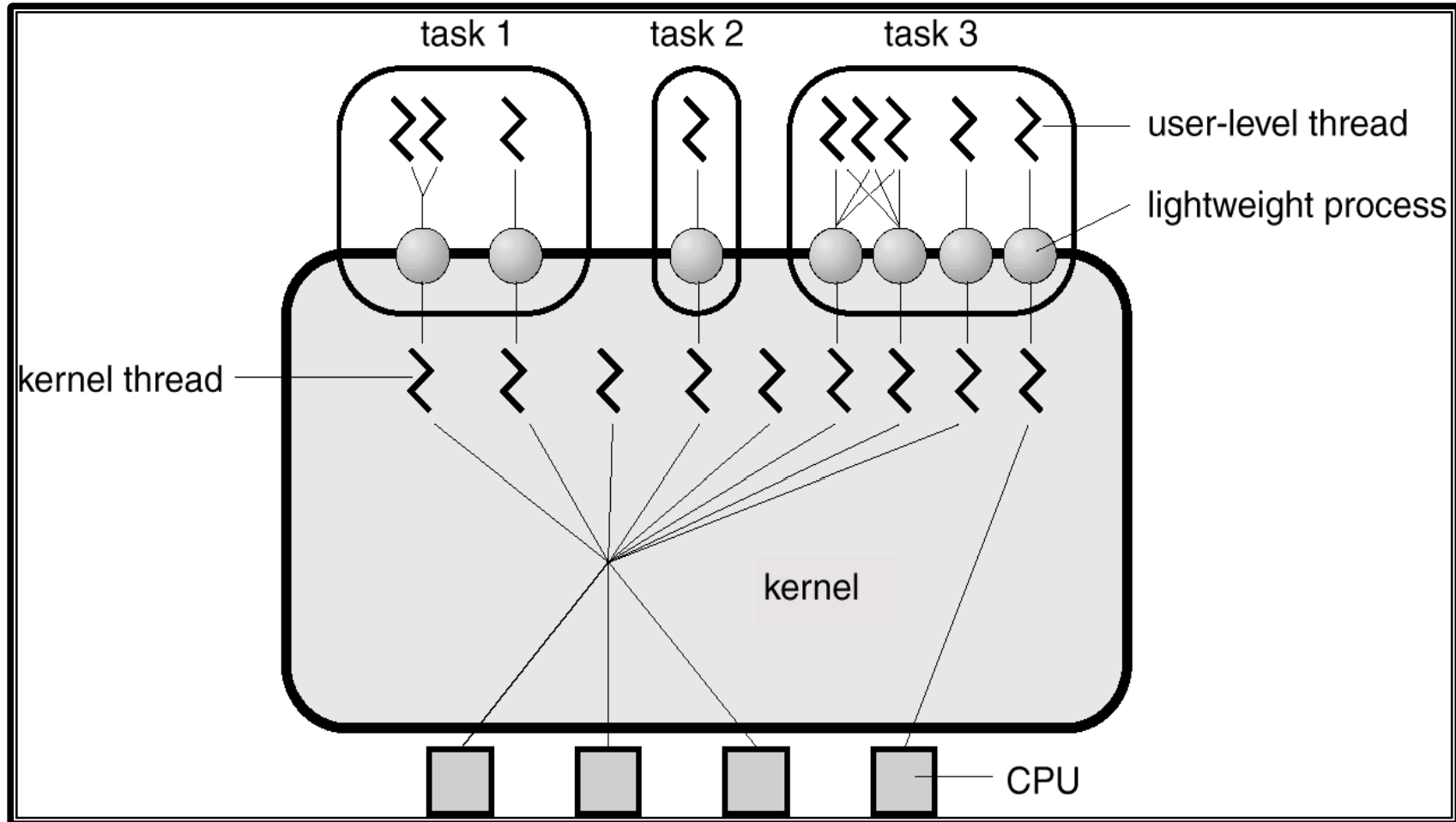


- Lightweight processes (LWPs)
 - ▣ Between user- and kernel- threads.

- Each process contains at least one LWP.
- Each LWP has a kernel-level thread.

- A bound user-level thread
 - ▣ Permanently attached to an LWP. (quick response time)
- An unbound thread
 - ▣ Multiplexed onto the available LWP pool.

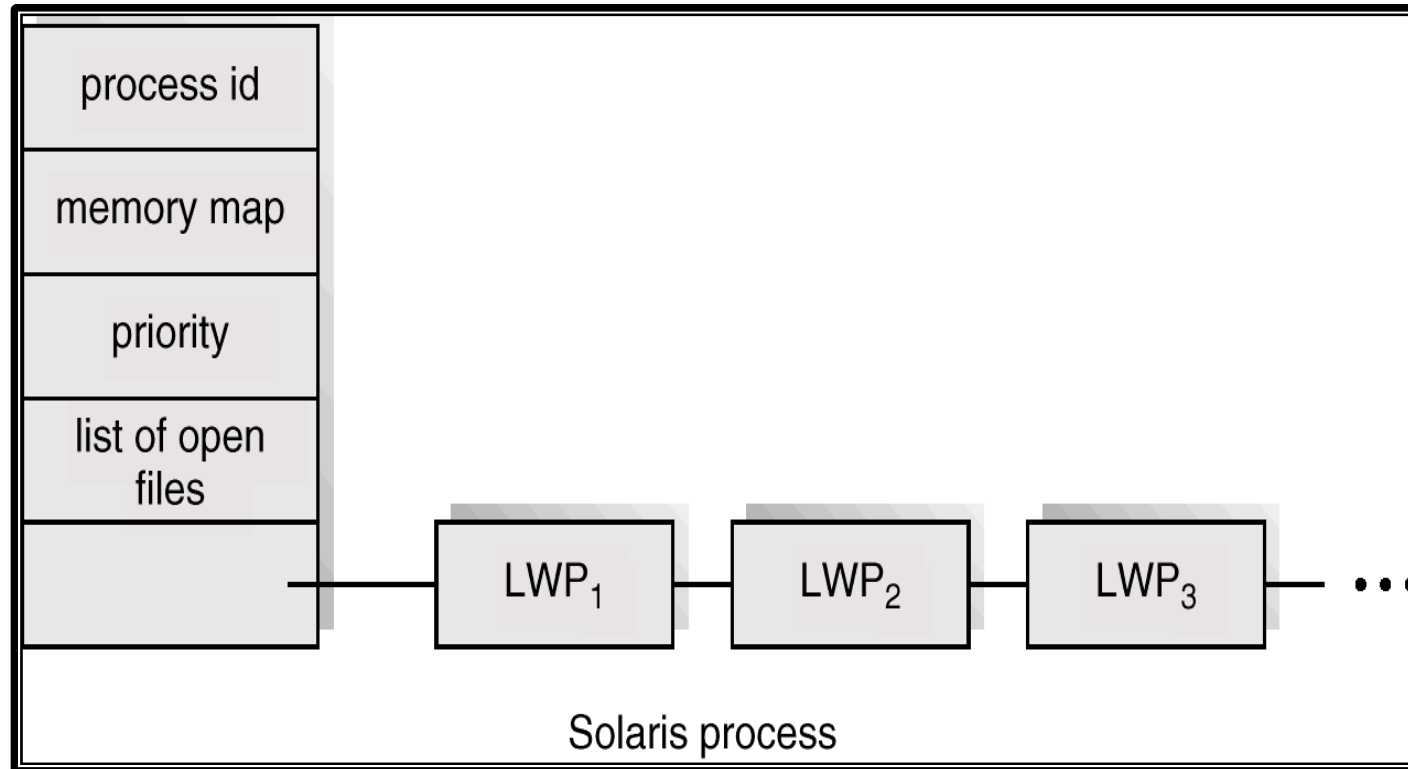
Solaris 2 threads



Solaris 2 threads (cont.)

- User-level threads are scheduled and switched among the LWPs by thread library.
- The thread library dynamically adjusts the number of LWPs.
 - ▣ Creates another LWP if all LWPs in a process are blocked
 - ▣ Deletes unused LWPs (about 5 minutes)
- User-level thread: thread ID, register set, stack, priority..
- LWP: a register set (for its running user-level thread), misc. info.
- Kernel thread: stack, kernel registers, a pointer to the LWP, priority and scheduling info.

Solaris process



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



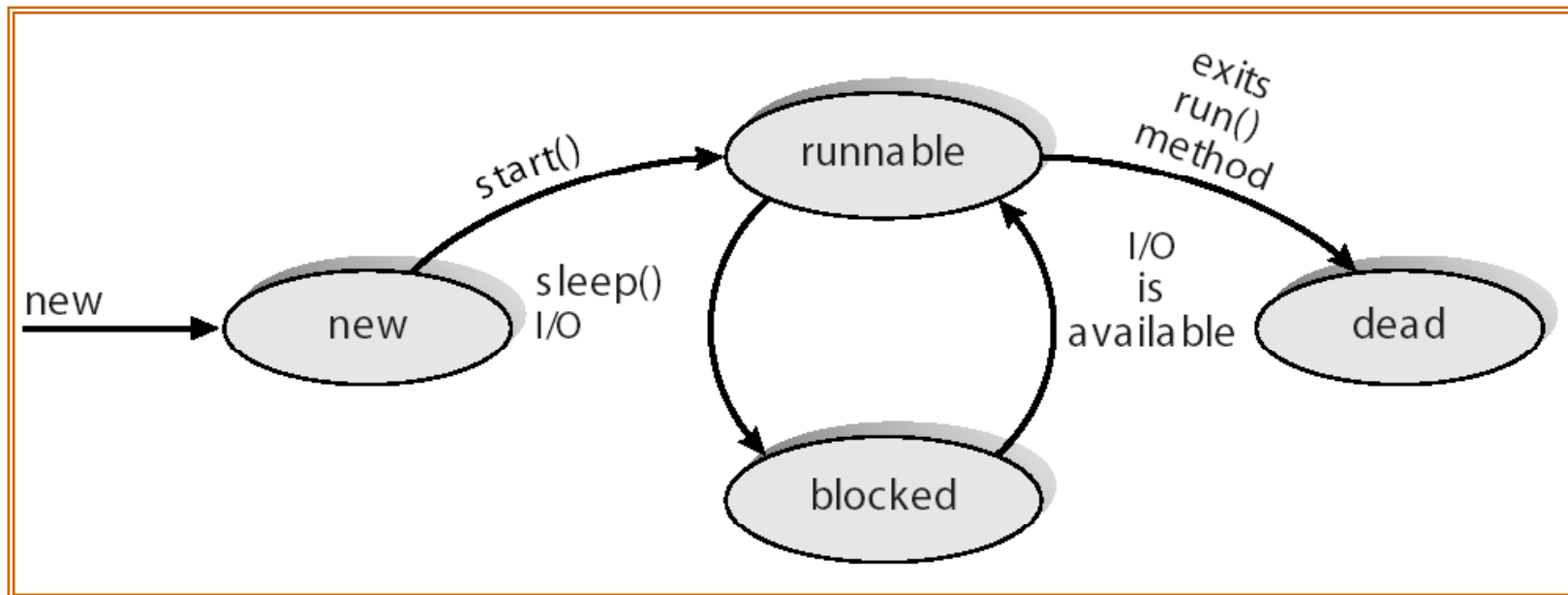
- Linux refers to them as *tasks* rather than *threads*
- Thread creation is done through **clone()** system call
- **clone()** allows a child task to share the address space of the parent task (process)

Java Threads



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

Java Thread States



END OF CHAPTER 4

