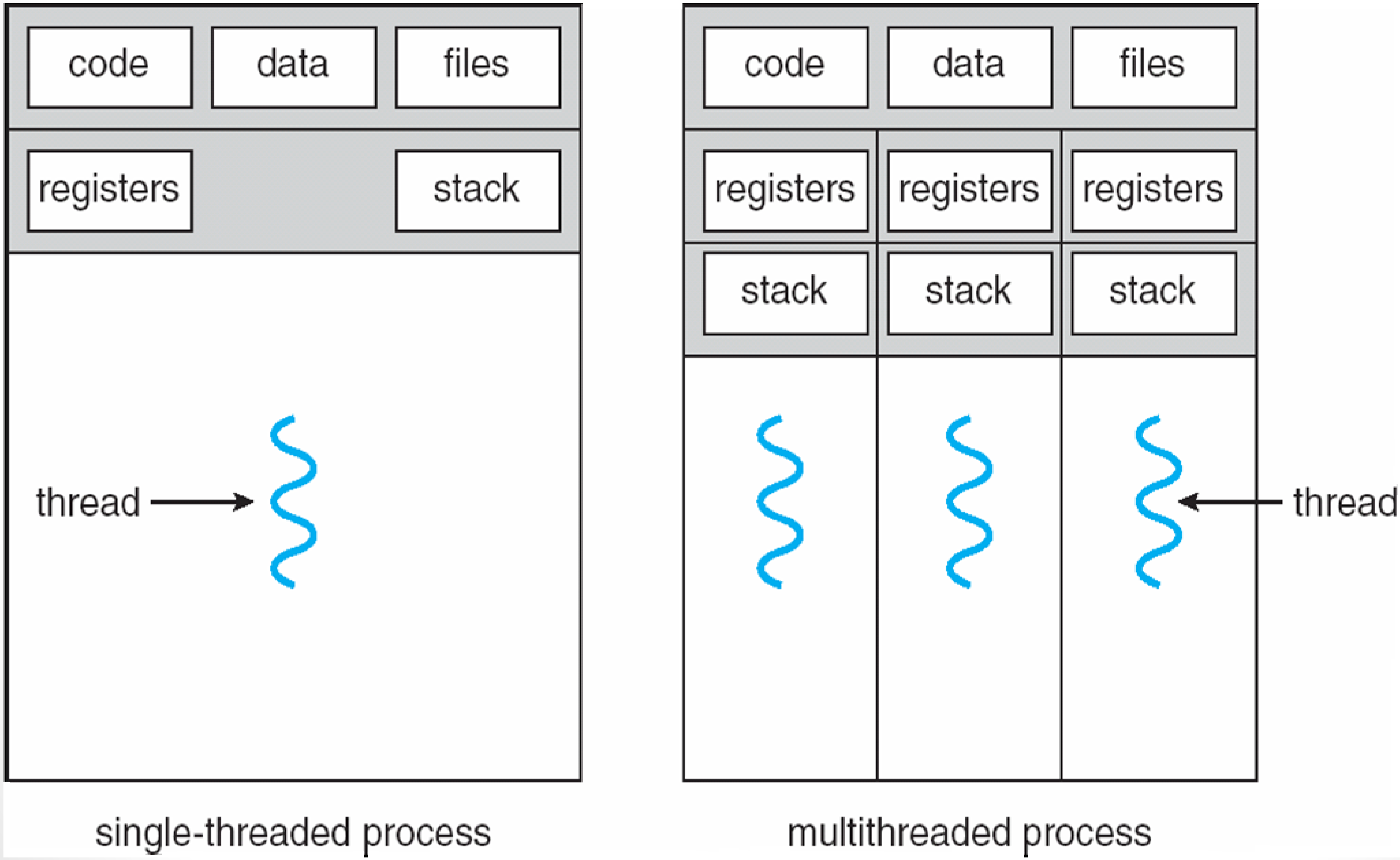
Multithreaded programming

D. Geraldine Bessie Amali
Assistant Professor
SCOPE

Definition

- A thread is a basic unit of CPU utilization.
- It comprises of a thread Id, a program counter , register set and a stack.
- It shares the code , data and other resources with other threads belonging to the same process.
- Multithreading-The ability of an OS to support multiple, concurrent paths of execution within a single process

Single and Multithreaded Processes



Single thread Vs multi thread

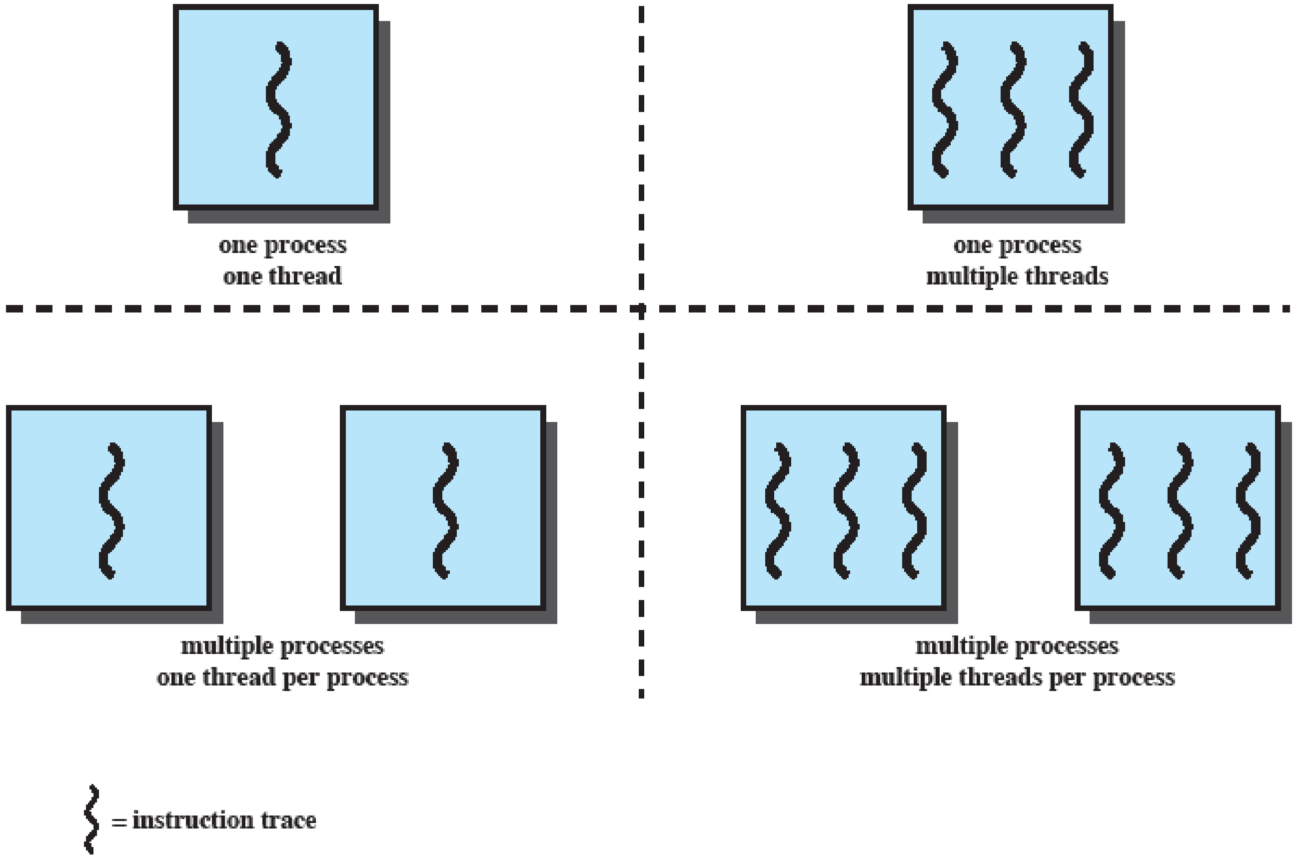


Figure 4.1 Threads and Processes [ANDE97]

Each thread has:

- an execution state (Running, Ready, etc.)
- saved thread context when not running (TCB)
- an execution stack
- some per-thread static storage for local variables
- access to the shared memory and resources of its process (all threads of a process share this)

Benefits

- Responsiveness
- Resource Sharing
- Economy
- Scalability

Takes less time to create a new thread than a process

Less time to terminate a thread than a process

Switching between two threads takes less time than switching between processes

Threads enhance efficiency in communication between programs

Threads

- In an OS that supports threads, scheduling and dispatching is done on a thread basis
- Most of the state information dealing with execution is maintained in thread-level data structures
  - suspending a process involves suspending all threads of the process
  - termination of a process terminates all threads within the process

Thread Execution States

The key states for a thread are:

- Running
- Ready
- Blocked

Thread operations associated with a change in thread state are:

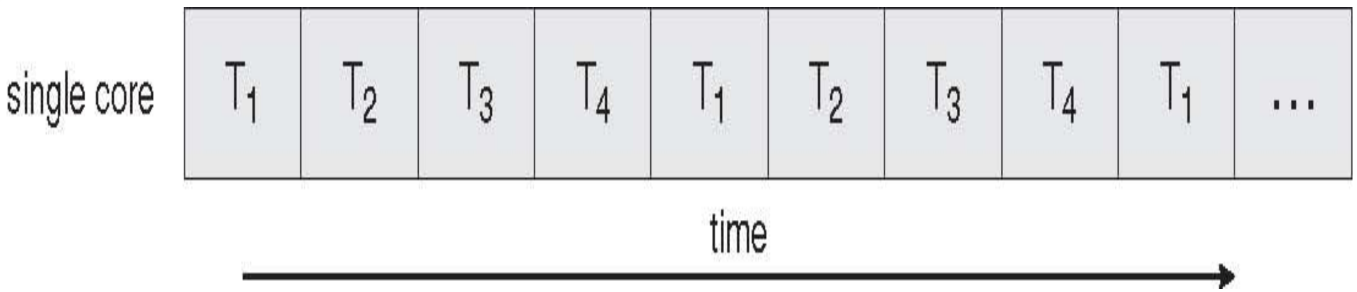
- Spawn (create)
- Block
- Unblock
- Finish

Thread Execution

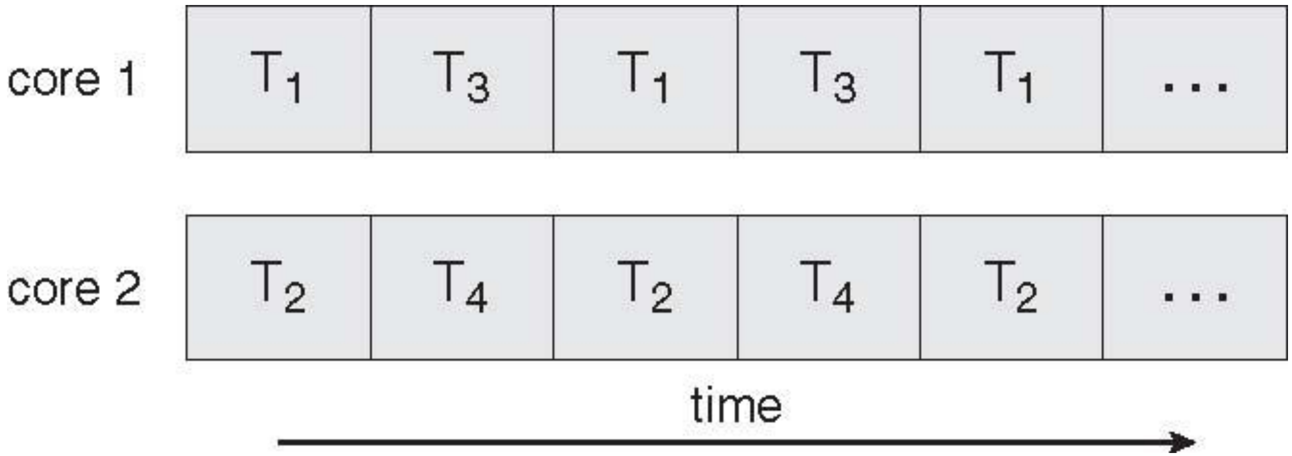
- A key issue with threads is whether or not they can be scheduled independently of the process to which they belong.

- Or, is it possible to block one thread in a process without blocking the entire process?
  - If not, then much of the flexibility of threads is lost.

Multicore programming



Concurrent execution on a single core system

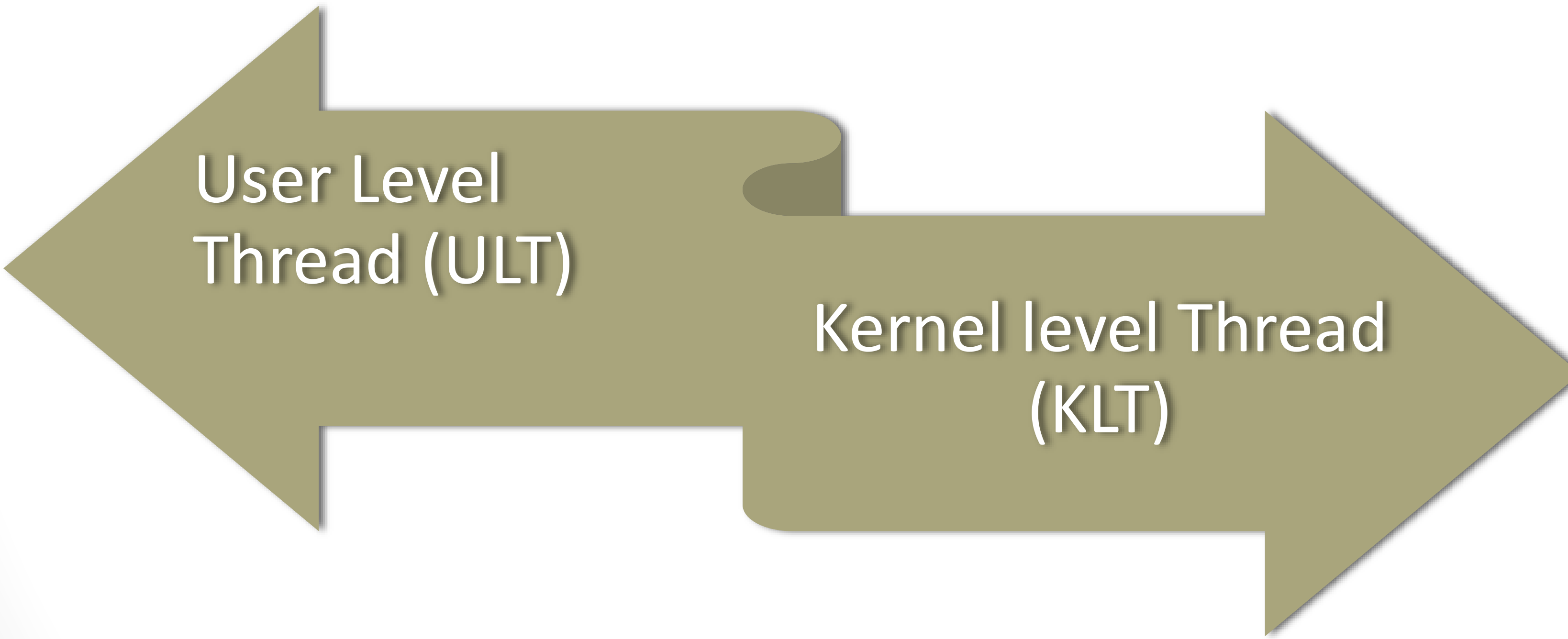


Parallel execution on a multi core system

Challenges in programming for multi core systems

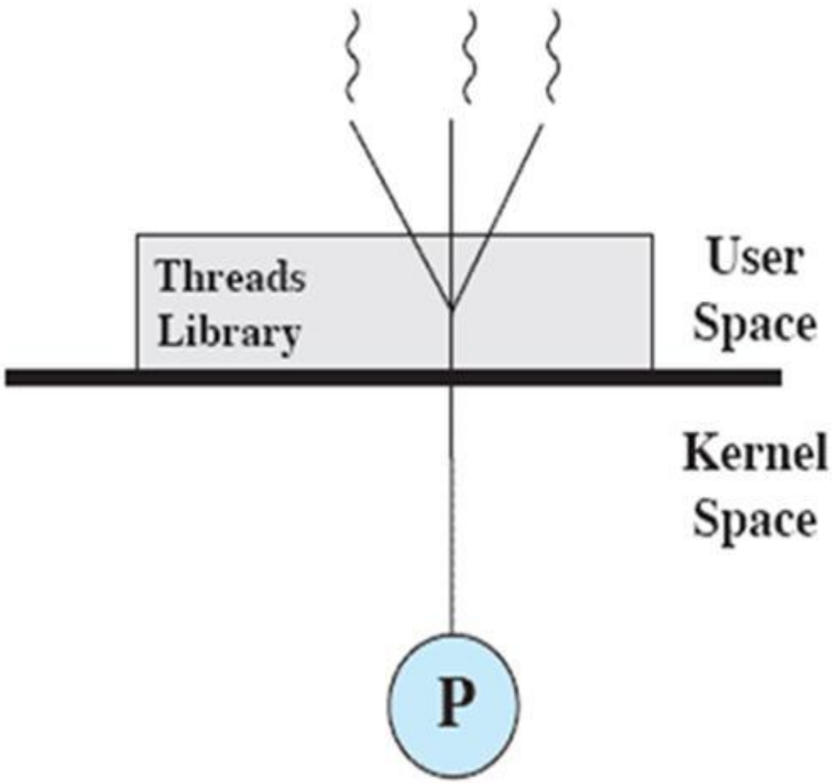
- Dividing activities
- Balance
- Data splitting
- Data dependency
- Testing and debugging

Types of Threads



User-Level Threads (ULTs)

- Thread management is done by the application
- The kernel is not aware of the existence of threads

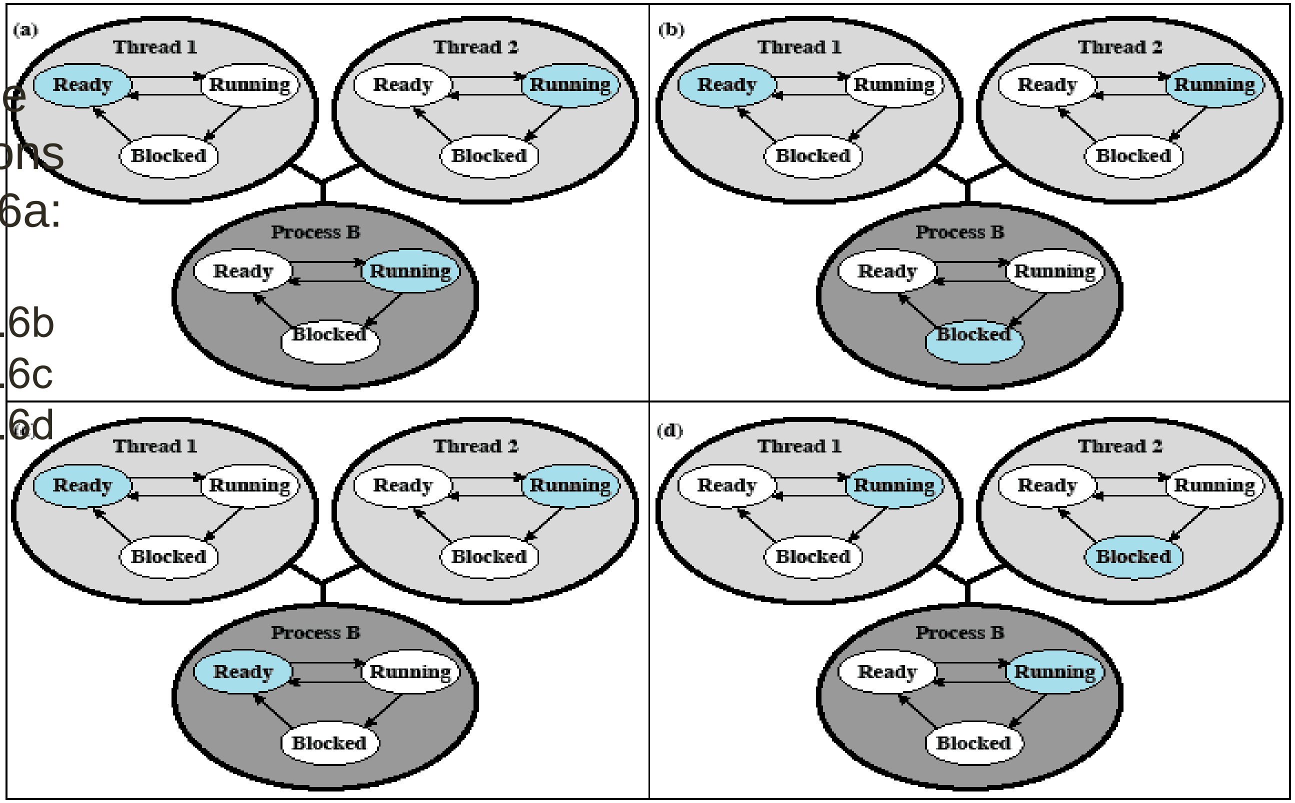


(a) Pure user-level

Relationships Between ULT States and Process States

Possible transitions from 4.6a:

4.6a→4.6b
4.6a→4.6c
4.6a→4.6d



Colored state is current state

Figure 4.6 Examples of the Relationships between User-Level Thread States and Process States

Figure 4.7 Examples of the Relationships Between User-Level Thread States and Process States

Advantages of ULTs

Thread switching does not require kernel mode privileges (no mode switches)

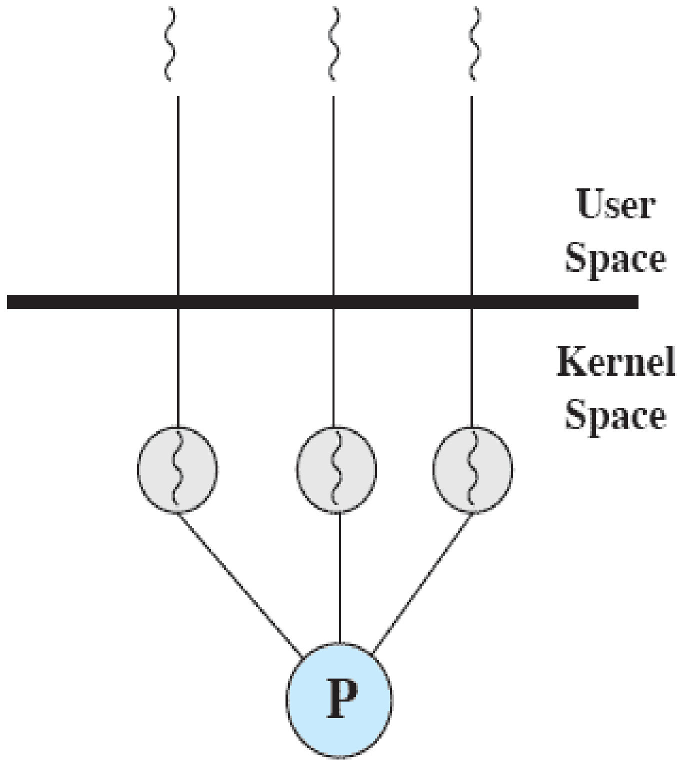
Scheduling can be application specific

ULTs can run on any OS

Disadvantages of ULTs

- In a typical OS many system calls are blocking
  - as a result, when a ULT executes a system call, not only is that thread blocked, but all of the threads within the process are blocked
- In a pure ULT strategy, a multithreaded application cannot take advantage of multiprocessing

Kernel-Level Threads (KLTs)



(b) Pure kernel-level

- Thread management is done by the kernel (could call them K*M*T)
  - no thread management is done by the application
  - Windows is an example of this approach

Advantages of KLTs

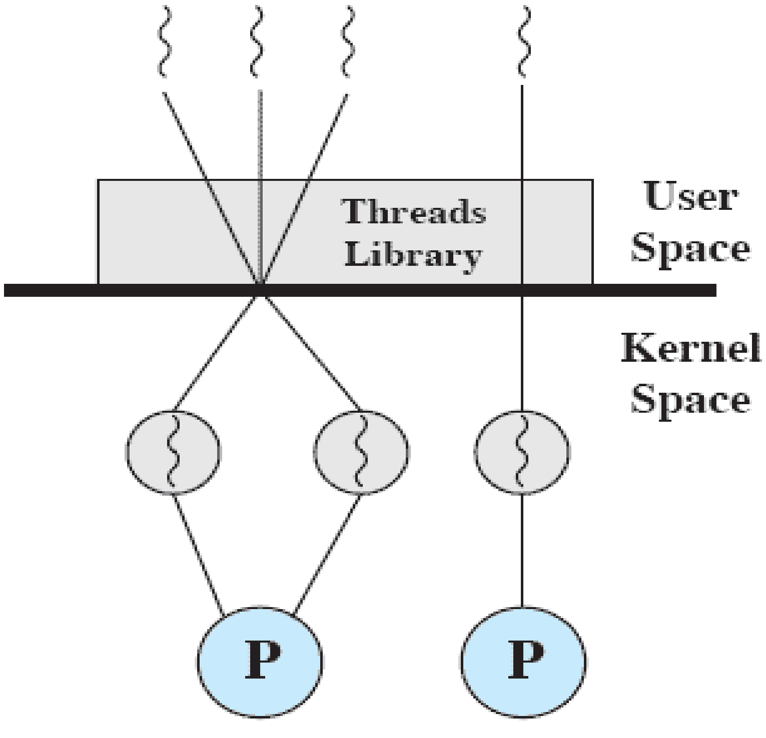
- The kernel can simultaneously schedule multiple threads from the same process on multiple processors
- If one thread in a process is blocked, the kernel can schedule another thread of the same process

Disadvantage of KLTs

- The transfer of control from one thread to another within the same process requires a mode switch to the kernel

Combined Approaches

- Thread creation is done in the user space
- Bulk of scheduling and synchronization of threads is by the application
- Solaris is an example

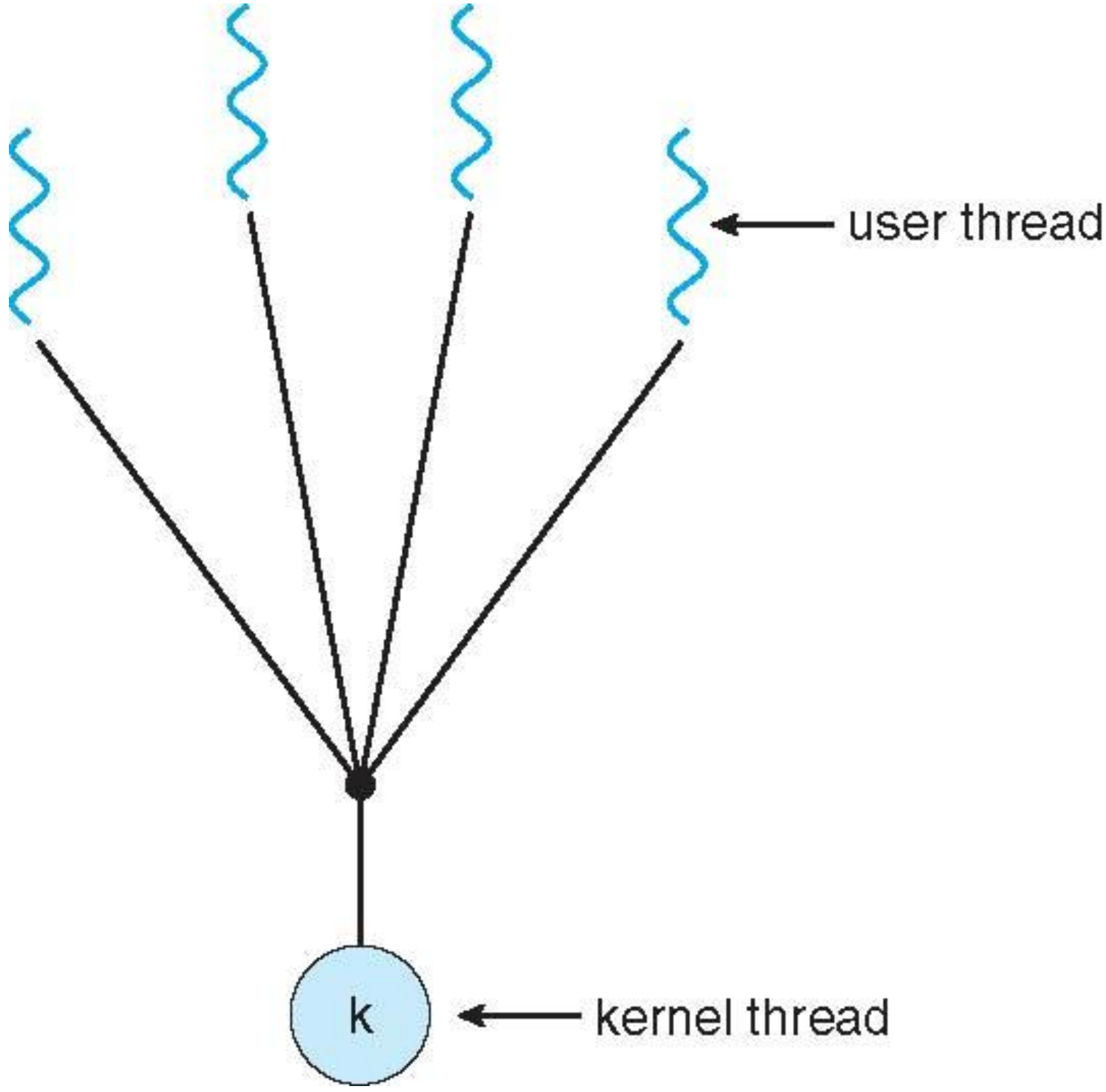


(c) Combined

Multithreading Models

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

Many-to-One Model



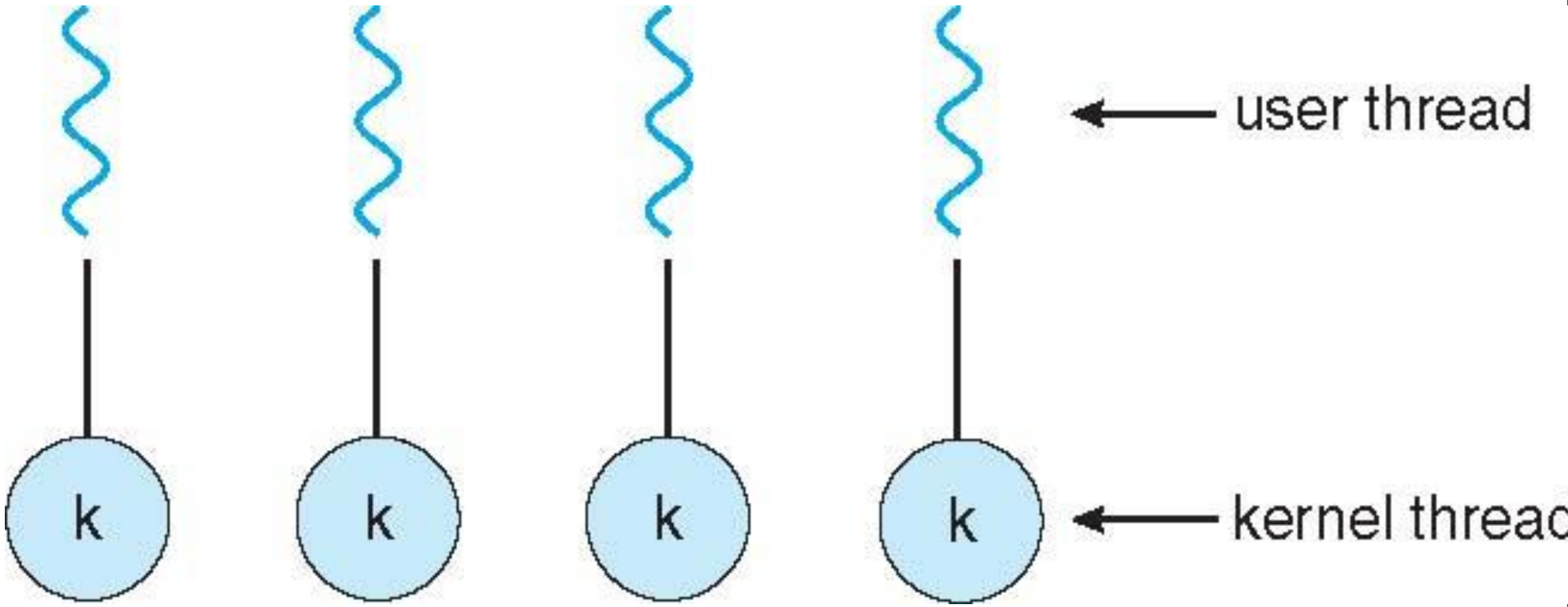
Many-to-One Model

- Many user-level threads mapped to single kernel thread

- User-level threads can be concurrent without being parallel, thread switching incurs low overhead, and blocking of a user-level thread leads to blocking of all threads in the process.

- Examples:
  - **Solaris Green Threads**
  - **GNU Portable Threads**

One-to-one Model

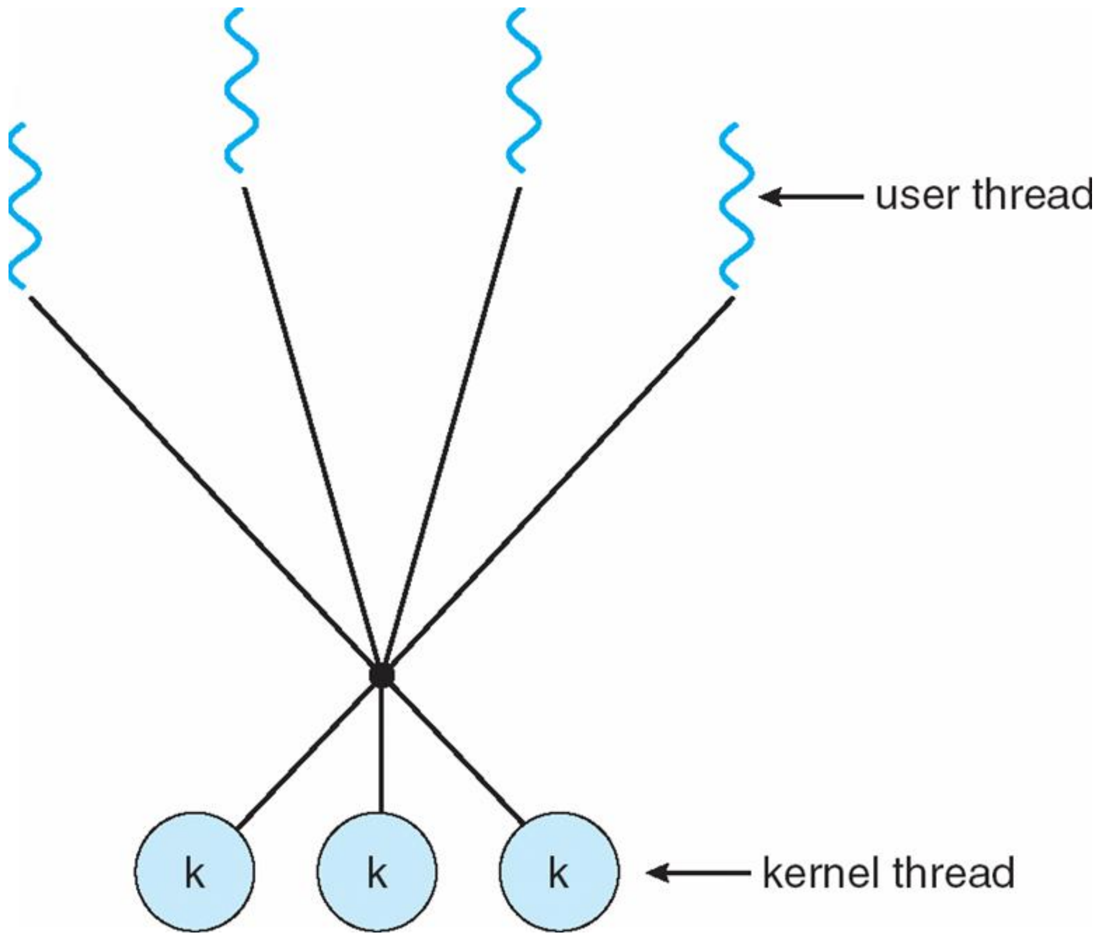


One-to-One

- Each user-level thread maps to kernel thread
- Threads can operate in parallel on different CPUs of a multiprocessor system; however, switching between threads is performed at the kernel level and incurs high overhead.
- Blocking of a user-level thread does not block other user-level threads of the process because they are mapped into different kernel-level threads.

- Examples
  - Windows NT/XP/2000
  - Linux
  - Solaris 9 and later

Many-to-Many Model



Many-to-Many Model

- Allows many user level threads to be mapped to many kernel threads

- It provides parallelism between user-level threads that are mapped into different kernel-level threads at the same time, and provides low overhead of switching.

Examples

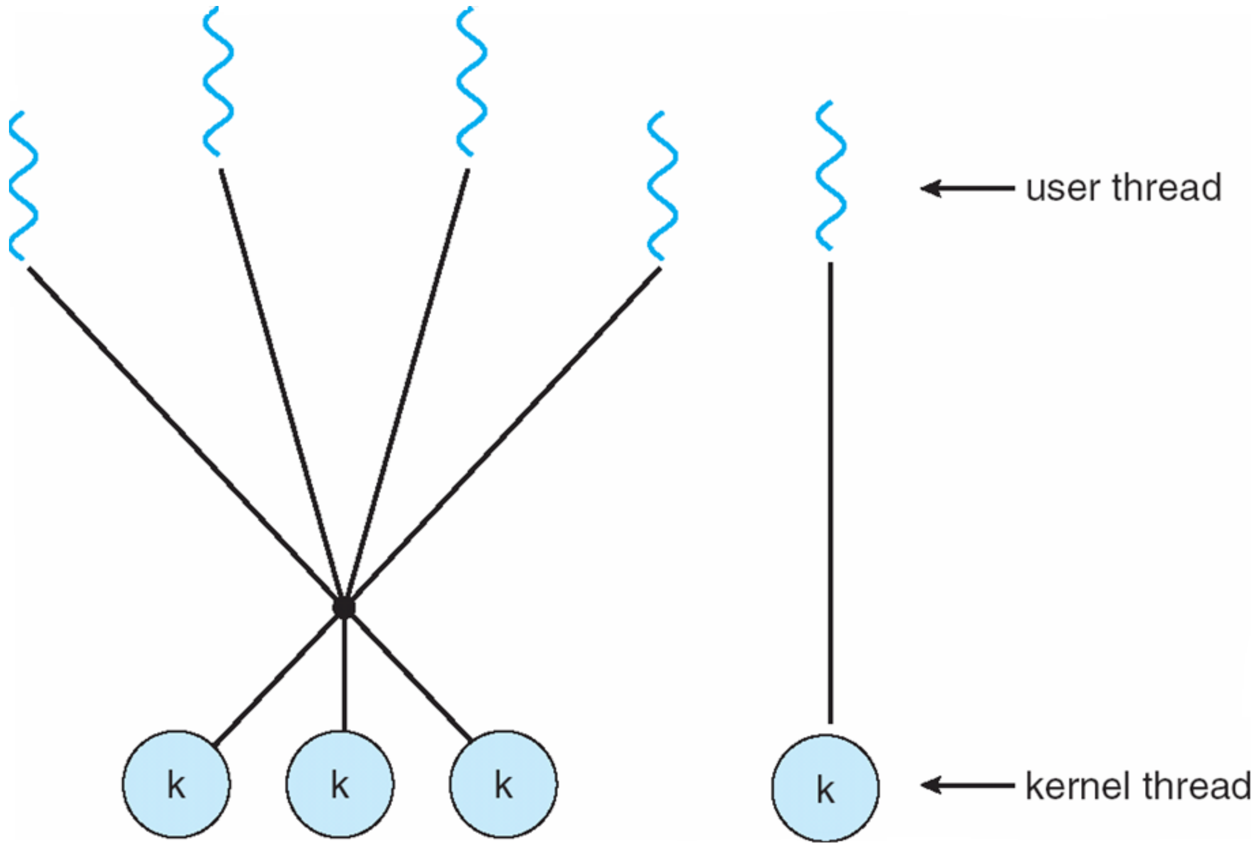
- Solaris prior to version 9
- Windows NT/2000 with the *ThreadFiber* package

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

Two-level Model



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

Pthreads

- The ANSI/IEEE Portable Operating System Interface (POSIX) standard defines the pthreads application program interface for use by C language programs.
- May be provided either as 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)