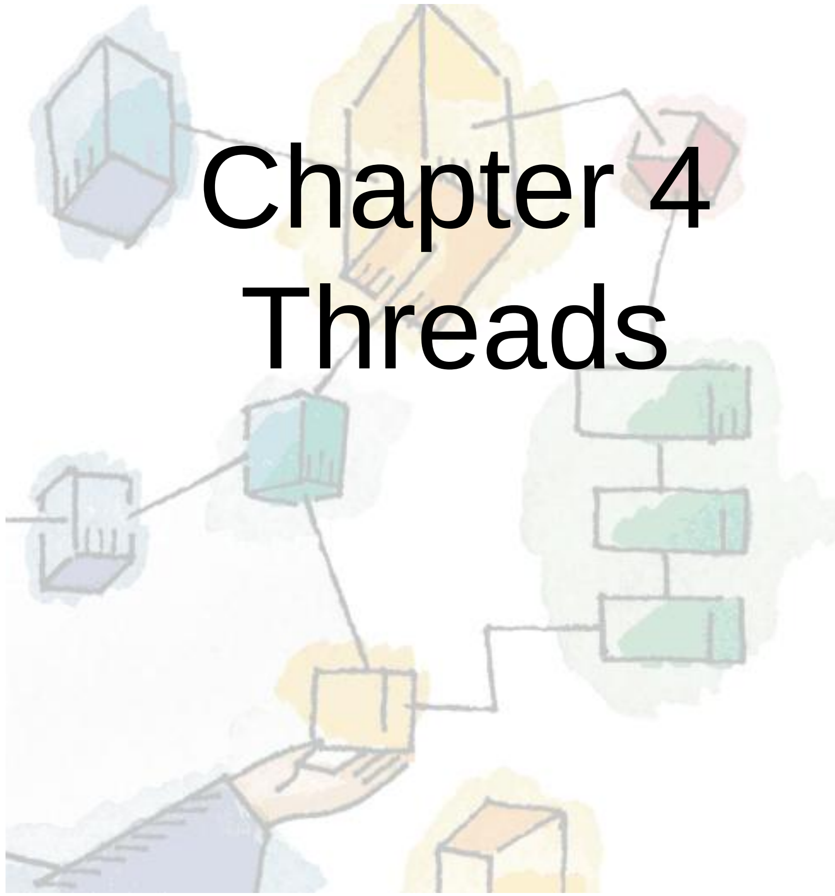
Operating Systems:
Internals and Design Principles, 6/E

William Stallings

Chapter 4
Threads



Processes and Threads

- Processes have two characteristics:
  - **Resource ownership** - process includes a address space to hold the process image, can be assigned resource ownership
  - **Scheduling/execution** - follows an execution path that may be interleaved with other processes

- These two characteristics are treated independently by the operating system

Processes and Threads

- The unit of dispatching is referred to as a ***thread*** or lightweight process

- The unit of resource ownership is referred to as a process or ***task***

Multithreading

- The ability of an OS to support multiple, concurrent paths of execution within a single process.

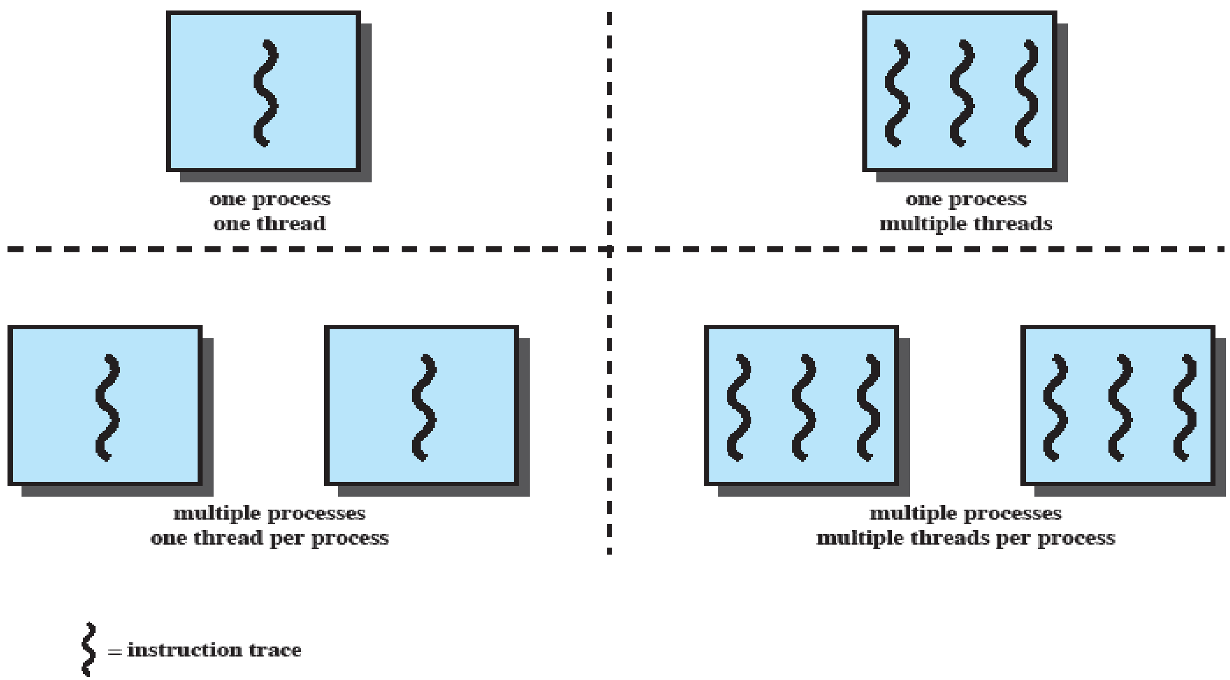


Figure 4.1 Threads and Processes [ANDE97]

Single Thread Approaches

- MS-DOS supports a single user process and a single thread.
- Some UNIX, support multiple user processes but only support one thread per process

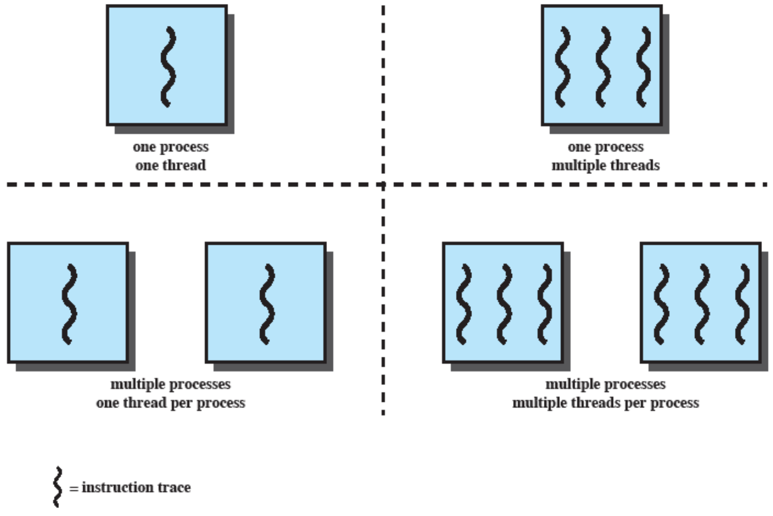


Figure 4.1 Threads and Processes [ANDE97]

Multithreading

- Java run-time environment is a single process with multiple threads
- Multiple processes ***and*** threads are found in Windows, Solaris, and many modern versions of UNIX

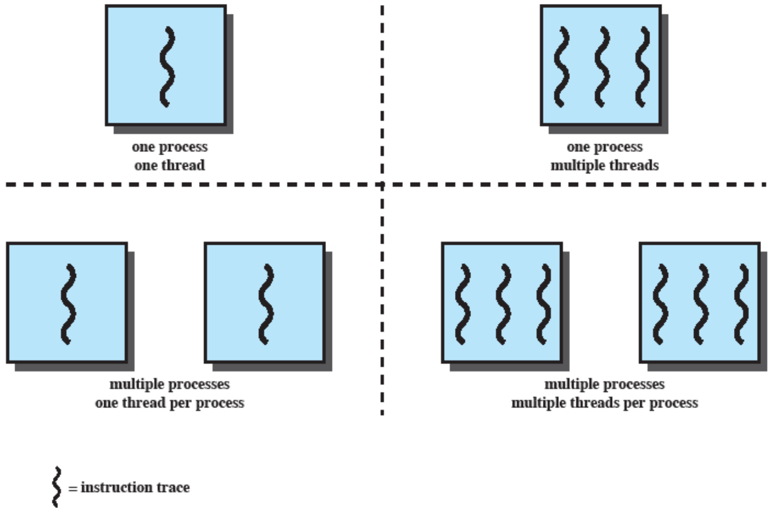


Figure 4.1 Threads and Processes [ANDE97]

Processes

- An address space which holds the process image
- Protected access to
  - Processors,
  - Other processes,
  - Files,
  - I/O resources

One or More Threads in Process

- Each thread has
  - An execution state (running, ready, etc.)
  - Saved thread context when not running
  - An execution stack
  - Some per-thread static storage for local variables
  - Access to the memory and resources of its process (all threads of a process share this)

Threads vs. processes

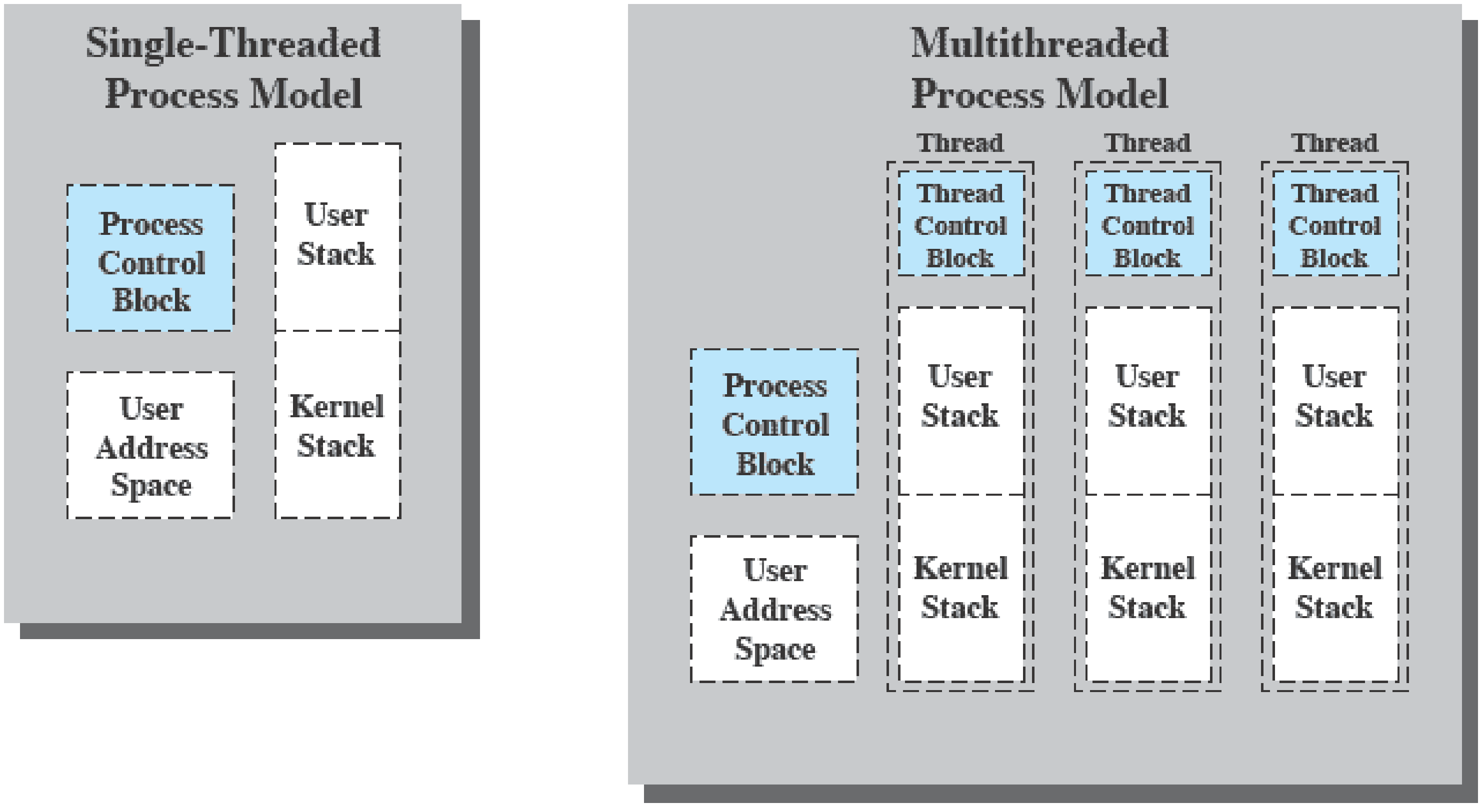


Figure 4.2 Single Threaded and Multithreaded Process Models

Benefits of Threads

- 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 that switching processes
- Threads can communicate with each other
  - without invoking the kernel

Thread use in a Single-User System

- Foreground and background work
  - One thread for display, other for input etc
- Asynchronous processing
  - Periodic backup of main memory to disk
- Speed of execution
  - Multiple threads can run concurrently for multiprocessor systems
- Modular program structure
  - Easy design

Threads: Issue

- Several actions that affect all of the threads in a process
  - The OS must manage these at the process level.

- Examples:
  - Suspending a process involves suspending all threads of the process
  - Termination of a process, terminates all threads within the process

Thread States and Operations

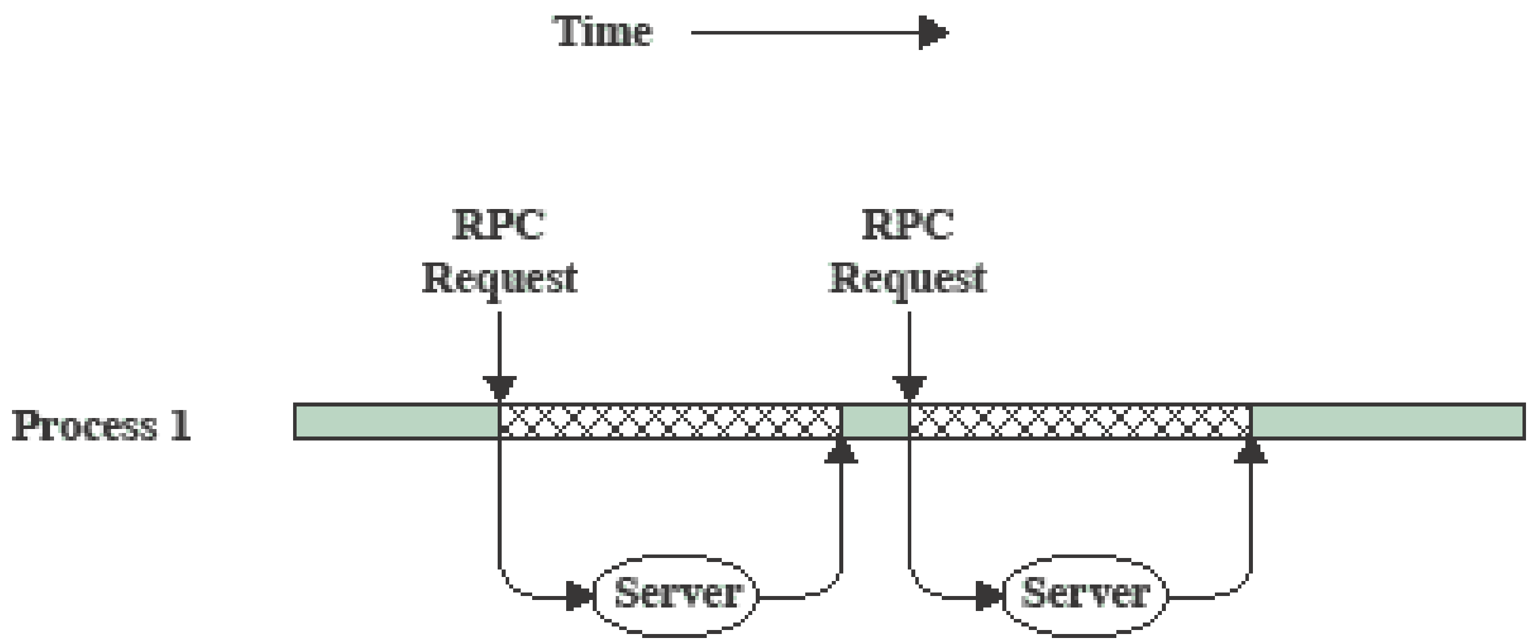
- States:
  - Running
  - Ready
  - Blocked

- Operations:
  - Spawn
  - Block
  - Unblock
  - Finish

Example: Remote Procedure Call

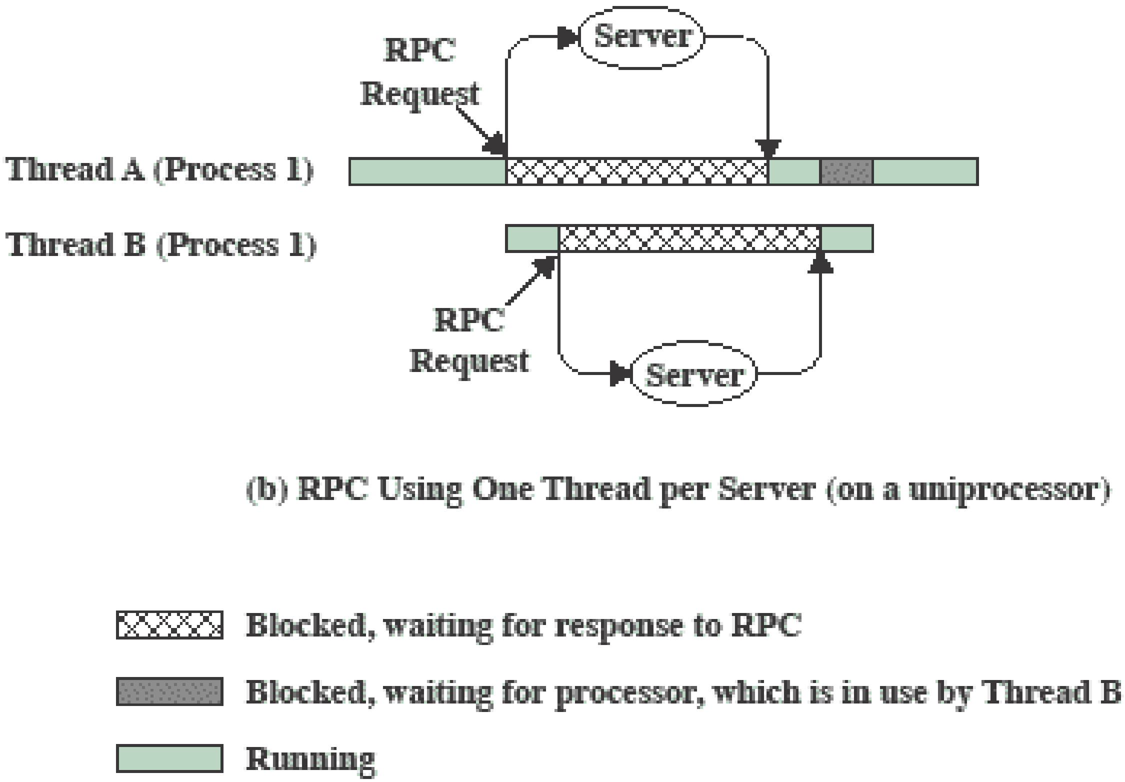
- Consider:
  - A program that performs two remote procedure calls (RPCs)
  - to two different hosts
  - to obtain a combined result.

RPC
Using Single Thread



(a) RPC Using Single Thread

RPC Using One Thread per Server



(b) RPC Using One Thread per Server (on a uniprocessor)

Multithreading on a Uniprocessor

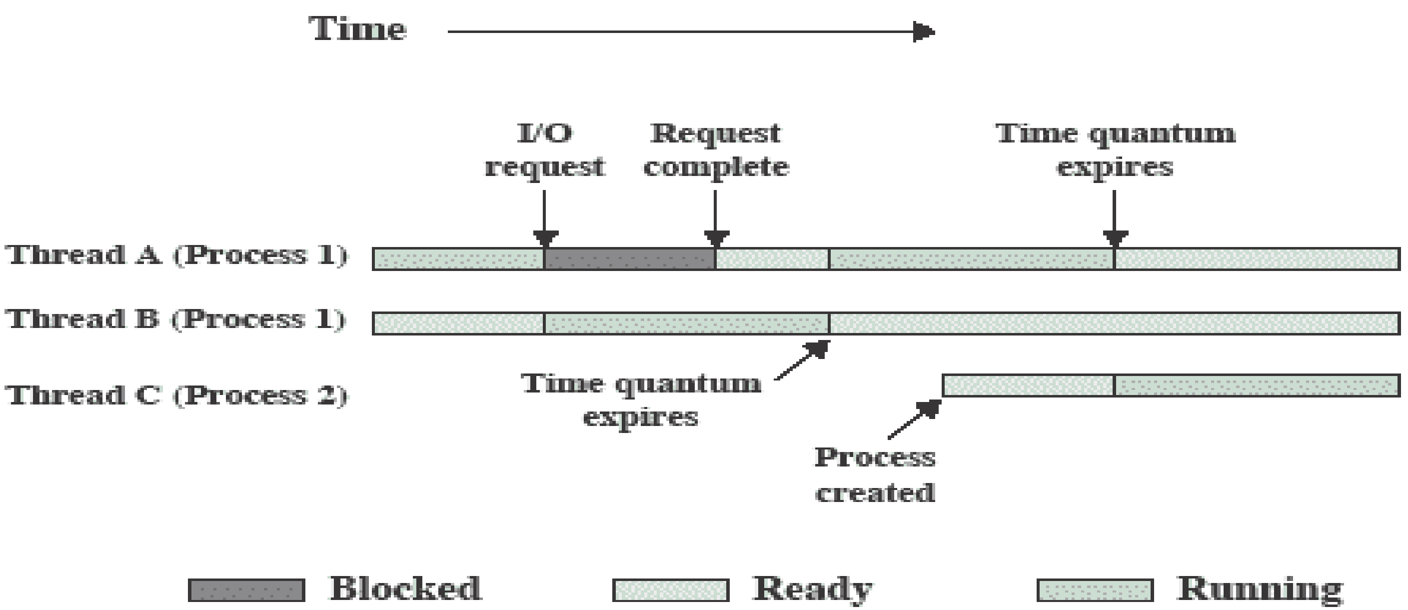


Figure 4.4 Multithreading Example on a Uniprocessor

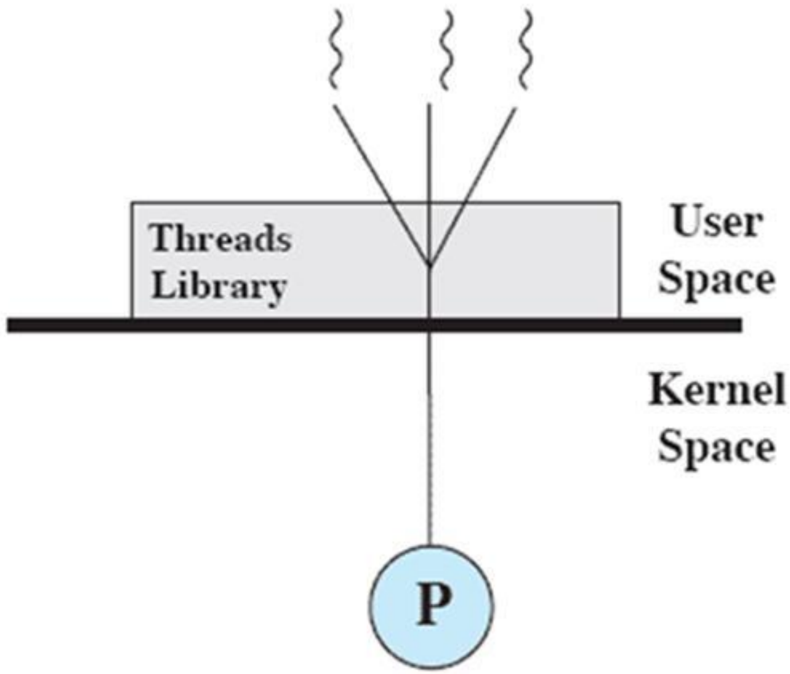
Categories of Thread Implementation

- User Level Thread (ULT)

- Kernel level Thread (KLT)

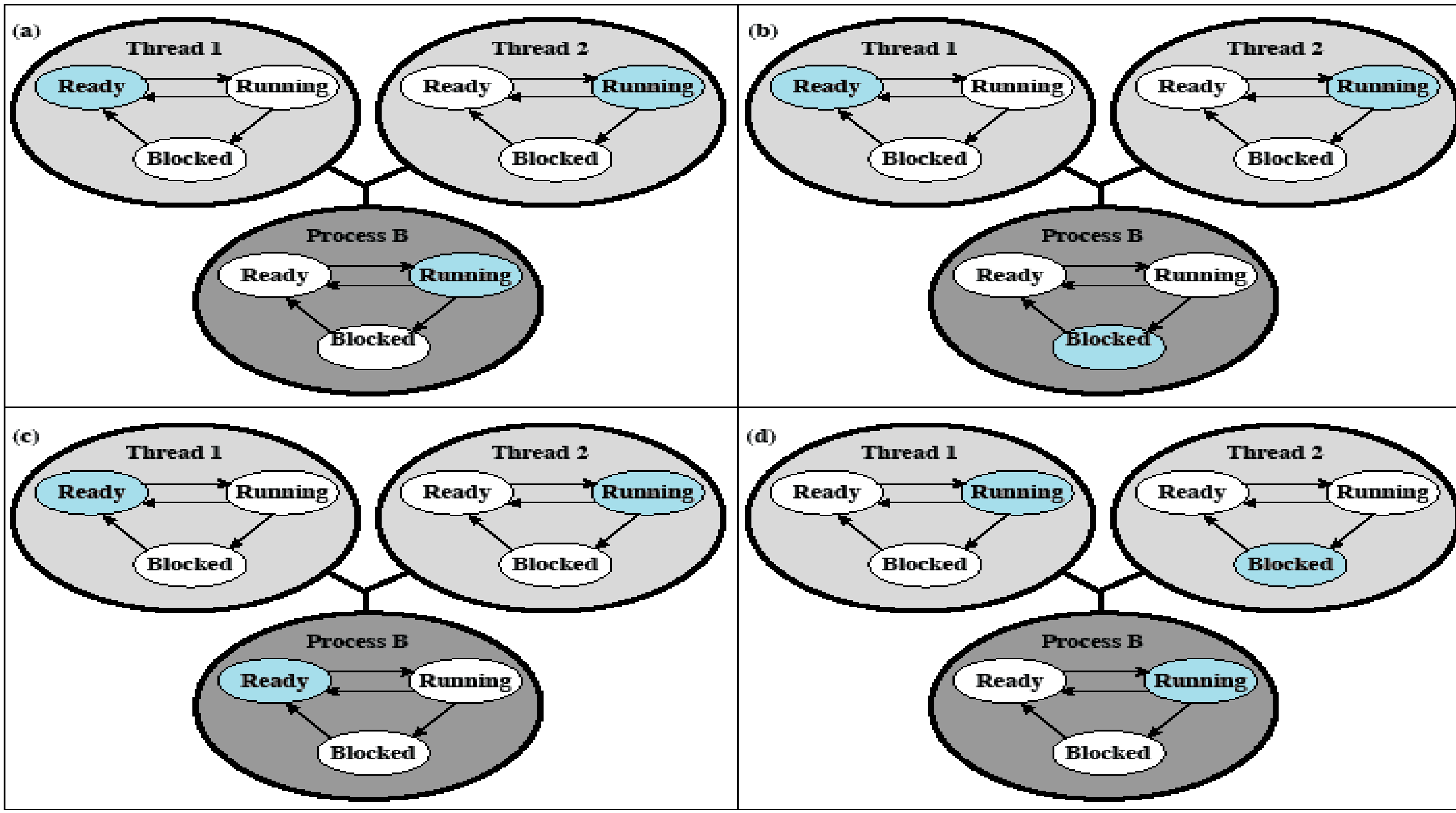
User-Level Threads

- All thread management is done by the application
- The kernel is not aware of the existence of threads
- Application begins with single thread
  - When the process is in running state, new thread can be spawned



(a) Pure user-level

Relationships between ULT Thread and Process States



Colored state
is current state

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

Relationships between ULT Thread and Process States

a) Process B is running, using Thread 2

b) Thread 2's application makes system call and blocks B

– Kernel Switches to other process

– Thread 2 still remains in running state (To maintain data structures) (control needs to return to Thread 2)

c) Clock Interrupt

– B is placed in ready state, thread state remains same

d) Thread 2 needs some action to be performed by Thread 1

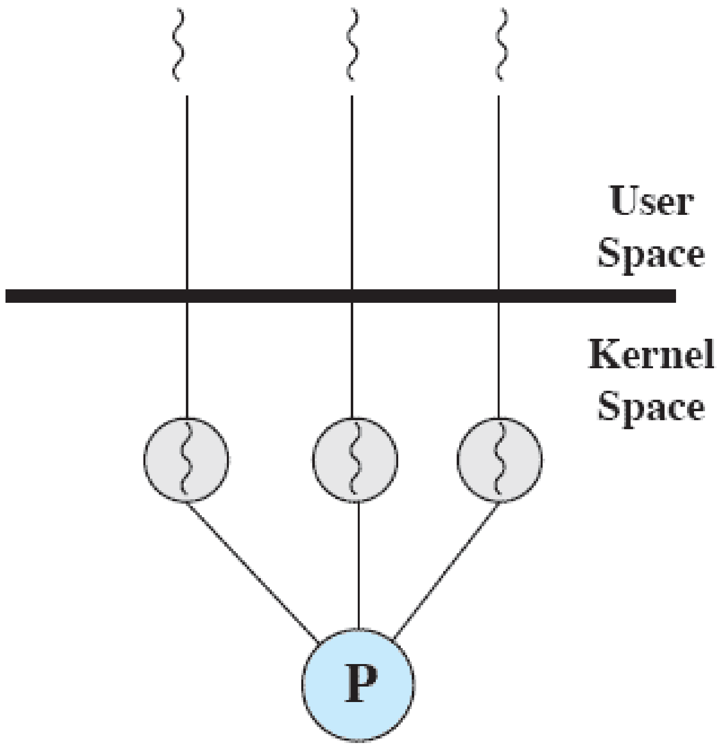
Advantages of ULT

- No need of kernel privileges for thread switching (no mode switch)
- Scheduling can be application specific
- ULT can run on any OS

Disadvantages of ULT

- Many system calls are blocking
  - Process is ultimately blocked
- Kernel assigns one process to only one processor at a time
  - A single thread can execute at a time
- Solution: Jacketing
  - Convert blocking system call to non blocking system call
    - Before requesting I/O, check whether it is busy (using jacketing routine)
    - If busy, then block the thread and transfer control to other thread

Kernel-Level Threads



(b) Pure kernel-level

- Kernel maintains context information for the process and the threads
  - No thread management done by application
- Scheduling is done on a thread basis
- Windows is an example of this approach

Advantages of KLT

- 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.
- Kernel routines themselves can be multithreaded.

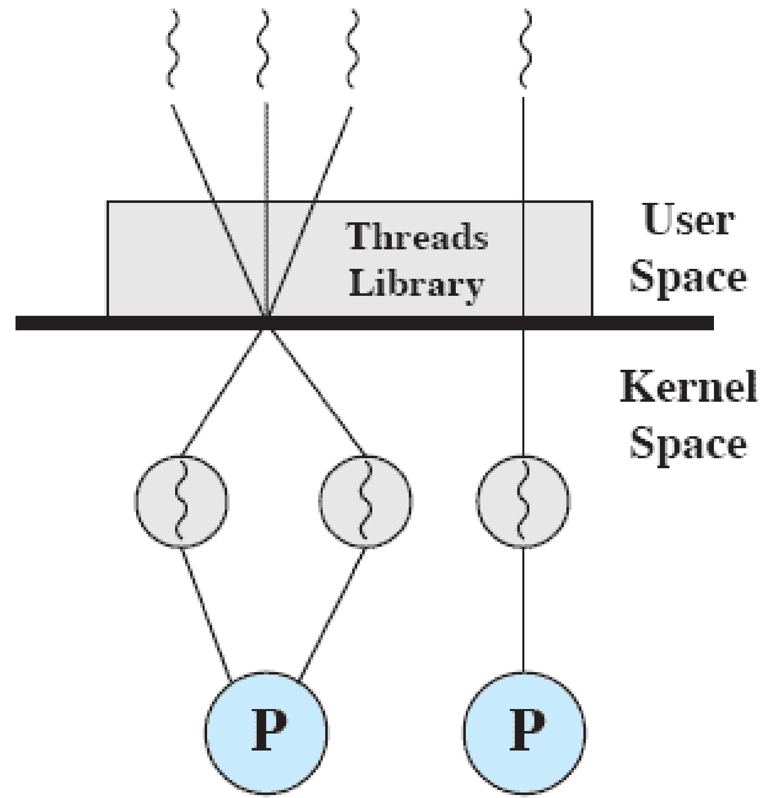
Disadvantage of KLT

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

Combined Approaches

- Thread creation done in the user space
- Bulk of scheduling and synchronization of threads by the application
- Multiple ULTs are mapped to smaller or equal number of KLTs

- Example is Solaris



(c) Combined

Relationship Between Thread and Processes

Table 4.2 Relationship Between Threads and Processes

Threads:Processes	Description	Example Systems
1:1	Each thread of execution is a unique process with its own address space and resources.	Traditional UNIX implementations
M:1	A process defines an address space and dynamic resource ownership. Multiple threads may be created and executed within that process.	Windows NT, Solaris, Linux, OS/2, OS/390, MACH
1:M	A thread may migrate from one process environment to another. This allows a thread to be easily moved among distinct systems.	Ra (Clouds), Emerald
M:N	Combines attributes of M:1 and 1:M cases.	TRIX