Operating Systems

Threads, SMP, and Microkernels

Chapter 4

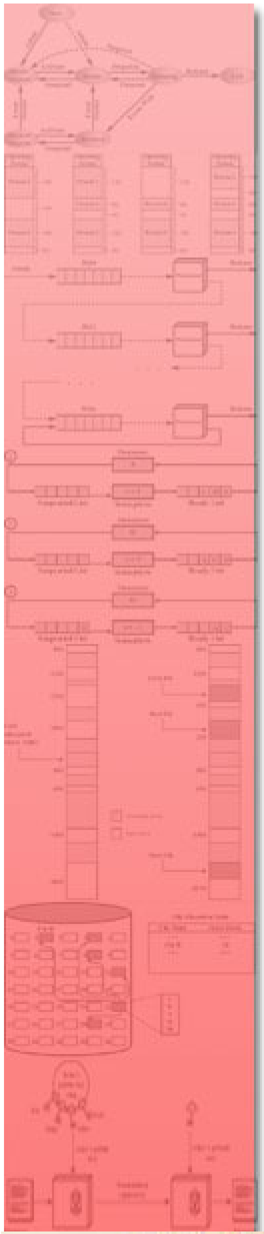
Outline

- Section 4.1 → Processes and Threads

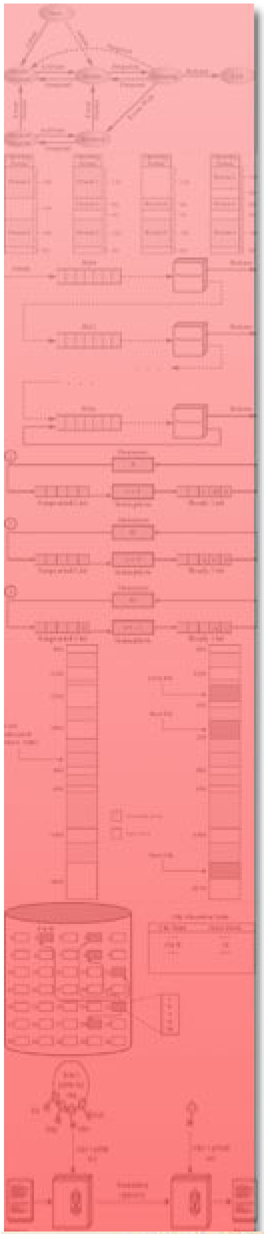
Process so Far

- *Resource ownership* - process is allocated a virtual address space to hold the process image
- *Scheduling/execution*- follows an execution path that may be interleaved with other processes
- These two characteristics are treated independently by the operating system

Process so Far



- Unit of dispatching is referred to as a *thread* or *lightweight process*
- Unit of resource of ownership is referred to as a process or task



Multithreading

- Operating system supports multiple threads of execution within a single process
  - MS-DOS supports a single thread
  - Many flavors of UNIX support multiple user processes but only support one thread per process
  - Windows 2000, Solaris, Linux, Mach, and OS/2 support multiple threads

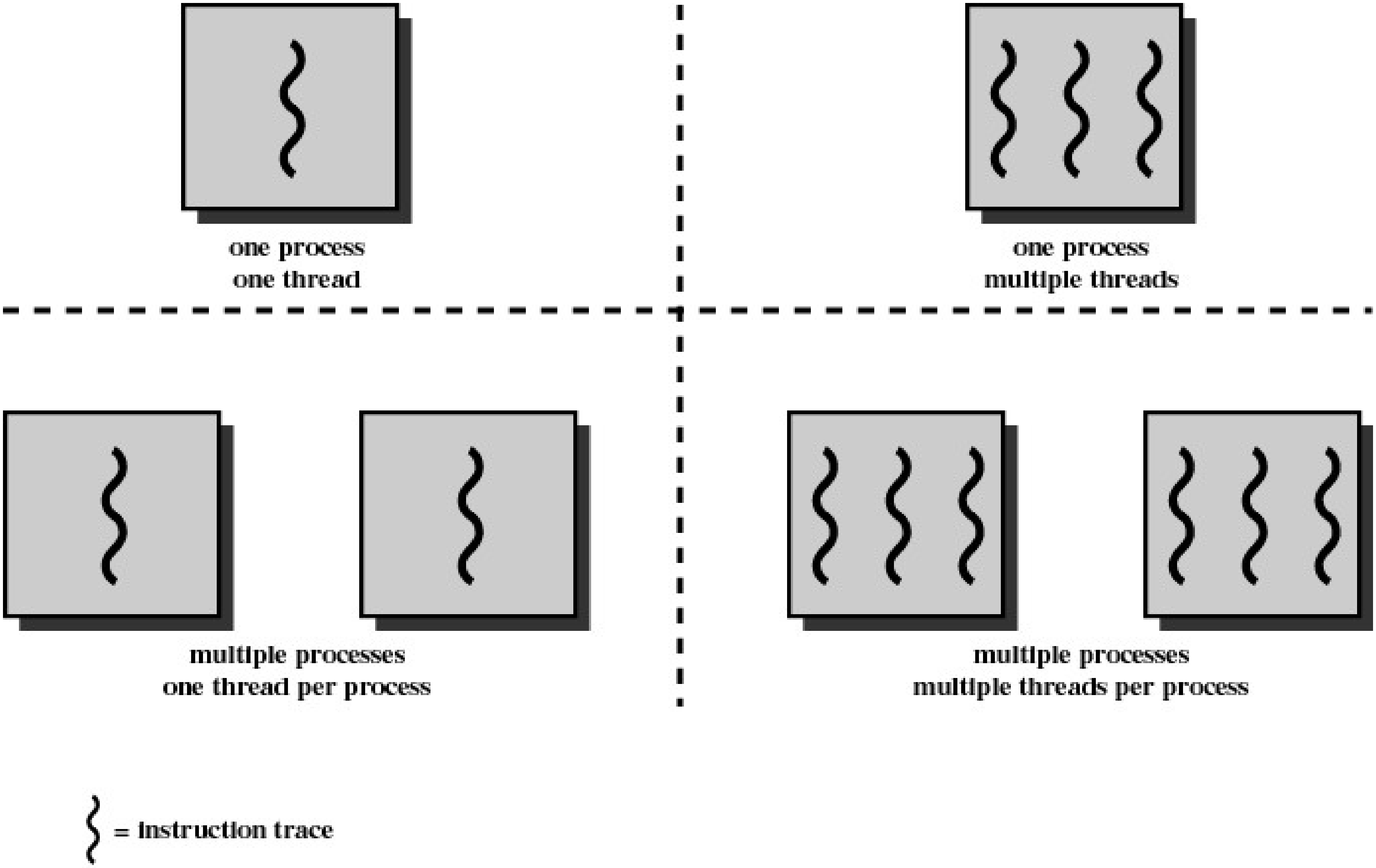


Figure 4.1 Threads and Processes [ANDE97]

Process

- In a multithreaded environment, a process is defined as the unit of resource allocation and a unit of protection
- Associated with processes
  - A virtual address space which holds the process image
  - Protected access to processors, other processes, files, and I/O resources
- Within a process, there may be one or more threads

Thread

- An execution state (running, ready, etc.)
- Saved thread context when not running
- Has 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
  - When one thread alters an item of data in memory, other threads can see the result and access this item

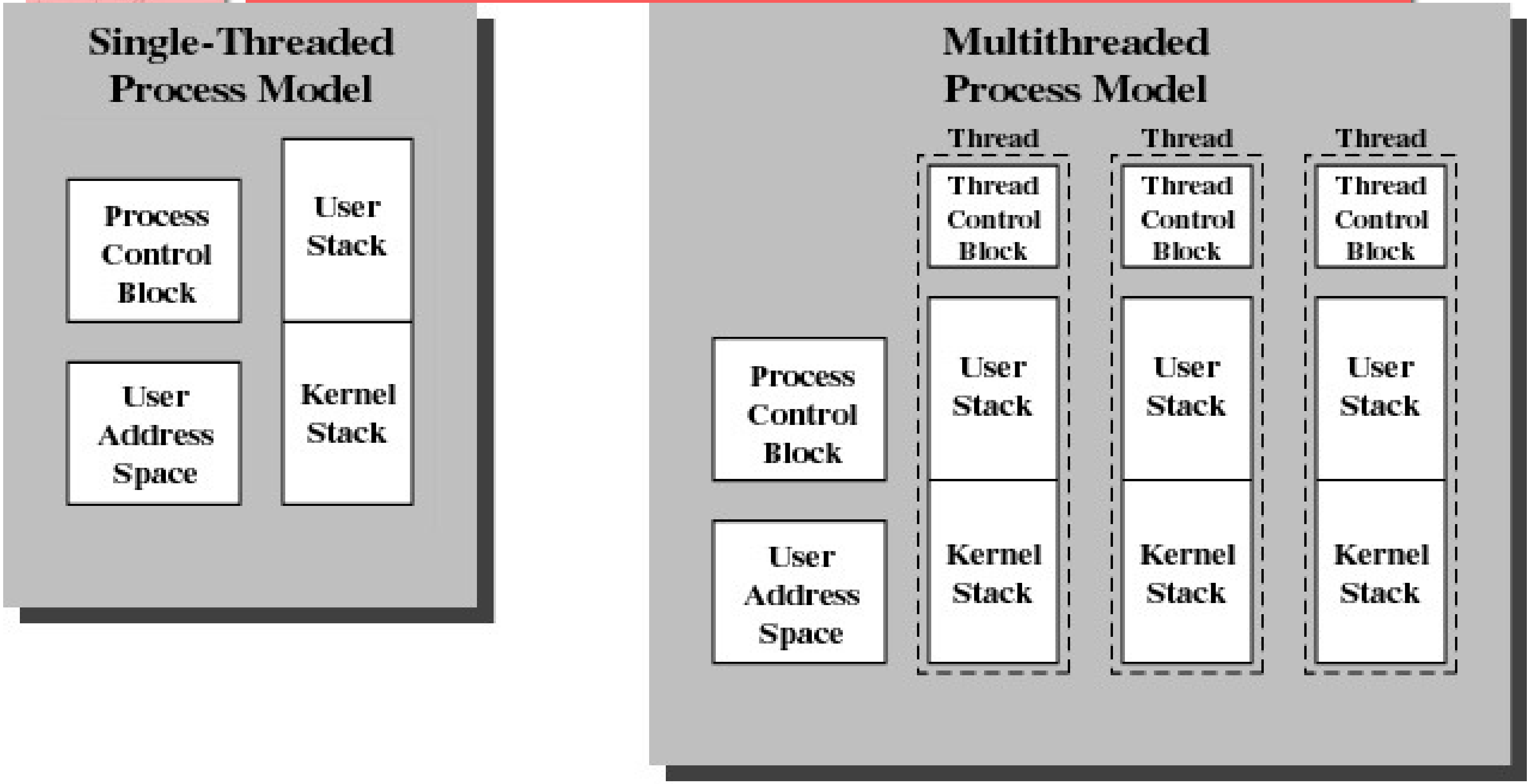
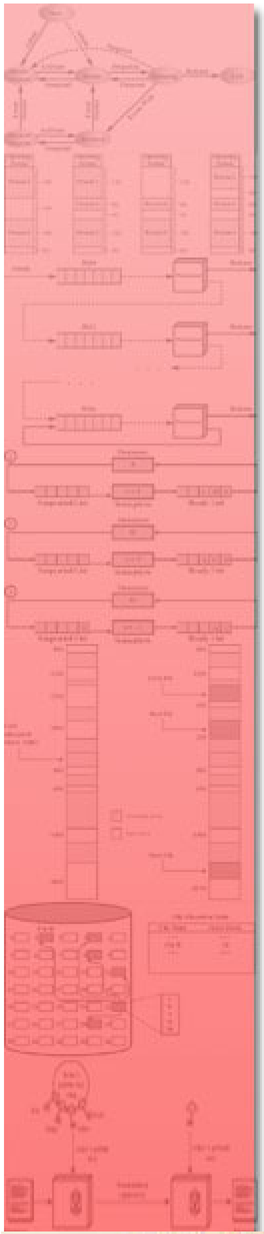


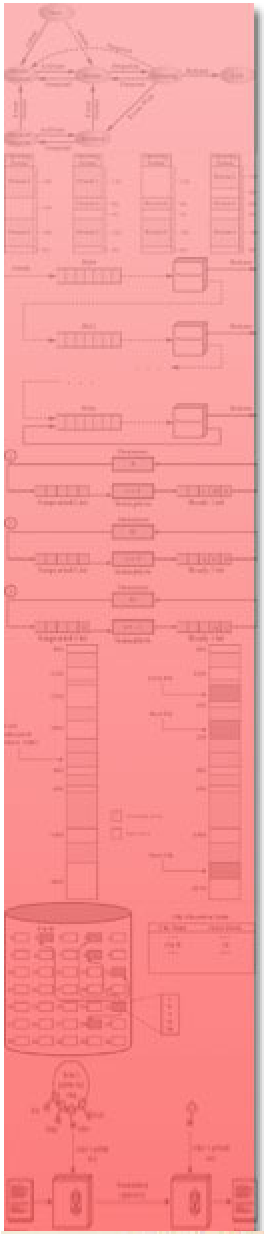
Figure 4.2 Single Threaded and Multithreaded Process Models



Benefits of Threads

- Takes less time to create a new thread than a process
- Less time to terminate a thread than a process
- Less time to switch between two threads within the same process
- Since threads within the same process share memory and files, they can communicate with each other without invoking the kernel
- Threads and sharing versus Processes and message passing

Uses of Threads in a Single-User Multiprocessing System



- Foreground to background work
  - One thread displays menus and reads user input, while another thread executes user commands and updates spreadsheet
- Asynchronous processing
  - Periodic backup
- Speed execution
  - Compute one batch of data while reading next batch from a device
- Modular program structure

Threads

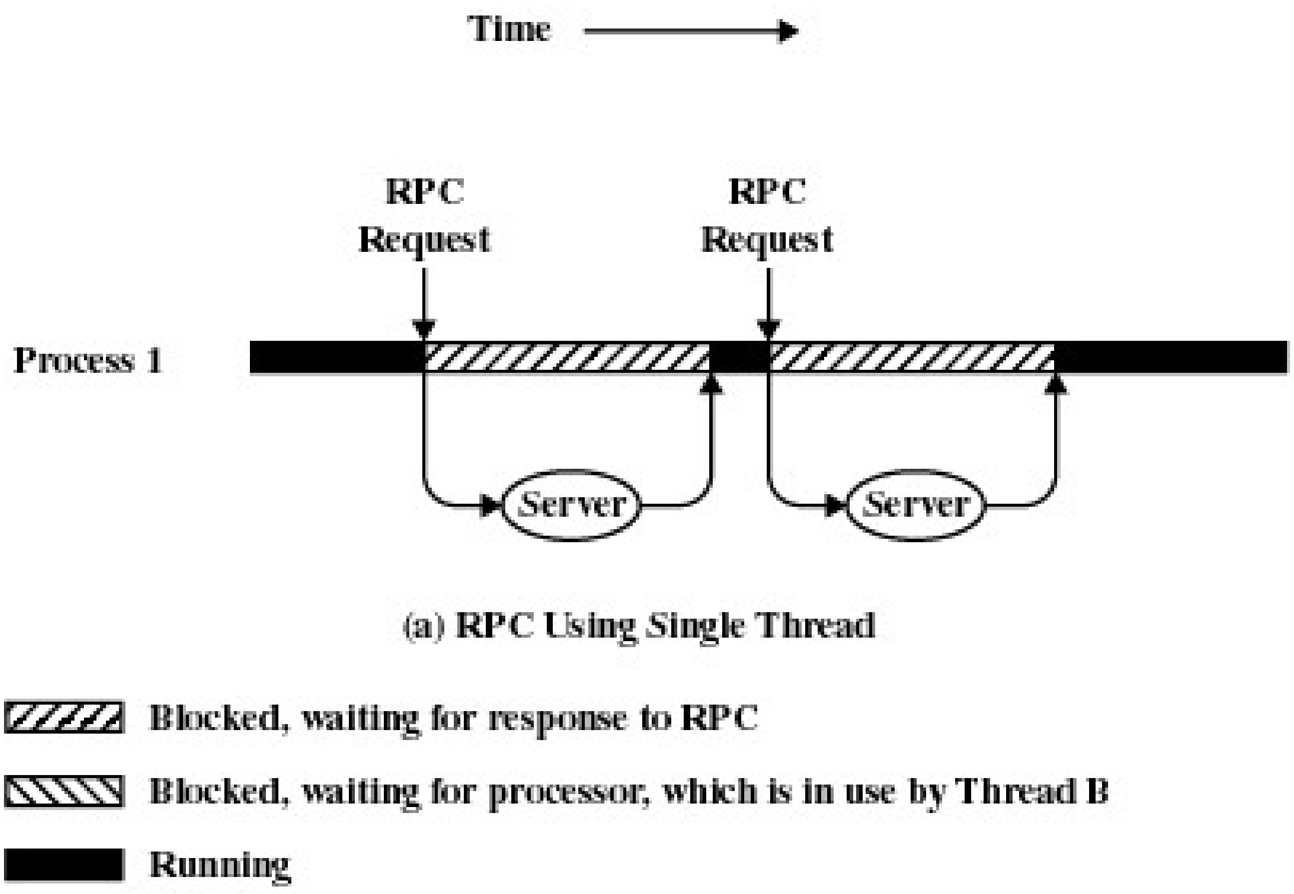
Scheduling and dispatching

- Suspending a process involves suspending all threads of the process since all threads share the same address space
- Termination of a process, terminates all threads within the process

Thread States

- Key states: Running, Ready, and Blocked
- 4 basic operations associated with a change in thread state
  - Spawn
    - Spawn another thread
  - Block
  - Unblock
  - Finish
    - Deallocate register context and stacks
- *Does the blocking of a thread result in the blocking of the entire process?*

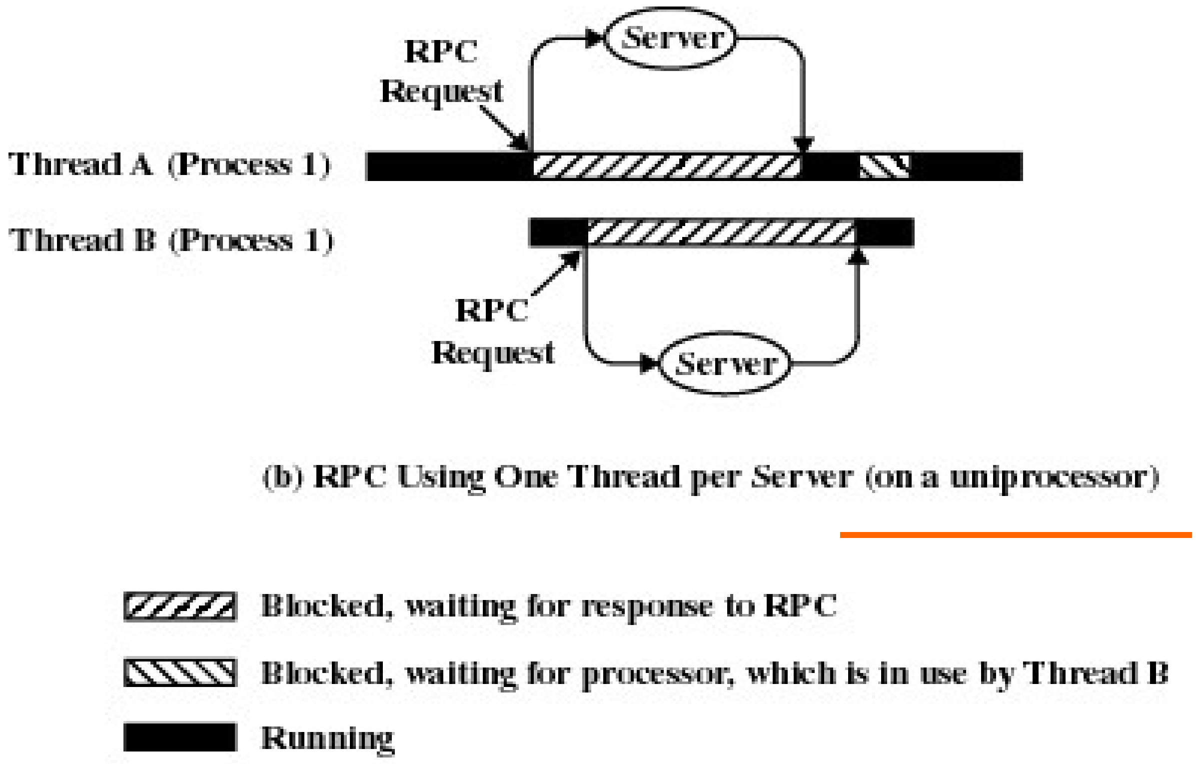
Remote Procedure Call Using Threads



© Dr. Ayman Abdel-Hamid, OS

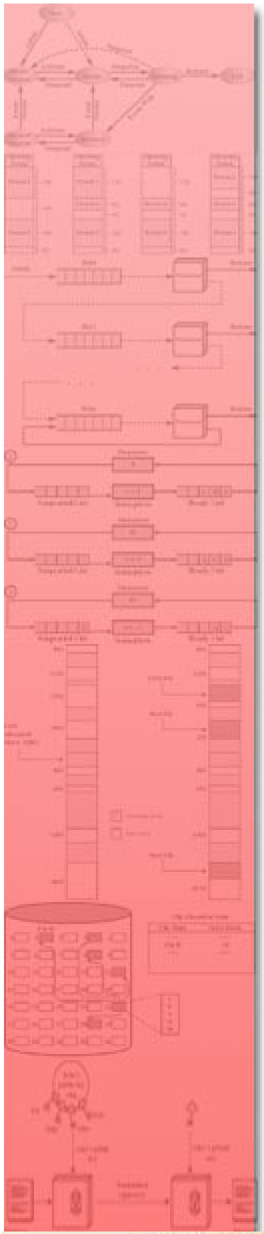
Figure 4.3 Remote Procedure Call (RPC) Using Threads

Remote Procedure Call Using Threads



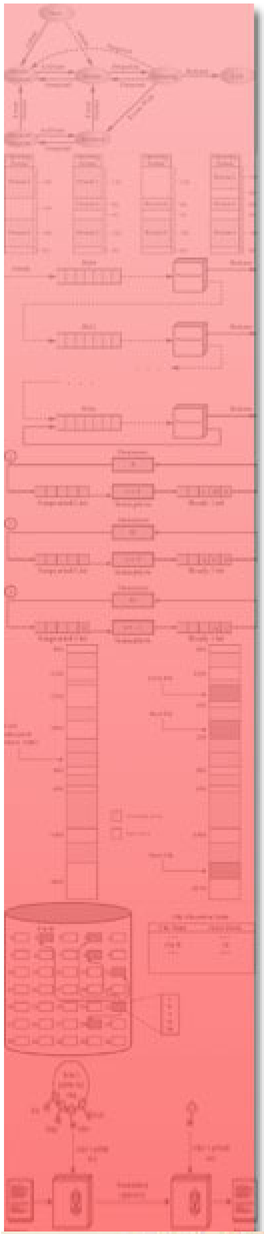
(b) RPC Using One Thread per Server (on a uniprocessor)

Figure 4.3 Remote Procedure Call (RPC) Using Threads



User-Level Threads (ULT)

- All thread management is done by the application through a threads library
- The kernel is *not aware* of the existence of threads
- Threads library creates a data structure for the new thread and passes control to one of the threads within the process using some scheduling algorithm



User-Level Threads (ULT)

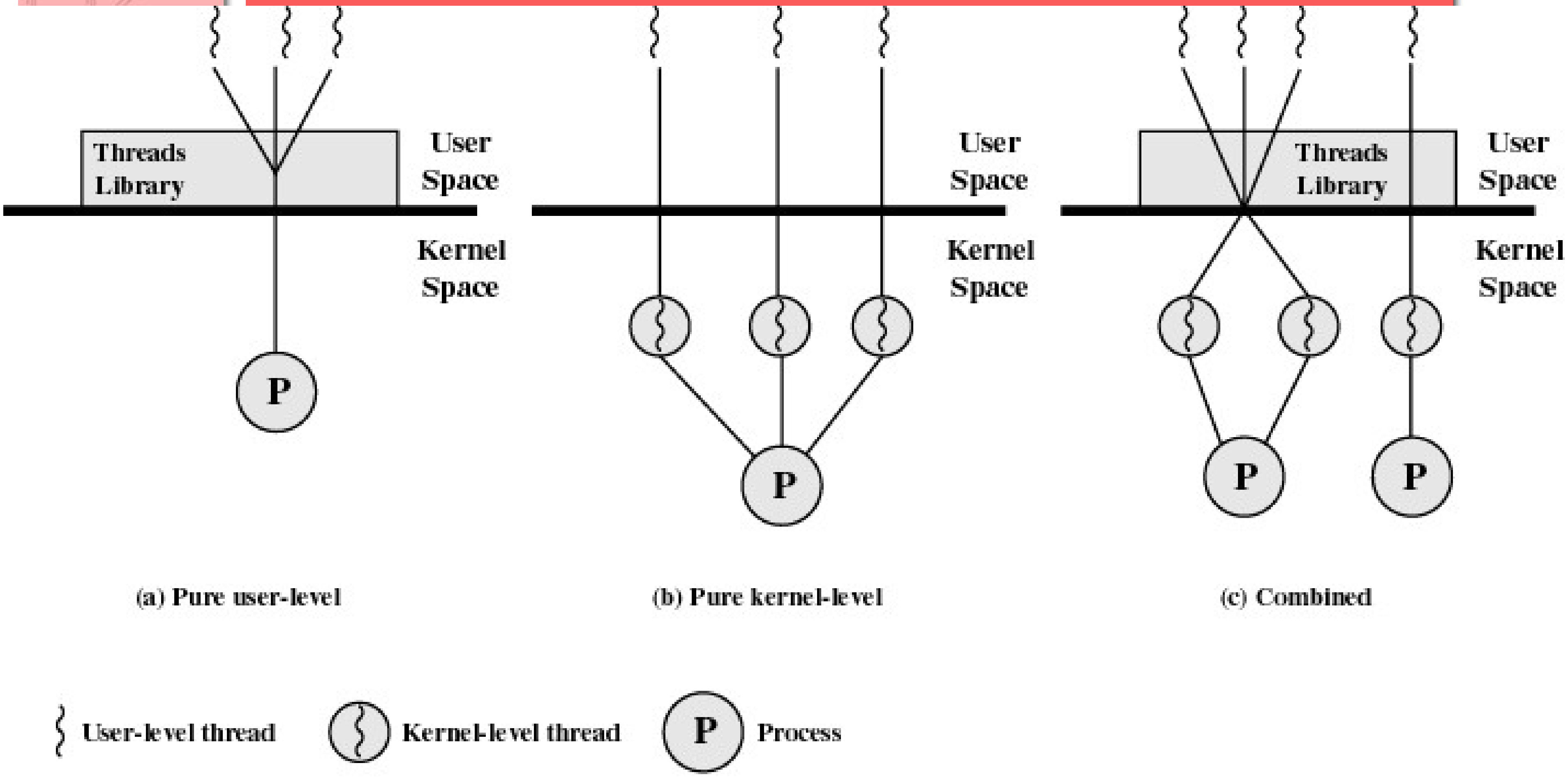
- Advantages
  - Thread switching does not require kernel mode privileges
  - Can run on any OS (no changes to underlying kernel)
  - Scheduling can be application specific
- *Disadvantages*
  - When a thread is blocked, all threads within the process are blocked
  - Cannot take advantage of multiprocessing

Kernel-Level Threads (KLT)

- W2K, Linux, and OS/2 are examples of this approach
- Kernel maintains context information for the process and the threads
- Scheduling is done on a thread basis
- Overcomes drawbacks of ULT
- *Disadvantage*: transfer of control from one thread to another within the same process requires a mode switch to the kernel (see Table 4.1)

Combined Approaches

- Example is Solaris
- Thread creation done in the user space
- Bulk of scheduling and synchronization of threads done in the user space
- Multiple ULTs from a single application are mapped onto some (smaller or equal) number of KLTs



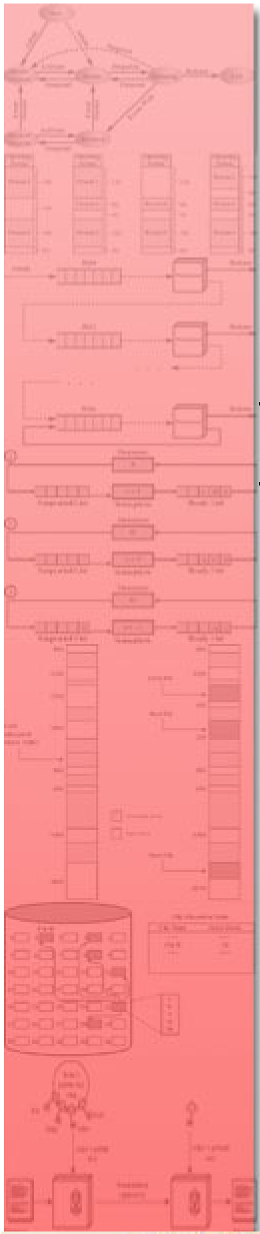
(a) Pure user-level (b) Pure kernel-level (c) Combined

Figure 4.6 User-Level and Kernel-Level Threads

Relationship Between Threads and Processes

Threads:Process	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, OS/2, OS/390, MACH

Relationship Between Threads and Processes



Threads:Process	Description	Example Systems
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:M	**Combines attributes of M:1 and 1:M cases**	**TRIX (experimental)**