Continued………

Unit -2

Threads and Processes

Objectives

- To introduce the notion of a thread — a smallest dispatchable unit of CPU utilization that forms the basis of multithreaded computer systems
- To examine issues related to multithreaded programming

1. Process

- 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

- Dispatching is referred to as a thread
- Resource of ownership is referred to as a process or task
- Protected access to processors, other processes, files, and I/O resources

2. Multithreading

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

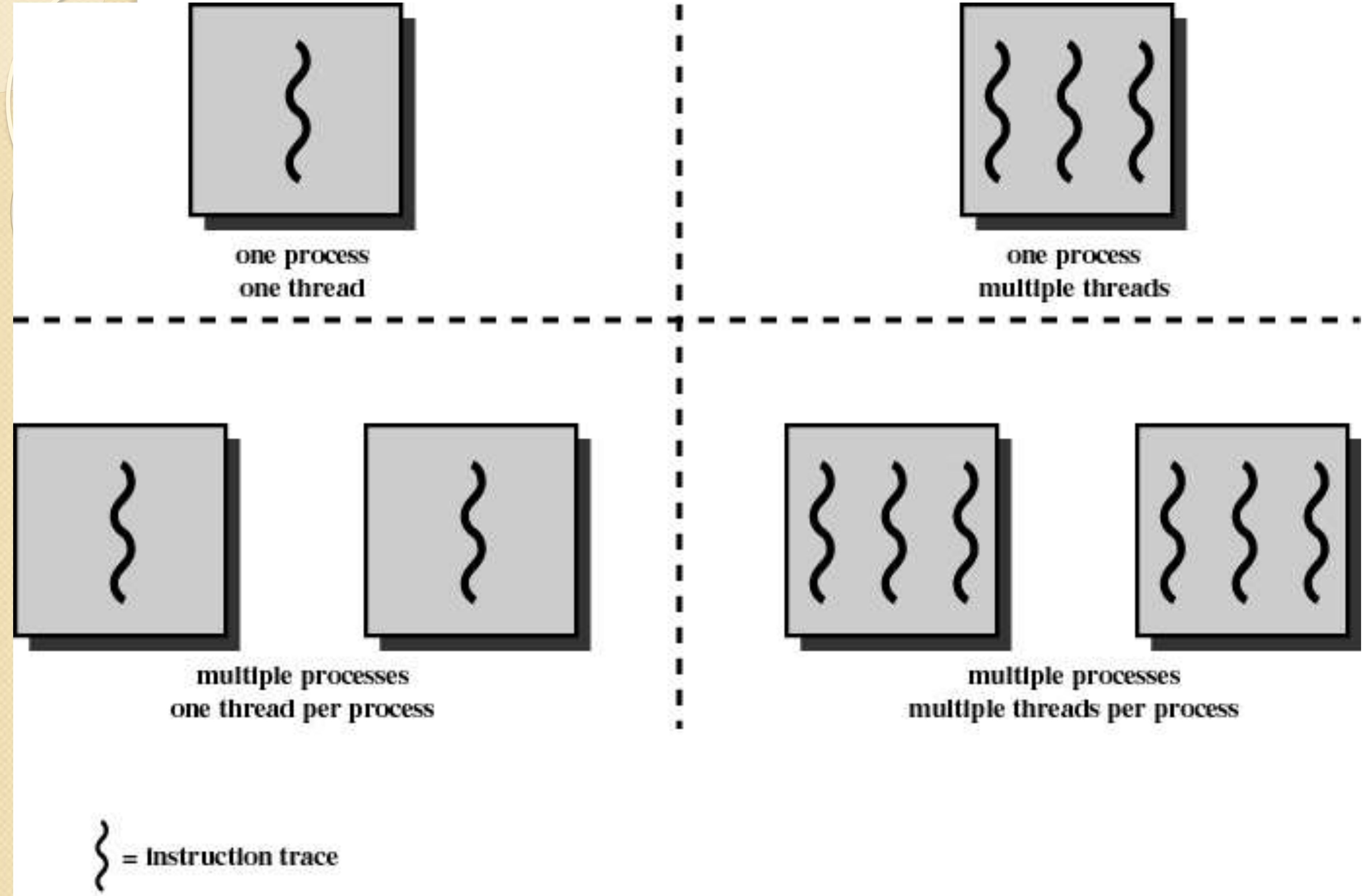
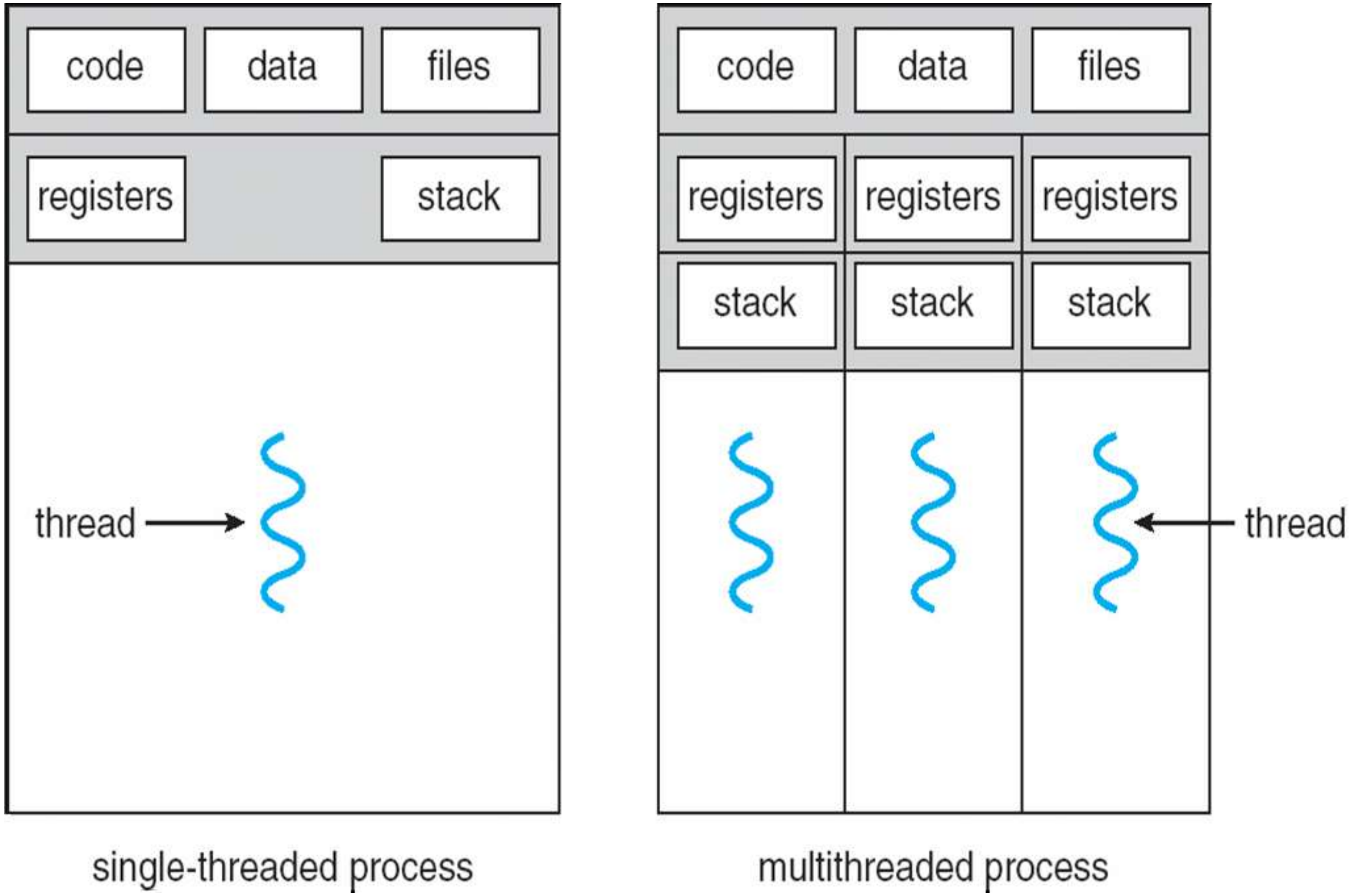


Figure 4.1 Threads and Processes [ANDE97]

Single and Multithreaded Processes



3. 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

Threads

- 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

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

Uses of Threads in a Single-User Multiprocessing System

- Foreground to background work
- Asynchronous processing
- Speed execution
- Modular program structure

4. Thread States

- States associated with a change in thread state
  - Spawn
    - Issue another thread
  - Block
  - Unblock
  - Finish
    - Deallocate register context and stacks

5. Levels of Threads

5.1 User-Level Threads

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

5.2 Kernel-Level Threads

- 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

5.3 Combined Approaches

- Example is Solaris
- Thread creation done in the user space
- Bulk of scheduling and synchronization of threads done in the user space

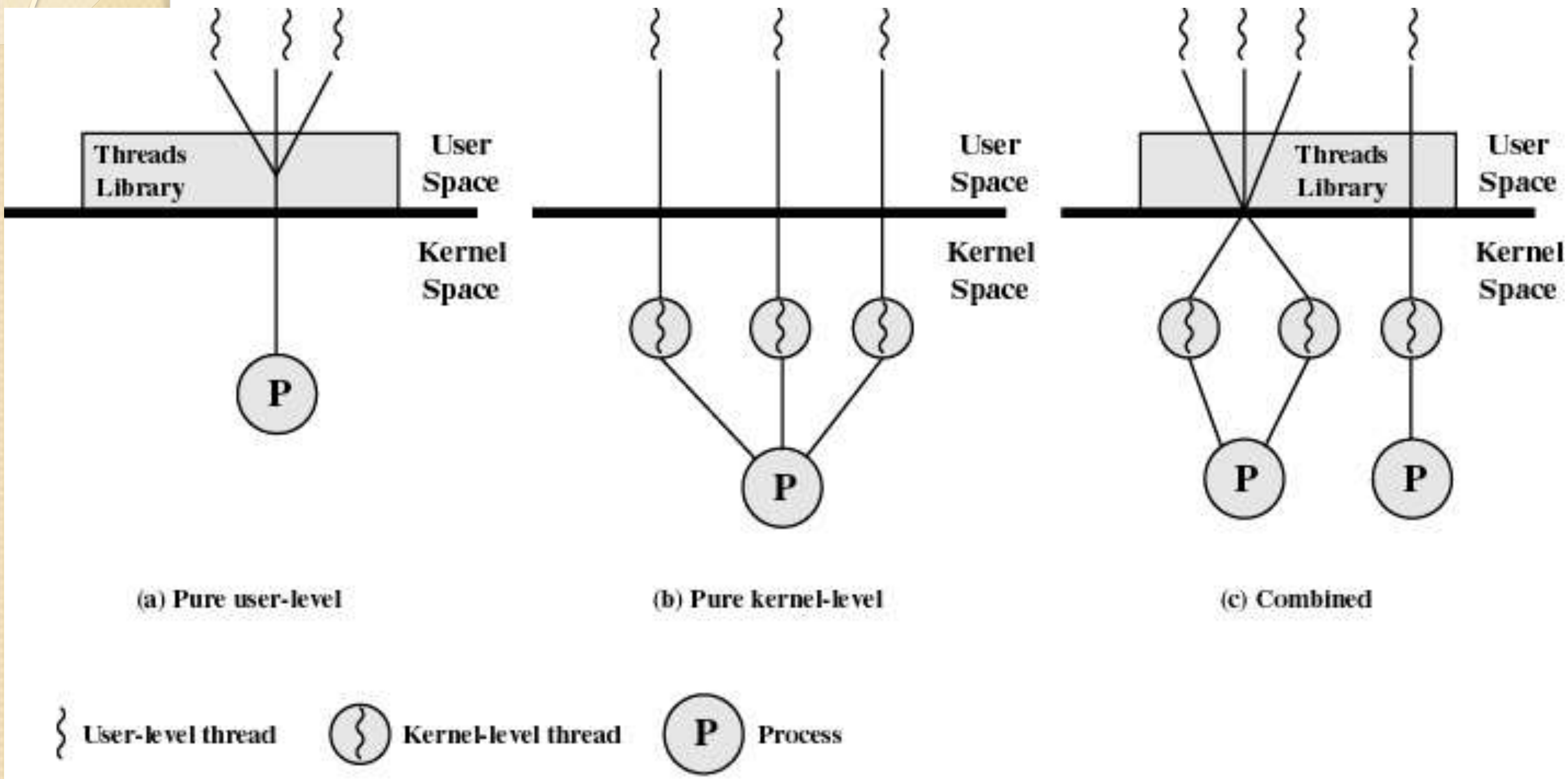


Figure 4.6 User-Level and Kernel-Level Threads

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