Operating Systems: Internals and Design Principles

Chapter 4 and 5 Threads, Concurrency, and Mutual Exclusion

Ninth Edition

By William Stallings

Processes and Threads

Resource Ownership

Process includes a virtual address space to hold the process image

- The OS performs a protection function to prevent unwanted interference between processes with respect to resources

Scheduling/Execution

Follows an execution path that may be interleaved with other processes

- A process has an execution state (Running, Ready, etc.) and a dispatching priority, and is the entity that is scheduled and dispatched by the OS

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

Single Threaded Approaches

- A single thread of execution per process, in which the concept of a thread is not recognized, is referred to as a single-threaded approach
- MS-DOS is an example

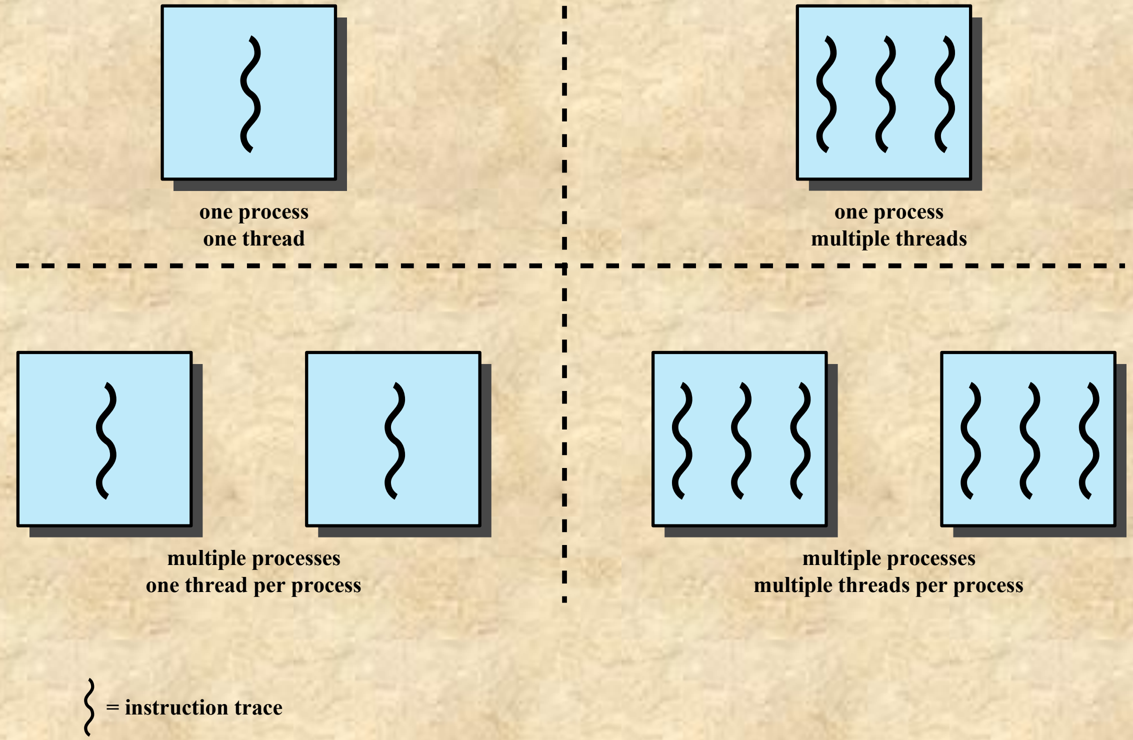

Figure 4.1 Threads and Processes

Multithreaded Approaches

- The right half of Figure 4.1 depicts multithreaded approaches
- A Java run-time environment is an example of a system of one process with multiple threads

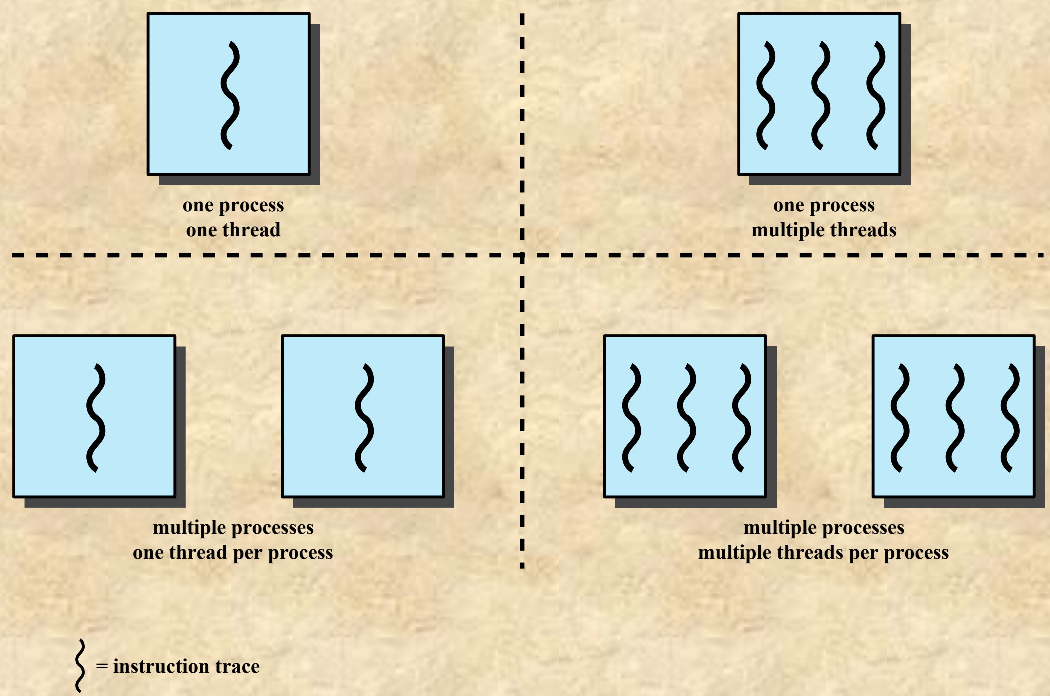

Figure 4.1 Threads and Processes

Process

- Defined in a multithreaded environment as “the unit of resource allocation and a unit of protection”
- Associated with processes:
  - A virtual address space that holds the process image
  - Protected access to:
    - Processors
    - Other processes (for interprocess communication)
    - Files
    - I/O resources (devices and channels)

One or More Threads in a Process

Each thread has:

- An execution state (Running, Ready, etc.)
- A 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 processes, shared with all other threads in that process

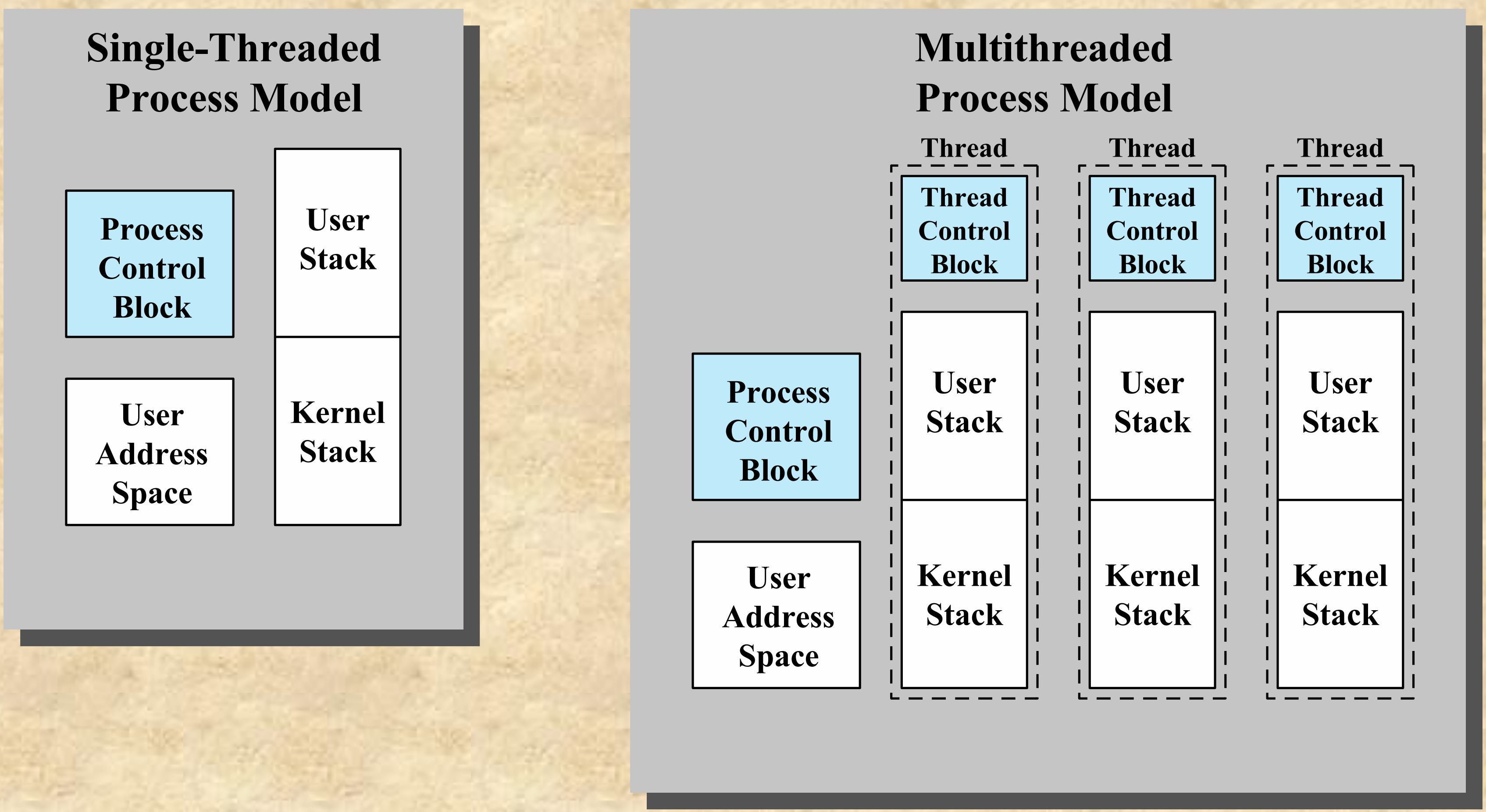

Figure 4.2 Single Threaded and Multithreaded Process Models

Key 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 than switching between processes

Threads enhance efficiency in communication between programs

Thread Use in a Single-User System

- Foreground and background work
- Asynchronous processing
- Speed of execution
- Modular program structure

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
- Block
- Unblock
- Finish

Concurrency

- Ability to run multiple processes or tasks simultaneously or in an overlapping manner.
- Concurrency and Parallelism are related concepts **but not the same.**
- **In parallel execution, processes are truly executed simultaneously.**
- **Parallelism is type of concurrency.**

Difficulties of Concurrency

- Sharing of global resources
- Difficult for the OS to manage the allocation of resources optimally
- Difficult to locate programming errors as results are not deterministic and reproducible

Race Condition

- Occurs when multiple processes or threads read and write data items
- The final result depends on the order of execution
  - The “loser” of the race is the process that updates last and will determine the final value of the variable

Resource Competition

- Concurrent processes come into conflict when they are competing for use of the same resource
  - For example: I/O devices, memory, processor time, clock

In the case of competing processes three control problems must be faced:

- **The need for mutual exclusion**
- **Deadlock**
- **Starvation**

How to ensure mutual exclusion?

- Hardware support
  - Interrupt disabling
  - Atomic Instructions
    - Compare&Swap instruction.
    - Exchange Instruction.
- OS and programming language support
  - Mutex
  - Semaphore
  - Monitor

Mutual Exclusion: Hardware Support

- **Interrupt Disabling**
  - In a uniprocessor system, concurrent processes cannot have overlapped execution; they can only be interleaved
  - A process will continue to run until it invokes an OS service or until it is interrupted
  - Therefore, to guarantee mutual exclusion, it is sufficient to prevent a process from being interrupted
  - This capability can be provided in the form of primitives defined by the OS kernel for disabling and enabling interrupts

- **Disadvantages:**
  - The efficiency of execution could be noticeably degraded because the processor is limited in its ability to interleave processes
  - This approach will not work in a multiprocessor architecture

Mutual Exclusion: Hardware Support

- Compare&Swap Instruction
  - Also called a "compare and exchange instruction"
  - A **compare** is made between a memory value and a test value
  - If the values are the same a **swap** occurs
  - Carried out atomically (not subject to interruption)

OS and language support

- Mutex
- Semaphore
- Monitor
- Condition variables

Table 5.3

Common Concurrency Mechanisms

Semaphore	An integer value used for signaling among processes. Only three operations may be performed on a semaphore, all of which are atomic: initialize, decrement, and increment. The decrement operation may result in the blocking of a process, and the increment operation may result in the unblocking of a process. Also known as a **counting semaphore** or a **general semaphore**
Binary Semaphore	A semaphore that takes on only the values 0 and 1.
Mutex	Similar to a binary semaphore. A key difference between the two is that the process that locks the mutex (sets the value to zero) must be the one to unlock it (sets the value to 1).
Condition Variable	A data type that is used to block a process or thread until a particular condition is true.
Monitor	A programming language construct that encapsulates variables, access procedures and initialization code within an abstract data type. The monitor's variable may only be accessed via its access procedures and only one process may be actively accessing the monitor at any one time. The access procedures are *critical sections*. A monitor may have a queue of processes that are waiting to access it.
Event Flags	A memory word used as a synchronization mechanism. Application code may associate a different event with each bit in a flag. A thread can wait for either a single event or a combination of events by checking one or multiple bits in the corresponding flag. The thread is blocked until all of the required bits are set (AND) or until at least one of the bits is set (OR).
Mailboxes/Messages	A means for two processes to exchange information and that may be used for synchronization.
Spinlocks	Mutual exclusion mechanism in which a process executes in an infinite loop waiting for the value of a lock variable to indicate availability.

Table 5.1

Some Key Terms Related to Concurrency

atomic operation	A function or action implemented as a sequence of one or more instructions that appears to be indivisible; that is, no other process can see an intermediate state or interrupt the operation. The sequence of instruction is guaranteed to execute as a group, or not execute at all, having no visible effect on system state. Atomicity guarantees isolation from concurrent processes.
critical section	A section of code within a process that requires access to shared resources and that must not be executed while another process is in a corresponding section of code.
deadlock	A situation in which two or more processes are unable to proceed because each is waiting for one of the others to do something.
livelock	A situation in which two or more processes continuously change their states in response to changes in the other process(es) without doing any useful work.
mutual exclusion	The requirement that when one process is in a critical section that accesses shared resources, no other process may be in a critical section that accesses any of those shared resources.
race condition	A situation in which multiple threads or processes read and write a shared data item and the final result depends on the relative timing of their execution.
starvation	A situation in which a runnable process is overlooked indefinitely by the scheduler; although it is able to proceed, it is never chosen.