Lecture 6: Kernel Structures and Threading

Summary

1. Computer system overview (Chapter 1)
2. Basic of virtual memory; i.e. segmentation and paging (Chapter 7 and part of 8)
3. Process (Chapter 3)
4. Mutual Exclusion and Synchronization (Chapter 5 section 1-4)
   - Conditions for race avoidance.
   - Strict alternation.
   - Semaphores.
   - Producers and Consumers problem.
   - Hardware support for mutual exclusion.
   - Monitors.

User Mode vs. Kernel Mode

1. Kernel architectures
   - Monolithic kernel (Unix, Windows 9X, MS-DOS)
   - Layered kernel (THE, MULTICS) (see Figure 4.10)
   - Microkernel (Symbian, Singularity, Minix)
   - Hybrid Kernel (Windows NT kernel)
2. Transition from user mode to kernel mode (see Figure 3.17)
   - System calls.
   - Exceptions.
   - Interrupts.

Threads

1. Processes
   - Resource ownership.
   - Scheduling/execution.
2. Thread: An execution path within a process (see Figure 4.1)
   - MS-DOS (single-process/single-thread) (see Figure 4.2).
   - Early flavor of UNIXs (muti-process/single-thread).
   - Windows, Solaris (multi-process/multi-thread).
3. Distinguishing between process and thread

- Process: Unit of resource allocation and protection.
- Thread: Execution unit.
  - (a) Execution state.
  - (b) Execution context (PC, stack, per-thread storage for local variable, access to resources).

4. Benefits of Threading

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

5. Thread states

   (a) Spawn.

   (b) Block.

   (c) Unblock.

   (d) Finish.

6. Synchronization and mutual exclusion (same as process)

7. User-level versus kernel-level threads (see Figure 4.6)

   - User-level threads: Threads are not recognized by kernel.
     - Thread switching requires no kernel intervention.
     - Scheduling policy can be application specific.
     - Portable, threading library provides as utilities.
     - Blocking call in a thread can block all threads in the same processes.
     - Multithreaded application cannot take advantage of multiprocessing.
   - Kernel-level threads: User-level threads are mapped to kernel threads.
     - Support scheduling of threads in the same process in multiprocessor environment.
     - A blocked threads does not cause other threads in the same process to block.
     - Scheduling is done entirely in the kernel.
   - Comparing thread management schemes between Solaris and Linux (see Figure 4.15).

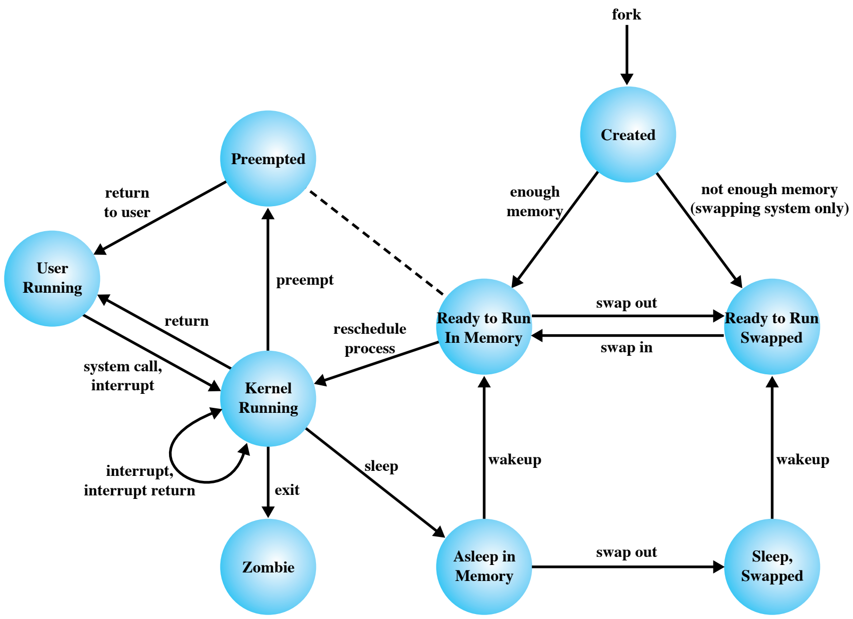

Figure 3.17 UNIX Process State Transition Diagram

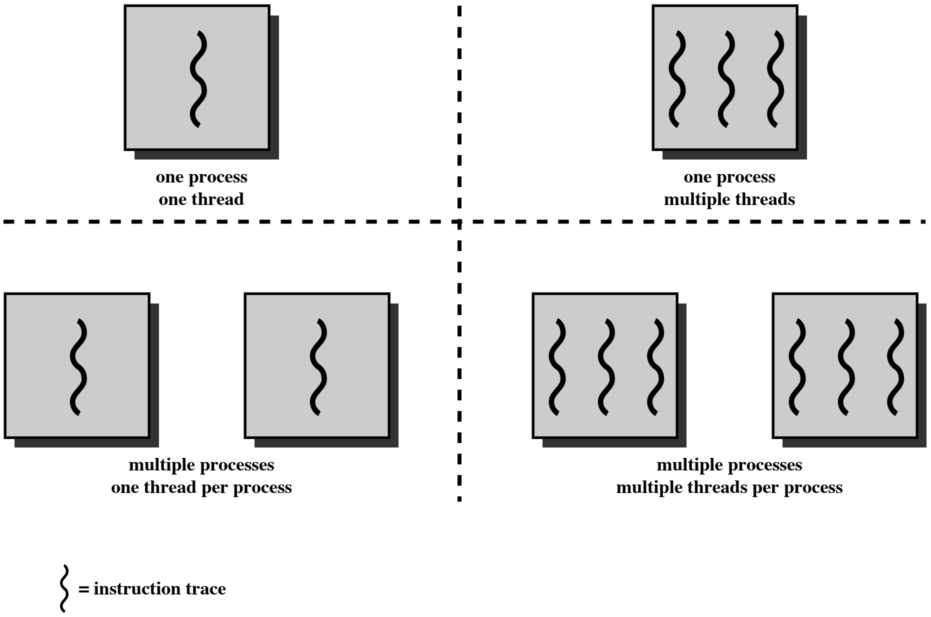
one process one thread
one process multiple threads
multiple processes one thread per process
multiple processes multiple threads per process
= instruction trace

Figure 4.1 Threads and Processes [ANDE97]

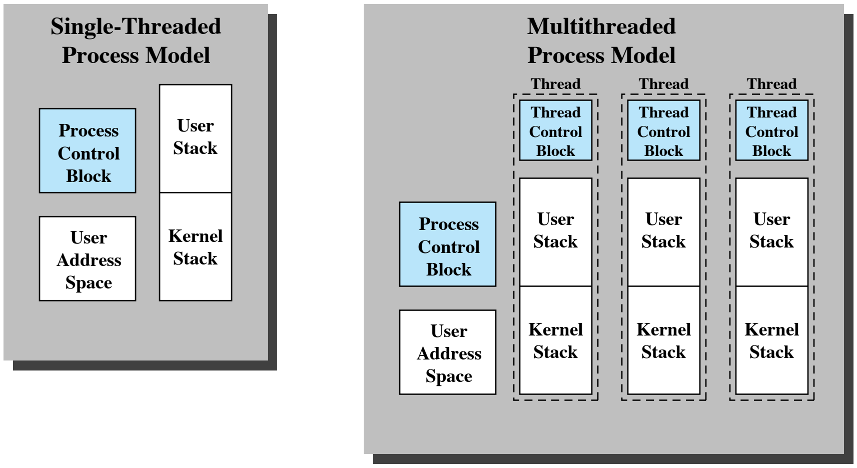

Figure 4.2 Single Threaded and Multithreaded Process Models

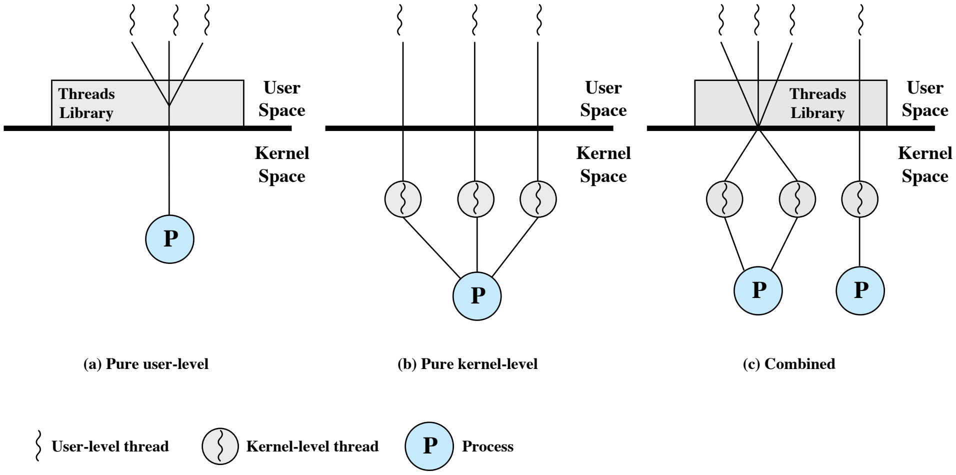

Figure 4.6 User-Level and Kernel-Level Threads

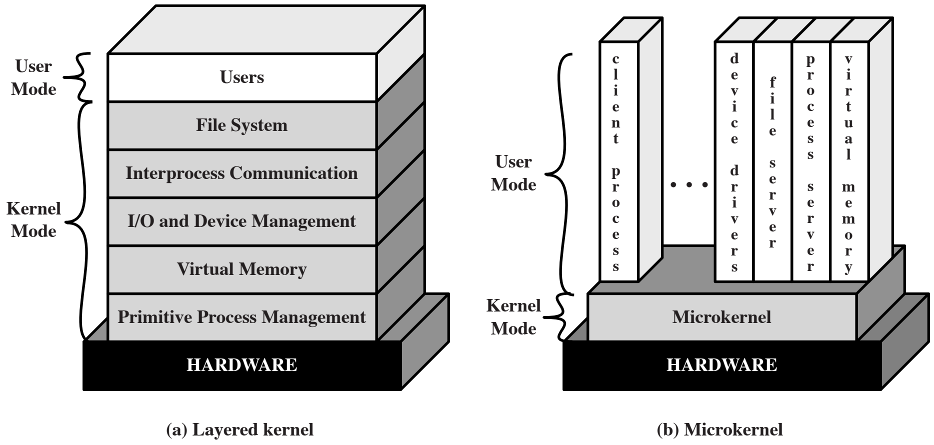

Figure 4.10 Kernel Architecture

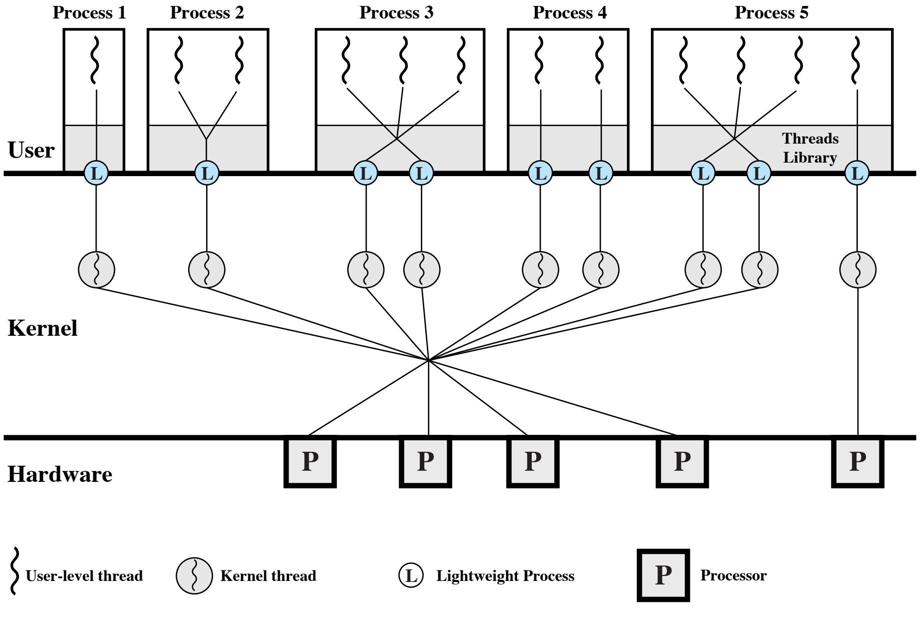

Figure 4.15 Solaris Multithreaded Architecture Example