

The following are topics that should be covered in addition to the Fall 2005 topics:

➤ **Threads**

→ **Many-to-one:** Process creates many user-level threads that are mapped to one kernel thread

- **Advantages:**

- **Fast Context Switching:** It is more efficient to perform a context switch at the user level than at the kernel level
- **Easier to implement**

- **Disadvantages:**

- **Blocking:** Since all threads are governed by one kernel thread, if one thread calls a blocking system call, all threads will be blocked
- **Non-parallelism:** Since there is only 1 kernel thread, program cannot be executed on multiple processors simultaneously

- **One-To-One:** Each user thread is mapped to exactly one kernel thread

- **Advantages:**

- **Parallelism:** 1 program can be executed across several processors, each controlling a different thread

- **Disadvantages:**

- **Kernel Thread Overhead:** Since each thread requires a kernel thread, programs with a lot of threads will begin experiencing overhead due to thread creation and context switching.

- **Many-To-Many:** Many user threads are created, which are controlled (pass around) by many kernel threads

- **Advantages:**

- No limitation to the number of threads that can be created

➤ **Scheduling:**

→ Circumstances for Preemptive Scheduling:

- Process switches from the running state to waiting state (ex, waiting for an IO request)
- Process switches from running state to ready state (ex, process's time slice expires)
- Process switches from waiting state to ready state (ex, IO request completed)
- Process terminates

→ Key Terms:

- CPU Utilization
- Throughput
- Turnaround Time
- Waiting Time
- Response Time

→ Types of schedules:

- First-Come, First-Serve
- Shortest Job First
- Priority Scheduling
- Round-Robin Scheduling
- Multilevel Queue Scheduling

- Multilevel Feedback Queue Scheduling
- **Critical-Section Problem:**
 - Must satisfy following 3 requirements:
 - **Mutual Exclusion:** Only 1 process can be in the critical section at a given time
 - **Progress:** Only the processes that are waiting to enter the critical section will decide which process is allowed to enter
 - **Example:** Process id's are entered into a queue as they wait for the critical section to become available. They are given access on a first-come, first-serve basis
 - **Non-Example:** OS performs a context switch if critical section is not available (ie, process goes to sleep). Periodically, process wakes back up and checks if critical section is available again
 - **Bounded Waiting:** Processes will always enter critical section after some finite period of time
 - **Synchronization Hardware:**
 - **Test-And-Set**
 - **Swap**
 - **Synchronization Constructs:**
 - Semaphores
 - Mutexes
 - Monitors
 - Critical Regions