**Chapter 4: Threads**

**4.1 Overview: Benefits of Multi-Programming**

* Multi-programming increases **CPU utilization** by organizing jobs so that the CPU always has one to execute.
* **Concurrency** is achieved through thread-level parallelism, enabling processes to make progress on multiple tasks without waiting for others to complete.

**4.2 Multicore Programming**

* Multicore systems require efficient handling of parallel tasks.
* Key concepts include **task parallelism** and **data parallelism**, which describe distributing tasks or data among multiple cores.
* Challenges in multicore programming: **load balancing, data dependency**, **race conditions**, and **scalability**.

**4.3 Multithreading Models**

* **Many-to-One Model**: Multiple user threads are mapped to a single kernel thread, limiting parallelism.
* **One-to-One Model**: Each user thread maps to a kernel thread, allowing parallel execution but consuming more system resources.
* **Many-to-Many Model**: Multiplexes user threads across a smaller or equal number of kernel threads, balancing efficiency and resource usage.
* Be able to **draw** and **interpret diagrams** of these models to illustrate thread management and limitations.

**Chapter 5: CPU Scheduling (Sections 5.1 - 5.4)**

**5.1 Basic Concepts**

* **CPU-I/O Burst Cycle**: The alternating phases of CPU and I/O activity in processes.
* **Preemptive vs. Non-Preemptive Scheduling**:
  + *Preemptive*: The OS can interrupt a process, allowing another to execute.
  + *Non-Preemptive*: Once a process starts, it runs to completion or until it voluntarily yields.

**5.2 Scheduling Criteria**

* **CPU Utilization**: Maximizing CPU usage.
* **Throughput**: Number of processes completed per time unit.
* **Turnaround Time**: Time taken from process submission to completion.
* **Waiting Time**: Time a process spends in the ready queue.
* **Response Time**: Time from request submission to the first response.

**5.3 Scheduling Algorithms**

* **First-Come, First-Served (FCFS)**: Non-preemptive; simple but can cause the *convoy effect*.
* **Shortest Job Next (SJN)**: Non-preemptive; minimizes waiting time but requires accurate burst time estimation.
* **Priority Scheduling**: Preemptive or non-preemptive; assigns priorities to processes, risking starvation.
* **Round Robin (RR)**: Preemptive; each process is given a small time slice, reducing response time but increasing context switches.
* **Multilevel Queue** and **Multilevel Feedback Queue**: Different queues for different priority levels.
* Practice **Gantt charts** for visualizing scheduling order, and know how to calculate metrics like **average wait time** and **throughput**.

**5.4 Multiple-Processor Scheduling**

* **Asymmetric Multiprocessing (AMP)**: Only one processor accesses the OS data structures, simplifying scheduling.
* **Symmetric Multiprocessing (SMP)**: Each processor is self-scheduling; may lead to load imbalances.
* **Processor Affinity**: Binding processes to specific processors to reduce cache issues.
* **Load Balancing**: Redistributing tasks to prevent certain processors from being overloaded.

**Chapter 6: Synchronization (Sections 6.1 - 6.6)**

**6.1 Background**

* Concurrency introduces challenges like **race conditions**, where multiple processes access shared data simultaneously, potentially causing data inconsistencies.

**6.2 The Critical-Section Problem**

* **Critical Section**: Code segment where shared resources are accessed.
* Solutions must ensure **mutual exclusion**, **progress**, and **bounded waiting**.

**6.3 Peterson's Solution**

* An algorithm for two processes, using a flag and a turn variable to achieve mutual exclusion. (This may be optional, depending on your instructor.)

**6.5 Semaphores**

* **Counting Semaphores**: Used for managing a pool of resources.
* **Binary Semaphores** (Mutex): Simplified version, acting as a lock for mutual exclusion.
* Common semaphore functions: wait() (P operation) and signal() (V operation).

**6.6 Classic Problems of Synchronization**

* **Bounded Buffer (Producer-Consumer)**: Ensures producers do not add to a full buffer and consumers do not remove from an empty buffer.
* **Readers-Writers Problem**: Ensures multiple readers can access data simultaneously, but writers have exclusive access.
* **Dining Philosophers Problem**: Models resource sharing and deadlock avoidance.

**Chapter 7: Deadlocks (Sections 7.1 - 7.7)**

**7.1 System Model**

* Deadlocks occur when processes are in a state of permanent waiting due to resource contention.

**7.2 Deadlock Characterization**

* Four **Necessary Conditions**:
  + **Mutual Exclusion**: Only one process can use a resource at a time.
  + **Hold and Wait**: Processes holding resources can request additional ones.
  + **No Preemption**: Resources cannot be forcibly taken.
  + **Circular Wait**: Processes form a cycle, each waiting on the next.
* **Resource-Allocation Graphs**: Visual representations to identify potential deadlock cycles.

**7.3 Methods of Handling Deadlocks**

* **Prevention**: Altering conditions to prevent deadlocks (e.g., preventing hold and wait).
* **Avoidance**: Requires prior knowledge of resource requests (e.g., Banker’s Algorithm).
* **Detection and Recovery**: Allows deadlock and then takes action to recover.

**7.4 Deadlock Prevention**

* Modifying resource allocation strategies to break at least one of the necessary conditions for deadlock, such as enforcing no hold and wait or circular wait.

**7.5 Deadlock Avoidance**

* **Banker’s Algorithm**: Allocates resources safely by ensuring a system’s state remains safe after each request.
* Note: Skip **Resource-Allocation-Graph Algorithm** in Section 7.5.2.

**7.6 Deadlock Detection**

* Detection methods check for deadlocks by identifying resource allocation cycles or unmet requests.
* Continues coverage of the Banker’s Algorithm as a method to detect unsafe states.

**7.7 Recovery from Deadlock**

* **Process Termination**: Stopping processes to break the deadlock.
* **Resource Preemption**: Temporarily removing resources from processes to resolve deadlock.