26 Lecture - CS410

Important Subjective

Question 1: What is a thread?

Answer: A thread is the smallest unit of execution within a process. It represents an independent sequence of instructions that can run concurrently with other threads, sharing the same resources such as memory and files.

Question 2: What is thread synchronization?

Answer: Thread synchronization is the coordination of multiple threads to ensure orderly execution and proper data sharing. It prevents race conditions and ensures data integrity when multiple threads access shared resources simultaneously.

Question 3: Explain the concept of a race condition.

Answer: A race condition occurs when two or more threads access shared data concurrently, leading to unexpected and potentially incorrect behavior due to the unpredictable order of execution. This can result in data corruption or inconsistent results.

Question 4: What is a critical section?

Answer: A critical section is a portion of code that must be executed by only one thread at a time to prevent race conditions. Synchronization mechanisms are used to ensure that only one thread can access the critical section at any given moment.

Question 5: What is the purpose of using locks in thread synchronization?

Answer: Locks are used to control access to shared resources by allowing only one thread to acquire the lock and access the resource at a time. They prevent multiple threads from concurrently modifying the resource, ensuring data consistency.

Question 6: Explain the difference between a mutex and a semaphore.

Answer: A mutex is a synchronization primitive that ensures mutual exclusion by allowing only one thread to access a resource at a time. A semaphore, on the other hand, can allow a specified number of threads to access a resource concurrently, based on its count.

Question 7: How does deadlock occur in multithreaded programming?

Answer: Deadlock occurs when two or more threads are blocked, each waiting for a resource that is held by another thread in the set. This leads to a situation where no thread can proceed, resulting in a standstill.

Question 8: What is thread contention, and how can it impact performance?

Answer: Thread contention refers to multiple threads competing for the same resource, leading to delays and inefficiencies. Excessive thread contention can reduce performance as threads spend more time waiting for access to resources rather than doing useful work.

Question 9: How does a Read-Write Lock differ from a regular lock (mutex)?

Answer: A Read-Write Lock allows multiple threads to read a shared resource simultaneously while ensuring exclusive access for writing. This is more efficient for scenarios where reading is more frequent than writing, as it reduces contention.

Question 10: Why is it important to carefully design the order in which locks are acquired to prevent deadlocks?

Answer: The order in which locks are acquired is crucial to preventing deadlocks. If threads acquire locks in different orders, it can lead to a situation where each thread is waiting for a lock held by another, resulting in a deadlock where none of the threads can make progress. Proper lock ordering helps avoid such scenarios.