25 Lecture - CS410

Important Subjective

Question 1: What is a thread, and how does it differ from a process?

Answer: A thread is a basic unit of execution within a process. Unlike processes, threads within the same process share the same memory space, making communication and data sharing more efficient.

Question 2: How does thread synchronization contribute to program correctness?

Answer: Thread synchronization ensures that multiple threads interact with shared resources in an orderly manner, preventing conflicts and race conditions, thus maintaining program correctness.

Question 3: Describe the concept of thread priority. Why might it be important?

Answer: Thread priority determines the order in which threads are scheduled for execution. Threads with higher priority are executed before lower-priority threads. Priority is important to manage resource allocation and responsiveness in multithreaded applications.

Question 4: What is a deadlock in the context of threading, and how can it be avoided?

Answer: Deadlock is a situation where two or more threads are unable to proceed due to circular dependencies. It can be avoided through techniques like resource allocation hierarchy, ensuring that threads request resources in a consistent order.

Question 5: Explain the terms "multithreading" and "concurrency."

Answer: Multithreading refers to the ability of a CPU or a single program to execute multiple threads concurrently. Concurrency refers to the concept of making progress on multiple tasks simultaneously.

DLLs:

Question 6: What is a Dynamic Link Library (DLL), and how does it contribute to software development?

Answer: A DLL is a modular file containing code and resources that multiple programs can share. It promotes code reusability, modularity, and efficient memory usage by allowing functions to be dynamically loaded and shared among different programs.

Question 7: How does dynamically linking to a DLL differ from statically linking code?

Answer: Dynamically linking to a DLL involves loading the DLL's code at runtime, reducing executable size and allowing for updates without recompilation. Statically linking includes all code in the final executable, making it larger and harder to update.

Question 8: Describe a scenario where DLL versioning issues might arise and explain how to mitigate them.

Answer: DLL versioning issues can occur when an application relies on a specific version of a DLL that changes or becomes unavailable. Mitigation involves maintaining backward compatibility, using version information, and implementing proper dependency management.

Question 9: How can DLLs contribute to code modularity and reusability?

Answer: DLLs allow functions or modules to be encapsulated into separate files, promoting modular design. These modules can be shared among multiple programs, enhancing code reusability and reducing redundancy.

Question 10: Explain the term "DLL Hell" and suggest strategies to avoid it.

Answer: "DLL Hell" refers to conflicts arising from incompatible or missing DLL versions. To avoid it, use versioning, distribute necessary DLLs with the application, implement proper dependency management, and prioritize backward compatibility.