**Q6. How to avoid Race conditions and deadlocks?**

Race conditions and deadlocks are common issues in C programming, they occur when working with multiple threads or processes. Some strategies to avoid them are as follows:

* **Synchronize Access to Shared Resources:**

Using synchronization primitives i.e., mutexes to ensure that only one thread accesses a shared resource at a time. This prevents race conditions where multiple threads try to modify the same resource simultaneously.

* **Use Locks Correctly:**

Deadlocks can occur when threads acquire locks in a different order, leading to a situation where each thread is waiting for a resource held by another thread. So, acquiring locks in a consistent order is important to prevent deadlocks.

* **Avoid Shared Mutable State:**

Minimizing the use of global variables or shared memory regions that can be modified by multiple threads concurrently. Instead, using message passing or other communication mechanisms to share data between threads.

* **Testing and Debugging:**

Thoroughly testing the code to identify and fix concurrency issues and using debugging tools to detect race conditions and deadlocks early in the development process.

* **Documentation and Code Review:**

Document concurrency-related decisions in our code and conducting code reviews to ensure knowledge sharing to improving the overall quality of our codebase.