Advanced C++ Multithreading Exam

Instructions

- Answer all questions.
- Assume C++17 or later standards unless specified otherwise.
- Write code snippets where required.
- Explain your reasoning for theoretical questions.

Section 1: Multiple Choice Questions (10 Questions)

1. **What is the primary purpose of `std::mutex` in a multi-threaded program?**
   (a) To allow threads to wait for a condition
   (b) To protect shared resources from concurrent access
   (c) To manage thread execution order
   (d) To signal between threads
2. **Which of the following is true about `std::condition_variable`?**
   (a) It can be used without a `std::mutex`
   (b) It must always be used with a `std::mutex`
   (c) It is used to lock resources
   (d) It replaces the need for `std::mutex`

3. **What happens if a thread tries to lock a `std::mutex` that is already locked by another thread?**

   (a) The thread terminates immediately

   (b) The thread continues execution without locking

   (c) The thread blocks until the mutex is unlocked

   (d) The thread throws an exception

4. **What is the purpose of `std::queue` in a producer-consumer scenario?**

   (a) To store threads

   (b) To store shared data between threads

   (c) To lock resources

   (d) To signal threads

5. **Which of the following is true about `std::thread`?**

   (a) A thread cannot be joined after it has been detached

   (b) A thread must always be joined

   (c) A thread can be both joined and detached

   (d) A thread cannot be detached

6. **What is the purpose of `std::condition_variable::wait()`?**

   (a) To lock a mutex

   (b) To wait for a signal while releasing the mutex

   (c) To terminate a thread

   (d) To unlock a mutex

7. **What happens if a `std::condition_variable` is signaled but no thread is waiting?**

   (a) The signal is ignored

   (b) The program crashes

   (c) The signal is stored for the next waiting thread

(d) An exception is thrown

8. **Which of the following is true about `std::unique_lock`?**

   (a) It cannot be used with `std::condition_variable`

   (b) It automatically unlocks the mutex when it goes out of scope

   (c) It is less flexible than `std::lock_guard`

   (d) It cannot be used with `std::mutex`

9. **What is the purpose of `std::condition_variable::notify_all()`?**

   (a) To wake up a single waiting thread

   (b) To wake up all waiting threads

   (c) To terminate all threads

   (d) To lock all mutexes

10. **What is a deadlock in the context of multi-threading?**

    (a) A situation where a thread terminates unexpectedly

    (b) A situation where two or more threads are blocked forever

    (c) A situation where a mutex is unlocked twice

    (d) A situation where a condition variable is signaled multiple times

Section 2: Code Analysis (10 Questions)

1. **Analyze the following code and identify any potential issues:**

```
std::mutex mtx;
std::queue<int> queue;
std::condition_variable cv;

void producer() {
    for (int i = 0; i < 10; ++i) {
        std::unique_lock<std::mutex> lock(mtx);
        queue.push(i);
        cv.notify_one();
```

```
    }
}

void consumer() {
    while (true) {
        std::unique_lock<std::mutex> lock(mtx);
        cv.wait(lock);
        int value = queue.front();
        queue.pop();
        std::cout << value << std::endl;
    }
}
```

2. **What is the purpose of the `std::unique_lock` in the above code?**
3. **What happens if the producer finishes before the consumer starts?**
4. **How can you modify the consumer to handle the case where the queue is empty?**
5. **What is the purpose of `cv.notify_one()` in the producer?**
6. **What happens if `cv.notify_all()` is used instead of `cv.notify_one()`?**
7. **What is the risk of not using `std::unique_lock` in the consumer?**
8. **How can you ensure the consumer thread terminates gracefully?**
9. **What is the purpose of the `std::queue` in this code?**
10. **What is the significance of the `std::mutex` in this code?**

Section 3: Code Implementation (10 Questions)

1. **Implement a producer-consumer model using `std::queue`, `std::mutex`, `std::condition_variable`, and `std::thread`.**
2. **Modify the above implementation to handle multiple producers and consumers.**
3. **Add a mechanism to gracefully shut down the consumer threads.**
4. **Implement a thread-safe queue using `std::queue`, `std::mutex`, and `std::condition_variable`.**
5. **Write a program where two threads increment a shared counter using `std::mutex` for synchronization.**
6. **Write a program where a thread waits for a signal from another thread using `std::condition_variable`.**
7. **Implement a barrier synchronization mechanism using `std::mutex` and `std::condition_variable`.**
8. **Write a program to demonstrate a deadlock scenario involving two threads and two mutexes.**
9. **Fix the deadlock in the above program.**
10. **Write a program to demonstrate the use of `std::async` with `std::mutex` and `std::condition_variable`.**

Section 4: Theoretical Questions (10 Questions)

1. **Explain the difference between `std::mutex` and `std::recursive_mutex`.**
2. **What is spurious wakeup, and how can you handle it in `std::condition_variable`?**
3. **Explain the difference between `std::lock_guard` and `std::unique_lock`.**
4. **What is the purpose of `std::condition_variable::wait_for()`?**

5. **Explain the concept of thread safety and how it applies to `std::queue`.**
6. **What is the difference between `std::thread::join()` and `std::thread::detach()`?**
7. **Explain the concept of a race condition and how `std::mutex` prevents it.**
8. **What is the purpose of `std::atomic` in multi-threading?**
9. **Explain the difference between `std::condition_variable` and `std::future`.**
10. **What is the role of the C++ memory model in multi-threading?**