Shared Mutex Solutions

Shared Mutex

- Explain briefly what a shared mutex is
- A shared mutex can be locked in two different ways
  - Exclusive lock. If a thread has an exclusive lock on a shared mutex, no other thread can acquire a lock until the first thread releases the lock
  - Shared lock. If a thread has a shared lock on a shared mutex, other threads can acquire a shared lock without having to wait for this thread to release it
  - If a thread wishes to acquire an exclusive lock, it must wait until all the threads which have a shared lock release their locks

shared_mutex usage

- Write a program which has two task functions
  - A "writer" task which modifies shared data
  - A "reader" task which accesses shared data but does not modify it
- Use an std::shared_mutex to synchronize these tasks
  - The reader task should sleep for 100ms before unlocking the mutex
  - This is to simulate activity
- The program creates twenty reader threads, then two writer threads, then another twenty reader threads
- How long do you expect it will take the program to run?
- Explain the results

shared_mutex usage

- With std::mutex, reader threads were forced to execute sequentially in their critical sections
- With std::shared_mutex, reader threads can execute concurrently in their critical sections
  - Unless a writer thread has an exclusive lock
- This reduces the amount of blocking
- The program can execute much more quickly

Data Race Avoidance

- Explain how using a shared mutex avoids data races
  - The writer thread cannot get an exclusive lock, until all other threads have left their critical sections
  - The writer thread's exclusive lock prevents all other threads from locking the mutex while the writer thread is in its critical section
  - A reader thread can only get a shared lock if there are no writer threads which have an exclusive lock
  - The reader thread's shared lock allows other reader threads to obtain a shared lock and execute their critical sections concurrently
  - There is no scenario in which a writer thread and a reader thread can concurrently execute in a critical section
  - The conditions required for a data race cannot occur