Lock Guard Solutions

Exception Thrown in Critical Section

- Explain what happens if an exception is thrown in a critical section
  - The program will immediately leave the current scope and start executing an exception handler
  - If the critical section is protected by calling lock() on std::mutex, the unlock() call will never be executed
  - All threads which are waiting to acquire a lock on the mutex will be blocked
  - All threads which are waiting or joined on these threads will be blocked
  - Usually, the entire program is deadlocked

Exception Thrown in Critical Section

- What approaches can programmers use to manage this situation?
  - C++ provides a number of "wrapper" classes that use the RAII idiom to manage mutexes
  - These take advantage of the fact that, when an exception is thrown, the destructors are called for every object in scope
  - These wrapper classes lock the mutex in their constructor and unlock it in their destructor
  - This guarantees that the mutex will always be unlocked if the function returns

Drawbacks of std::mutex

- Are there other situations where these approaches are useful?
  - Yes
- If so, give some examples
  - Thread function with multiple return paths ("return" statements)
  - Loop which locks a mutex and has "break" or "continue" statements
  - The programmer forgets to call unlock(!)

std::lock_guard

- Rewrite the unscramble program
  - Instead of locking and unlocking a mutex directly, use std::lock_guard
  - Add a sleep statement at the end of the loop
- Do you notice anything unusual?
  - There is a noticeable delay between each output
- Why might this be the case?
  - The mutex is not unlocked until the end of the loop
  - The sleep statement is executed while the mutex is still locked
  - Other threads cannot enter their critical section while this thread is sleeping

std::lock_guard

- Change your program
  - Throw an exception in the critical section
  - Add a try-catch block to handle the exception
- What happens when you run the program?
  - The program appears to run normally
  - Except that each output is followed by the exception handler's output
- Explain your results
  - When the exception is thrown, the destructors are called for all objects in scope
  - std::lock_guard's destructor unlocks the mutex
  - Another thread which is waiting will acquire the lock and be able to run
  - No threads are blocked

std::lock_guard

- Suggest one feature that could usefully be added to std::lock_guard
  - A member function to unlock the mutex
  - This would give programmers more control over when the mutex is unlocked
  - It would prevent unnecessary blocking of other threads when executing code after the critical region