Mutex Introduction Exercises

Copyright © James Raynard 2020

Mutex introduction

- Explain what is meant by the terms "critical section" and "mutex" in relation to multi-threaded programs
- How can mutexes be used with critical sections?
- Briefly describe the C++ std::mutex

Copyright © James Raynard 2020

Rewrite using std::mutex

- Rewrite the "scrambled output" program using a mutex to protect the output operations
- Verify that the output is not scrambled when there are concurrent threads running
- What happens if the mutex is not unlocked? Explain the results

Copyright © James Raynard 2020

Rewrite using std::mutex

- Alter your program so that an exception is thrown between the output statement and the unlock call
- Add a catch handler at the end of the loop to handle the exception
- What happens when you run the program? Explain your results.

Copyright © James Raynard 2020

Mutexes and data

- Implement a simple thread-safe wrapper for the std::vector class, Vector
  - This only stores ints
  - It has a push_back() member function. This uses a mutex to protect calls to the std::vector push_back()
  - Provide a print() function to display all its elements
- Write a thread entry function that calls push_back 5 times, with a 50ms sleep after each call
- Write a program with a global Vector instance that starts ten of these threads and then prints out all the elements of the Vector
- Replace the Vector instance with a standard vector<int>
- Explain your results

Copyright © James Raynard 2020

try_lock()

- Write a program which runs two task functions in separate threads
- The first task function locks a mutex, sleeps for 500ms and releases the lock
- The second task function sleeps for 100ms, then calls try_lock() to lock the mutex. If unsuccessful, it sleeps again for 100ms and calls try_lock() again. If successful, it unlocks the mutex
- Add suitable print statements and run the program
- What do you observe?

Copyright © James Raynard 2020