Condition Variables with Predicate Solutions

Lost Wakeup

- What is meant by the term "lost wakeup"?
  - A lost wakeup occurs when a thread notifies a condition variable, but there are no threads which are waiting on the condition variable
  - This can happen if a writer thread calls notify() before the reader thread calls wait()
  - This can cause the reader thread to block indefinitely in the wait() call
- Write a program which demonstrates a lost wakeup

Spurious Wakeup

- What is meant by the term "spurious wakeup"?
  - A spurious wakeup occurs when a condition variable wakes up a thread, without any threads notifying the condition variable
  - This is due to the way that the C++ std::condition_variable is implemented

Lost Wakeup Avoidance

- Suggest how to avoid lost wakeups
  - Add a shared bool variable
  - This bool will indicate to the reader whether there are any pending notifications
  - When a thread calls notify(), it sets the bool
  - Before a thread calls wait(), it checks the bool
  - If the bool is false, there have been no notifications and the thread calls wait()
  - If the bool is true, there has been a notification and the thread continues
- Modify your program so that it is not affected by lost wakeups

Spurious Wakeup Avoidance

- Suggest how to avoid spurious wakeups
  - The same approach as with lost wakeups
  - Add a shared bool variable
  - This bool will indicate to the reader whether there are any pending notifications
  - When a thread calls notify(), it sets the bool
  - Before a thread calls wait(), it checks the bool
  - If the bool is false, there have been no notifications and the thread calls wait()
  - If the bool is true, there has been a notification and the thread continues

Multiple Threads

- Modify your program so that there are three reader threads
- Modify the reader's task function to display the thread's ID
- Modify the writer's task function so that it
  - Calls notify_one() once
  - Calls notify_one() three times
  - Calls notify_all() once

Multiple Threads

- Explain the results
  - When the writer thread calls notify_all(), the condition variable will wake up all three reader threads. In each reader, wait() returns, the mutex is locked and the thread can display the modified value before it exits
  - When the writer thread calls notify_one() three times, the condition variable will also wake up all three reader threads
  - Calling notify_one() once will cause one reader thread to be woken up. The other two readers will continue to sleep. If no further notifications are sent to the condition variable, the program will be blocked indefinitely
  - In all three cases, the choice of which thread to wake up, and the order in which to wake them up, is made by the scheduler. Different executions may result in different output