CMPU-224 Lab13 Quiz

Spring 2026

Name: _____

This is a closed book, closed notes quiz. No electronic devices are allowed. You have until 3:30pm to complete the quiz. There are a total of 5 questions and 10 points.

There should be enough space on the quiz for your answers. If you need more space to work out a problem, blank paper will be available, just ask.

If you finish with time remaining, raise you hand and I will come and collect your quiz. You may then work on the lab assignment.

Good Luck!

1. (2 points) A thread calls `pthread_cond_wait(&c, &m)` while holding mutex `m`. Which statement most accurately describes what happens to `m`?

   A. The mutex stays locked the whole time the thread sleeps.

   B. The mutex parameter is only used when debugging the threads in `gdb`.

   C. The call atomically releases `m` and sleeps, then re-acquires `m` before returning.

   D. The thread spins on `m` in a busy loop until the condition is signaled.

2. (2 points) The canonical pattern around `pthread_cond_wait` is

   ```
   pthread_mutex_lock(&m);
   while (!predicate)
       pthread_cond_wait(&c, &m);
   /* predicate is true here */
   pthread_mutex_unlock(&m);
   ```

   Why must the check be a `while` loop rather than a single `if`?

   A. `pthread_cond_wait` is non-blocking and must be invoked repeatedly until the predicate becomes true.

   B. A thread returning from `pthread_cond_wait` is not guaranteed that the predicate still holds, so it must be re-checked before proceeding.

   C. `pthread_cond_signal` is unreliable on Linux and must be retried until it succeeds.

   D. The loop ensures fairness: without it, one thread could monopolize the condition variable and starve the other waiters.

3. (2 points) Suppose you are tyring to implement a solution to the producer consumer problem with one condition variable and one lock. You run a four-producer / four-consumer stress test and the test eventually deadlocks. Which explanation is correct?

    A. POSIX requires at least one condition variable per participating thread, so a single condition variable can support at most one producer and one consumer.

    B. `pthread_cond_signal` can only wake a thread on the same core, so producers and consumers miss each other's wakeups on a multi-core system.

    C. Calling `pthread_cond_signal` while holding the mutex prevents other threads from aquiring the lock.

    D. A `pthread_cond_signal` can wake a same-side waiter (e.g., consumer wakes consumer), leaving the waiter on the other side unsignaled.

4. (2 points) In the multi-threaded allocator discussed in lecture and implemented in lab, multiple threads can be blocked at the same time requesting *different* sizes, and a single release of X bytes may satisfy some waiters but not others. The lab asks you to use one mutex and one condition variable. To avoid stranding a waiter that *could* proceed when another waiter (woken first) cannot, the `alloc_release` function must call

    _____ instead of `pthread_cond_signal`.

5. (2 points) The cyclic barrier implemented in lab stores both an arrival counter *and* a monotonically increasing `phase` field that is incremented once per round. A waiter snapshots `phase` on entry and sleeps until it changes, rather than sleeping until the arrival counter takes a particular value. Why is the `phase` field necessary?

    A. The POSIX standard requires every barrier implementation to expose a `phase` counter as part of its API, so it is added to match `pthread_barrier_t`.

    B. Without `phase`, the mutex would have to be held across the entire wait, preventing any thread from making forward progress between rounds.

    C. Without it, a fast thread can re-enter `barrier_wait` and increment the counter off of zero before slow waiters re-check the predicate, leaving them asleep forever.

    D. The `phase` field guarantees that threads are released from `barrier_wait` in the same order they arrived, preventing slower threads from being starved across many rounds.