Condition Variables

Chapter 30

Previously in CS212…

- We've talked using locks with certain data structures
- Problematically, when using locks, we were mostly spin/busy waiting
- This isn't very efficient as it wastes valuable CPU time to maybe do no meaningful work
- Perhaps there might be a better way…

Condition Variables

- Allow a thread to wait until a condition is true
- This is an explicit queue where threads wait while the state of execution is not desirable (e.g. data is not ready to be read)
- Once the state changes, we can signal one or more of those threads to wake up and do work (e.g. data is available for processing)

Basic Ops

- Wait()
  - Releases the lock AND puts the calling thread to sleep atomically
  - When it returns, re-acquires the lock
- Signal()
  - Wake up at **least one** thread waiting for a certain condition variable
- Broadcast()
  - Wake up **all threads** waiting for a certain condition variable
  - Useful, but can mask design flaws where signal is a better choice

Producer/Consumer Problem

- Also called the bounded-buffer problem
- Common use case when working with data and using threads
- Some threads need to process some data, while others are responsible for creating it
- Using a queue where the consumer enqueues jobs/data for processing and consumers dequeue data to perform operations

Gotchas

- Hold the lock when calling signal or wait to be safe!
- Don’t use one conditional variable for multiple purposes
  - E.g. waking consumer and producers when you only want one or the other
- You need a lock AND a conditional variable
- When checking a state (e.g. done) use a while loop and not an if to prevent issues with **spurious wakeup**

Next Time

- We look at semaphores!