Condition Variables and Semaphores

CSC 213 – November 3, 2021
Agenda for Today

1. Q&A
2. Condition Variables
3. Semaphores
4. Reminders
Lettercount Assignment Q&A

How do we use threadIdx.x in our code?

That's a CUDA-only feature. With pthreads, you can make two threads that do different things by changing the arguments you pass to them.
Sudoku Lab Q&A
Condition Variables
Condition Variable Basics

What does a condition variable do?
A cv allows threads to be placed on a queue where the thread will wait. Later, we can wake the thread back up.

Why would we use a condition variable with a mutex instead of just the mutex?
A thread that waits on a condition variable goes to sleep instead of spinning. Waiters no longer compete with the thread that will allow them to proceed.
Condition Variable Interface

`pthread_cond_wait(pthread_cond_t* cv, pthread_mutex_t* m)`

Unlock the mutex m and block the calling thread. Later, when the calling thread wakes up, the thread will re-acquire the mutex m.

`pthread_cond_signal(pthread_cond_t* cv)`

Wake up one of the threads waiting on the condition variable. The calling thread (usually) should hold the mutex that was released when the waiter(s) waited.

`pthread_cond_broadcast(pthread_cond_t* cv)`

Wake up all of the threads waiting on the condition variable. Same restriction as above w.r.t. the mutex.
A Simple Parallel Program

**Main Function**
Create two threads, tick_thread and tock_thread

While there is input from the user:
  - If the user typed "go", allow both threads to run
  - Wait for more input.

Join with tick_thread and tock_thread

**Thread Function**
Wait for the main thread to allow this thread to run.

Repeat five times:
  - Print this thread's message ("tick" or "tock")
  - Pause for one second

How should we make the threads wait, and how do we resume them?
*Discuss strategies with your group. We'll complete the implementation together.*
Implement Tick/Tock using Condition Variables
Semaphores
Semaphore Basics

What is a semaphore for?
A semaphore is a bit like a combination of a lock and condition variable. It can do what those two can do together.

How is a semaphore different from a mutex or condition variable?
The condition that a semaphore uses to decide whether or not a thread waits is tied to an internal counter in the semaphore.
Semaphore Interface

sem_init(sem_t* s, int cross_process, int state)
    Pass in a pointer to the semaphore to initialize. Pass in zero for cross process, unless the semaphore has to work across processes. The state parameter is the initial value of the semaphore's counter (which is ++ or --)

sem_wait(sem_t* s)
    Decrement the value in the semaphore's counter. If the value becomes negative, increment the counter and the thread sleeps. Eventually that thread wakes, and we repeat the process.

sem_post(sem_t* s)
    Increment the value of the semaphore's counter. If the counter is now 1, wake up a thread waiting waiting on the semaphore.
A Simple Parallel Program

Main Function
Create two threads, tick_thread and tock_thread

While there is input from the user:
If the user typed "go", allow both threads to run
Wait for more input.

Join with tick_thread and tock_thread

Thread Function
Wait for the main thread to allow this thread to run.

Repeat five times:
Print this thread's message ("tick" or "tock")
Pause for one second

How can we implement this same program with semaphores?
Discuss strategies with your group. We'll complete the implementation together.
Implement Tick/Tock using Semaphores
Reminders
Reminders

Lab
Sudoku lab is due on Wednesday, November 10th (next week)

Reading
Read about concurrency bugs for Friday

My Office Hours
A bit different this week!
Monday 10–10:45am, Tuesday 3:15–4:30pm, Friday 10–11am

Mentor Office Hours
Sunday 1–4pm, Tuesday 7–9pm, Wednesday 8–9pm
More Reminders

Department Info Session
Are you a 2nd year declaring a CS major this spring?
Attend this week's CS Extra at 4pm on Thursday to learn more about the major

Preregistration
If you have enjoyed the material in this class, you might want to take
CSC 395: Advanced Operating Systems