Condition Variables and Semaphores

CSC 213 – November 2, 2022
Agenda for Today

1. Connecting to MathLAN
2. Department Events
3. Q&A
4. Condition Variables
5. Semaphores
6. Reminders
Connecting to MathLAN
Department Events

CS Extra  4pm on Thursday, November 3rd
Graduate School in Computer Science
Discussion with Professors Eliott and Autry

CS Table  noon on Tuesday, November 8th
Digital Redlining (watch for an email today)

US Midterm Election  Tuesday, November 8th (no extra credit)
Tuesday, November 8th is the last day to vote in the US Midterm Election.
If you are eligible to vote, please do so!
Sudoku Lab Q&A
Condition Variables
Condition Variable Basics

What does a condition variable do?
A c.v. allows you to share information about a condition between threads more easily. That information is used to synchronize threads. Specifically, c.v.s allow threads to wait, and other threads to resume them.

Why would we use a condition variable with a mutex instead of just the mutex?
If you know two threads are accessing the same variable at once, you need a lock. If two threads are constantly acquiring and releasing a lock, they spend lots of time locking and little time making progress. A condition variable makes it possible to put the waiting thread to sleep, and the working thread can wake it up later.
Condition Variable Interface

`pthread_cond_wait(pthread_cond_t* cv, pthread_mutex_t* m)`
The thread that calls this function should hold mutex m. When called, `pthread_cond_wait` will release the mutex and put the thread to sleep on the `c.v`. Before this function returns, the thread will re-acquire mutex m.

`pthread_cond_signal(pthread_cond_t* cv)`
One thread waiting on cv will wake up, and will wait to reacquire the mutex it was holding when it called `pthread_cond_wait`. We don't know which thread will wake up. If there are no threads waiting, nothing happens.

`pthread_cond_broadcast(pthread_cond_t* cv)`
Wake up all threads waiting on cv. If there are no threads waiting, nothing happens.
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?
Semaphores capture the behavior of both locks and condition variables. A semaphore has a counter value. Threads can synchronize using the counter value. A thread can wait on the semaphore, or post to the semaphore. Waiting decrements the counter, and posting increments the counter. Any wait when the counter is $\leq 0$ will block the thread. A post will unblock the thread.

How is a semaphore different from a mutex or condition variable?
A semaphore that starts with a count of 1 is *exactly the same* as a mutex. Unlike a condition variable, if we use a semaphore we don't need a mutex to go with it. The semaphore's counter allows us to specify exactly how many waiters can proceed. Semaphores have no spurious or lost wakeups.
Semaphore Interface

```c
sem_init(sem_t* s, int cross_process, int state)
    Initialize a semaphore. If we were using this in multiple processes we'd set cross_process to 1. The state parameter is the initial counter value.
```

```c
sem_wait(sem_t* s)
    Try to decrement the semaphore's counter. If it goes below zero, sleep. Otherwise proceed.
```

```c
sem_post(sem_t* s)
    Increment the semaphore's counter. Wake up one sleeping thread.
```
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 9th (next week)

Reading
Read about concurrency bugs for Friday

My Office Hours
A bit limited this week. Sorry!

Mentor Office Hours
Sunday 1–4pm, Monday 8–9pm, Tuesday 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