The Perils of Multithreading

44-550: Operating Systems

How Can Multithreading Be Dangerous?

- Well, it's not, if you know everything about the state of your computer, including the order threads will be switched to
- Chaos theory says otherwise
- Memory is shared between threads
  - If your threads aren't sharing any of the data, it doesn't matter
  - In the bitonic sort example, I could prove (theoretically) that threads weren't trying to access the same piece of data simultaneously
- We run into *Race Conditions*
  - Tricky to debug and fix (Heisenbugs)
  - What if some operations aren't *atomic*?

Race Conditions Example

"Knock Knock"
"Race condition"
"Who's there?"

A Slightly More Worrisome Example

Thread 1	Thread 2	Account Balance
Read Balance: $200		Balance: $200
	Read Balance: $200	Balance: $200
	Deposit: $500	Balance: $200
Cash Check: $500		Balance: $-300
	Update Balance (200 + 500)	$700

Working with Shared Memory

- When working with shared memory, it is a good idea to ensure that only one thread is accessing a given resource at once
- Some problems allow you to do this without additional constructs
- Other problems, notsomuch...
- Several constructs are available to the skilled programmer when using multiprogramming
  - Mutexes
  - Semaphores
  - Condition Variables

Mutexes

- Mutual Exclusion
- Concept is simple:
  - The resource you are needing is behind a door
  - There is one key to the door
  - You can only get to the resource if you have the key
  - If someone else has the key, you MUST wait until the key becomes available
  - Once you acquire the key, nobody else can use it
- Basic operations
  - lock/acquire
  - release

Semaphores

- If a mutex is a unique key to one door, semaphores are n copies of a key, to n identical locks
- Basically a counter:
  - If the value is positive, decrement the value and continue on your way. When done, increment the semaphore
  - If the value is zero (if it's negative, you have problems), wait until it's positive and then do previous thing
- Basic operations:
  - Wait
  - Signal
- An interesting note from Stanford's notes on concurrency and sempahore notes: Historically, P is a synonym for SemaphoreWait. You see, P is the first letter in the word prolagen which is of course a Dutch word formed from the words proberen (to try) and verlagen (to decrease). V historically is a synonym for SemaphoreSignal since, of course, verhogen means "to increase" in Dutch.

Condition Variables

- Way to perform *thread synchronization*
- Threads will not continue until some condition is true
- Basic operations
  - wait
  - signal: wake one thread waiting
  - broadcast: wake all threads waiting on a condition

So we've solved all the problems, right?

- Well...
- No
- The system can still enter states in which it is impossible to continue
  - Starvation
  - Deadlock
  - Livelock

Starvation, Deadlock, Livelock

- Starvation
  - System reaches a state in which a thread is constantly denied resources required to run
- Deadlock
  - System cannot continue (and cannot change state)
- Livelock
  - System can continue to change state, but makes no progress toward the goal

The Dining Philosophers

- N philosophers are sitting at around table, and a bowl of rice sits in front of each
- To the right and left of each philosopher sits a single chopstick
- Philosophers must alternatively think and eat
- A chopstick can only be held by one philosopher
- A philosopher can only eat if he/she has both the left and the right chopstck

A Failed Solution

```
while true:
    while left chopstick is unavailable:
        think/sleep
    grab left chopstick
    while right chopstick is unavailable:
        think/sleep
    grab right chopstick
    eat/work
    put down right chopstick
    put down left chopstick
```

Other, Non-Obvious Solutions

- Dijkstra's solution
  - Establish an order on the chopsticks
- Arbitrator
  - Have a "waiter" that arbitrates who has what resources when
- Chandy/Misra solution
  - A much more generalized solution
- See:
  `https://en.wikipedia.org/wiki/Dining_philosophers_problem`