More Synchronization and Exam Logistics

CS439: Principles of Computer Systems
October 2, 2017
Last Time

• The Importance of Safety
  – Safety First!

• Advanced Synchronization
  – Conservative Two-Phase Locking
  – Transactions

• How to Program Multi-threaded Code
Today’s Agenda

• Review
  – Atomicity
  – How we get it
  – Tradeoffs and Problems
• Readers/Writers
• Pemberley!
Synchronization Review
Concurrent execution in C:

```c
int a=1, b=2;
main() {
    createThread(fn1, 4);
    createThread(fn2, 5);
    thread_join(all);
}

fn1(int arg1){
    if(a) b++;
}

fn2(int arg1){
    a=arg1;
}
```

What are the values of a and b after execution?

A. a=1, b=2  
B. a=1, b=3  
C. a=5, b=2  
D. a=5, b=3
... but can be problematic

```c
int a=1, b=2;
main() {
    createThread(fn1, 4);
    createThread(fn2, 5);
    thread_join(all);
}

fn1(int arg1){
    if(a) b++;
}

fn2(int arg1){
    a=0;
}
```

What are the values of `a` and `b` after execution?

A. `a=0, b=2`
B. `a=0, b=3`
C. `a=1, b=2`
D. `a=1, b=3`
Can both critical sections execute during a single execution of the code?

A. Yes
B. No
Atomicity

• Required to reason about multi-threaded code without considering all interleavings
• Requires mutual exclusion
• Locks provide that solution
• Looked at lock implementation
  – Requires waiting
  – Requires hardware support
• Use software abstractions
  – Semaphores
  – Monitors (lock+condition variables)
Tradeoff and Problems: Difficult to Get Right

- Ensure safety
- Ensure liveness
- No race conditions
- No starvation
- No priority inversion
- No deadlock
In Addition... the Cost of Parallelization

```c
for (k = 0; k < n; k++)
    a[k] = b[k] * c[k] + d[k] * e[k];
```

How would you parallelize this?
How many threads?
The Six Commandments

- Thou shalt always do things the same way
- Thou shalt always synchronize with locks and condition variables
- Thou shalt always acquire the lock at the beginning of a function and release it at the end
- Thou shalt always hold lock when operating on a condition variable
- Thou shalt always wait in a while loop
- (Almost) Never sleep()
Why Thread Coding Standards?

• History has tested this approach
• If you follow these commandments, you will find it easier to write correct code.
• In this class, you must use them or lose points.
• We highly recommend that you continue to do so after this class
But...

• After this class, if you can come up with something better, please use it!

• BUT...
  – Lots of really smart people have thought really hard about this already, so a day or two of thought is unlikely to change the best practice
  – The consequences of getting code wrong can be atrocious
  – People who are confident about their abilities tend to perform *worse*. If you think you are a Threading and Concurrency Ninja and truly understand, then you may wish to re-evaluate...
    • Dunning-Kruger effect
In this class...

• Six commandments
• Coarse-grained locking
• Order your resources
Readers and Writers
A Different Type of Problem

• We’ve looked at problems where we protect shared data by only allowing one thread in the critical section at a time

• Is this always appropriate? When might we want to let more threads access shared data at once?
Readers/Writers Problem

• Data is shared among several threads
  – Some only read
  – Some only write

• To get correct results, we allow multiple readers at a time, but only one writer at a time

• How can we control access to the object to permit this protocol?
Correctness Criteria

• Each read or write of the shared data \textit{must} happen within a critical section
• Guarantee mutual exclusion for writers
• Allow multiple readers to execute in the critical section at once
• Allow one writer (and no readers) to execute in the critical section at once
Readers and Writers: Monitor Solution

- What methods do we need?
- How many locks?
- How many condition variables?
- What should we name them?
- Any other variables?

- Assume we’re going to say `<read>` and `<write>` for accesses to the shared data.
Readers and Writers: Monitor Solution

Variables:

read()
write()

Is our solution fair?
A. Yes
B. No, favors readers
C. No, favors writers
Understanding Our Solution

It works, but it favors readers over writers

- Any reader blocks all writers
- All readers must finish before a writer can start
- Last reader will wake any writer, but a writer wakes all readers and writers
- If a writer exits and a reader goes next, then all readers that are waiting will get through
Readers and Writers: Monitor Solution

Variables:

```c
read()

write()

read()

}
```

Alternative Semantics

• It may be that you would like a writer to enter its critical section as soon as possible.

• How could we implement that?
Pemberley!
Exam Logistics
What might be on the exam?

A. Information from lectures and reading
B. Coding questions
C. Concept questions (general understanding/thought)
D. All of the above (and more!)
Exam Procedures

• Arrive on time
  – No one may start the exam after the first person leaves

• Bring your UT ID

• Find your EID and assigned seat on the chart outside the classroom

• Do not enter the room until told to do so

• When you enter, proceed to your seat
Exam Procedures

• Leave all extra paper, electronics, hats, etc. in your bag.
• Do not begin the exam until told to do so
• No questions may be asked during the exam
  – Write any assumptions
• When finished
  – turn in exam and all scratch paper to myself or the proctor
  – present your ID
iClicker Question

What should you bring to the exam?

A. A writing utensil and your ID
B. Nothing
My Best Advice

Do NOT panic!

You have been taught how to do each question, and you can do it.
Announcements

• Class on Wednesday is canceled
• No discussion sections this week
• My office hours are canceled
• Project 1 is due Friday night
• See you Monday!
  – Virtual Memory, here we come!