More Synchronization

CS439: Principles of Computer Systems

February 25, 2015

Last Time

- How to Program Multi-threaded Code
- Dining Philosophers
- Deadlock
  - when a set of threads cannot progress because each requires a resource held by another member of the set
- Prevent deadlock through resource ordering
- Advanced Synchronization
  - Fine-grained locking (efficiency)
  - 2-Phase locking
  - Transactions

Today's Agenda

- The Importance of Safety (Therac-25)
- Review
  - Atomicity
  - How we get it
  - Tradeoffs and Problems

Therac-25
or The Importance of Safety

Background

- Linear accelerator
- Used to treat patients …
- Picture

Modes of Operation

	Beam Energy	Beam Current	Beam Modifier
For electron therapy	5-25 MeV	low	magnets
For X-ray therapy, photo mode	25 MeV	high (100x)	flattener
For field light mode	0	0	none

What Went Wrong?

- Two software problems
  - Let's look at pseudocode…
- Tons of bad software design/human failures that might have prevented this:
  - No end-to-end consistency checks
  - Errors reported by number only *and there was no documentation!*
  - False alarms
  - No quality control
  - No clearinghouse for mistakes *and company hid failures from other users*
  - Don't trust software---hardware should have prevented this, too

What about more recent disasters?

- We don’t know for sure
- Possibly software lost treatment plan and defaulted to “all leaves open”

- Software should have sensible defaults!

Lessons

- Complex systems fail for complex reasons
- Be tolerant of inputs
- Be strict on outputs

- Assume buggy software and protect against it!

Synchronization Review

iClicker Question

```
int flag1=0, flag2=0;

int main(){
   tid id=thread_create(p1, NULL);
   p2(); thread_join(id);
}

void p1 (void *ignored){
   flag1=1;
   if(!flag2){
      critical_section_1();
   }
}

void p2(void * ignored){
   flag2=1;
   if(!flag1){
      critical_section_2();
   }
}
```

Can both critical sections in 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

```
for(i=0; i<N; i++){
    for(j=0; j<N; j++){
        for(k=0; k<N; k++){
            C[i][j] += A[i][k] * B[j][k];
}}}
```

How would you parallelize this?

How many threads?

How many locks?

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 locks

Summary

- Please Think!
- Safety first!
  - Coarse-grained locking is the easiest to get right, so do that
  - Don’t worry about performance at first
  - In fact, don’t even worry about liveness at first

Announcements

- Exam 1 is NEXT Wednesday at 7p UTC 2.112A
  - If you have a conflict, you should have already told me and received instructions
- Topics list, sample exam will be posted tomorrow
- Homework 4 is due Friday at 8:45a
- Project 1 due NEXT Friday at 11:59p