The Importance of Safety

CS439: Principles of Computer Systems
February 17, 2016
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
What is the Therac-25?

• Linear accelerator
• Used to treat patients ...
## Modes of Operation

<table>
<thead>
<tr>
<th>Mode</th>
<th>Beam Energy</th>
<th>Beam Current</th>
<th>Beam Modifier</th>
</tr>
</thead>
<tbody>
<tr>
<td>For electron therapy</td>
<td>5-25 MeV</td>
<td>low</td>
<td>magnets</td>
</tr>
<tr>
<td>For X-ray therapy, photo mode</td>
<td>25 MeV</td>
<td>high (100x)</td>
<td>flattener</td>
</tr>
<tr>
<td>For field light mode</td>
<td>0</td>
<td>0</td>
<td>none</td>
</tr>
</tbody>
</table>
What Went Wrong?

• Two (major) software problems
• Tons of bad software design/human failures that might have prevented this:
  – False alarms
  – Errors reported by number only and there was no documentation!
  – No clearinghouse for mistakes and company hid failures from other users
  – No end-to-end consistency checks
  – No quality control
  – 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
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 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 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

• Homework 4 due Friday in section
• Exam 1 is next Wednesday at 7p WEL 2.224
  – If you have a conflict, you should have already told me (if you don’t receive instructions by noon Tuesday, contact me again)
  – Show up ON TIME
• Project 1 due next Friday 11:59p