CS 240 in context
How Computers Work

1

Hardware

- Devices (transistors, etc.)
- Digital Logic
- Microarchitecture
- Instruction Set Architecture
- Operating System
- Compiler/Interpreter
- Programming Language
- Program, Application, Algorithm

Software
2 Big Ideas in CS, Systems, and beyond

Abstraction
Do not start every project with transistors. Abstraction is beautiful and empowering, but real abstractions have leaks and wrinkles.

Translation
Between layers of abstraction. Structured computation.

Representation
No representation without taxation. Representations have costs.

Performance
Memory: clever, imperfect abstraction. Tiny code changes, huge impact.

Security + Reliability
Trickiest exploits & errors involve multiple layers, even hardware!

These things matter more every day.
The **GHOST vulnerability** is a buffer overflow condition that can be easily exploited locally or remotely, which makes it extremely dangerous. This vulnerability is named after the `GetHOST` function involved in the exploit.
Ariane 5 Rocket, 1996
Exploded due to cast of 64-bit floating-point number to 16-bit signed number. Overflow.

1998
Mars Climate Orbiter
Disintegrated due to mismatched units in Lockheed-Martin / NASA software components.
Toyota "Unintended Acceleration Events"

Oklahoma jury:
"Spaghetti Code" = "reckless disregard"

>10,000 global variables
81,514 violations of MISRA-C coding rules
   Expect 3 minor bugs + 1 major bug per 30 violations

Task/process monitoring failed to monitor tasks/processes
Memory corruption

(Wait, it was written in C?!?!?!)

http://www.safetyresearch.net/blog/articles/toyota-unintended-acceleration-and-big-bowl-%E2%80%9Cspaghetti%E2%80%9D-code
"... a **Model 787 airplane** that has been powered continuously for 248 days can lose all alternating current (AC) electrical power due to the generator control units (GCUs) simultaneously going into failsafe mode ... This condition is caused by a **software counter** internal to the GCUs that will **overflow** after **248 days** of continuous power. We are issuing this AD to prevent loss of all AC electrical power, which could result in **loss of control of the airplane.**" --FAA, April 2015
How could we improve computer systems?

Security

Efficiency

Speed

Space

Programmer

Cost, availability

What a simple phone can do for people: https://opendatakit.org/about/deployments/

Energy, materials

A few of the impacts we usually don't see:


Reliability

...
one thing to make the world a better place...
shared-memory multithreading

private execution context:
- program counter
- call stack
- registers

shared memory:
- heap (objects)
- global variables

thread 1
memory access
thread 2
thread 3
class Candidate {
    int votes;
    ...
    void addVote() {
        votes++;
    }
}

candidate votes: 13

Thread 1

Thread 2

Thread 3
### concurrent accesses

<table>
<thead>
<tr>
<th>Thread 1</th>
<th>Thread 2</th>
<th>Votes</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>0</td>
</tr>
</tbody>
</table>

- \texttt{votes++;}

- \texttt{votes++;}
## concurrent accesses

<table>
<thead>
<tr>
<th>thread 1</th>
<th>thread 2</th>
<th>votes</th>
</tr>
</thead>
<tbody>
<tr>
<td>int v1 = <code>votes</code>;</td>
<td></td>
<td>0</td>
</tr>
<tr>
<td><code>votes</code> = v1 + 1;</td>
<td></td>
<td>1</td>
</tr>
<tr>
<td>int v2 = <code>votes</code>;</td>
<td></td>
<td>1</td>
</tr>
<tr>
<td><code>votes</code> = v2 + 1;</td>
<td></td>
<td>2</td>
</tr>
</tbody>
</table>
Problem 1: each thread’s increment should happen “as one.”

<table>
<thead>
<tr>
<th>thread 1</th>
<th>thread 2</th>
<th>votes</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>0</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>int v1 = votes;</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>votes = v1 + 1;</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>votes = v2 + 1;</td>
<td>1</td>
<td></td>
</tr>
</tbody>
</table>

Problem 2: two threads accessed votes “at the same time.”
(data race)
Northeast Blackout, 2003
caused in part by a software concurrency error
despite “in excess of 3 million online operational hours” -Mike Unum, GE Energy
data race

Two memory accesses:

1. to the same memory location
2. by different threads
3. at least one access is a write
4. the accesses are not ordered by synchronization

thread 1

```c
int v1 = votes;
votes = v1 + 1;
```

thread 2

```c
int v2 = votes;
votes = v2 + 1;
```
synchronization with locks

Synchronization orders events in separate threads to control access to shared data.

class Candidate {
    int votes;
    Lock l = new Lock();

    void addVote() {
        l.lock();
        votes++;
        l.unlock();
    }
}

Zero or one threads can hold a mutual exclusion lock at a time.

only one thread at a time
synchronization with locks

Is there a data race?

data race
two memory accesses:
• to the same memory location
• by different threads
• at least one write
• not ordered by synchronization.
synchronization with locks

thread 1

\[
\text{lock();}
\]
\[
\text{int } v1 = \text{votes};
\]
\[
\text{votes } = v1 + 1;
\]
\[
\text{unlock();}
\]

thread 2

\[
\text{lock();}
\]
\[
\text{int } v2 = \text{votes};
\]
\[
\text{votes } = v2 + 1;
\]
\[
\text{unlock();}
\]

votes

\[
\begin{array}{c|c|c}
\text{thread 1} & \text{thread 2} & \text{vote} \\
\hline
 v1 & v2 & 1 \\
\end{array}
\]
Data races are errors!

- unpredictable outcome
- unintuitive semantics in Java
- undefined semantics in C/C++

Data traces should be exceptions.

[Elmas et al., PLDI 2007; Marino et al., PLDI 2010; Lucia et al., ISCA 2010; Adve and Boehm, CACM Aug. 2010; ...]
data race exceptions

thread 1

```c
int v1 = votes;
votes = v1 + 1;
```

thread 2

```c
int v2 = votes;
```

---

"data race"

exception
implementing exceptions

<table>
<thead>
<tr>
<th>Exception Type</th>
<th>Code Example</th>
</tr>
</thead>
<tbody>
<tr>
<td>Null Pointer Dereference</td>
<td>if (c == null) { throw new NullPointerException(); } c.addVote();</td>
</tr>
<tr>
<td>Array Index Out of Bounds</td>
<td>if (i &lt; 0</td>
</tr>
<tr>
<td>Data Race</td>
<td>????????? votes = 13;</td>
</tr>
</tbody>
</table>
Software data race exceptions are slow.

**Recipe:**
- Access history of every memory location
  - Check/update on every memory access.
- Sync history of every lock
  - Update on every lock operation.
Radish: faster + accurate

Radish slowdown vs. native execution

PARSEC benchmarks (C)

[ISCA 2012]
software reliability toolbox

software engineering tools

- programming languages & program analysis
- compilers
- run-time systems
- computer architecture
- hardware implementation

Knowledge of program

data race exceptions

Efficient checking mechanisms

[ASPLOS 2014]
Translation affects data races.

reuse memory locations

move objects to new memory locations

add accesses

add synchronization

schedule language threads on hardware threads

[ASPLOS 2014]
Fast (HW) + Accurate (SW)

Java data-race exceptions

JVM + LARD events

x86 + LARD interface

Radish + LARD support

[ISCA 2012, ASPLOS 2014]
Skills for Thinking and Programming

Few of you will build new HW, OS, compiler, but...

1. Effective programmers understand their tools and systems.
2. The skills and ideas you learn here apply everywhere.

Reason about computational models, translation.

Debug for correctness and performance (with tools to help).

Assess costs and limits of representations.

"Figure it out" via documentation, experiments, critical thinking.
4 Foundations

- CS 301: Compilers and Runtime Systems
- CS 342: Computer Security
- CS 242: Computer Networks
- CS 251: Programming Languages
- CS 240: Computer Systems/Organization
- CS 349: Distributed Computing
- CS 3??: Operating Systems
- CS 249: Scientific and Parallel Computing
- CS 3??: Computer Architecture
- CS 304: Databases with Web Interfaces