15-410

"My other car is a cdr" -- Unknown

Exam #1
Mar. 1, 2010

Dave Eckhardt

Synchronization

Computer Club movie night

- **“The Net”**
  - **“Her driver's license. Her credit cards. Her bank accounts. Her identity. DELETED.”**
- **Tuesday 17:30, Wean 7500**

However….

Synchronization

Checkpoint 2 –Wednesday

- Please read the handout warnings about context switch and mode switch and IRET *very carefully*
  - Each warning is there because of a big mistake which was very painful for previous students

Asking for trouble

- If your code isn't in your 410 AFS space every day, you are asking for trouble
- If your code isn't built and tested on Andrew Linux every two or three days, you are asking for trouble
- If you aren't using source control, that is probably a mistake

Synchronization

Upcoming events

- 15-412 (Fall)
  - If you want more time in the kernel after 410...
  - If you want to see what other kernels are like, from the inside

Google “Summer of Code”

- http://code.google.com/soc/
- Hack on an open-source project
  - And get paid
  - And quite possibly get recruited
- Projects with CMU connections: Plan 9, OpenAFS (see me)

CMU SCS “Coding in the Summer”?

Synchronization

Crash box

- How many people have had to wait in line to run code on the crash box?
  - How long?

Synchronization

Debugging advice

- Once as I was buying lunch I received a fortune

Synchronization

Debugging advice

- Once as I was buying lunch I received a fortune

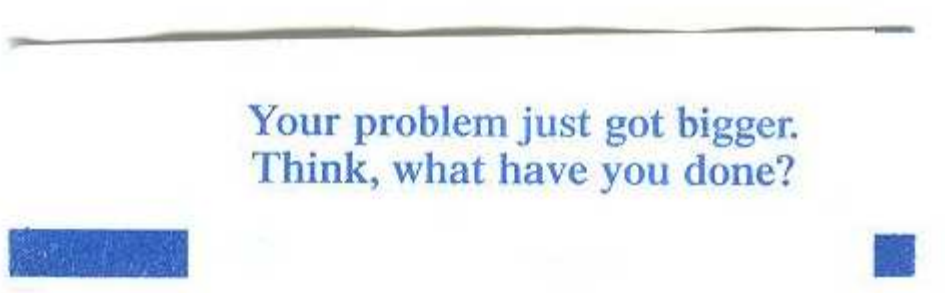

Image credit: Kartik Subramanian

A Word on the Final Exam

Disclaimer

- Past performance is not a guarantee of future results

The course will change

- Up to now: “basics” - What you *need* for Project 3
- Coming: advanced topics
  - Design issues
  - Things you won't experience via implementation

Examination will change to match

- More design questions
- Some things you won't have implemented (text useful!!)
- Still 3 hours, but more stuff (~100 points, ~7 questions)

“See Course Staff”

If your paper says “see course staff”...

- **...you probably should!**

This generally indicates a serious misconception...

- **...which we fear will seriously harm code you are writing now...**
- **...which we believe requires personal counseling, not just a brief note, to clear up.**

Outline

Question 1

Question 2

Question 3

Question 4

Question 5

Q1a – "exception" vs. "interrupt"

Related concepts abound!

- Trap, fault, interrupt, "machine check", "NMI"
- Each is a "surprise" to the processor
- Key differences
  - Is ___ synchronous to the instruction stream?
  - Can ___ be recovered from?
  - Is ___ "a normal surprise" or really a big surprise?
  - If we go back to the old instruction stream, which instruction will we go back to?

Q1b – "Segment register"

Key underlying concept: "segment"

- Range of memory of one kind
  - Defined by: (base, limit, flags)
    - » This is true on multiple architectures
  - x86 calls (base,limit,flags) a "descriptor"

x86 architecture

- Descriptors live in a table
  - A table index ("selector") selects which entry to use
    - » Of course, selector = index plus flags

Segment register

- Selectors are stored in registers –segment registers!
- Code fetch uses %CS; PUSH/POP use %SS; ...

Q1b – "Segment register"

How to get into trouble

- "Segment register stores memory"
  - Too small!
  - Abstraction-layer collapse

A common issue

- "Segment register stores (base,limit)"
  - Not impossible, but segments are frequently used in multiple contexts (everybody on a machine uses the same kernel code segment...)

Q2 –Dining Philosophers

Good news

- Most people got 100%
- Many people were close

Key issues

- Impossible traces
- Knowing “the key idea” but not the specifics
  - When your kernel deadlocks, it will be specific!

Q3 –SIGSEGV

The problem

- A poorly-written “handler” for SIGSEGV results in a program hanging

Conceptually

- “Segmentation fault” is an *exception*
  - Some instruction couldn't be executed
    - » (key issues: which one, why not?)
  - The instruction stream will *re-execute* that instruction
    - » Unless we fix the reason why it broke, or
    - » Unless we divert that instruction stream

Details

- This instruction can't be fixed, so we must divert.

Q4 –“Pair locks”

Thread-synchronization code with a problem

- Actually, two problems

Q: show progress failure

Common issues

- Unreadable traces
- Impossible traces
- Showing a bounded-waiting failure instead
  - Easy to confuse - though the problem gave the definition!
  - Good “bounded-waiting failure” answers did ok
- Many people found the problem - good

Q4 –“Pair locks”

Hint

- You need only three threads
  - One thread “of each type”
- Show “Some thread(s) want to acquire something(s) available for acquisition, but for an indefinite period of time this does not happen”.

Q4 –"Pair locks"

Q: solve the problem

Common issues

- Problem still exists even after the code changes
- New code causes other problems (deadlock, too many lockers)

"One giant lock"

- Serialize everybody no matter what they ask for –if one person wants A and the other wants B, second waits for first to be done
  - This is the moral equivalent of global variables –avoid unless *no* other solution is possible

Q4 –“Pair locks”

“Yield loop”

- “If I wake up and things aren't right for me, wake up somebody else”
  - The moral equivalent of “yield(-1)”
  - Problem: constant churn: *somebody is always running!*
    - » Remember: stopping when it's time to stop is very important

Good solutions

- “Impose a lock order” - works pretty well in this case
- Use some other P2 thread-synchronization primitive
  - (one of the higher-level ones)
- Change the cond_signal() call to something else

Q5 – print_ints()

Goal

- Write a “mini-printf()”

Concepts

- Rehash of P0 knowledge: where are the ints?
  - Get one register value via assembly code, and off we go!

The vararg/stdarg solution

- Not what we were looking for, but acceptable if done right
  - Now you know what's inside!

Common issues

- “Assume a convenient int array as a parameter”
- “Stack grows the other way”

Breakdown

90% = 63.0 16 students (3 got 69/70)

80% = 56.0 26 students

70% = 49.0 20 students

60% = 42.0 9 students

50% = 35.0 4 students

<50% 2 students

Comparison

- **Scores were “reasonably shaped”**
  - **Probably a few more A's than typical**

Implications

Score 42..49?

- Form a theory of “what happened”
  - Not enough textbook time?
  - Not enough reading of partner's code?
  - Lecture examples “read” but not grasped?
  - Sample exams “scanned” but not *solved*?
- Probably plan to do better on the final exam

Score below 42?

- Something went *rather* wrong
  - It's important to figure out what!
- Passing the final exam may be a serious challenge
- To pass the class you must demonstrate some proficiency on exams (not just project grades)