15-410

"My other car is a cdr" -- Unknown

Exam #1
Oct. 13, 2010

Dave Eckhardt

Garth Gibson

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 (possibly get recruited, probably not a lot)
- 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

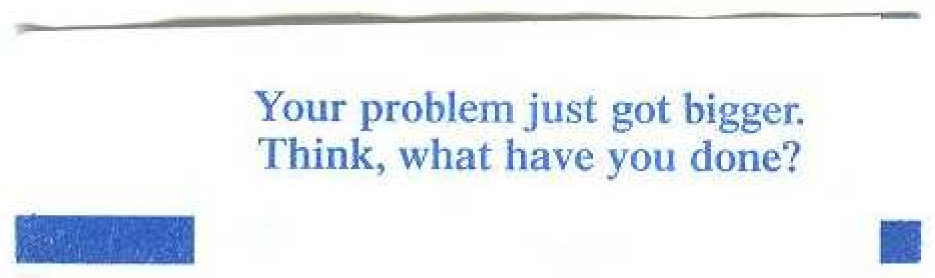
Debugging advice

- Once as I was buying lunch I received a fortune

Synchronization

Debugging advice

- Once as I was buying lunch I received a fortune



Your problem just got bigger.
Think, what have you done?

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 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

Question 6

Q1a –“three 'kinds' of register”

Many “kinds” were acceptable

- Caller-save, callee-save, ...
- General-purpose, control, ...

Hardware has its quirks

- Usually to make something go faster
- We need to keep the ***details*** of this “finite state machine” in mind
  - Some x86-32 quirks are just quirks
  - But many do represent how most hardware works

Q1b –"kernel stack"

In C, "the action" centers on the stack

In kernels, "the action" centers on *kernel* stacks

- ...which are structurally different from user stacks

Key features

- Must not be accessible (read-write *or* read-only) to user code
- Generally fixed-size (small)
- "Must always exist"

How to get into trouble

- Talk only about stacks in general

Q2 –Scheduler state transitions

Good news

- Most people did at least “ok”

Frequent problems

- Confusing “sleeping” with “blocked”
  - It is *possible* to conceive of “sleeping” as “a kind of blocked”
    - » (Implementation often a bit different)
    - » We gave you two states (hint: we think they're different)
- You should have *two* single-ended arcs
- Be sure to understand the key running ⇔ runnable interchange (there are multiple reasons each way)
- Blocked generally goes to Runnable, *then* to Running
  - Scheduler usually needs to evaluate the new runnable

Q3 –cvars atop rendezvous()

The problem

- Implement condition variables in an unfamiliar situation

Conceptually, a cvar includes...

- ...queue of sleeping threads
- ...solution to “atomic block” problem

Common problems

- Each cvar uses rendezvous() tags: 0, 1, 2, ...
  - This means it's impossible for a program to use two cvars
- cond_signal() blocks until some thread calls cond_wait()
  - That may never happen!
  - Cvar's job is to block *waiters* indefinitely, not signallers
- See course staff if you have a malloc() list storing 0, 1, 2...

Q4 –“rwlock_promote()”

Q: What if we non-atomically upgrade our lock?

- People pervasively saw what is wrong here

Q: What's wrong with rwlock_promote() “spec”?

- Key problem: “block awaiting __X__ while forbidding all others to achieve __X__” can be implemented, but it's a recipe for deadlock...
- Some answers were based on mis-readings of the “spec”
  - “Sequential atomic upgrade” isn't atomic for the second thread, so that reduces to the part (a) problems

Q4 – "rwlock_promote()"

"Be careful out there..."

- "Insertion could be lost" – mis-ordered, but not actually *lost*
- "Read/write of free()'d memory causes an exception"
  - This is not a rule! If you use bad data *as a pointer*, maybe...

Q5 –Critical-section algorithm

Overall

- Most people correctly identified one problem
- Quite a few didn't find a second one
    - Don't worry, we swapped (a) and (b) points so your correct solution got 10 points and the incorrect one got 5

Q5 –Critical-section algorithm

Common problems

- Notation
  - i vs. j caused some people to spin on the wrong variable
  - Arithmetic doesn't really work for “thread 1” and “thread 2”
- One problem class: impossible traces
  - “`do { ... } while (!...false)`” does run a second time
  - A few other impossible sequences
- *Common* problem: stopping a trace too early
  - If you want to show a steady state, make sure you trace long enough to show it *is* steady!
  - Once through a loop isn't enough if key values change
    - » Need to show them stuck in the new value, or changing back to the old value
  - Be very clear about what sub-trace you believe repeats

Q6 –Nuts & Bolts

Overall

- People often identified the bad register (good)
- “What went wrong” claims were less plausible
  - The register dumps we showed were from short code with plausible bugs
    - » Accidental stack crash due to array overflow
    - » thread_fork wrapper gone awry
- “How could this happen?” can save a lot of debugging time in P3

Advice

- Grader claimed your code wouldn't die the way you said?
  - Try running your code in the P2 environment and see how it does die

Breakdown

90% = 67.5	3 students
80% = 60.0	16 students
70% = 52.5	23 students (52 and up)
60% = 45.0	10 students
50% = 37.5	0 students
<50%	0 students

Comparison

- Noticeably fewer “A's” than typical
- Also noticeably fewer “R's”

Implications

Score under 55?

- 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

Reminders

- Final exam will focus more on “design”
  - On this exam, most represented by cvars & rwlock_promote() - if both were trouble for you, be warned!
- To pass the class you must demonstrate proficiency on exams (not just project grades)