15-410

"Democracy is three wolves and a sheep...
...voting on what's for dinner."

Exam #1
Oct. 25, 2006

Dave Eckhardt

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 implemention

Examination will change to match

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

Outline

Question 1

Question 2

Question 3

Question 4

Question 5

Q1 –Short Answer

"Core concepts"

- Mode switch
  - Change of privilege level, so devices and VM can be accessed
    - » Kernel mode, user mode
  - Happens on "surprise" (interrupt, system call, exception)
- Context switch
  - When a processor stops running a runnable and starts running a different runnable
  - May be a *consequence* of a mode switch
  - Is not part *part of* of a mode switch
  - Does not *require* a mode switch
- Please understand how these are independent things

Q1 –Short Answer

“One for the road”

- Not something you need in 15-410
- Something increasingly part of the cruel world
- Memory barrier
  - An instruction which makes memory temporarily “pre-modern”
  - Roughly: flushes the write pipe
  - Less roughly: ensures all processors agree on the ordering of certain memory events
  - “Read barrier”, “Write barrier”, “Barrier”

Q2 –Out of Business

We provide find_work() code

- **...sensitive to “going_out_of_business” boolean**

(a) Fill in add_work()

- **Must cond_signal() a find_work() thread**
- **Probably shouldn't add when going out of business**

(b) Fill in go_out_of_business()

- **Must set the flag**
- **Must cond_broadcast() the find_work() threads!!**
- **Should really drain the queue**

This is a “how pieces fit together” question

Q3 – Traveling Traceback

The mission

- Port traceback() to run on top of reference kernel

One key obstacle

- “No SIGSEGV, msync(), write(), /proc”

(a) How to resolve?

- Need another way to presence-check/safe-copy memory!
  - That's what SIGSEGV, msync(), write(), /proc were for in P0
- Three approaches
  - new_pages() - some wrinkles, but an acceptable approach
  - “send out a thread” - to respond “ok” or die trying
  - There's actually another relevant syscall... hmm....

Q3 –Traveling Traceback

Notable variant on (a)

- "Add a new system call which checks memory presence"
  - Not what we were looking for ("P0 on top of reference kernel" is a situation where you can modify P0 but not modify the reference kernel)
  - But ok –as long as you explained how that system call would work!

(b) Show us your code for the %ebp ⇒ %ebp case

- Goal: ***safely*** fetch next %ebp (given one which was already safely fetched)
- That is, "do the induction step"

Design: what is the job, what primitives do I have?

Q4 –Getting Grilled

Story

- Grill party
- Some people want burgers with cheese, others don't
- Thread code

One amusing glitch

- The grill isn't supposed to catch fire –a case arm was missing in some exam versions (oops)

The punch line?

Q4 –Getting Grilled

Story

- Grill party
- Some people want burgers with cheese, others don't
- Thread code

One amusing glitch

- The grill isn't supposed to catch fire –a case arm was missing in some exam versions (oops)

The punch line?

- This wasn't a deadlock question

Q4 –Getting Grilled

Story

- Grill party
- Some people want burgers with cheese, others don't
- Thread code

One amusing glitch

- The grill isn't supposed to catch fire –a case arm was missing in some exam versions (oops)

The punch line?

- This wasn't a deadlock question
  - It was a *starvation* question!

Q4 –Getting Grilled

What's starvation?

- The system makes forward progress
- There isn't a deadlock (a cycle of ...)
- But a certain class of user/customer/thread/... indefinitely can't get its job done due to structural unfairness
  - A popular form: it's “much too hard” to get the locks you need
    - » Maybe because you need many more locks than others
    - » Maybe because there is a lock-ordering problem
  - This form: the system solves the “special case” only randomly

Q4 –Getting Grilled

The grill problem

- There is a narrow window when cheese-seekers can add cheese to a burger
- But the system is designed to alert cheese-seekers to the opening of that window
- Anybody can add cheese to any burger in the window, so it can happen arbitrarily often
- Insight: there is no “anti-cheese protection” in the system
  - It produces un-cheesed burgers *only by accident*
    - » As a function of load, not as a function of “fairness”
    - » The people who like cheese determine how often the people who don't like cheese succeed...guess what?

Q4 –Getting Grilled

Solving the grill problem

- Queue
  - Attractive theoretically
  - Possible “hot spot” in a large multi-processor system
    - » “Many” partygoers, “many” grill slots
  - A fine approach
- Embed cheese-refusers into system's goal structure
  - More flag bits

Open-ended question, graded gently

Q5 –Broken Mutual Exclusion

Key concepts

- Tracing through a race condition without printf()
  - A valuable skill
- Writing an execution trace which *clearly* shows the problem
  - Training for submitting a bug report to the development team for the other half of the product
- Getting the right name for which mutual exclusion requirement you found broken
  - This part was graded gently

Summary

90% = 67.5 10 students

80% = 60.0 18 students

70% = 52.5 17 students (52 and up)

60% = 45.0 6 students

<60% 1 student

Comparison

- **This is a roughly-typical mix for the mid-term**
- **More C's, fewer D's, fewer R's**

Implications

Score below 52?

- Figure out what happened
- Probably plan to do better on the final exam

Warning...

- To pass the class you must demonstrate reasonable proficiency on exams (project grades alone are not sufficient)
- See syllabus