15-410

"My other car is a cdr" -- Unknown

Exam #1
Oct. 15, 2008

Dave Eckhardt

Roger Dannenberg

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

Crash box

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

Synchronization

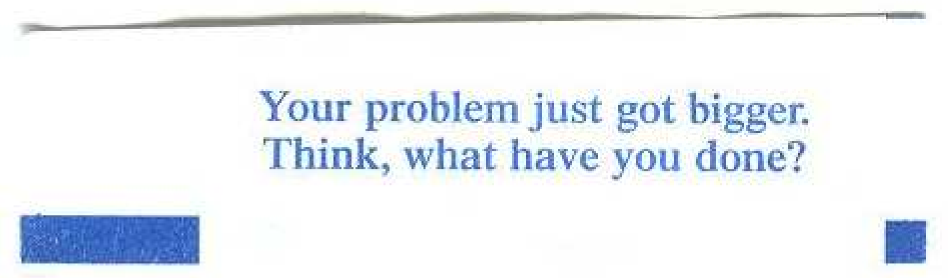
Debugging advice

- Last year as I was buying lunch I received a fortune

Synchronization

Debugging advice

- Last year 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)

Outline

Question 1

Question 2

Question 3

Question 4

Question 5

Question 6

Q1 –Reasons for using threads

Answers straightforward

- For at least the next 5 years we will live in a “multi-core world”
- For full credit, be sure not only to state an answer but to do so in a way which makes it clear you understand what the concepts mean

Q2 –Dining Philosophers

Part A (setup)

- Three philosophers, one pool of four chopsticks
- As with the homework question—though for a totally different reason—you can't have a cycle in the wait graph
    - Note: you *can* have “hold & wait”
        - » acquire_one_chopstick() is called twice in a row
        - » If it returns immediately once and blocks once, that is *exactly* “waiting while holding”, i.e., hold & wait!

Part B (“Along comes an octopus...”)

- Assume you have a deadlock. How many chopsticks are held (what is the *largest* number possible)?

Q2 –Dining Philosophers

Discussion

- If you firmly understand deadlock, the question might be a little novel but shouldn't be tough
- If there is a deadlock question on the final it will probably be “different from this”.

Q3 –"Philosophers Dining"

The mission

- Write a chopstick-pool object
  - Involves locking and synchronization
  - Not too hard (actually, it's a "trick question")

Common issues

- Confusion about pointers and malloc()
  - Message from the universe: it is really time to have a solid grasp on this issue. As necessary, see course staff. Really.
- "Paradise lost"
  - If somebody can revoke your happiness, you'd better check.
    - » This is a key concept.
    - » Review lecture if necessary.

Q4 –Critical-section algorithm

“Dannenberg's Algorithm”

- *Zero* of the three critical-section properties hold!
- Failure of “bounded waiting” is a little tricky to show
  - But then, that property is frequently hard work
  - Warning: do *not* show “lock is not acquired in FIFO order” - that is not a requirement!
  - Three threads is enough
- Failures of mutual exclusion and progress can be shown via *short* traces
  - Two threads “inside the gate” suffice; traces are short

Advice

- Being able to write a short trace showing how code goes awry is important. You might have some code “like that”.

Q5 –Nuts & Bolts

head.S double-fault handler

- “Stuff values into registers” - why?
- “Set stack pointer to something” - why?
- “Set stack pointer to something *else*” - why??

Common misconceptions

- %SS, etc., are “spare” or “temporary” registers
  - Sadly, they are critical for correct execution. You can't ignore them.
- %EAX contains only “return values”
  - *Most* of the time it's just another caller-save register
- Subtracting from %ESP makes stack space available
  - It *consumes* space (for a useful purpose?)

Q6 –Design

To SIGSEGV or not to SIGSEGV?

- Best answers are about costs
- Kernel entry/exit is a relevant cost... but there are others
  - Memory-copy operations are costly
  - Disk operations are more costly
  - Costs should be combined according to a model of how execution proceeds
- Various costs are associated with code
  - Portability was mentioned sometimes (though sometimes in ways we found confusing)
  - Modularity, predictability, ...

Graded fairly gently

Breakdown

90% = 67.5	27 students
80% = 60.0	25 students
70% = 52.5	13 students (52 and up)
60% = 45.0	4 students (44 and up)
50% = 37.5	2 students
<50%	3 students

Comparison

- Scores are higher than typical (3-5 points)

Implications

Score below 70%?

- Something went *really* wrong!
- You are *strongly* advised to debug the situation
- To pass the class you must demonstrate reasonable proficiency on exams (project grades alone are not sufficient)
- See syllabus

Above 70%?

- Probably a 50/50 chance that final-exam score will be one grade lower...