15-410

"My other car is a cdr" -- Unknown

Exam #1
Mar. 16, 2009

Dave Eckhardt

Synchronization

Checkpoint 2 –Friday

- 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

Google "Summer of Code"

- http://code.google.com/soc/
- Hack on an open-source project
  - And get paid
  - And quite possibly get recruited

CMU SCS "Coding in the Summer"

Synchronization

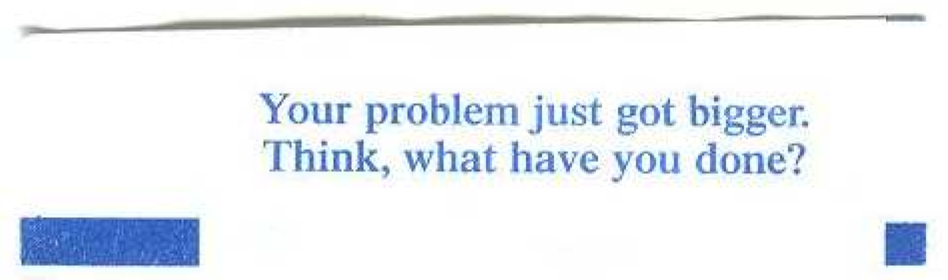
Debugging advice

- 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

Q1a –"Blocked"

Many had trouble here

- This is a *key* concept
- Blocked most vitally means "*executing zero instructions*" (until a specific state change happens later)
  - It is the state which results from "voluntary descheduling"
- Blocked is *not* spinning, yielding, whistling, etc.
- This is a difficult distinction...
- ...but it is *very important* for your kernel
  - Sometimes some of your threads should block
  - If instead they spin, yield, whistle, etc., your kernel will lose points
    - » Maybe a lot!

Q1b –“Asynchronous Thread Cancellation”

Most-common mistake: defining the *other* thing

- No big deal (1 point)

Other misconceptions

- Cancellation is what an angry kernel does
  - No, it's an operation invoked within an application, e.g., pthread_cancel()
- This somehow involves wait() or thr_wait()
  - No... cancellation is used exactly when you don't want to wait.

Q2 –Interrupts/PUSHA

This is an "execution environment" question

- (of the "hardware" variety)

Q2a –Can an interrupt stop a PUSHA "in the middle"?

- The general answer, across all instruction sets, is "no".
- If you stop an instruction "in the middle", you need to write down not just the program counter but a "fractional program counter", meaning a checklist of which *parts* of the instruction were completed so you don't re-do them.
- Two exceptions
  - On x86, mysterious string instructions, starting with REP
  - A few architectures have "imprecise interrupts"
    - » This is painful and unpopular
- "Interrupt pending?" is asked *between* instructions

Q2 –Interrupts/PUSHA

Q2b –What about a page fault? Protect via "`CLI`"?

Three concepts in play

- A page fault is not an interrupt, so `CLI` can't help
  - Faults (and traps) are "synchronous" to the instruction stream: if the instruction gets to execute, then the fault/trap will result.
- `PUSHA` *can* generate a page fault...
  - But if/when it does, it does so *before* starting to work...
    - » So `PUSHA` doesn't need "protection" to work correctly.
- Regardless, there are no page faults in the P1 run-time environment!

Q3 –Semaphore Problem

Problem statement

- Add `sem_broadcast()` to semaphores: “wake up all threads waiting on a semaphore”.
- What's wrong with this code?

Two undeniable utter failures

- `sem_wait()`/`sem_signal()` suffer from “paradise lost”
- `sem_wait()`/`sem_broadcast()` deadlock
- It is to your advantage to train yourself to see these errors in code... such as your partner's code!

Be careful to write a *short, compelling* trace

Q4 –Rendezvous

The mission

- Write a rendezvous object
  - Involves locking and synchronization

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.
  - In this question, the “third thread” was generally the first thread, “coming around again too quickly”

Q4 –Rendezvous

Other issues

- Deadlock, various race conditions, viewing unlocked data

Having a *plan* is critical

- "3-state" version
  - Object contains no value, 1st value (not 2nd), 2nd (not 1st)
  - That third state is important, becomes here comes the next thread!
- "2-pointer" version
  - Each party provides a pointer, second party does the swap
  - At that point the object is "empty" - don't need a third state
- "2-slot" version often worked; 2-count semaphore, too
- With a plan, you can check that other paths don't happen
  - Otherwise, it's easy to get some cases, miss some

Q5 –Process Model

Declare some variables which are named by region

- `const char rodata[] = “Can't touch this!”`
  - That string will live in the “read-only data” region
- If you can't list the other interesting regions or can't figure out how to get a variable “into” one, this is a problem with your understanding of the C run-time environment
  - ...which will hamper debugging your kernel...
  - Note that the C run-time environment is simpler than that of almost any other language... you should really “get” this before leaving this class!

Breakdown

90% = 67.5	3 students
80% = 60.0	34 students
70% = 52.5	28 students (52 and up)
60% = 45.0	13 students (44 and up)
50% = 37.5	11 students
<50%	0 students

Comparison

- Scores are a bit under typical (3-5 points)

Implications

Score 45..50?

- Figure out what happened
  - Not enough textbook time?
  - Not enough reading of partner's code?
  - Lecture examples “read” but not grasped?
- Probably plan to do better on the final exam

Score below 45?

- Something went *very* 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 (project grades alone are not sufficient)