15-410

"My other car is a cdr" -- Unknown

Exam #1
Oct. 24, 2016

Dave Eckhardt

Dave O'Hallaron

Synchronization

Checkpoint 2 – Wednesday, in Wean 5207 cluster

- Arrival-time hash function will be different

Checkpoint 2 - alerts

- Reminder: context switch ≠ timer interrupt!
  - Timer interrupt is a *special case*
  - Looking ahead to the general case can help you later
- 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

Synchronization

Asking for trouble?

- If your code isn't in your 410 AFS space every day, you are asking for trouble
  - Roughly 2/3 of groups have blank REPOSITORY directories...
- 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
- GitHub sometimes goes down!
  - S'13: on P4 hand-in day (really!)

Synchronization

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

Book report!

- **Try not to forget about it until the last minute!**

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

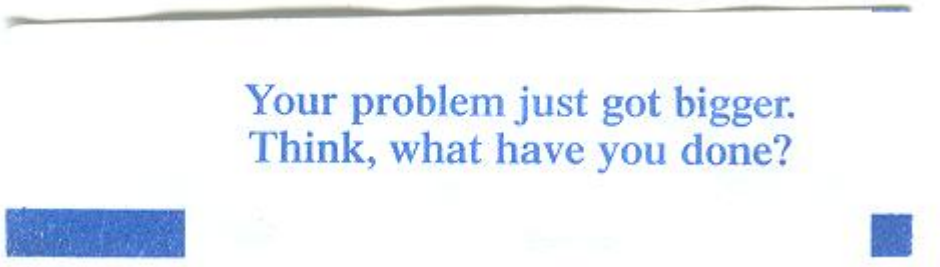

Image credit: Kartik Subramanian

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 could be more stuff (~100 points, ~7 questions)

“See Course Staff”

If your exam 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

Q1a – Pebbles “tasks” vs. “threads”

Purpose: Show a clear understanding of the distinctions between tasks and threads

- Task is a container for resources like separate virtual address space, IPC endpoints, and threads.
  - IPC endpoints: vanish() and wait()
- Thread is a schedulable register set
  - Shares the task address space with other threads
  - Cannot access the address space of another task
  - Usually operates on its own stack

Outcomes

- Generally reasonable answers
- Don't confuse Pebbles tasks with Linux processes
  - Linux “process”/“thread” distinctions are “odd”

Q1b – “thread safety”

Purpose: Show a clear understanding of what makes a function thread-safe

- A thread-safe function can be called by multiple threads and still produce correct results
- Properties
  - Protect all accesses to shared variables (e.g., mutex)
  - Doesn't maintain state in static local variables (e.g., gethostent(), strtok())

Outcomes

- Thread-safety != re-entrant (thread-safe functions can have and share internal state)
- Be careful about conflating “thread-safe” with “ideal mutex”
  - A function could be thread-safe but not provide “bounded waiting”
- Only 1 student discussed static variables

Q2 – Faulty Condition Variables

What we were testing

- Depth of understanding of cvar atomic-block problem
- Or: ability to find a race condition split between two small-ish functions

Good news

- Many people figured out that a thread gets stuck because something happens too early

Bad news

- Some people had alarming ideas about semaphores
  - "Buffering" the availability of old events/deposits is a key semaphore job!
  - People will expect you to know how semaphores work
- Some traces were longer than necessary (not necessary to show execution of entire program if you carefully specify that your trace starts with some later state)

Q2 – Faulty Condition Variables

Fix

- **A clear understanding of the problem suggests a *very* simple fix**

Q3 – Deadlock

Parts of the problem

- Find the deadlock
- Suggest a fix

Results – finding

- Most people correctly described a reachable deadlock
- Roughly 1/3 found a minimal-thread-count deadlock
  - The problem structure strongly implies how many that is
  - Some people used 1 extra thread (ok)
  - Some people didn't attempt an explanation of how many threads are necessary

Most-common mistakes

- Insufficient justification of a claimed deadlock state
- Impossible traces (too many copies of a book)
  - » Writing a clear trace is an important mental tool

Q3 – Deadlock

Results – fixing

- Many solutions are plausible and received credit
- Terminology note: preemption is taking a resource from somebody else

Overall

- While analysis, thought, and tracing were required, this was a mostly straightforward question

Q4 – "Condition Locks"

Question goal

- Slight modification of typical "write a synchronization object" exam question

General conceptual problems

- "x() takes a pointer" does *not* mean "x() must call malloc()"
- Assigning to a function parameter changes the *local copy*
  - It has no effect on the calling function's value
  - C isn't C++ or Pascal (luckily!)
- See course staff about any general conceptual problems revealed by this specific exam question

Q4 – “Condition Locks”

Alarming things

- Spinning is *not ok*
- Yield loops are “arguably less wrong” than spinning
  - Motto: “When a thread can't do anything useful for a while, it should block; when a thread is unblocked, there should be a high likelihood it can do something useful.”
- cond_wait() really must accept a *locked* mutex
- cond_wait() 97.3% must be invoked inside an if()
  - “Unconditional condition wait” is roughly as bad as it sounds

Q4 – “Condition Locks”

Most-common issues

- If you cond_signal() one random thread, and that thread can't proceed, what (doesn't) happen next?
- If N threads cond_broadcast() a pool of N threads, that's N^2 thread activations but probably only N successes
  - If there is no feasible way to figure out which thread(s) should be awakened, that may be the only option
  - In this problem it *is* possible to “figure out” – that approach got more credit
- If threads will be “stuck for a while”, try to use something other than a mutex (why?)

Q5a – Nuts & Bolts: “capture %eip”

Purpose: Think about using familiar asm instructions in unfamiliar ways.

- Can be solved with one or two lines of code
- Two approaches
  - Use a (very) common instruction that manipuates %eip
  - Use linker's ability to assign absolute addresses to symbols

Outcomes

- Reasonable distribution of scores
- Not legal to use %eip as an instruction argument (x86-32)
- Partial credit given for some kind of valid %eip manipulation

Q5b – Nuts & Bolts: variable locations

Purpose: Review your understanding of a basic idea.

- 2 in BSS
- 1 in data
- 3 in stack (2 in a special place)

Outcomes

- This should be an easy/fast question
  - For the rest of the semester you will spend a lot of time debugging stacks
- But there were very few perfect scores

Breakdown

```
90% = 63.0    8 students (69/70 is top)
80% = 56.0    9 students
70% = 49.0   10 students
60% = 42.0    4 students
50% = 35.0    3 students
<50%          4 students
```

Comparison

- Median grade was 75%, so this probably wasn't a “killer exam”

Implications

Score below 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?
- It is important to do better on the final exam
  - Historically, an explicit plan works a lot better than “I'll try harder”
  - Strong suggestion: draft plan, see instructor

Implications

Score below 40?

- Something went *dangerously* wrong
  - It's *important* to figure out what!
- Beware of "triple whammy"
  - Low score on *all three* "middle" questions
    - » Those questions are the "core material"
    - » Strong scores on Q1+Q5 don't make up for serious trouble with core material
- Passing the final exam may be a *serious* challenge
- *Passing the class may not be possible!*
  - To pass the class you must demonstrate proficiency on exams (not just project grades)
- See instructor

Implications

"Special anti-course-passing syndrome":

- Only "mercy points" received on several questions
- Extreme case: *no* question was convincingly answered
  - It is *not possible to pass the class* if both exams show no evidence that the core topics were mastered!