15-410

"My other car is a cdr" -- Unknown

Exam #1
Mar. 6, 2017

Dave Eckhardt

Dave O'Hallaron

Synchronization

Checkpoint schedule

- Wednesday during class time
- Meet in Wean 5207
  - If your group number *ends* with
    - » 0-2 try to arrive 5 minutes early
    - » 3-5 arrive at 10:42:30
    - » 6-9 arrive at 10:59:27
- Preparation
  - Your kernel should be in mygroup/p3ck1
  - It should load one program, enter user space, gettid()
    - » Ideally lprintf() the result of gettid()
  - We will ask you to load & run a test program we will name
  - Explain which parts are “real”, which are “demo quality”

Synchronization

Book report!

- **Hey, “Mid-Semester Break” is just around the corner!**

Synchronization

Asking for trouble?

- If your code isn't in your 410 AFS space every day, you are asking for trouble
  - Roughly 1/2 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

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

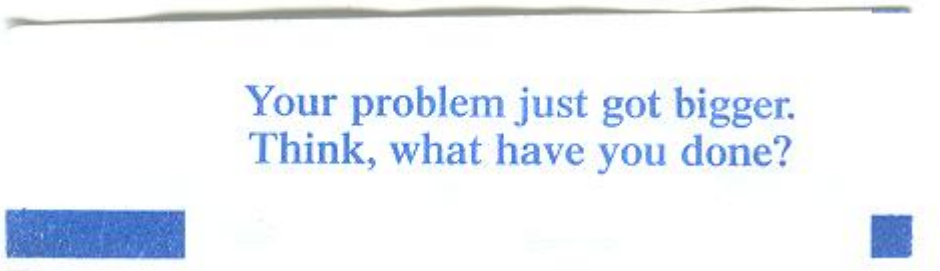

Image credit: Kartik Subramanian

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 – "Can I assume ___?"

Purpose: demonstrate familiarity with key mental tools for design

- These tools will be more necessary in P3 than P2
- And maybe even more necessary after P3!

Outcomes

- Generally reasonable answers

Q1b – IDT-entry contents

Purpose: Demonstrate understanding how an interrupt / trap handler is specified

- Fundamental: where is the code for the handler?
  - x86 special detail: "program counter" has two parts: %eip and %cs
- Other features are mostly "x86 details"

Outcomes

- Answers generally good
- If you got a low score on this, probably address the issue: interrupts/traps/faults/exceptions are important material for this class

Q2 – Critical-Section Algorithm

What we were testing

- Primarily: ability to find and show race conditions
- Also: knowledge of what a c.s. algorithm should do

Good news

- Many people got a perfect score (nearly half the class)

A common problem

- Trace executes loop body from top to bottom once but doesn't go back and do it again

A conceptual problem

- "If the scheduler permanently quits running one of the threads, it will never acquire the lock"
  - True, but *no* critical-section algorithm can solve the "some thread runs at zero speed" problem, so this isn't a valid criticism

Q3 – “Pair Matcher”

Administrative announcement

- Question was advertised as 15 points (true)
- Part A was advertised as 5 points and Part B was advertised as 15 points (false)
- Actual values: A⇒3 B⇒12

Q3 – “Pair Matcher”

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

Q3 – "Pair Matcher"

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."

"Will not work out well"

- *Any* examination of part of a multi-part data structure without holding a lock is *very* likely to cause a problem
  - Unlocked "`if (stage == 0)`" – it can change!
  - Unlocked "`return sp->result`" – it can change!

Q3 – “Pair Matcher”

“Generally try to avoid”

- “Evil third thread syndrome”
  - Generally: some thread is signalled but somebody else gets the lock first, “Paradise Lost” ensues
  - In this problem it's “evil second pair of threads”
  - This is an important phenomenon to avoid, so if you ran into it please study it carefully

Other general advice

- It's a good idea to trace through your code and make sure that at least the simplest (“good”) case works without threads getting stuck

Q3 – "Pair Matcher"

Solutions with queues often didn't work out well

- Most queue solutions where the queue could possibly contain more than one element ran into some sort of trouble
- If a queue never contains more than one item then a queue isn't needed

Awakening the *right* number of threads is important

- Awakening too many (`cond_broadcast()`) can be a big efficiency problem
- Awakening too *few* causes progress failures
- This problem was harder than typical in this regard
  - We saw a lot of progress failures

Q3 – “Pair Matcher”

“Too many locks”

- Most solutions with too many locks (4, 5, …) got into some sort of trouble
- Even correct solutions with too many locks were hard to understand; locking isn't super-cheap
- So a minor deduction was applied

Outcome

- ~40% of the class did well
- ~30% of the class had a lot of trouble
- Note that this was easier than a typical “write synch object” question

Q4 – Deadlock

Parts of the problem

- Find the deadlock
- Suggest a fix

Results – finding

- Most people correctly described a reachable deadlock

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

Q4 – Deadlock

Results – fixing

- This was hard!
- The most common "just flip things around" solutions *caused* some other problem (race/deadlock)
- Most "just use one giant lock" solutions didn't do well
  - A giant lock is rarely a good solution
  - If what's inside the lock is `sleep()` or O(N) operations, consider other approaches!

Notes about approaches

- We provided a "status" field that we didn't really use... hmm....
- Some people changed the type of what was enqueued on some queues
- Some people added some cvars (plus a cute trick)

Q4 – Deadlock

Outcomes

- Around 1/6 of the class got under 70% (14/20)
  - That probably indicates something should be addressed

Q5 – Nuts & Bolts: Broken Adder

Purpose: Think about integer arithmetic

- At a high level: implement 32-bit add with 16-bit add plus shifts
- Why? Debugging P3 will require staring at bits to figure out what's wrong... this is a good way to figure out if some practice is needed

Key Issues

- Fundamentally, a loop is not needed
  - There were some "not so great" loop solutions and one "really alarming" loop solution
- Carry is a function of *all* lower-order bits (you can't sample just one or two bit positions)
- Watch out for callee-save registers when using assembly code

Q5 – Nuts & Bolts: Broken Adder

Outcomes

- Around 75% of class “passed” (7/10)
- There were some *very* low scores

Breakdown

```
90% = 63.0    8 students (70/70 is top)
80% = 56.0   24 students
70% = 49.0   22 students
60% = 42.0    6 students
50% = 35.0    3 students
<50%          0 students
```

Comparison

- Median grade was 80%, so this wasn't a “killer exam”
  - (Median grade last semester was 75%)

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 42?

- 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!