15-410

"My other car is a cdr" -- Unknown

Exam #1
Mar. 5, 2018

Dave Eckhardt

Brian Railing

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 you aren't using source control, that is probably a mistake
- If your code isn't in your 410 AFS space every day, you are asking for trouble
  - GitHub sometimes goes down!
    - » S'13: on P4 hand-in day (really!)
  - 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

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

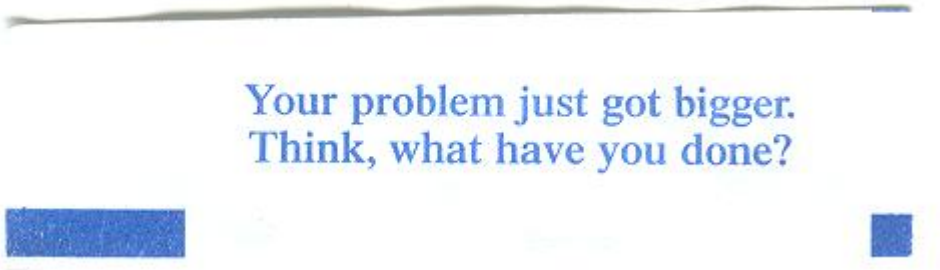

Image credit: Kartik Subramanian

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 – Recursion

Purpose: demonstrate suspicion of a dangerous practice

- Baseline definition: self-calling (maybe via another function: mutual recursion)
- Key ideas: consumes stack space, *stack space is tight in most kernel run-time environments*

Outcomes

- Many reasonable answers
- Good scores were not rare

Q1b – "Paradise Lost"

Purpose: Demonstrate understanding of a concurrency anti-pattern

- Key points
  - A condition was true; then revoked; expected to be true later
  - It is possible to be unlucky and observe while revoked
  - Can often be fixed by replacing "if" with "while"

Outcomes

- Many solid answers
- Some alarming answers
  - "Something involving 3 threads and dequeue()"
  - "Paradise Lost == TOCTTOU == race condition"
    - » Arguably there is a subset relationship
    - » But causes and fixing are *very* different
      - "Add locks" != "Change 'if' to 'while'"

Q2 – Pair-matcher race

What we were testing

- Find a race condition
- Write a convincing trace

Good news

- 2/3 of the class got 8/10 or better (it was an easy race)

Other news

- 1/3 of the class got 9/10 or 10/10... not a lot

Common issues

- Omitting part of the trace, e.g., unlock
- Not making state changes clear
- Not stating the problem in words before writing the trace

Q3 – "Reducing deadlock"

Question goals

- Diagnose a deadlock situation, based on deadlock principles
- Design (state) a solution

Good news / bad news

- A/B: 20%
- A/B/C: 42%

Observations

- The deadlock was not easy to find
- Finding it without applying principles was probably infeasible

Q3 – “Reducing deadlock”

Approach

- “Just trying out traces” isn't likely to work
  - Too many threads are required
  - Threads have too many options

Q3 – “Reducing deadlock”

Approach

- “Just trying out traces” isn't likely to work
  - Too many threads are required
  - Threads have too many options
- Part (a) – “list deadlock elements” – is an opportunity
  - There are multiple hold&wait sites in the code (~5)
  - A detailed list enables careful evaluation of which sites can be involved in a cycle
  - Some things look suspicious but can be proven to be safe

Q3 – “Reducing deadlock”

Approach

- “Just trying out traces” isn't likely to work
  - Too many threads are required
  - Threads have too many options
- Part (a) – “list deadlock elements” – is an opportunity
  - There are multiple hold&wait sites in the code (~5)
  - A detailed list enables careful evaluation of which sites can be involved in a cycle
  - Some things look suspicious but can be proven to be safe
- Once you know how/where threads can deadlock, getting the necessary setup is a much simpler problem
  - Partial credit was assigned for “setup” problems

Q3 – “Reducing deadlock”

Notes

- One frequent mistake asserted a 3-thread deadlock that requires the reservation system to be broken..
  - But we don't think it is
  - This was a partial-credit case too
- The 0'th operand is special, so handling it in a trace requires care

Alarming

- Some answers relied on misunderstanding of how semaphores work (“early” signals are *stored*)
  - This is an important thing to clear up!
- Some answers asserted patterns of acquire() and release() that ignored how the code in operator() calls them

Q4 – Abortable condition variables

Question goal

- Slight modification of typical "write a synchronization object" exam question
- This was toward the easier end of questions in this class

Alarming core issue

- When you signal a thread because you want it to run, it will run right away (before any other thread)
  - Note that Q2 was about this being false!

Less alarming but common

- Excessive use of the "world mutex" passed into the acv results in excessive serialization

Q4 – Abortable condition variables

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 – Abortable condition variables

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

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."
  - Special case: mutexes should not be held for genuinely indefinite periods of time

Q4 – Abortable condition variables

Important general advice!

- It's a good idea to trace through your code and make sure that at least the simplest cases work without races or threads getting stuck

Other things to watch out for

- Memory leaks
- Memory allocation / pointer mistakes
- Forgetting to shut down underlying primitives
- Parallel arrays (use structs instead)

Q4 – Abortable condition variables

Outcome

- ~35% of the class "did ok" (scored 70% or better)
- There were a *lot* of 8/20 (== 40%), some below that"

Q5 – Nuts & Bolts: exec() vs. registers

Question goals

- Test understanding of process model
  - fork(), exec(), how values get into registers

Expectations – Part A

- Descriptions of how the non-specified registers get initialized naturally by the new program
  - Straightforward: %eax, %ebx, etc.
  - Important case: %ebp
    - » Need not be initialized by exec(); handled by prologue

Expectations – Part B

- Description of how a program could launch with access to information it should not know

Q5 – Nuts & Bolts: exec() vs. registers

Alarming claims – Part A

- "exec() is a function" - discussion based on caller-save and callee-save registers
  - But exec() is very much not a function
  - And the question's focus was on getting the right values into registers *before* the first actual C function is called
- "The new program doesn't need any values from the old program"
  - But part of exec()'s job is providing values from the old program to the new program

Alarming claims – Part B

- If %ebp is not initialized, the program/kernel might crash

Breakdown

```
90% = 63.0    0 students
80% = 56.0    1 student   (58/70 is top)
70% = 49.0   11 students
60% = 42.0   11 students
50% = 35.0    7 students
<50%          3 students
```

Comparison

- Top score was low, so this wasn't an easy exam
- Median grade was 67%, so this wasn't an easy exam

Implications

Some “curving” seems likely

- Details TBD

Score below 47?

- 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!**

Implications

Special note for S'18

- **If you didn't get 13/20 on either Q3 *or* Q4 we should probably talk**