15-410

"My other car is a cdr" -- Unknown

Exam #1
Mar. 15, 2021

Dave Eckhardt

Synchronization

Book report!

- This your approximately-mid-semester reminder about the book report assignment

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 40% 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
  - Don't forget about CC=clang / CC=clangalyzer
- Running your code on the crash box may be useful
  - But if you aren't doing it fairly regularly, the first "release" may take a *long* time

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

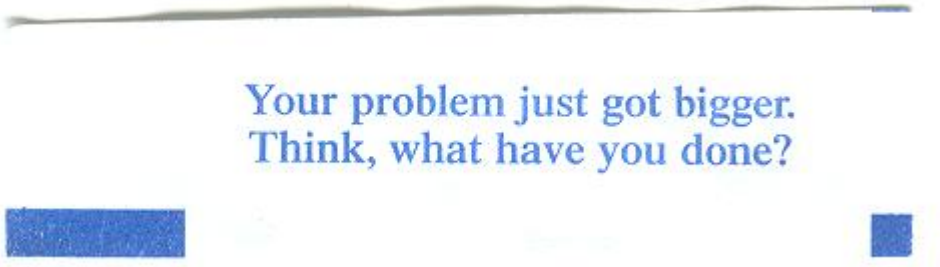


Image credit: Kartik Subramanian

A Note for Posterity

The S'21 mid-term exam occurred during COVID-19

This was an atypical exam

- "2 hours of material"
- 4-hour exam session
- Personal start time in a 36-hour window
- Open book, open notes (including submitted P0/P1/P2 code)
  - Honor system
- Reduced weight at the end of the semester

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

What will that mean for the final exam?

- Current thinking (based on F'20)
  - Final exam will be very similar to mid-term exam
    - » Open book/notes, floating time, not many questions
  - May well have more design focus, more kernel focus
- Early advice
  - “Attend” lectures, do readings
  - Review your code and your partner's code
  - Review ink comments from the course staff

“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.**

...though it might instead indicate a complex subtlety...

- **...which we believe will benefit from personal counseling, not just a brief note, to clear up.**

“See Instructor”...

- **...means it is probably a good idea to see an instructor...**
- **...it does not imply disaster.**

“Low Exam-Score Syndrome”

What if my score is really low????

- **It is frequently possible to do *dramatically* better on the final exam**
- **Specific suggestions later**

Outline

Question 2

Question 3

Question 4

Question 5

Q2 – Three kinds of error

Purpose: demonstrate grasp of a robustness practice

- Hopefully P2 involved careful error handling
- Hopefully P3 will involve careful error handling
- "Robust code is *structurally different* than fragile code"
- P3 requires not just code but *structurally non-fragile code*.

If you were lost on this question...

- We had a lecture on this topic (February 19)
- Other "odd" lectures to possibly review
  - Debugging, Questions
  - #define, #include
  - We expect you to know *and apply* all of this material

Q2 – Three kinds of error

Alarming problems (conceptual)

- "Design flaw" is not one of the "three kinds of error"
  - "Rewrite code" is not something that goes in {…}
- "Lock contention" is not one of the "three kinds of error"
  - Lock contention is not visible to the code while it's running
    - » So code doesn't really respond to lock contention
    - » Code *authors* need to profile and rewrite

Q2 – Three kinds of error

Alarming problems (practice)

- "return;" from a void function
  - That is *covering up* a problem, not *handling* it
- yield loop
  - Hoping somebody else can solve the problem won't work well if nobody does
  - "Hold & yield" is basically "hold & wait"...uh-oh...
- silent vanish
  - This is not supportive of anybody fixing anything

Q2 – Three kinds of error

Suggestions

- Try to have a centralized reporter
  - Java, Rails, … produce stack traces
    - » Useful for many errors
  - The Pathos reference kernel produces register dumps
    - » Useful for many errors
- Try to have a good invocation pattern
  - assert(0) is not a very good invocation pattern

Q2 – Overall

Scores

- 50% of the class got 4/4 (100%)
- 33% got 3/4

Q3 – Register Dump

Question goal

- Stare at a register dump and form a plausible hypothesis
  - 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

Hint

- Two things that should be vastly different aren't

Common issues

- It is necessary to say *why/how* a wrong register leads to an exception
  - Since there was a fault, it ought to be possible to say why *some particular instruction* failed to execute

Q3 – Register Dump

Specific issues

- Copying values between registers can't page-fault
  - Page faults happen to memory references
- %eip isn't really a register
  - Most architectures since the 1980's don't allow you to name the program counter
- Some code examples would have faulted too early
- Mis-alignment may not result in a user-visible fault
  - x86-32: completely optional
  - Other plaftorms: exception handler may secretly fix

Q3 – Overall

Scores

- **~35% of the class scored 4/4**
- **~35% of the class scored 3/4**

Q4 – Synch or Swim

What we were testing

- Find a deadlock (important skill)
- Write a convincing trace (demonstrates understanding)

Good news

- ~60% "got an A" (14/15 or 15/15)
- ~25% "got a B" (12/15 or 13/15)
- The first question (10 points) went well for almost all students
  - So lots of people can identify and trace a simple two-thread deadlock

Q4 – Synch or Swim

Noticeable issues

- Omitting too many lines of trace
  - A very terse trace might summarize very-different executions (one deadlock, one not)
    - » That does not clearly demonstrate understanding

Alarming issues

- Deadlock definition does not mathematically require multiple threads
- Deadlock definition *does* require the four ingredients
  - Waiting for an event that will never come is not a deadlock

Q4 – Synch or Swim

Warning

- Many deadlock questions have #threads != 2
  - It may be necessary to reason from principles
    - » Scrutinize all of the hold&wait sites

Q5 – Testing Semaphores

Question goal

- *Atypical* variant of typical "write a synchronization object" exam question
  - Make sure you can block and unblock threads without things going wrong due to race conditions
- Writing test code is hard!

Hint

- It is hard to ensure/determine that a thread is blocked
  - Ensuring the relative ordering of two blockings is harder

Q5 – Testing Semaphores

Common issues

- "X; sem_wait(&s);"
  - ∀x: a timer tick can happen between x and sem_wait()
- "sem_signal(&s); sem_signal(&s);"
  - The first-awakened thread might not run very long before a timer tick causes the second thread to run instead

Q5 – Testing Semaphores

Common issues

- "X; sem_wait(&s);"
  - ∀x: a timer tick can happen between x and sem_wait()
- "sem_signal(&s); sem_signal(&s);"
  - The first-awakened thread might not run very long before a timer tick causes the second thread to run instead

Democracy

- If your solution relies on "50%+1 majority rule", you are making a strange assumption about what counterexamples imply...
  - » If something looks "98% FIFO", it *may* be 100%
  - » If something looks "51% FIFO", it also looks quite a bit non-FIFO

Q5 – Testing Semaphores

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
- Maybe figure out which operation is “the hard one” and pseudo-code that one before coding the easy ones?

Other things to watch out for

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

Time

Our target: “2 hours of exam content”

- **Provided: 4 hours of exam time**

Time

Our target: "2 hours of exam content"

- **Provided: 4 hours of exam time**

Observations

- **Min: 1.65 hours**
- **Median: 3.8 hours**
- **Max: 4 hours**

Wow, almost nobody took under 2 hours?

Time

Our target: "2 hours of exam content"

- Provided: 4 hours of exam time

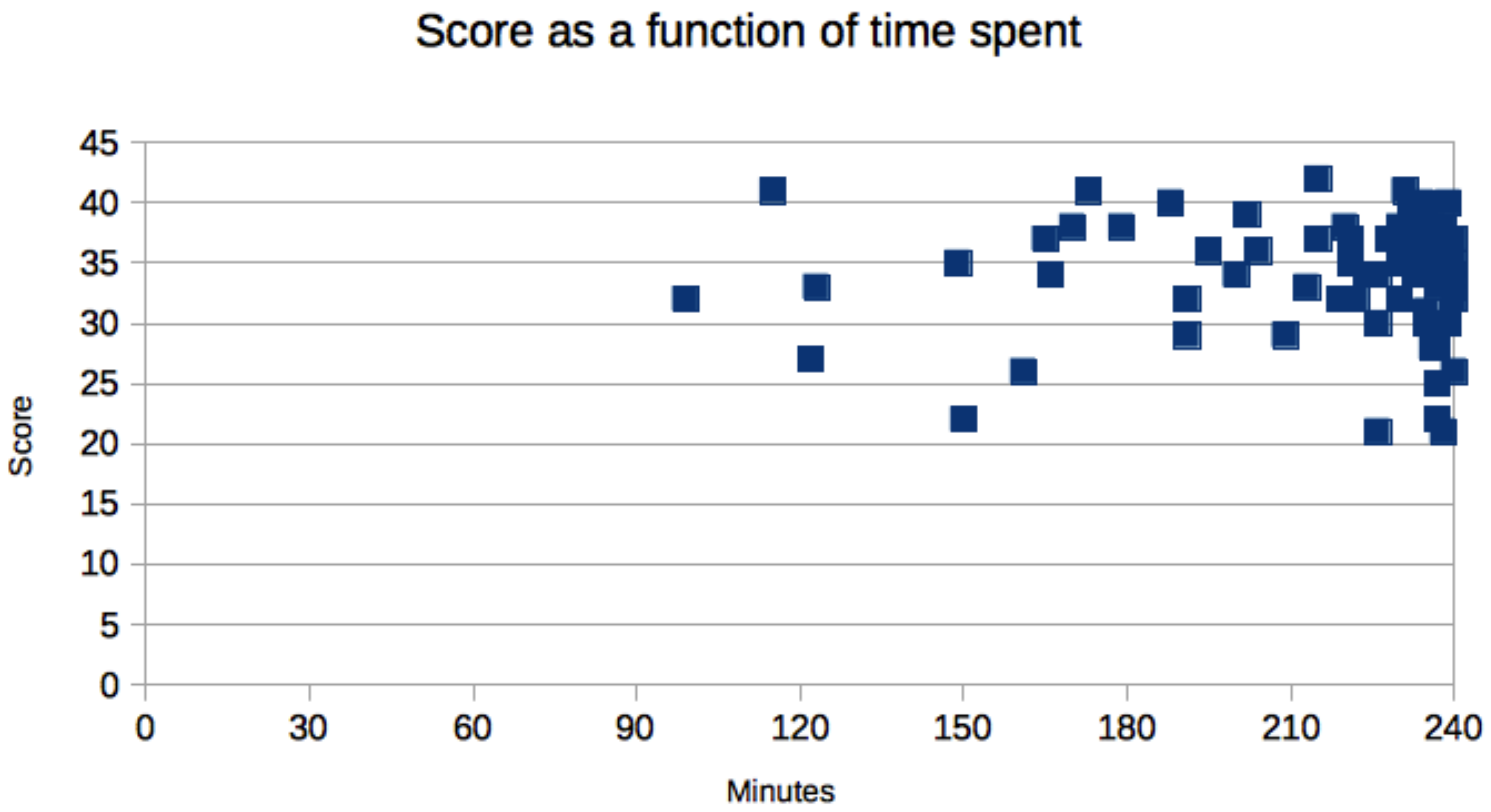
Observations

- Min: 1.65 hours
- Median: 3.8 hours
- Max: 4 hours

Wow, almost nobody took under 2 hours?

- "Correct, but wrong"

Time


Score as a function of time spent
Score
45
40
35
30
25
20
15
10
5
0
0
30
60
90
120
150
180
210
240
Minutes

Time

Our target: "2 hours of exam content"

- Provided: 4 hours of exam time

Observations

- Min: 1.65 hours
- Median: 3.8 hours
- Max: 4 hours

Wow, almost nobody took under 2 hours?

- Reasonable score range visible by ~2 hours
- Range at 4 hours is very similar to range at 3 hours
- Low-ish scores range from ~2.5 hours to 4 hours
- High-ish scores range from ~2.0 hours to 4 hours

Breakdown

```
90% = 38.7   11 students (39 and up)
80% = 34.4   24 students (35 and up)
70% = 30.1   23 students (30 and up)
60% = 25.8    4 students (26 and up)
50% = 21.5    5 students
40% = 17.2    0 students
<40%          0 students
```

Comparison/calibration

- Scores are high compared to a typical 410 mid-term
  - Low of 49%, median of 80%
- But they are *a lot* like last semester's mid-term!

Implications

Score below 30?

- 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*:
    - » Identify causes, draft a plan, see instructor

Implications

Score below 26?

- Something went *noticeably* wrong
  - It's *important* to figure out what!
- Passing the final exam could be a challenge
- *Passing the class may be at risk!*
  - To pass the class you must demonstrate proficiency on exams (not just project grades)
  - We don't know the format of the final exam yet, but a strong grasp of key concepts, especially concurrency, is important
- Try to identify causes, draft a plan, see instructor
  - Good news: explicit, actionable plans usually work well

Action plan

Please follow steps in order:

1. Identity causes
2. Draft a plan
3. See instructor

Action plan

Please follow steps in order:

1. Identity causes
2. Draft a plan
3. See instructor

Please avoid:

- "I am worried about my exam, what should I do?"
  - *Each person should do something different!*
  - The "identify causes" and "draft a plan" steps are individual, and depend on some things not known by us

Action plan

Please follow steps in order:

1. Identity causes
2. Draft a plan
3. See instructor

Please avoid:

- "I am worried about my exam, what should I do?"
  - *Each person should do something different!*
  - The "identify causes" and "draft a plan" steps are individual, and depend on some things not known by us

General plea

- Please check to see whether there is something we strongly recommend that you have been skipping because you never needed to do that thing before
  - This class is different