15-410

"My other car is a cdr" -- Unknown

Exam #1
Mar. 14, 2022

Dave Eckhardt

Synchronization

Checkpoint schedule

- Friday during class time
- Meet in Wean 5207
  - If your group number *ends* with
    - » 0-2 arrive at 11:44:27
    - » 3-6 try to arrive 5 minutes early
    - » 7-9 arrive at 11:27:30
- Preparation
  - Your kernel should be in mygroup/p3ck2
  - We are expecting everybody (even if not quite done)
    - » Unless you notify us by noon on Thursday

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 70% 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
  - Using a variety of compilers is likely to expose issues
- 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

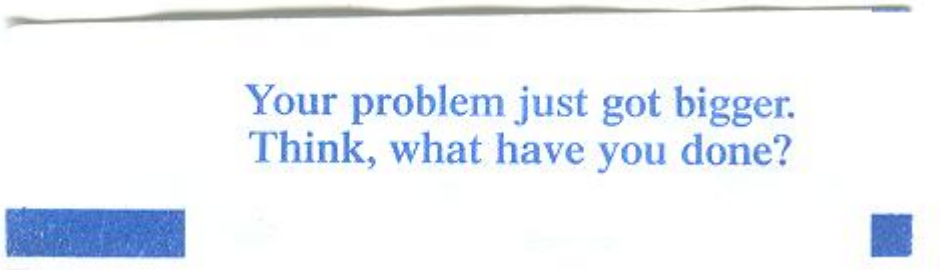

Image credit: Kartik Subramanian

A Note for Posterity

The S'22 mid-term exam occurred during COVID-19

This was *semi*-typical exam

- **Arguably one question shorter than typical**
- **But there was one "monster" question, so maybe not?**

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 (~85 points, ~6 questions)

Please Avoid Faint Pencil!

Some people wrote using pencil

- Some wrote with *very faint* pencil!
- Please do not do this on the final exam!
  - In any class!

“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 1

Question 2

Question 3

Question 4

Q1 – Short Answer

Three parts

- **“Can I assume _____?”**
- **Register dump**
- **readline() buffer**

Q1a – "I would like to assume..."

Basic idea: cost-benefit analysis

- What might you *gain* by assuming X?
  - Is it really a noticeable gain?
- What might you *lose* by assuming X?
  - If !X is wildly unlikely and easy to detect, then maybe the loss is "once in a long while I need to apologize and nobody will be mad"
  - If !X is plausible and would lead to disaster, then assuming X will plausibly lead to disaster

As system designers:

- You will need to "bake assumptions into your design"
- You should give real thought to which assumptions to "bake in"
- This pattern represents the most-basic "real thought"

Q1a – “I would like to assume...”

Most students did well

- **75% got 6/6 or 5/6**

Q1b – 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

Hints

- A critical register has a value which is *very* implausible
- A second register has that value too; this strongly suggests one or two specific scenarios

Q1b – Register Dump

Selected issues

- "Every page is writable xor executable" is a good idea, but not a law
  - Marking individual pages as non-executable wasn't even *possible* in x86-land before AMD shipped it in 1999 (Intel caught up in 2004?); ARM XN is ~2002
  - NX isn't implemented by Pathos, and if you look into what you'd need to do in your kernel you'll see why
- It's a good idea throughout P2 and P3 to be familiar with the Pebbles memory layout
- "X seems odd" does not mean that X will result in an exception; this sort of question requires a (plausible) specific exception reason

Q1b – Register Dump

Scores

- **~40% of the class scored 5/5 or 4/5**
- **~20% of the class scored *below* 3/5**

Q1c – readline() buffer contents

Question goals

- A bit of API design
- Thinking about readline() as a bridge from P1 to P3

Selected issues

- Note that system calls shouldn't do large amounts of extra work “to be nice”
  - If extra work is built into the kernel then people who don't benefit still must pay
- Various claims were made that aren't always true (e.g., related to strlen() after readline()) or were about implausible usages (strlen() after readline())

Scores

- ~90% of the class scored 4/4 or 3/4

Q2 – “Faulty Mutexes”

Which critical-section requirement doesn't hold?

- ? **Mutual exclusion**
- ? **Progress**
- ? **Bounded waiting**

Q2 – “Faulty Mutexes”

Which critical-section requirement doesn't hold?

- ✓ Mutual exclusion
- ✓ Progress
- ✓ Bounded waiting

Q2 – “Faulty Mutexes”

Which critical-section requirement doesn't hold?

- ✓ Mutual exclusion
- ✓ Progress
- ✓ Bounded waiting

“What a terrible mutex implementation!”

- – The author (Jin Lee)

Q2 – “Faulty Mutexes”

Which critical-section requirement doesn't hold?

- ✓ Mutual exclusion
- ✓ Progress
- ✓ Bounded waiting

“What a terrible mutex implementation!”

- The author (Jin Lee)

Scores

- Many people did very well (75% did 8/10 or better)
- Many people got lost in the jungle (20% under 6/10)
  - Remedying whatever went wrong here will be fruitful

Q2 – “Faulty Mutexes”

Problematic concepts

- “This mutex doesn't handle a thread locking and then dying!”
  - True... but essentially none do
- “This mutex doesn't handle an evil thread which randomly calls unlock() while it doesn't hold the lock!”
  - True... but …
- “This mutex doesn't work if thread 13 gets halfway through locking but the scheduler never runs it again”
  - It is very difficult to write locks (or critical sections!) if some thread can run at zero speed
- Meanwhile, this artifact fails *when used correctly*!

Q2 – "Faulty Mutexes"

Problematic traces

- Traces should generally not show multiple effects on the same line
- "while(...)"
  - Doesn't really identify which loop is running
  - Doesn't indicate how many times it's running
  - These matter enough that they should be clarified!
- "Assume this situation holds: ____, then ____"
  - If that situation isn't the result of mutex_init(), then some steps should probably be shown (is the assumed state *possible*?)

Suggestion

- The top-clarity traces made *data* values clear, not just program-counter values
  - Multiple notations were fine

Q3 – Grading Deadlock

What we were testing

- Find a deadlock (important skill)
- Write a convincing trace (demonstrates understanding)
- Fix a deadlock (and argue that the fix works)

Good news

- ~50% scored 13/15 or better
- ~80% scored 11/15 or better
- So lots of people can identify and trace a fairly typical deadlock

Q3 – Grading Deadlock

Noticeable issues

- Various trace issues
  - A very terse trace might summarize very-different executions (one deadlock, one not)
    - » That does not clearly demonstrate understanding
- Trace does not state or follow assumptions
  - Sometimes happens when trace is missing too many details
- Some very-complicated fixes were proposed
  - A very-simple fix is possible
- Stating a fix without explaining *why* it's a fix lost points
- Confusing circular wait with hold&wait was somewhat common

Q3 – Grading Deadlock

General issues to watch out for

- "Global mutex" is an *emergency* solution to deadlock
  - Not a good solution
- Memorizing the four deadlock ingredients probably is a good idea
  - If something is a fix, that thing should clearly ensure the absence of one of the ingredients – it should be easy to say which and how
- Generally, avoid traces with multiple operations in a single row
  - Unless clarity is genuinely improved
- Not all "tabular traces" were tabular
  - A paragraph isn't really a trace

Q4 – “Event Manager”

Question goals

- Variant of typical “write a synchronization object” exam question
- This one was “hard” rather than “easy” or “typical”

Q4 – "Event Manager"

Question goals

- Variant of typical "write a synchronization object" exam question
- This one was "hard" rather than "easy" or "typical"

Scores varied!

- 25% of class got 16/20 (80% score) or better
- 50% of class got 14/20 (70% score) or better
- But ~25% of class got 10/20 (50% score) or worse
  - Low scores often resulted from multiple synchronization design/usage problems
    - » Global(ish) mutexes
    - » Holding a lock too long
    - » "Paradise Lost"
    - » Destroying objects in a racy way

Q4 – "Event Manager"

Common issues

- Some submissions can't actually reach EM_WAITED (or EM_RAN)
- When possible, consider multiple options
  - mutexes are not the *sole* locking tool available
- We said "ok to assume malloc() doesn't fail on an exam"
  - That is a structurally-more-reasonable assumption for em_init() than for em_request()!
    - » Please review P2 material on "return values"
    - » P3 faces similar considerations!
  - But malloc() was not necessary at all, so if you used it lots this may suggest P2/P3 design problems
- We said to *not* use deschedule()/make_runnable()
  - It is always wise to consider the standard tools (mutex, cvar, semaphore, rwlock)
    - » It is hard to do better than using them
    - » It is easy to do worse

Q4 – “Event Manager”

General synchronization calamities

- Deadlock
- Progress failures (e.g., losing threads)
  - Unlocking not-held locks
- Mutual exclusion failures
- 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 – “Event Manager”

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/case 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)

Breakdown

```
90% = 54.0    4 students
80% = 48.0   17 students
70% = 42.0   14 students
60% = 36.0   12 students (35 and up)
<60%          2 students
```

Comparison/calibration

- Scores are a little low for a typical 410 mid-term
- But honestly not crazy low
  - Low 46%, median 76%, max 92%

Implications

Score below 42?

- 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

Implications

Score below 42?

- 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 38?

- 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

Implications

Score below 38?

- 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. Identify 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