15-410

"My other car is a cdr" -- Unknown

Exam #1
Oct. 24, 2022

Dave Eckhardt

Dave O'Hallaron

Synchronization

Checkpoint 2

- Monday during class time
- Most likely in Wean Hall
- Your kernel should be in mygroup/p3ck2

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

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 50% 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

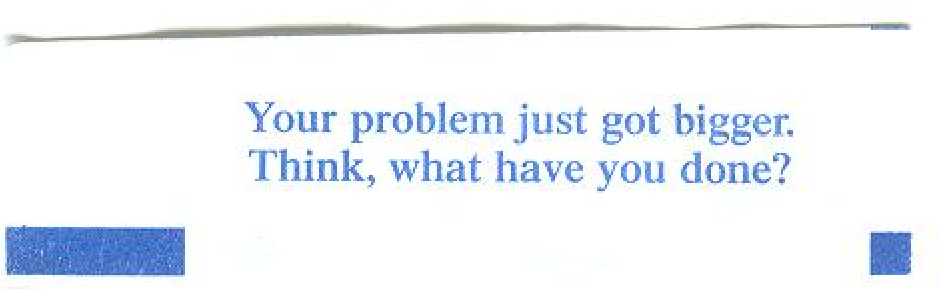
Debugging advice

- Once as I was buying lunch I received a fortune

Synchronization

Debugging advice

- Once as I was buying lunch I received a fortune



Your problem just got bigger.
Think, what have you done?

Image credit: Kartik Subramanian

A Note for Posterity

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

But it was an “arguably typical” exam

- 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 *faint* pencil!
  - Luckily we did not use Gradescope this time
  - But some graders expressed some concern
- Please do not write faintly with pencil 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
- Question 5

Q1 – Short Answer

Two parts

- Register dump
- Design matrix (LRU implementations)

Q1a – 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 "in deep space"
    - That register value *will* be used rather than ignored, and that value *will* lead to tears

Q1a – Register Dump

Selected issues

- It's a good idea throughout P2 and P3 to be familiar with the Pebbles memory layout
- One particular exception is basically guaranteed
- Straightforward to say which instruction would trip
  - And what that instruction was intended to accomplish

Q1b – Design decision

Purpose: demonstrate grasp of a design tool

- Hopefully P2 involved deliberate design
- Hopefully P3 is involving deliberate design
- "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 (September 2)
- Other "odd" lectures to possibly review
  - Debugging, Errors
  - #define, #include
  - We expect you to know *and apply* all of this material

Q1b – Design decision

Specific guidance

- There are *two* ways to find a blobby in under O(N)
  - There is no rule that each blobby must come from malloc()
  - Even “malloc() blobbies()” can be found in O(1)-ish time

General guidance

- Use numbers when possible
  - Quoting space as “O(1)” in Systems is probably wrong
    - 128 is not the same as 4
  - O(N) time is *not* always worse than O(1) in Systems work
    - If N can be bounded at 4, O(N) is *less than* O(1000)
- If one operation is invoked millions of times more than the other, that matters a lot. Workload matters!

Q1 – Results

Scores

- ~60% of the class scored 8/10 or above
- ~20% of the class scored *below* 7/10

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!
  - Knowing what semaphores do is not optional
- Problem is in cvar code, not application code. Traces need not show lots of application code but ***must clearly show*** cvar code!

Q2 – Faulty Condition Variables

FIx

- A clear understanding of the problem suggests a *very* simple fix
- Some suggested fixes work only if a cvar has just one waiter thread *ever*

Q2 – Faulty Condition Variables

Fix

- A clear understanding of the problem suggests a *very* simple fix
- Some suggested fixes work only if a cvar has just one waiter thread *ever*

Scores

- Half the class got 13/15 (86%) or above

Q3 – Deadlock

Good news

- Lots of people identified the deadlock

Q3 – Deadlock

Good news

- Lots of people identified the deadlock

Interesting news

- There are *two* deadlocks!
- Also a race condition!
  - But if you didn't find a deadlock please practice until you can

Q3 – Deadlock

Good news

- Lots of people identified the deadlock

Interesting news

- There are *two* deadlocks!
- Also a race condition!
  - But if you didn't find a deadlock please practice until you can

Key issues

- A process/resource graph is a specific tool
  - Three circles with lines connecting them isn't that tool
- `sem_signal()` before `sem_wait()` is generally *not* a deadlock ingredient
- Be careful that traces can actually happen!
- If you provided only a textual narrative, please practice tabular traces

Q3 – Deadlock

Scores

- ~75% scored 13/15 (86%) or better
- So lots of people can identify and trace a fairly typical deadlock
- Scores below 10 are definitely concerning

Q4 – Testing RWLocks

Question goal

- *Atypical* variant of typical “write a synchronization object” exam question
- Writing test code is hard!

Q4 – Testing RWLocks

Question goal

- *Atypical* variant of typical “write a synchronization object” exam question
- Writing test code is hard!

This question was *hard!*

- Two course-staff members quickly dashed off solutions that were quite wrong (score under 50%)
- Grading was relatively gentle

Q4 – Testing RWLocks

Question goal

- *Atypical* variant of typical “write a synchronization object” exam question
- Writing test code is hard!

Suggestion

- Must “prove” that threads have/haven't done specific things
  - “Prove” N readers have acquired the lock
  - If a writer is injected without such a proof, that writer's experience doesn't pass/fail the rwlock

Q4 – Testing RWLocks

Question goal

- *Atypical* variant of typical “write a synchronization object” exam question
- Writing test code is hard!

Suggestion

- Must “prove” that threads have/haven't done specific things
  - “Prove” N readers have acquired the lock
  - If a writer is injected without such a proof, that writer's experience doesn't pass/fail the rwlock
  - Encapsulating “prove N readers” in a helper function is probably a really good idea
  - Breaking the problem into stages and encapsulating stages into functions is a really good idea
    - » Not just on an exam!

Q4 – Testing RWLocks

Common issues

- FIFO implies starvation-free, but...
  - RWLocks can be starvation-free without being strictly FIFO
  - And there are good performance reasons for being *boundedly* non-FIFO
- Test works if scheduler behaves in exactly one way
  - This is not a good assumption for any code you write!

Q4 – Testing RWLocks

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)

Q4 – Testing RWLocks

Scores

- 35% scored 16/20 (80%) or better
- Median: 14/20 (70%)

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
  - 25% of class got 10/10

Breakdown

90%	= 63.0	7 students
80%	= 56.0	11 students
70%	= 49.0	9 students
60%	= 42.0	9 students
50%	= 35.0	8 students
<50%		1 student

Comparison

- Median grade was 52 (74%)

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

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

Implications

Score below 42?

- 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 42?

- 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. Identify 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