15-410

"My other car is a cdr" -- Unknown

Exam #1
Mar. 12, 2023

Dave Eckhardt

Synchronization

Checkpoint schedule

- Friday during class time
- Meet in Wean 5207
  - If your group number *ends* with
    - » 0-2 try to arrive 10:55-11:00 (5 minutes early)
    - » 3-5 arrive at 11:12:30
    - » 6-9 arrive at 11:30:27
- 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

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

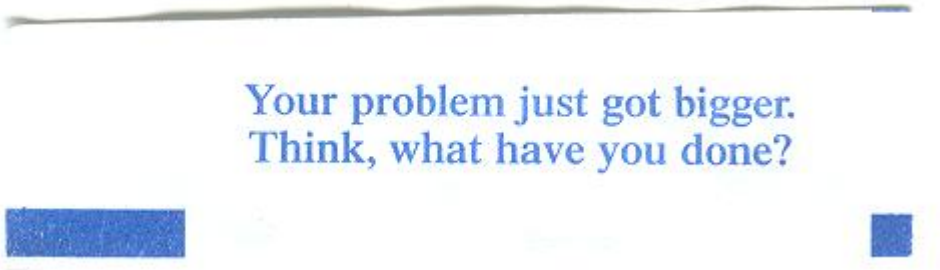

Image credit: Kartik Subramanian

A Note for Posterity

The S'23 mid-term exam occurred during COVID-19

But it was an “arguably roughly typical” exam

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

Three parts

- **“Three kinds of error”**
- **P2 examples of two kinds**
- **“Paradise Lost”**

Q1a/b – 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 3)
- Other "odd" lectures to possibly review
  - Debugging, Questions
  - #define, #include
  - We expect you to know *and apply* all of this material

Q1a/b – Three kinds of error

Official trichotomy

- Resolvable – so resolve it
- Reportable – so report it
- "It's over"
  - Involve the developer, because the program is *broken*
  - Stop the program before propagating lies

Not really in the same space

- "I shouldn't have written this code, so I need to re-design"
- That was generally accepted anyway

Q1a/b – Three kinds of error

Not the core issue: "common error vs. rare error"

- That doesn't help with, e.g., "page fault"
  - Page faults aren't super-common
  - Some page faults are resolvable
  - Some page faults are fatal

Q1a/b – Three kinds of error

Not the core issue: "common error vs. rare error"

- That doesn't help with, e.g., "page fault"
  - Page faults aren't super-common
  - Some page faults are resolvable
  - Some page faults are fatal
- The core issue is which {…} code is needed
  - It is important to write *different* code for {…}
    - » xmalloc() is wrong (for robust code) exactly because it is a way to write the same code for different cases
  - It is important to be confident about which case is which

Q1a/b – Three kinds of error

Not the core issue: "common error vs. rare error"

- That doesn't help with, e.g., "page fault"
  - Page faults aren't super-common
  - Some page faults are resolvable
  - Some page faults are fatal
- The core issue is which {…} code is needed
  - It is important to write *different* code for {…}
    - » xmalloc() is wrong (for robust code) exactly because it is a way to write the same code for different cases
  - It is important to be confident about which case is which

Extraneous

- "Lock contention"
- Forgot to increment loop variable
- O(N**2) instead of O(log log N)

Q1a/b – 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

Q1a/b – Three kinds of error

Practice 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

Q1c – "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'"

Q1 – Results

Scores

- **~60% of the class scored 8/10 or above (good)**
- **~25% of the class scored *below* 6/10 (… … ...)**

Q2 – Critical-Section Problem

What we were testing

- Ability to find a bounded-waiting problem
- Ability to write a clear execution trace
- Ability to solve a bounded-waiting problem

Odd feature of the problem

- This code was discussed in class!

Many scores were high

- Good!

Q2 – Critical-Section Problem

Some disturbing features were observed

- Some traces were not easy to read
  - It is to your benefit to be good about thinking scenarios through, and notation matters
  - Plus, you still have a final exam to take...
- A few people misinterpreted the code (that can happen)
- Roughly 10% of suggestions for fixing the problem made it worse
  - Spin-waiting
  - Deadlock

If you had trouble with this question...

- ...please figure out why, and how to practice. This is core material.

Q3 – Library Deadlock

Parts of the problem

- Find the deadlock
- Suggest a fix

Results – finding

- Most people correctly described a reachable deadlock
- Roughly 1/3 found a minimal-thread-count deadlock
  - The problem structure strongly implies how many that is
  - Some people used 1 extra thread (ok)
  - Some people didn't attempt an explanation of how many threads are necessary

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

Q3 – Library Deadlock

Results – fixing

- Many solutions are plausible and received credit
- Terminology note: preemption is taking a resource from somebody else

Overall

- While analysis, thought, and tracing were required, this was a mostly straightforward question
- 75% of the class scored 80% or better

Q4 – “Simulation Clock”

Question goals

- Variant of typical “write a synchronization object” exam question
- This one was “roughly typical” (*maybe* “medium-hard”)
  - Requirements / solution structure were a little atypical
  - Spec and test code were arguably better than typical

Q4 – "Simulation Clock"

Question goals

- Variant of typical "write a synchronization object" exam question
- This one was "roughly typical" (*maybe* "medium-hard")

Scores varied!

- Median score was 14/20 (70%)
- 30% of class got 16/20 (80% score) or better
- 60% of class got 14/20 (70% score) or better
- But ~33% of class got 10/20 (50% score) or worse
  - Primary low-score causes
    - » Parts missing (tick() not waiting *ever*)
    - » Progress failure (wait before ack)
    - » "Double churn", "Churn"
    - » Yield loop(!) / *spinning(!!)*

Q4 – “Simulation Clock”

Alarming memory mishaps

- mutex_init() passed an uninitialized pointer
- init() refusing to work on random pieces of memory
- free() called on memory that didn't come from the heap

These alarming things should be fixed *soon*!

Q4 – "Simulation Clock"

"Structurally not ok"

- #define MAX_THREADS 1000
  - A thread cap is so rare that it must be explicitly authorized
  - The problem provides a handy alternative
- Assuming thr_getid() returns values between 0 and 1000
  - This can happen only in super-special-case situations
  - So rare it must be explicitly authorized
  - The problem has *two* workable alternatives (at least)
- malloc() on demand for linked-list nodes
  - This is a "structurally wrong meme" - always strive to avoid!
  - The problem provides a handy alternative
  - Please review P2 handout material on "return values"
  - Beware: P3 faces similar considerations!

Q4 – "Simulation Clock"

Synchronization problems

- Waiting before acking is simple progress failure
- "Double churn"
  - Each waiter is awakened many times, not once
  - tick() thread is awakened many times, not once
- "Excessive tick() serialization"
  - tick() must awaken N threads
  - tick() must hear back from N threads
  - But the N threads should be allowed to run in parallel!
- Holding a mutex for O(N)
  - Mutexes are not the *sole* locking tool available
- Scanning a collection without holding any lock
- Returning a random value
  - mutex_unlock(&m); return (ptr->field);

Q4 – “Simulation Clock”

“Glitches”

- lock() twice on the same mutex
- Forgot cond_wait() takes *two* parameters
  - It is really hard to write correct code without this
- Forgot unlock()
- Forgot signal()
- Forgot destroy()
- Forgot free()

Q4 – “Simulation Clock”

Approach

- Pseudo-code/outline *strongly* suggested
  - block(), register(), ack(), collect(), awaken()
  - Pseudo-code/outline all parts before coding any part
  - Consider writing helper functions!
- “First I'll code up wait(), then I'll code up tick()” is much less likely to result in correct code

Q4 – “Simulation Clock”

General synchronization-calamity checklist

- 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 – “Simulation Clock”

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)

Q5 – Nuts & Bolts: “wrapper()”

Purposes

- Verify “stack planning”
- Confirm x86-32 asm coding conventions

Outcomes

- 75% of class got 8/10 or better

Q5 – Nuts & Bolts: “wrapper()”

Concerning

- Not restoring %esp / %ebp
- Forgetting to call f()
- Forgetting that x86 stacks grow down
  - Quick reference by a former student: stackgrowsdown.com

Common

- Off-by-one: storing into *stack_high
- Inverting order of parameter pushes
- Forgetting f() can trash caller-save registers

Breakdown

```
90% = 58.5    9 students (57.0 and up)
80% = 52.0    7 students
70% = 45.5   17 students (45.0 and up)
60% = 39.0    3 students (38.0 and up)
50% = 32.5    2 students (32.0 and up)
40% = 26.0    1 student
<40%          0 students
```

Comparison/calibration

- These scores don't look blatantly problematic

Implications

Score below 45?

- 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 45?

- 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 39?

- 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 39?

- 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