15-410

"My other car is a cdr" -- Unknown

Exam #1
Oct. 23, 2023

Dave Eckhardt

Dave O'Hallaron

Synchronization

Checkpoint schedule (NOTE NEW HASH FUNCTION)

- *Next Monday* 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

A Word on the Final Exam

Disclaimer

- Past performance is not a guarantee of future results

The class 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)

Thanks for Avoiding Faint Pencil!

It wasn't a problem on the mid-term

- Let's keep it that way for the final exam!

“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
- The eternal battle: blobbies vs. timestamps

“And Now, A Word From Our Sponsor...”

Three Kinds of Error

An actionable *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 (September 20)
- Other "odd" lectures to possibly review
  - Debugging, Questions
  - #define, #include
  - We expect you to know *and apply* all of this material

Three Kinds of Error

Official trichotomy

- Resolvable – so resolve it
- Reportable – so report it
- “Rebootable”(?)
  - Involve the developer, because the program is *broken*
  - Stop the program before propagating lies

Not exactly in the same space

- “Rewritable”(??) - “I shouldn't have written this code, so I need to re-design and rewrite”
- That was generally accepted anyway

Q1a/b – Three Kinds of Error

Purpose

- Demonstrate understanding of the three kinds of error in 410 orthodoxy

Selected Issues

- Tying *frequency of occurrence* to *type of error*
- No specific example for fixable error
- Missing explanation for “why that response is what should be done for that kind of error”
- For fatal error, no mention of panic/abort/exit/stop

Q1b – Specific Examples of P2 Errors

Purpose

- Demonstrate understanding of error handling in your P2

Selected issues

- Proposed fix is a “deadlock factory”
  - “Hold & yield” == “hold & wait”... uh-oh...
- Example/discussion not specific to your P2 code

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
    - » affirm(0) is only “arguably less wrong”

Q1c – Design Decision

Purpose

- Demonstrate grasp of a design tool
- Hopefully P2 involved deliberate design
- Hopefully P3 is involving deliberate design

Selected issues

- Equivocating instead of *making a decision*
  - "If we assume X, then A1, but if we assume !X, then A2"
  - But once the metrics and values are in, it is necessary to *decide* and *express a rationale*
- No position on the relative frequencies of references vs. evictions
  - P(hit) >> P(miss) or something is be *very very* wrong
    - » This is true of any/all caches
    - » Evictions are due to misses, not hits!

Q1 – Results

Scores

- ~66% of the class scored 8/10 or above (good)
- ~10% of the class scored *below* 7/10 (… … ...)

Q2 – Bridge Problem

What we were testing

- Ability to find comon synchronization problems
- Ability to support a diagnosis with a clear trace

Odd feature of the problem

- As it happens, HW1 somewhat constituted a hint

Many scores were high

- ~66% had 13/15 or better

Q2 – Bridge Problem

Warning

- 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...

Q2 – Bridge Problem

Warning

- 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...

Selected disturbing features

- “This is a Paradise Lost problem”
  - It isn't (maybe review the Paradise Lost lecture)
- “This is a deadlock”
  - Not if nobody is holding anything!
- “Assume mutexes don't work”... (!!!)
- “Assume cvars are anti-FIFO”
  - Maybe... but there is a much better answer

Q2 – Bridge Problem

How to fix the second problem?

- “Solving the problem” by holding a mutex too long is not a great solution
- “Solving the problem” by deleting logging information is not a great solution

Q2 – Bridge Problem

If you had trouble with Q2...

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

Q3 – Parallel-sort Deadlock

Question goals

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

Q3 – Parallel-sort Deadlock

Question goals

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

Notes

- The code won't let two threads deadlock (hmm...)
- Some ingredients were mis-attributed
  - “Mutual exclusion” does exist, but not because the code contains `mutex_lock()`/`mutex_unlock()` !
  - “No preemption ever” isn't quite right!
- A simple fix does exist

Q3 – Parallel-sort Deadlock

Traces

- Many traces were *very* clear
  - That's the goal!

Trace issues

- Writing “`wait()`” once for each thread, without indicating what condition(s) led to waiting or what they are waiting for is not super-convincing
- If a thread is permanently stuck, it's good to say that
- Showing critical values as columns is a nice touch
- Exiling function calls and parameter values to the margin is “not a best practice”
  - Writing a draft on scrap paper is a good idea

Q3 – Parallel-sort Deadlock

Good solution

- It is possible to *very* succinctly indicate which code should be changed and what should be added/changed
  - That is strong evidence of understanding the problem

Less-good solution

- It is also possible to write vague words about some deadlock ingredient
  - That is less-strong evidence of understanding

Q3 – Parallel-sort Deadlock

Scoring...

- Part A/B: traces were graded pretty gently
  - So if you got a trace deduction, please take it to heart
- Part C: Solution quality counted
  - Thinking of multiple solutions could be a good tactic

Good news / bad news

- A/B: ~80% of class
  - Deadlock derivable by applying principles to the code
- Below C: ~10% of class

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

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 thought glitch

- When you signal a thread because you want it to run, it will run right away (before any other thread)
  - Nope nope nope
  - Best to assume the thread you want to run will immediately encounter a timer tick and *every other thread* will run first

Less alarming but common

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

Q4 – Abortable condition variables

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
- Skipping a core design requirement
  - `abort()` has a *requirement* for indefinite-waiting code
- Starvation factories
  - When FIFO is wanted, LIFO is problematic
- "Strength amplification"
  - It is *possible* to implement a thread counter via a linked list of threads and O(N) list searches, but integers work faster!

Q4 – Abortable condition variables

Approach guidance

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

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
    - If the question provides example traces, it's prudent to check that your code does the right thing for those traces!

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

- ~60% of the class “did ok” (scored 70% or better)
- ~16% of the class “did not do ok” (under 60%)

Q5 – Nuts & Bolts

Four subquestions

- Q5(a/b/c): Variable locations in memory
- Q5(d): Most-reliable allocation

Q5(a/b/c) – Variable Locations

Purpose: Review understanding of a basic idea

- How C variables map to different memory regions
- Encourage use of `malloc()` to be deliberate, not reflexive
    - Especially important in P2 and P3

Selected issues

- Allocated memory not initialized to zero
- Allocating array in text region
    - Rare since the 1980's

Q5(d) – Most-Reliable Allocation

Purpose: Provide practice justifying choices

- Some options are less reliable than others

Selected issues

- Unconvincing arguments were observed
    - Hopefully P3 decisions are based on convincing arguments!

Q5 – Results

Outcome

- ~66% of the class got 10/10

Breakdown

90% = 63.0	7 students
80% = 56.0	15 students
70% = 49.0	6 students
60% = 42.0	1 students
50% = 35.0	0 students
<50%	0 students

Comparison

- Median grade was 58 (83%)
  - This is high!

Implications

Score below 52?

- 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 52?

- 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 46?

- 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 46?

- 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