15-410

"My other car is a cdr" -- Unknown

Exam #1
Oct. 21, 2024

Dave Eckhardt

Synchronization

Checkpoint schedule (NOTE NEW HASH FUNCTION)

- 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 11:13:17
    - » 6-9 arrive at 11:31:19
- 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*
  - Some timer interrupts will *not* cause context switch
    - » Really!
  - Most context-switch invocations will have nothing to do with the timer
    - » Really!
- 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

Google "Summer of Code"

- http://code.google.com/soc/
- Hack on an open-source project
  - And get paid
  - And quite possibly get recruited
- Projects with CMU connections: Plan 9, OpenAFS (see me)

CMU SCS "Coding in the Summer"?

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

Two parts

- mutex vs. rwlock(WRITE)
- P2 memory-allocation design matrix

Q1a - Mutex vs. rwlock(WRITE)

High-level principle

- Different locks for different situations
  - Contention expectation
  - What is being protected
  - Need for waiting/handoff

Mutex is one *specific* kind of lock

- Use in the right situation
- *Don't* use in other situations

Q1a - Mutex vs. rwlock(WRITE)

"Mutex doctrine"

- *Short* sequence
  - Not true of all locks
- Must avoid "interfering executions"
  - This one is true of all locks
- Contention *expected to be rare*
  - Not true of all locks

Who can think of counter-examples?

Q1a - Mutex vs. rwlock(WRITE)

"Mutex doctrine"

- ***Short*** **sequence**
  - **Not true of all locks – rwlock is a counter-example!**
- **Must avoid "interfering executions"**
  - **This one is true of all locks**
- **Contention *expected to be rare***
  - **Not true of all locks – "barrier" is a counter-example!**
- **Be able to say why each matters**

Q1a - Mutex vs. rwlock(WRITE)

Common issues

- Some answers discussed rwlock(READ) – not part of the question
- Some answers didn't answer the “Use X or use Y” question
- “That's not what rwlocks are for”
  - Perhaps, but using a system's fastest lock for very-slow tasks is *more* problematic

Lost on this question?

- Discussed in *two* lectures (Synch 1, Paradise Lost)

Q1b – P2 memory-allocation matrix

Purpose

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

Q1b – P2 memory-allocation matrix

Purpose

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

Common issues

- Describing one reasonable decision and one unreasonable decision
  - *Almost always* if enough analysis has been done a decision will be among reasonable decisions!
- Matrix cells should not be essays!
  - Refer to "Questions" lecture
- Complete answers must *describe a decision*
  - Often: "The value V1 for metric M1 is more important than the value V2 for metric M2, because rationale R".

Q1 – Results

Scores

- **~66% of the class scored 7/10 or above (good)**
- **~25% of the class scored *below* 6/10**

Q2 – Faulty Mutex

What we were testing

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

Odd features of this problem

- This code fails *everything*
  - Mutual exclusion, progress, bounded waiting

Many traces got full credit

- Thus, prudent to follow up on any point deductions

Q2 – Faulty Mutex

Conceptual warnings

- Showing that a mutex doesn't work right if it is invoked illegally is not a good approach
  - In general, *no* implementation catches *all* abuse
    - » Especially not under time pressure!
  - Meanwhile, the code really does fail *everything*
    - » Being able to locate errors like that is an important skill
- Also illegal to assume some thread runs at zero speed
- Incorrect definition of "unbounded" will cause trouble

Q2 – Faulty Mutex

Concerning trace issues

- Abbreviation to the point of ambiguity
  - "`xchg()`" without a variable name is damaging in code with two different `xchg()` sites
- Leaving out 100% of `lock()` and 50% of `unlock()`, then starting trace in the middle of `unlock()`, is genuinely too terse

Q2 – Faulty Mutex

Outcomes

- ~75% had 13/15 or better
- ~20% had 10/15 or below

If you had trouble with Q2...

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

Q3 – Dining Philosophers

Goals

- Identify a deadlock
- Clearly communicate a scenario
- Suggest a fix

Common/concerning trace issues

- "This thread must awaken!!" was written frequently
  - If T1 is blocked on a cvar and T2 signals, then T1 must be seen to run!
- Frequent deductions for not showing *any* `cond_signal()` calls
  - This is a fundamental part of this problem!
- Trace should show numbers (not just "`left`", "`right`")
- Trace can't rely on mutexes not working

Q3 – Dining Philosophers

Fixes

- Reasonable fixes exist via Prevention, Avoidance, and Detection
- Also less-reasonable fixes
  - Given problem structure, generic graph-based algorithms are overkill if counters will suffice
- Beware starvation
  - Generic "allocate all-at-once" and generic "preempt" answers

Concerning

- "Impose strict turn-taking on diners"
  - Quite unreasonable when DINERS > 6
  - "Remove all parallelism" *does* help with deadlock, but...

Q3 – Dining Philosophers

Outcomes

- 60% had 12/15 or better
- 20% had below 10/15

Q4 – Atomic Matching

Question goal

- Slight modification of typical "write a synchronization object" exam question
  - Inspired by a traditional UCB OS question!
- This one was *atypically easy*
  - Scores below 70% (14/20) are concerning

Q4 – Atomic Matching

Interesting question features

- Can be done well with or without semaphores
  - If you solved it one way, maybe try again a different way?

Things to watch out for

- Starvation during entry was somewhat common
  - Avoidable, so avoiding it is good
- When possible, `cond_signal()` outside of a mutex is better than inside
- Avoid “thundering herd” aka “churn”
  - One giant cvar
  - Unbounded number of threads of all types waiting on it for different things
  - `cond_broadcast()` wakes everybody up and many threads must block again

Q4 – Atomic Matching

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 – Atomic Matching

Outcomes

- ~50% of the class scored 18/20 or better ("A")
  - This question is arguably "not super hard"
- ~30% of the class "did not do ok" (under 60%)
  - These outcomes are concerning

Other questions in this category are harder

- Perhaps a final-exam question might be harder

Q5 – Nuts & Bolts

Quick question

- What happens given a specific silly kernel_main()?

Key idea

- The stack will overflow

Details?

- Good
  - Plausible identification of a specific plausible thing and how paving over that thing would stop execution
- Less good
  - "Page fault" – there is no paging
  - "`SWEXN_CAUSE_PAGEFAULT`" – there is no swexn()

Q5 – Nuts & Bolts

Concerning

- "IDT race condition"
  - Overwriting 0xFE77458E with 0xFE77458E is *not a race condition*, regardless of write size or write order!
- Overlooking unbounded stack growth
  - Code contains *two* different growth patterns

Q5 – Nuts & Bolts

Outcomes

- 20% got 8/10 or better
  - That is a low fraction of the class
- 30% got 3/10 or worse
  - This is a high fraction of the class

Breakdown

```
90% = 63.0    4 students
80% = 56.0   13 students
70% = 49.0    9 students
60% = 42.0    3 students
50% = 35.0    6 students
<50%          3 students
```

Comparison

- Median score was 54/70 (77%)
  - This is not low

Implications

Score below 54?

- 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 54?

- 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 *much* better than “I'll try harder”
  - *Strong suggestion*:
    - » Identify causes, draft a plan, see instructor

Implications

Score below 44?

- 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 44?

- 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. 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

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