15-410

"My other car is a cdr" -- Unknown

Exam #1
Mar. 10, 2025

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 necessarily 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
  - Please execute those instructions in order

Outline

Question 1

Question 2

Question 3

Question 4

Question 5

Q1 – Short Answer

Two parts

- **“Can I assume _____?”**
- **Register dump**

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"

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

- 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

Q1 – Results

Scores

- **~60% of the class scored 7/10 or above (good)**
- **~20% of the class scored *below* 5/10**

Q2 – Deadlock

What we were testing

- Knowledge of deadlock ingredients
- Understanding of problems *and design space* of deadlock detection/recovery

Odd features of this problem

- This was a “conceptual” deadlock problem instead of a “read code” deadlock problem
- Neither Prevention nor Avoidance was tested

Q2 – Deadlock

"Policy" vs. "implementation"

- A policy problem/challenge is when it is *difficult to decide what to do*
  - Policy discussions likely include "Which?" and/or "should", etc.
- An implementation problem/challenge is when it is *difficult to carry out a task*
  - Implementation challenges likely include "expensive", "slow", "brittle", etc.

Q2 – Deadlock

Before **deploying detection & recovery...**

- **It is necessary to *decide* among approaches**

After **deciding to do detection & recovery...**

- **Whichever approach was selected will be *expensive* in some way**

Q2 – Deadlock

Before **deploying detection & recovery...**

- **It is necessary to *decide* among approaches**

After **deciding to do detection & recovery...**

- **Whichever approach was selected will be *expensive* in some way**

System designers should be able to frame decisions in terms of policy decisions and implementation costs

Q2 – Results

Scores

- ~66% of the class scored 8/10 or above (good)
- Nobody scored below 6/10

Q3 – “Nemo's Algorithm”

What we were testing

- Primarily: ability to find *and show* race conditions
- Also: knowledge of what a c.s. algorithm should do

Good news

- Many people got a perfect score (60% of the class)

Bad news

- Traces should not have multiple threads writing in a single table row!
- “Not FIFO in terms of who executed the first instruction of lock() first” is not the definition of a bounded-waiting failure!
- A repeatable sequence must consider *data values*!
  - If `turn` has a different value it's not a repeat!
  - This is an important thing to get right

Q3 – Results

Scores

- **~70% of the class scored 14/15 or 15/15 (good!)**
- **~20% of the class scored *below* 10/15**

Q4 – Pausable Semaphore

Question goal

- Slight modification of typical "write a synchronization object" exam question

Interesting question features

- Can be done well with or without semaphores
  - If you solved it one way, maybe try again a different way?
- Both short solutions and long solutions are reasonable

Q4 – Pausable Semaphore

Question goal

- Slight modification of typical "write a synchronization object" exam question

Interesting question features

- Can be done well with or without semaphores
  - If you solved it one way, maybe try again a different way?
- Both short solutions and long solutions are reasonable
  - Some short solutions are not stellar, though
    - » Piling a bunch of threads up on a mutex for an indefinite period of time is short *but probably turns the fans on*
    - » An rwlock is arguably *anti-good* at stopping threads promptly

Q4 – Pausable Semaphore

General note on blocking

- Threads that can't do productive work should *stop running*
- Once stopped, a thread should remain stopped *until there is a reasonable likelihood that it can do productive work*

Things to watch out for

- *When possible*, `cond_signal()/cond_broadcast()` 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 – Pausable Semaphore

General conceptual problems

- "x() takes a pointer" does *not* mean "x() must call malloc()"
- Assigning to a function parameter changes the *local copy*
  - It has no effect on the calling function's value
  - C isn't C++ or Pascal (luckily!)
- See course staff about any general conceptual problems revealed by this specific exam question

Q4 – Pausable Semaphore

Approach guidance

- This question mixes counting with blocking for two very-different reasons (but maybe it's *three* different reasons?)
  - Existing primitives implement counting and blocking and unblocking
    - » So it is possible to offload lots of work
    - » But it is important to keep track of who should receive priority to take various steps
- 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 signal()" is much less likely to result in correct code

Q4 – Pausable Semaphore

Outcomes

- ~50% of the class scored 15/20 or better (75%+)
  - 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's in a P1 interrupt/exception frame – and why?

Common issue

- Providing a rationale for %eflags
  - Some things in %eflags change *a lot*, and those values must be correct!

Outcomes

- 70% of the class score 8/10 or above (good)
- 25% of the class scored 5/10 or below
  - Some far below!

Breakdown

```
90% = 58.5    5 students (58.0 and up)
80% = 52.0    7 students (52.0 and up)
70% = 45.5    3 students (44.0 and up)
60% = 39.0    3 students (39.0 and up)
50% = 32.5    1 student  (32.0 and up)
40% = 26.0    0 students
<40%          1 student
```

Comparison/calibration

- Overall scores don't look blatantly problematic

Implications

Score below 50?

- 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 50?

- 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 40?

- Something went *noticeably* wrong
  - It's *important* to figure out what!
- Beware of "triple whammy"
  - Low score on three "core" questions
  - Generally Q2, Q3, Q4
- 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)

Implications

Score below 40?

- Something went *noticeably* wrong
  - It's *important* to figure out what!
- Beware of "triple whammy"
  - Low score on three "core" questions
  - Generally Q2, Q3, Q4
- 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)
- 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