15-410

"My other car is a cdr" -- Unknown

Exam #1
Oct. 15, 2013

Dave Eckhardt

Todd Mowry

Synchronization

Checkpoint 2 – Wednesday, in cluster

- Arrival-time hash function will be different

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

PGP key signing

- **Friday, October 25**
- **16:30**
- **GHC 4301**

Synchronization

Asking for trouble

- If your code isn't in your 410 AFS space every day, you are asking for trouble
  - Roughly half 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
- If you aren't using source control, that is probably a mistake
- GitHub sometimes goes down!
  - S'13: on P4 hand-in day (really!)

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

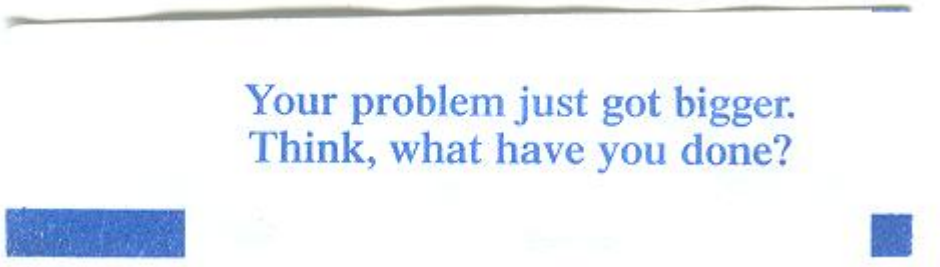

Image credit: Kartik Subramanian

Synchronization

Crash box

- How many people have had to wait in line to run code on the crash box?
  - How long?

Upcoming Events

Google "Summer of Code"

- http://code.google.com/soc/
- Hack on an open-source project
  - And get paid (possibly get recruited, probably not a lot)
- Projects with CMU connections: Plan 9, OpenAFS (see me)

CMU SCS "Coding in the Summer"?

15-412 (Fall)

- If you want more time in the kernel after 410...
- If you want to see what other kernels are like, from the inside

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 more stuff (~100 points, ~7 questions)

“See Course Staff”

If your paper 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.**

Outline

Question 1

Question 2

Question 3

Question 4

Question 5

Q1a – "Paradise Lost" or "TOCTTOU"

"Paradise Lost"

- This is an *amazingly* common bug pattern
  - Somebody makes the world ready for you.
  - They signal you.
  - You run.
  - Oops! You didn't run soon enough.
- Often solve via "if() vs. while()"
- This is *so amazingly* common...
  - ...that we made up a name for it...
  - ...then we wrote a lecture about it...
  - ...then we put it on an exam...

TOCTTOU error

- "Time-of-Check to Time-of-Use"
- The "general case" of "Paradise Lost"
- *Not* generally solved by "if() vs. while()"

Q1a – “Paradise Lost” or “TOCTTOU”

Grading

- 5 correct
- 4 almost correct
- 3 right general idea
- 2 plausibly headed toward the right general idea
- 1 something plausibly relevant

Q1b – “mixing fprintf() and signal handlers”

Background

- P0 handout warns about sprintf()
- P0 handout warns about “fprintf() and printf() in certain contexts”
- Warning says “Please … reacquaint yourself with the details of these issues and their implications”

Q1b – “mixing fprintf() and signal handlers”

What does fprintf() do?

- Prints to a “FILE *”
  - Which, fundamentally, is a buffer...
  - ...which is to say, “stateful object containing invariants”

What does a signal handler do?

- It runs at a surprising time

Q1b – “mixing fprintf() and signal handlers”

What does fprintf() do?

- Prints to a “FILE *”
  - Which, fundamentally, is a buffer
  - ...which is to say, “stateful object containing invariants”

What does a signal handler do?

- It runs at a surprising time

So?

- signal(SIGSEGV, surprise);
- fprintf(foofile, “%s %s\n”, “foo”, 0);

Q1b – “mixing fprintf() and signal handlers”

What does fprintf() do?

- Prints to a “FILE *”
  - Which, fundamentally, is a buffer
  - ...which is to say, “stateful object containing invariants”

What does a signal handler do?

- It runs at a surprising time

So?

- signal(SIGSEGV, surprise);
- fprintf(foofile, “%s %s\n”, “foo”, 0);

!FAQ

- What does this have to do with OS?
  - It's almost like another thread, isn't it?
- Ok, I get it...say, what's so wrong about sprintf(), anyway?

Q2 – “Faulty Mutexes”

Which critical-section requirement doesn't hold?

- ? **Mutual exclusion**
- ? **Progress**
- ? **Bounded waiting**

Q2 – “Faulty Mutexes”

Which critical-section requirement doesn't hold?

- ✓ Mutual exclusion
- ✓ Progress
- ✓ Bounded waiting

Q2 – “Faulty Mutexes”

Which critical-section requirement doesn't hold?

- ✓ Mutual exclusion
- ✓ Progress
- ✓ Bounded waiting

“What a terrible mutex implementation!”

- – The author

Q2 – "Faulty Mutexes"

Which critical-section requirement doesn't hold?

- ✓ Mutual exclusion
- ✓ Progress
- ✓ Bounded waiting

"What a terrible mutex implementation!"

- – The author

Grading

- Many people did very well
- Many people got lost in the jungle
- Not much middle ground

Q2 – "Faulty Mutexes"

Popular and problematic submissions

- "This mutex doesn't handle a thread locking and then dying!"
  - True... but essentially none do
- "This mutex doesn't handle an evil thread which randomly calls unlock() while it doesn't hold the lock!"
  - True... but …
- Overall: if an artifact fails *when used correctly*, we are not enraged by failures due to abuse

Misconceptions

- "Takes a bunch of steps" is not a bounded-waiting failure
- "Somebody holds the lock for a long time" doesn't violate progress
  - Progress is a requirement on the *choosing protocol*
  - If the lock is held choosing need not begin

Q3 – Battleship Deadlock

Good news

- **Most people found the deadlock**
- ***Lots* of full-credit answers, lots of "very close"**

Q3 – Battleship Deadlock

Good news

- Most people found the deadlock
- *Lots* of full-credit answers, lots of "very close"

The other good news

- Most people also solved the deadlock correctly

Q3 – Battleship Deadlock

Bad news

- Most people found the deadlock
- Most people also solved the deadlock correctly
- ⇒ A final-exam deadlock question could be harder

Q3 – Battleship Deadlock

Bad news

- Most people found the deadlock
- Most people also solved the deadlock correctly
- ⇒ A final-exam deadlock question could be harder

The other bad news

- A common problem was changing the spec
  - ...which the problem statement said not to do
- Often your boss won't let you change the spec
- And it's usually better to keep fixes internal when possible

Q4 – "Master/slave Mutexes"

Question goal

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

What we asked for

- Lock
- Master (creator) thread gets priority
- Slave threads are awakened (with an error) if master chooses to destroy the lock

Q4 – "Master/slave Mutexes"

Question goal

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

What we asked for

- Lock
- Master (creator) thread gets priority
- Slave threads are awakened (with an error) if master chooses to destroy the lock
  - Key sub-issue: master is allowed to free()/pave the memory as soon as destroy() completes

Q4 – “Master/slave Mutexes”

Hazards - “lock”

- Spin loop / yield() loop / “cond_signal() loop” / sleep(20)
  - These are *not* handy multi-tools!
  - What must you *prove* before using a spin loop?
  - What does a yield() loop really do?

Q4 – “Master/slave Mutexes”

Hazards - “lock”

- Spin loop / yield() loop / “cond_signal() loop” / sleep(20)
  - These are *not* handy multi-tools!
  - What must you *prove* before using a spin loop?
    - » Hint: #1 is “on a multi-processor”
  - What does a yield() loop really do?
    - » Hint: what if you're on a multi-processor?
  - *Essentially always: use a genuine synchronization method*
    - » This is a *core* concept. You *must* show proficiency.

Q4 – "Master/slave Mutexes"

Hazards - "lock"

- Spin loop / yield() loop / "cond_signal() loop" / sleep(20)
  - These are *not* handy multi-tools!
  - What must you *prove* before using a spin loop?
    - » Hint: #1 is "on a multi-processor"
  - What does a yield() loop really do?
    - » Hint: what if you're on a multi-processor?
  - *Essentially always: use a genuine synchronization method*
    - » This is a *core* concept. You *must* show proficiency.
- Multiple threads can acquire the lock
  - Most often: "Paradise Lost"
- unlock() has insufficient locking
- Nothing beyond a mutex is used to stall threads
  - ...lots of threads... ...who face a low-priority wait...
  - "Stall a thread for an unknown time until a specific condition is met" should not sound like "mutex" to you

Q4 – "Master/slave Mutexes"

Hazards - "master priority"

- Common: master is *awakened* preferentially, but slave threads can rush in quickly and defeat it
  - Lock must "look different" during "handoff from slave to master" (or else it will look the same!)
- Less common: master and slave arms of lock() code are too similar

Hazards - "slaves awakened on master destroy()"

- destroy() just paves over the object and returns
- destroy() issues a broadcast() and flees immediately
- Some slaves are awakened, but not all
- Some awakened slaves might not be 100% done with the lock when the master starts paving

Q4 – "Master/slave Mutexes"

Warning about cvars

- In P2 and on the exam it's ok to assume the relationship between cond_signal() and awakenings is "1:1 and onto"
- POSIX condition variables *do not have this property*
  - If you call pthread_cond_signal() *once*, you may awaken *multiple* threads
  - "man 3 pthread_cond_signal" and see "spurious wakeup"
  - Sorry!
    - » How to fix?
    - » See "if() vs. while()"...

Q5 – "far_call()"

Question goals

- Verify basic assembly-language skills, stack understanding

Outcomes

- *Lots* of A & B scores
- If not, make sure you figure out what went wrong!

Selected common/dangerous issues

- Not going back to the original stack
- General stack-discipline problems
- Some stack-discipline register not restored
- Clobbers callee-save registers

Breakdown

90% = 67.5

80% = 60.0

70% = 52.5

60% = 45.0

50% = 37.5

<50%

Breakdown

```
90% = 67.5   13 students (67 and up)
80% = 60.0    7 students
70% = 52.5    9 students
60% = 45.0    9 students
50% = 37.5    4 students
<50%          0 students
```

Comparison/calibration

- Scores were higher than typical, more A's

Low exams all got clobbered on Q2 and Q4

Implications

Score 45..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?
- Probably plan to do better on the final exam

Score below 45?

- Something went *dangerously* wrong
  - It's important to figure out what!
- Passing the final exam may be a *serious* challenge
- *Passing the class may not be possible!*
  - To pass the class you must demonstrate proficiency on exams (not just project grades)
- See instructor

Implications

"Special anti-course-passing syndrome":

- Only "mercy points" received on several questions
- Extreme case: *no* question was convincingly answered
  - It is *not possible to pass the class* if both exams show no evidence that the core topics were mastered!

“Design” in this exam

Reminder...

- Final exam will focus more on “design”
  - This was not a design-heavy mid-term exam
  - You may need to review other mid-term exams for design material if you haven't already