15-410

"My other car is a cdr" -- Unknown

Exam #1
Mar. 4, 2013

Dave Eckhardt

Synchronization

Checkpoint schedule

- Wednesday during class time
- Meet in Wean 5207
  - If your group number *ends* with
    - » 0-2 try to arrive 5 minutes early
    - » 3-5 arrive at 10:42:30
    - » 6-9 arrive at 10:59:27
- Preparation
  - Your kernel should be in mygroup/p3ck1
  - It should load one program, enter user space, gettid()
    - » Ideally lprintf() the result of gettid()
  - We will ask you to load & run a test program we will name
  - Explain which parts are “real”, which are “demo quality”

Synchronization

Asking for trouble

- If your code isn't in your 410 AFS space every day, you are asking for trouble
  - "Many" 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

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

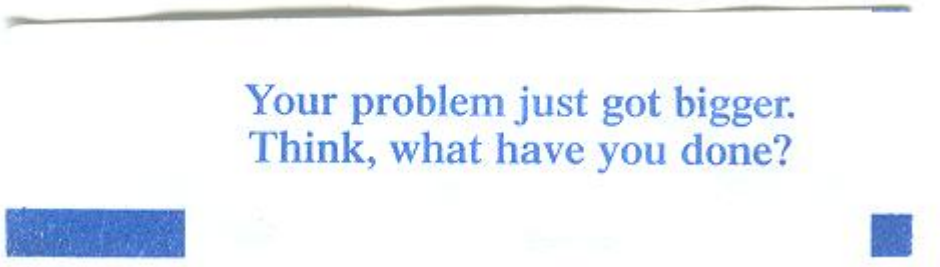

Image credit: Kartik Subramanian

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 – “Runnable”

Expected

- A scheduler state for a thread
- Not running
- Not blocked
- “Could be running except that we don't have enough processors right now”

Q1b – "Starvation"

Hoping to see

- A resource allocation problem
- A thread or class of threads might never get what it needs
- Meanwhile, other threads are getting what they need
- Not a deadlock (there is no circular wait, etc.)

Problematic answers

- "Starvation is another name for 'bounded waiting failure'"
- "Starvation is: #include <bounded_waiting_failure.h>"

The conceptual problem

- Starvation is *related* to bounded-waiting failures
  - But lots of things are related to each other
  - Ideally, we use a different name to convey a *different* concept
  - Using different names for different bad things helps us diagnose and avoid them

Q1b – “Starvation”

“Bad thing” list

- Some thread grabs a lock and never releases it
- When a bunch of threads try to grab a lock they all get stuck forever (“progress failure”)
- When a bunch of threads try to grab a lock maybe one gets stuck forever (“bounded-waiting failure”)

Q1b – "Starvation"

"Bad thing" list

- Some thread grabs a lock and never releases it
  - This is not a problem with the locking protocol (no protocol can overcome abuse)
- When a bunch of threads try to grab a lock they all get stuck maybe-forever ("progress failure")
  - Horrible bug in low-level lock code used by all threads
  - Threads may be running continuously
  - Must fix right away
- When a bunch of threads try to grab a lock maybe some get stuck for maybe-forever ("bounded-waiting failure")
  - Bad problem in low-level lock code used by all threads
  - One thread may be running continuously
  - Needs to be fixed or at least "seriously argued away"

Q1b – “Starvation”

“Bad thing” list

- When a bunch of threads with different needs try to satisfy their needs, threads with some needs might never be satisfied (“starvation”)

Q1b – "Starvation"

"Bad thing" list

- When a bunch of threads with different needs try to satisfy their needs, threads with some needs might never be satisfied ("starvation")
  - Serious problem
  - Usually application-level, not in low-level lock/synch code
  - Happens even if low-level synch code is perfect!
  - Fix usually involves adding an application-specific scheduler

Q1b – "Starvation"

"Bad thing" list

- When a bunch of threads with different needs try to satisfy their needs, threads with some needs might never be satisfied ("starvation")
  - Serious problem
  - Usually application-level, not in low-level lock/synch code
  - Happens even if low-level synch code is perfect!
  - Fix usually involves adding an application-specific scheduler

Starvation example

- One lock for a pool of N things
- Different people need 1..N things
- Plan: grab lock; loop on "things freed" cvar until N free
  - This works great for 1-clients, 2-clients … not so good for N
- Fix?

Q1b – "Starvation"

Starvation example

- One lock for a pool of N things
- Different people need 1..N things
- Plan: grab lock; loop on "things freed" cvar until N free
  - This works great for 1-clients, 2-clients … not so good for N
- Fix?
  - "Be strictly FIFO" ⇒ greatly reduces concurrency
  - "Some sort of age policy" ⇒ code is complicated
- Anyway, this is not the same problem as unfair locks

Q2 – "Exceptional Throwing"

Good news

- Lots of high scores (people found the bug and showed it)

Bad news

- Also lots of low scores

Common issues

- Showing impossible outcomes
  - Often by forgetting that some line is executed, e.g., a cond_signal()
- Missing initial part of trace
  - Showing something that would indeed go wrong if a non-obvious state were in place

Rare, but more serious

- Misconceptions about how condition variables work

Q3 – Cluster Deadlock

Good news

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

Things to be careful of

- Some people were unclear about deadlock requirements
- "Everything would be ok if the whole room were protected by a mutex"
  - *Danger!* Please review Dining Philosophers lecture example!
- Test-taking oops – if we write "Assume X is ok", it is unwise to claim X leads to a problem

Q4 – “Banking”

Question goal

- Slight modification of typical “write a synchronization object” exam question

Outcome

- Scores varied widely!

Structural hazards

- Interactions between long-waiting threads and object deactivation require care
- Interactions between fast operations and slow operations require care
- The simplistic `transfer()` can deadlock if two people try to transfer money into each other's accounts
- `close()` can't finish (`mutex_destroy()`) while threads are still awakening and finding out bad news

Q4 – "Banking"

Things to watch out for

- Fundamentally wrong plan
  - No condition variables (e.g., yield()-loop "synchronization")
  - This is *very* serious: *key* course concepts were not understood; it is *absolutely necessary* to fix this problem
- `malloc()`/pointer misunderstandings
  - *Very* serious: It is difficult to imagine how students can write passing kernels while confused about these issues
- "Paradise Lost" (if you were dinged for this, *definitely* review that lecture!)
- `broadcast()` where `signal()` should be used
  - A pattern for serious inefficiency
- `signal()` where `broadcast()` should be used
  - A pattern for getting threads stuck forever
- Lock leaks
- `mutex_unlock(&a->m); return (a->balance);`

Q5 – "`get_esp()`"

Question goals

- Verify basic assembly-language skills, stack understanding
- Discourage people from calling `get_esp()`
  - You can write the code, but what can you do with the answer you get?

Expected solutions

- Delta of 0: push/call/pop
- Delta of 4: push/call/no-need-to-pop-right-away
- Sometimes the Part B code wasn't "structurally different" from Part A (only a constant changed) – not what we were hoping for, given the *vast* diversity of possible code

Outcomes

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

Q5 – "get_esp()"

Common problems

- Clobbering callee-saved registers we used
- Forgetting that our callers clobber our caller-save registers
- Forgetting to restore %ebp
- Corrupting various registers, corrupting our return address, etc.
- Fracturing credibility (`PUSHA`)
- Returning y-x instead of x-y

An alarming common code sequence

- `movl $4, %eax`
- `pushl %eax`

Breakdown

```
90% = 67.5   13 students (66 and up)
80% = 60.0   23 students
70% = 52.5   16 students
60% = 45.0    6 students
50% = 37.5    4 students
<50%          2 students
```

Comparison/calibration

- Not obviously “too hard” / “too easy”

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":

- You got only the "mercy points" on several questions
- Extreme case: *no* question was convincingly answered
  - It is very important that you don't have *two* exams without evidence that *some* topics have been mastered!

“Design” in this exam

Reminder...

- Final exam will focus more on “design”
  - On this exam, design was best represented by
    - » Q4 (Banking)
  - But there wasn't a lot of design (so you will want to review other mid-term exams if you didn't while studying)