15-410

"My other car is a cdr" -- Unknown

Exam #1
Mar. 3, 2014

Dave Eckhardt

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

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

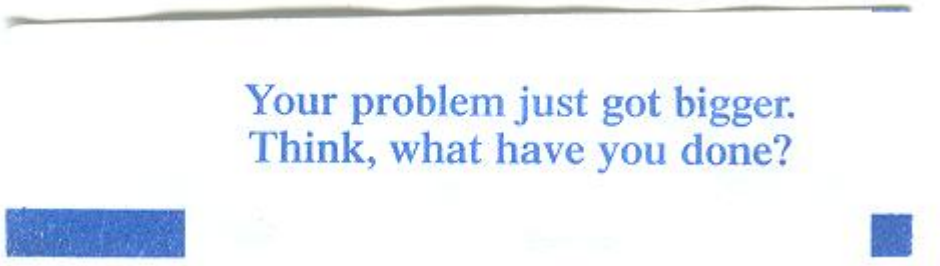

Image credit: Kartik Subramanian

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 – "Deadlock Prevention"

For full credit

- Here are the deadlock ingredients
- Deadlock *prevention* passes a static law against one ingredient
- Either
  - Explain why some of them are hard, or
  - Mention one or two that are typically used, or
  - Briefly summarize how other approaches work

Typical issues

- Describing a non-prevention approach
- Using language which makes it unclear the concept was grasped (e.g., well enough to distinguish it from other approaches)

Q1b – “User mode”

For full credit

- User mode protects kernel data structures
- User mode protects hardware
- User mode is enforced by hardware

Most-common notable issue

- Describing some mechanism (RPL bits; “different code segments) without reference to goals

Q2 – Deadlock

Good news

- Lots of people identified the deadlock

Q2 – Deadlock

Good news

- Lots of people identified the deadlock

Better news

- We thought there was *one* deadlock, but the class found *three*

Q2 – Deadlock

Good news

- Lots of people identified the deadlock

Better news

- We thought there was *one* deadlock, but the class found *three*

Key issues

- Misunderstandings about semaphores
  - `sem_signal()` before `sem_wait()` is generally *not* a deadlock
- A process/resource graph is a specific tool (three circles with lines connecting them isn't that tool)
- Be careful that traces can actually happen!

Q3 – "Channels"

Question goal

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

What we asked for

- mutex, cvar
- Common mutex issues
  - "Total exclusion", "Total inclusion", deadlock
  - Worrisome: limit on maximum number of threads
- Common/key cvar issues
  - Limit on maximum number of threads – "anti-pattern"
  - Mis-handling of "world mutex" (e.g., drop immediately)
  - Deadlock
  - "Anti-FIFO" (e.g., stack)
  - `broadcast()` not implemented, or seriously flawed

Q3 – “Channels”

Higher-level issues/suggestions

- `cond_signal()` shouldn't block indefinitely
- Try to avoid holding a lock while doing an O(N) operation
- Avoid “check for something bad happening; if so, return silently”

Meta-issues/suggestions

- In the async-send case, should “full buffer” mean “fail” or “block”?

Q3 – “Channels”

Higher-level issues/suggestions

- `cond_signal()` shouldn't block indefinitely
- Try to avoid holding a lock while doing an O(N) operation
- Avoid “check for something bad happening; if so, return silently”

Meta-issues/suggestions

- In the async-send case, should “full buffer” mean “fail” or “block”?
  - There are other options too!

Q3 – "Channels"

Higher-level issues/suggestions

- `cond_signal()` shouldn't block indefinitely
- Try to avoid holding a lock while doing an O(N) operation
- Avoid "check for something bad happening; if so, return silently"

Meta-issues/suggestions

- In the async-send case, should "full buffer" mean "fail" or "block"?
  - There are other options too!
    - » Return silently
    - » Delete oldest item
    - » Delete *all* items in buffer
    - » …
  - How to think about this?

Q3 – “Channels”

Higher-level issues/suggestions

- `cond_signal()` shouldn't block indefinitely
- Try to avoid holding a lock while doing an O(N) operation
- Avoid “check for something bad happening; if so, return silently”

Meta-issues/suggestions

- In the async-send case, should “full buffer” mean “fail” or “block”?
  - There are other options too!
    - …
  - How to think about this?
    - Claim: “Three kinds of error”...

Q3 – “Channels”

Higher-level issues/suggestions

- `cond_signal()` shouldn't block indefinitely
- Try to avoid holding a lock while doing an O(N) operation
- Avoid “check for something bad happening; if so, return silently”

Meta-issues/suggestions

- In the async-send case, should “full buffer” mean “fail” or “block”?
  - There are other options too!
    - » …
  - How to think about this?
    - » Claim: “Three kinds of error”...
- “Should our cvars be FIFO?”

Q3 – "Channels"

Higher-level issues/suggestions

- `cond_signal()` shouldn't block indefinitely
- Try to avoid holding a lock while doing an O(N) operation
- Avoid "check for something bad happening; if so, return silently"

Meta-issues/suggestions

- In the async-send case, should "full buffer" mean "fail" or "block"?
  - There are other options too!
    - » …
  - How to think about this?
    - » Claim: "Three kinds of error"...
- "Should our cvars be FIFO?"
  - Should they be *anti*-FIFO?

Q3 – "Channels"

Higher-level issues/suggestions

- `cond_signal()` shouldn't block indefinitely
- Try to avoid holding a lock while doing an O(N) operation
- Avoid "check for something bad happening; if so, return silently"

Meta-issues/suggestions

- In the async-send case, should "full buffer" mean "fail" or "block"?
  - There are other options too!
    - » …
  - How to think about this?
    - » Claim: "Three kinds of error"...
- "Should our cvars be FIFO?"
  - Should they be *anti*-FIFO?
  - In many situations "pretty darn FIFO" is all that can be done

Q4 – “Imprecise Peterson's”

Key idea

- Many algorithms are not ok if steps are taken out of order
- Imprecise interrupts result in steps being completed out of order
- Show how Peterson's Solution breaks if steps are taken out of order

How to solve (a)

- Find nearby things which must be in order
- Make them “less in order”
  - Subject to requirements stated in problem
- There are two breakable sequences (that we know of)

Q4 – “Imprecise Peterson's”

Selected common/dangerous issues (a)

- Trace shown doesn't contain a race
- Trace shown requires hardware to misbehave in a way that would break single-threaded programs

Common issues (b)

- Proposed solution fixes *exactly this code*, but other code with the same problem would not be fixed
- Proposed solution would work, but only if it were deployed differently than described
- Proposed solution fixes a non-problem identified in (a)

Q5 – Blocking

For full credit

- Blocked thread can't run until a specific event
- Blocked thread is *not in a run queue*

Dangerous idea

- "If a thread invokes `gettid()`, the thread's execution is suspended until the system call returns."
  - This is dangerously wrong.
  - The thread isn't suspended: it's *running* gettid()!

Q5 – Blocking

For full credit

- Blocked thread can't run until a specific event
- Blocked thread is *not in a run queue*

Dangerous idea

- "If a thread invokes `gettid()`, the thread's execution is suspended until the system call returns."
  - This is dangerously wrong.
  - The thread isn't suspended: it's *running* gettid()!

The "hierarchy"

- Running and doing useful work (user mode *or kernel mode*)
- [Running and doing "locking work"]
- Runnable but not running (in scheduler "run queue")
- Blocked = not running and not runnable

Q5 – Blocking

Common misconception

- Question text reminds: especially on a multiprocessor, "might need a lock" does *not* mean "likely to block"
  - Remember that we assume most locks are *usually not contested* and are *held briefly*

Common glitches

- Vagueness about non-runnability (common deduction: "-1 OOQ")
- Explaining why *half* of what `swexn()` does shouldn't block

Breakdown

```
90% = 63.0
80% = 56.0
70% = 49.0
60% = 42.0
50% = 35.0
<50%
```

Breakdown

```
90% = 63.0   13 students (68/70 is top)
80% = 56.0   22 students
70% = 49.0   17 students
60% = 42.0    7 students
50% = 35.0    5 students
<50%          2 students
```

Comparison

- Median grade was a low B, so this probably wasn't a "killer exam"

Implications

Score "sub-C" (40..49)?

- 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 40?

- 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!**