15-410

"My other car is a cdr" -- Unknown

Exam #1
Oct. 16, 2012

Dave Eckhardt

Synchronization

Checkpoint 2 - alerts

- 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

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

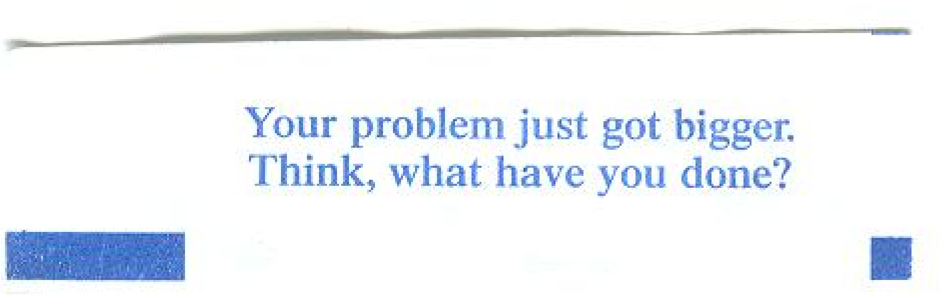
Debugging advice

- Once as I was buying lunch I received a fortune

Synchronization

Debugging advice

- Once as I was buying lunch I received a fortune



Your problem just got bigger.
Think, what have you done?

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 – “Atomic Instruction Sequence”

Expected

- Short sequence
- Must not be interleaved with some “related sequences”
- Typically nobody is trying to interleave “against us”
  - It *can* happen, but it's too rare for us to use a “big hammer” in the common case

Q1a – "Atomic Instruction Sequence"

Expected

- Short sequence
- Must not be interleaved with some "related sequences"
- Typically nobody is trying to interleave "against us"
  - It *can* happen, but it's too rare for us to use a "big hammer" in the common case

Most-common problem

- An atomic instruction sequence must not be interrupted
  - Actually, the problem is that it *will be* interrupted
    - » For sure if it's user-space code
    - » Probably even if it's kernel code (don't forget about multiprocessor machines!")
  - The key idea is that we must control the bad interleavings *even when* the sequence is interrupted
    - » "Atomic effect even if not atomic execution"

Q1b – “kernel mode”

Hoping to see

- PL0
- Can access hardware devices
- Can access “kernel-only” memory/data structures
- Can access processor control registers
- Provides crash isolation among users (referee)
- Is entered on syscall/trap/fault/exception

Q1b – "kernel mode"

Hoping to see

- PL0
- Can access hardware devices
- Can access "kernel-only" memory/data structures
- Can access processor control registers
- Provides crash isolation among users (referee)
- Is entered on syscall/trap/fault/exception

Two worrisome themes

- Kernel mode is the privileged mode that the kernel runs in
  - Ok, I guess so, but why?
- Kernel mode is for code that touches the kernel stack
  - True, but not really the heart of the matter (again: why?)

Q2 – Broken “Dekker's Algorithm”

Good news

- Most people saw a mutual exclusion failure

Common issues

- Leaving out part of the trace
    - Leaving out one observation of a key variable/value
    - “Really leaving stuff out” - something missing from both threads

Q2 – Broken “Dekker's Algorithm”

Good news

- Most people saw a mutual exclusion failure

Common issues

- Leaving out part of the trace
    - Leaving out one observation of a key variable/value
    - “Really leaving stuff out” - something missing from both threads

Less-common issues (carefully review your exam)

- Nobody who tried to show a bounded-waiting failure did
    - Key problem: incorrect definition of bounded waiting
- Some people wrote traces of the algorithm working
    - Advice: practice some old homework questions about Dekker or Bakery

Q3 – Graders' Algorithm

Good news

- Most people found the deadlock
- Dangerous (rare) issue
    - Misunderstanding how mutexes and cvars work (!!)
        - » `cond_wait()` drops and reacquires the mutex! This is a fundamental part of what it does, and this absolutely must be understood.
- Beware: Impossible/unclear execution traces
    - You need to be able to reason about these issues and communicate them to others.
    - Our exact format is not 100% necessary, but you need something at least that descriptive and clear.

Q3 – Graders' Algorithm

Some issues with specifying a fix

- Calling `examine_exam_number()` while holding a mutex is not a high-quality solution

Many issues about *explaining* a fix

- “Prevents hold&wait” isn't true if what is really happening is “Ensures at most one thread is holding and waiting”
  - That's “prevents cycles in the wait graph”

Q4 – “Channels”

Question goal

- “Write a synchronization object” - typical exam question

A word about (non-neutral) expectations

- Some people asked whether `receive()` should block or immediately return when nothing is queued

Q4 – “Channels”

Question goal

- “Write a synchronization object” - typical exam question

A word about (non-neutral) expectations

- Some people asked whether `receive()` should block or immediately return when nothing is queued
  - In general, if there is nothing for a thread to do, it should stop running! This is important!
  - Recall that we discussed the “offload the sleep(1) problem onto the caller” anti-pattern.

Q4 – “Channels”

Question goal

- “Write a synchronization object” - typical exam question

A word about (non-neutral) expectations

- Some people asked whether `receive()` should block or immediately return when nothing is queued
  - In general, if there is nothing for a thread to do, it should stop running! This is important!
  - Recall that we discussed the “offload the sleep(1) problem onto the caller” anti-pattern.
  - Occasionally a “`try_receive()`” or “`try_lock()`” operation is useful
    - » These are rare special cases, generally used to avoid deadlock in callbacks or interrupt handlers, and require care to use correctly
    - » They generally do not exist “alone” (without a blocking `receive()` which is used most of the time)

Q4 – “Channels”

Question goal

- “Write a synchronization object” - typical exam question

Hint (written in question text)

- “Synch” case and (normal) “asynch” case can be done with very similar code

Key design issue – who blocks when?

- Sender: buffer full (no space)
- Receiver: buffer empty (no data)
- “Synchronous Sender”: data stored but not yet removed

Unblocking

- Added data to buffer ⇒ unblock a receiver needing data
- Made space in buffer ⇒ unblock a sender needing space
- Made space in buffer ⇒ unblock a synchronous sender

Q4 – “Channels”

Grading

- 8 points for synch mode
- 12 points for asynch mode

Grader alarm

- Many solutions fail in *very* common (non-race) cases
  - “Init, then a sender sends an item” ⇒ crash
  - “Init, then a receiver arrives seeking an item” ⇒ crash

More-typical issues

- *Many* instances of “Paradise lost”
  - Please review the lecture, avoid that syndrome in kernel code
- {Sender,receiver} forgets to awaken {receiver,sender}
- One cvar used to indicate too many conditions

Q5 – Segmented Stack / ss_call()

Basic idea

- Call a function, but on a different stack area than the current one
  - Motivation: non-contiguous stacks avoid fragmentation issues

Solution ingredients

- Allocate the new stack area
- Switch to new stack area
- Run the function, remember the return value
- Switch back to old stack area
- Make sure all appropriate state is saved, transferred, restored

Hmm...

- "Kind of like": context switch/yield(), thr_create()

Q5 – Segmented Stack / ss_call()

Troublesome approaches

- `thr_create()/thread_fork`
  - Difficult to get right
  - HUGELY expensive (compared to malloc() + function call)
    - Multiple stacks, synchronization, thread create+destroy!
- `swexn()`
  - Also fundamentally not what was sought

Typical issues

- Minor calling-convention issues
- Omission of saving/restoring some particular thing
- Hand-writing `malloc()` in terms of `new_pages()` (it's easier and likely more correct to just use `malloc()`)

Suggestion

- Work from a checklist: alloc; save A, B, …; adjust A, B, ...

Q5 – Segmented Stack / ss_call()

“How to detect stack overrun?”

- Expected: sentinel/canary/magic-cookie
- Some solutions suggested things that are not feasible
  - “Protect last byte of ______”

Breakdown

90% = 63.0	10 students (66/70 is top)
80% = 56.0	13 students
70% = 49.0	9 students
60% = 42.0	12 students
50% = 35.0	6 students
<50%	3 students

Comparison

- If we count 48/70 == 49/70 the C/D break looks better
- Scores were “not high, not super low”

Implications

Score under 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 at/below 36?

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

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!
    - » So if this exam looks that way, you should definitely at least "see course staff" to reduce the likelihood that both do!

“Design” in this exam

Reminder...

- Final exam will focus more on “design”
  - On this exam, design was best represented by
    - » Q4 (channels)
    - » Q5 (ss_call)
  - If you were flummoxed by those two questions, try to figure out how to be less so in the future