15-410

"My other car is a cdr" -- Unknown

Exam #1
Oct. 14, 2014

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

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

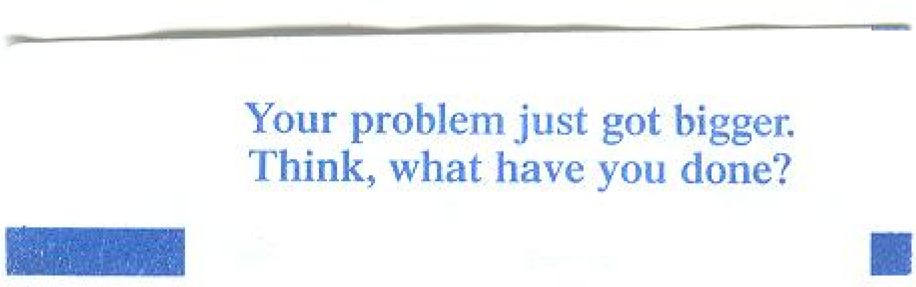
Debugging advice

- Once as I was buying lunch I received a fortune

Synchronization

Debugging advice

- Once as I was buying lunch I received a fortune



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?

"Andrew Linux" VM image?

- Issue-reporting hotline!
  - http://tinyurl.com/nqgedwu

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"

For full credit

- List *all three* assumptions we make
- All three matter in terms of architecture / implementation

Typical issues

- Missing one or two of the assumptions
- Over-claiming ("nothing else must run")
- Describing an atomic *instruction* (not a sequence)
- Getting something backwards
  - "It prevents interleaving" (correct: interleaving must be prevented on its behalf)

Q1b – “South Bridge”

For full credit

- Connects devices to CPU
- Something about which devices
  - Give examples, or
  - “The slower ones”
- Something about the connection (e.g, “via North Bridge”)

Most-common notable issues

- “SB == PIC”
- “It's in the CPU” (not all machine parts are CPU parts!)
- “It contains the IDT” (IDT is in RAM!)

Q2 – Yo!

Problem

- Find the race condition

Q2 – Yo!

Problem

- Find the race condition

Solution

- Well, there were two

Q2 – Yo!

Problem

- Find the race condition

Solution

- Well, there were two
  - “'Paradise Lost' ⇒ consume invalid work”
  - “Thread can get stuck indefinitely” (subtle)

Q2 – Yo!

Problem

- Find the race condition

Solution

- Well, there were two
  - “'Paradise Lost' ⇒ consume invalid work”
  - “Thread can get stuck indefinitely” (subtle)
    - » If you found the subtle one but not the simpler one, maybe go back and look at the problem again as practice

Q2 – Yo!

Good news

- ~25% of class got a perfect score
- Another ~20% came pretty close

Q2 – Yo!

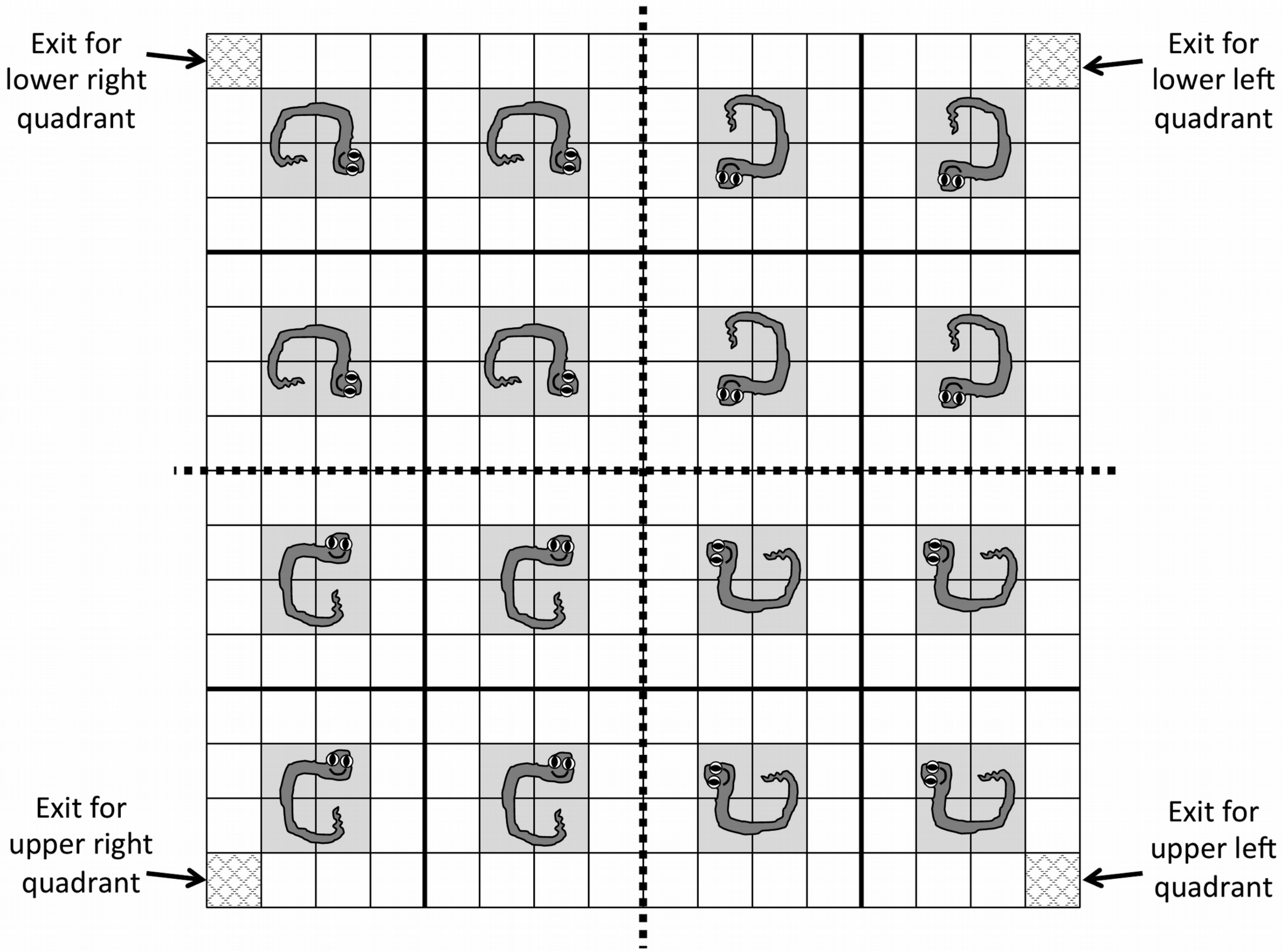
Good news

- ~25% of class got a perfect score
- Another ~20% came pretty close

Less-good news

- ~30% of the class had serious trouble
  - Finding race conditions is an important skill
    - This one wasn't super-easy, but it wasn't super-hard either
    - Suggestions
      - *Carefully* review "Synchronization" lectures
      - Be sure to practice this for final exam
  - Writing traces is an important skill too

Q3 – Deadlock


Exit for lower right quadrant
Exit for lower left quadrant
Exit for upper right quadrant
Exit for upper left quadrant

Q3 – Deadlock

Parts of the problem

- Basic deadlock explanation
- Deadlock prevention?
- Deadlock avoidance?

Q3 – Deadlock

Parts of the problem

- Basic deadlock explanation
  - Most people did well here
- Deadlock prevention?
  - It can be done – with understanding and creativity
- Deadlock avoidance?
  - It can be done – with understanding and creativity

Q3 – Deadlock

Deadlock *prevention*

- Pick a deadlock ingredient to permanently ban
  - Only one of the four is really plausible
- Figure out how to solve the problem with that ban in place
  - One approach tweaks initial snake locations using an initial override step
  - Another approach involves careful understanding of geometry and paths

Q3 – Deadlock

Deadlock *prevention*

- Pick a deadlock ingredient to permanently ban
  - Only one of the four is really plausible
- Figure out how to solve the problem with that ban in place
  - One approach tweaks initial snake locations using an initial override step
  - Another approach involves careful understanding of geometry and paths

Misconceptions / non-solutions

- “Try to lock the space I want else spin”
  - Two snakes can want each other's spaces
  - “Spin-trylock” isn't different than “lock”
- “Tweak strategy and hope”
  - Solving the problem requires banning something (and then making the new system work)

Q3 – Deadlock

Deadlock *avoidance*

- This is a trickier approach
  - Processes must *pre-declare* their *worst-case* usage
    - What they need before they can free things
  - Resource allocator must compute based on *future* collisions

Common “glitches”

- Not taking into account that each snake initially owns some resources
- Wrong avoidance algorithm
  - In this problem, each square is unique
  - An algorithm for multi-instance resources won't work

Q3 – Deadlock

Conceptual problems

- “Safe sequence” is not “execution sequence”
  - A safe sequence is part of a proof-by-example computation
    - “We can enter state X because we know a *bad* way to get out of state X”
    - We don't plan to *use* that bad way
    - Usually somebody will use *less than* their worst-case needs
    - So usually we will execute in parallel

Q3 – Deadlock

Conceptual problems

- "Safe sequence" is not "execution sequence"
  - A safe sequence is part of a proof-by-example computation
    - "We can enter state X because we know a *bad* way to get out of state X"
    - We don't plan to *use* that bad way
    - Usually somebody will use *less than* their worst-case needs
    - So usually we will execute in parallel
- There must be an initial "request lots of stuff" step
  - Avoidance isn't about careful consideration of each request in isolation
  - The key is considering requests vs. knowledge of the future

Q3 – Deadlock

“Run one snake at a time”

- This is *a* solution
  - *Every* concurrency problem can be solved by a global mutex
- It is never a *high-quality* solution

Other issues

- Solution described is prevention, not avoidance

Solution hints

- Mentally run one snake to completion to understand path properties
- Figure out which other snakes could run concurrently

Q4 – “Select Variables”

Question goal

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

General conceptual problems

- Everything must be initialized and destroyed
- “x() takes a pointer” does *not* mean “x() must call malloc()”
- Other “malloc() issues”
  - malloc()/free() must be paired
  - Prefer “list of objects” to “list of object pointers”
- See course staff about any conceptual problems revealed by this specific exam

Q4 – “Select Variables”

A particular anti-pattern

- “broadcast() and let threads fight it out”
  - This is usually *a* solution
    - Many synchronization problems can be “addressed” by having everybody spin all the time
  - It is not a *high-quality* solution
    - Threads should run when they can probably make progress, and should be blocked when they probably can't make progress
    - “Wake 1000 when only 1 can win” is not “can probably make progress”

Q4 – “Select Variables”

Synchronization/concurrency problems

- `cond_signal()` shouldn't block indefinitely
  - Taking locks is necessary, but the job is *awakening*
  - Blocking is a potential deadlock factory
- An awakened thread shouldn't be re-awakened later
  - “One wakeup per block”
- Condition variables don't “store up” awakenings
  - If nobody is awakened, the signal has no future effect
- Object-global state must be managed carefully
  - One “return code” can be set multiple times before anybody can view it
- Be sure an object isn't still in use before destroying it
- Beware “anti-FIFO” patterns (e.g., stack)

Standard issues

- “Paradise Lost”

Q5 – Process Model

Q: “PUSHL (PL3) … PUSHL (PL0)”; why?

- The question tests understanding of how/why execution enters kernel mode

Q5 – Process Model

Q: “PUSHL (PL3) … PUSHL (PL0)”; why?

- The question tests understanding of how/why execution enters kernel mode

What we expected

- Three reasons
- Sufficient detail to convince us
- No “dangerous visions”

Q5 – Process Model

Q: “PUSHL (PL3) … PUSHL (PL0)”; why?

- The question tests understanding of how/why execution enters kernel mode

What we expected

- Three reasons
  - One voluntary, two involuntary
  - One asynchronous, two synchronous
- Sufficient detail to convince us
  - “Context switch” isn't a cause; it's an effect
- No “dangerous visions”

Q5 – Process Model

Q: "PUSHL (PL3) … PUSHL (PL0)"; why?

- The question tests understanding of how/why execution enters kernel mode

What we expected

- Three reasons
  - One voluntary, two involuntary
  - One asynchronous, two synchronous
- Sufficient detail to convince us
- No "dangerous visions"

"Dangerous visions"

- "`swexn()` handlers run in kernel mode"
- "Some other thread might ..."

Q5 – Process Model

Q: “PUSHL (PL3) … PUSHL (PL0)”; why?

- The question tests understanding of how/why execution enters kernel mode

What we expected

- Three reasons
  - One voluntary, two involuntary
  - One asynchronous, two synchronous
- Sufficient detail to convince us
- No “dangerous visions”

“Dangerous visions”

- “`swexn()` handlers run in kernel mode”
  - They'd better not!
- “Some other thread might ...”
  - True, but how would that affect this processor's execution?

Breakdown

Data-integrity warning

- 5 students took a makeup exam
- Their scores are not included here

Breakdown

90% = 67.5

80% = 60.0

70% = 52.5

60% = 45.0

50% = 37.5

40% = 30.0

Breakdown

90% = 67.5	1 student	(top: 70/75 = 93%)
80% = 60.0	3 students	
70% = 52.5	18 students	(52 and up)
60% = 45.0	9 students	(44 and up)
50% = 37.5	8 students	(37 and up)
40% = 30.0	10 students	
<40%	6 students	

Breakdown

90% = 67.5	1 student	(top: 70/75 = 93%)
80% = 60.0	3 students	
70% = 52.5	18 students	(52 and up)
60% = 45.0	9 students	(44 and up)
50% = 37.5	8 students	(37 and up)
40% = 30.0	10 students	
<40%	6 students	

Comparison/calibration

- Scores were lower than typical, more “double peak”
- Very-low exams mostly clobbered on “Yo!” *and* deadlock
  - Some did ok on “select variables” - this is hopeful

Implications

Some scaling is likely

- TBD, pending missing scores

Score “sub-C” (~35..40)?

- 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

Implications

Score below 37?

- Something went *dangerously* wrong
  - It's important to figure out what!
- Beware of “triple whammy”
  - Low score on “Yo!” *and* deadlock *and* select-vars
    - Those questions are the “core material”
    - Strong scores on Q1+Q5 don't make up for serious trouble with core material
- 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!**