15-410

"My other car is a cdr" -- Unknown

Exam #1
Oct. 16, 2019

Dave Eckhardt

Dave O'Hallaron

Synchronization

Checkpoint 2 – Wednesday, in Wean 5207 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

Book report!

- Hey, “Mid-Semester Break” is just around the corner!

Synchronization

Asking for trouble?

- If you aren't using source control, that is probably a mistake
- If your code isn't in your 410 AFS space every day, you are asking for trouble
  - GitHub sometimes goes down!
    - S'13: on P4 hand-in day (really!)
  - Roughly 1/2 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
  - Don't forget about CC=clang / CC=clangalyzer
- Running your code on the crash box may be useful
  - But if you aren't doing it fairly regularly, the first "release" may take a *long* time

Synchronization

Google “Summer of Code”

- http://code.google.com/soc/
- Hack on an open-source project
  - And get paid
  - And quite possibly get recruited
- Projects with CMU connections: Plan 9, OpenAFS (see me)

CMU SCS “Coding in the Summer”?

Synchronization

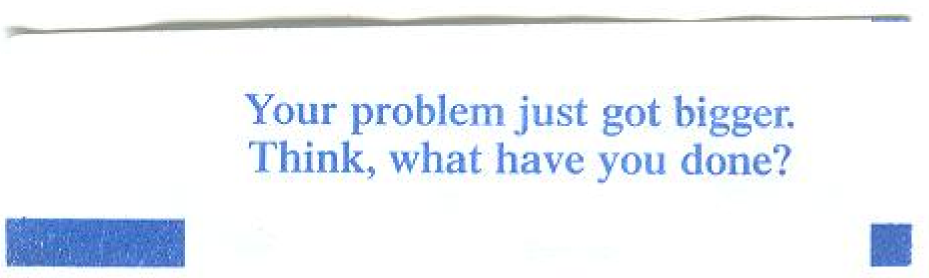
Debugging advice

- 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

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 could be more stuff (~100 points, ~7 questions)

“See Course Staff”

If your exam 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.

...though it might instead indicate a complex subtlety

“Low Exam-Score Syndrome”

What if my score is really low????

- It is frequently possible to do *dramatically* better on the final exam
- Specific suggestions later

Outline

Question 1

Question 2

Question 3

Question 4

Question 5

Q1a – P2 design decision

Purpose: demonstrate grasp of a design tool

- Hopefully P2 involved deliberate design
- Hopefully P3 is involving deliberate design
- “Robust code is *structurally different* than fragile code”
- P3 requires not just code but *structurally non-fragile code*.

If you were lost on this question...

- We had a lecture on this topic (August 30)
- Other “odd” lectures to possibly review
  - Debugging, Errors
  - #define, #include
  - We expect you to know *and apply* all of this material

Q1b – Register Dump

Question goal

- Stare at a register dump and form a plausible hypothesis
  - Why? Debugging P3 will require staring at bits to figure out what's wrong... this is a good way to figure out if some practice is needed

Hint

- Two registers are wrong with respect to a third register

Common issues

- It is necessary to say ***why/how*** a wrong register leads to an exception
  - "%xxx should point at Y, not at Z" is not a fault type in this situation
  - "Page fault" is actually fairly *un*likely
  - Some faults not really possible in P2/P3 were claimed

Q1 – Overall

Scores

- Almost everybody scored 8/10 or better

Q2 – Critical-section protocol

What we were testing

- Find a race condition (important skill)
- Write a convincing trace (demonstrates understanding)

Good news

- 33/39 students scored 14/15 or better

Minor issues

- Trace doesn't have an *exactly-repeating* part
- Trace doesn't *clearly identify* the exactly-repeating part

Alarming issues

- Trace requires a thread to "run at zero speed"
- Trace can't happen

Advice

- Don't "just start writing a trace" (start on scrap paper)

Q3 – Battleship Deadlock

Question goals

- Diagnose a deadlock situation, based on deadlock principles
- Show a trace
- Design a solution

Q3 – Battleship Deadlock

Common issues

- “Global mutex” is an *emergency* solution to deadlock
  - Not a good solution
- Memorizing the four deadlock ingredients probably is a good idea
- Generally, avoid traces with multiple operations in a single row
  - Unless clarity is genuinely improved
- Not all “tabular traces” were tabular
  - A paragraph isn't really a trace

Specific to this question

- If your solution requires rollback (not all do), forgetting rollback results in incorrect outcomes

Q3 – Battleship Deadlock

Scores

- 32/39 students (~82%) scored 13/15 (86%) or better

Q4 – Boolean Cyclic Barriers

Question goal

- Variant of typical “write a synchronization object” exam question
- This was was probably “typical” (not “easy”, nor “killer”)

Key issue

- Threads from Phase t+1 could arrive before threads from Phase t have finished leaving

Some workable architectures

- Preserving the “old” result
- Stalling premature arrivals
- Creating a “mailbox” per thread

Q4 – Boolean Cyclic Barriers

Common issues

- Violations of interfaces(!)
- Forgetting to reset state after one phase's arrivals have happened
- Forgetting to unlock
- cond_broadcast() inside a mutex
  - This is not generally necessary, and is a big concurrency lose

Alarming issues

- Reading fields before acquiring a lock
- bcb_destroy() calls free(bp)
- bcb_arrive() returns other than true/false
  - Review "Errors" lecture?

Q4 – Boolean Cyclic Barriers

Synchronization problems

- Spinning is *not ok*
- Yield loops are "arguably less wrong" than spinning
  - Motto: "When a thread can't do anything useful for a while, it should block; when a thread is unblocked, there should be a high likelihood it can do something useful."
  - Special case: mutexes should not be held for genuinely indefinite periods of time
- Blocking should use an underlying primitive (cvar, semaphore) rather than implementing one manually

Q4 – Boolean Cyclic Barriers

Important general advice!

- It's a good idea to trace through your code and make sure that at least the simplest cases work without races or threads getting stuck
- Maybe figure out which operation is “the hard one” and pseudo-code that one before coding the easy ones?

Other things to watch out for

- Memory leaks
- Memory allocation / pointer mistakes
- Forgetting to shut down underlying primitives
- Parallel arrays (use structs instead)

Q4 – Boolean Cyclic Barriers

Outcome

- 22/39 students (~50%) scored 14/20 (70%) or better
- 9/39 students (~25%) scored 10/20 (50%) or worse
    - “Severe tire damage” group is typically ~30%

Implications

- Being able to write this kind of code shows understanding of primitives and also hazards
- Life in P3 (and after) may involve embodying special-purpose synchronization patterns in code

Q5 – Stack Picture

Question goals

- Test understanding of stack
  - Quite important for P0, P2, P3
  - Somewhat important for P1
  - Probably important for P4
- Bonus: slightly test understanding of other regions

High-level inventory

- Enough stack frames
- Enough pieces in each stack frame
- Getting the struct in the right place
  - Getting fields in the right order
- Not putting strings in strange places

Q5 – Stack Picture

Specific issues

- "char *" means "4 bytes of pointer to a string that is stored somewhere else"
  - So the bytes H, i, r, o should not appear in the stack
  - It is *possible* for the bytes of a string to appear in the stack
    - But then they almost always are null-terminated
- Generally string *constants* ("Hiro") are stored in rodata
- Struct fields occupy increasing addresses

Somewhat alarming

- In C, stack frames are "reclaimed" in strictly-FIFO fashion
  - This is not true in ML, but C is not ML
- In x86-32, %ebp is saved on the stack
  - At least for this class, which adheres to the true convention

Q5 – Stack Picture

Outcome

- 20/39 students (~50%) scored 8/10 or better
- 6/39 students (~15%) scored 5/10 or worse

Breakdown

```
90% = 63.0   13 students
80% = 56.0   15 students
70% = 49.0    7 students
60% = 42.0    3 students
50% = 35.0    0 students
<50%          1 student
```

Comparison

- Median grade was 83%, so this was an easy-ish exam
  - Last semester's median was 61%

Implications

Score below 53?

- 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?
- It is important to do better on the final exam
  - Historically, an explicit plan works a lot better than “I'll try harder”
  - *Strong suggestion*:
    - Identify causes, draft a plan, see instructor

Implications

Score below 45?

- Something went *noticeably* wrong
  - It's *important* to figure out what!
- Beware of “triple whammy”
  - Low score on *three* questions
    - Generally Q2, Q4, Q5
- Passing the final exam could be a challenge
- *Passing the class may not be possible!*
  - To pass the class you must demonstrate proficiency on exams (not just project grades)
- Try to identify causes, draft a plan, see instructor

Action plan

Please follow steps in order:

1. Identity causes
2. Draft a plan
3. See instructor

Action plan

Please follow steps in order:

1. Identity causes
2. Draft a plan
3. See instructor

Please avoid:

- "I am worried about my exam, what should I do?"
  - *Each person should do something different!*
  - Thus "identify causes" and "draft a plan" steps are individual and depend on some things not known by us

Action plan

Please follow steps in order:

1. Identity causes
2. Draft a plan
3. See instructor

Please avoid:

- "I am worried about my exam, what should I do?"
  - *Each person should do something different!*
  - Thus "identify causes" and "draft a plan" steps are individual and depend on some things not known by us

General plea

- Please check to see whether there is something we strongly recommend that you have been skipping because you never needed to do that thing before
  - This class is different