15-410

"My other car is a cdr" -- Unknown

Exam #1
Oct. 13, 2009

Dave Eckhardt

Garth Gibson

Synchronization

Checkpoint 2 –Wednesday

- 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
- 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

Crash box

- How many people have had to wait in line to run code on the crash box?
  - How long?

Synchronization

Debugging advice

- Last year as I was buying lunch I received a fortune

Synchronization

Debugging advice

- Last year as I was buying lunch I received a fortune

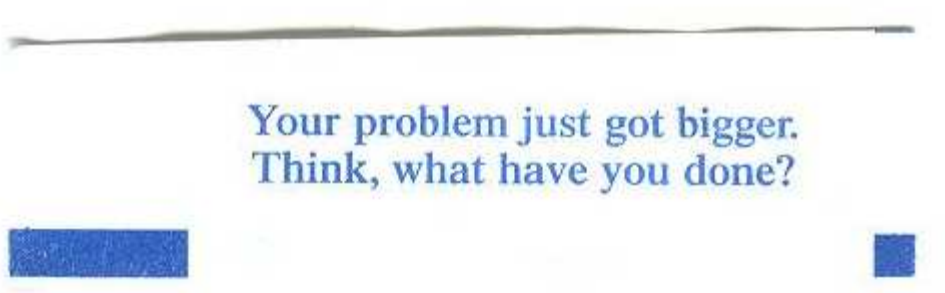

Image credit: Kartik Subramanian

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 more stuff (~100 points, ~7 questions)

“See Course Staff”

If your paper says “see course staff”...

- **...you probably 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 – “trap” vs. “fault”

Related concepts abound!

- Trap, fault, interrupt, “machine check”, “NMI”
- Each is a “surprise” to the processor
- Key differences
  - Is ___ synchronous to the instruction stream?
  - Can ___ be recovered from?
  - Is ___ normal or abnormal?
  - What is the next user-space instruction to be executed?

Q1b – "Bounded waiting"

Most-common mistake: defining the *other* thing

This is a useful concept

- **Is it easy or hard to obtain inside your kernel?**

Q2 –main() wrapper

Good news

- Many people got this substantially right (median 8/10)

Background issues

- Where are argc and argv stored?
- PUSHA (on "general principles"? What *must* be saved?)
- x86 vs. x86-64 (every system is different!)
- In stack-based languages, the stack is "where the action is". Every bit of detail you can grasp will enable you to debug some problem. Carpe diem!

Documentation issues

- Stack pictures are good! (See P2 & P3 handouts)
- If code changes, documentation may need to as well

Q2 –main() wrapper

Other issues

- Order of pushing things on stack
- Not setting up a legal stack frame
  - Running main() “lasts a while” - stack-trace should work!

Q3 –North/South Bridge

Two proposed algorithms to manage bridge crossings

- **while (!available) { cond_wait(&done, &bm); }**
- **while (ready_for != my_ticket) { cond_wait(&done, &bm); }**

Q3 –North/South Bridge

Two proposed algorithms to manage bridge crossings

- **while (!available) { cond_wait(&done, &bm); }**
- **while (ready_for != my_ticket) { cond_wait(&done, &bm); }**

This problem is about two ubiquitous threats to concurrent code

- **...?**
- **...?**

Q3 –North/South Bridge

- **Two proposed algorithms to manage bridge crossings**
  - **while (!available) { cond_wait(&done, &bm); }**
  - **while (ready_for != my_ticket) { cond_wait(&done, &bm); }**
- **This problem is about two ubiquitous threats to concurrent code**
  - **Starvation / “Unbounded waiting” (first algorithm)**
  - **Deadlock (second algorithm)**
- **Each version is thwarted by an “evil third thread”**
  - **Or a stream of them**

Q3 –North/South Bridge

Two proposed algorithms to manage bridge crossings

- **while (!available) { cond_wait(&done, &bm); }**
- **while (ready_for != my_ticket) { cond_wait(&done, &bm); }**

Common misconception - “Paradise lost”

- **Happiness does indeed phase in and out in in #1**
  - **The “evil third thread” can get in the way**
- **But we *defend* against the possibility with the loop**
  - **Recall outline of “Paradise Lost” lecture: “if() vs. while()”**

Q3 –North/South Bridge

Another common misconception: “++”

- x = ++y; // y's value must go through the ++ to get to =
- x = y++; // y's value is next to the =; ++ “off to the side”

Trouble

- “Hold a lock around the I/O” - avoid this when possible, since I/O takes “forever” (milliseconds!!)

Cautions

- 15-410 cvars, especially ones you write, are probably strictly FIFO. POSIX cvars are not, so don't burn that too deeply into your reasoning.
- “Clear” execution traces probably show all synch. ops

Q4 –TA-status server

“What's wrong with this picture?”

- A race between MSG_FIN and MSG_QUERY
  - A referenced object can be destroyed
  - A destroyed object can be referenced
- Most people found the problem - good

Q4 – TA-status server

Challenge

- Deleter must know when nobody else still has a pointer to the object
- An isomorphic problem “might” turn up in your kernel!

Non-scalable approaches

- “Solve the deletion problem” by never deleting!
  - This is a “memory leak” - not a good plan
  - Systems like this can't be extended (e.g., “add_new_ta()”)
- Add a “global lock” which serializes all execution
  - Defeats the goals using threads (esp. on multi-processors!)!
  - Design: locks affecting more threads must be held more briefly
  - Advice: name locks clearly (“big name” may mean trouble)

Q4 –TA-status server

Challenge

- Deleter must know when nobody else still has a pointer to the object
- An isomorphic problem “might” turn up in your kernel!

Approaches with promise

- “Lock handoff” - table lookup returns object already locked against disappearance
- Deleter flushes out inspectors with an rwlock
- If the problem is references others have... count them!

Q4 – TA-status server

Reference counts

- Object “knows” how many people have pointers to it
- Depending on circumstances, anybody may end up with “the last pointer”
  - Maybe the thread who is deleting it
    - » “Delete” now means “remove from table; flag as 'done'”
  - Maybe that pesky thread with the “old reference”

Q4 –TA-status server

```
foo_destroy(foo *fp) {
  lock(fp);
  if (--fp->refs > 1) { // Still live...
    unlock(fp);  return;
  }
  ...destroy parts...
  ...free object...
}
```

Notes

- Table presence counts as 1 reference, “cloned” on return
- Many calls to “destroy foo” - most don't really destroy it

Q5 –“Semaphores Rule!”

Goal

- Write mutex and cvar using (nothing but) semaphores

Key observation

- mutex = mutual exclusion, cvar = “expert waiting”
- semaphore = mutual exclusion *plus* “expert waiting”
- Fundamental “objects” recur throughout concurrent code
  - Understanding and being able to rearrange/redeploy is key

Two parts

- “Implement mutex” - widely solved
- “Implement cvar” - much more trouble!

Q5 –“Semaphores Rule!”

“Big” problems

- cas2i_runflag()
  - Forbidden by problem statement!
  - The world is not full of systems with cas2i_runflag()
  - The world *is* full of “use understanding of core principles to solve a problem with different constraints or tools”
- “semaphore == cvar”
  - cond_wait(c,m) { sem_wait(c->sem); } // “m” often unused!
  - cond_signal(c) { sem_signal(c->sem); }
  - This fundamentally doesn't work
    - » cvar “generally waits”
    - » semaphore “generally does not wait”

Q5 –"Semaphores Rule!"

Plausible approaches

- "mutex plus chain of semaphores"
  - Elaborate "atomic sleep" code is *not* necessary!
    - » Semaphore already encapsulates a solution for this!
- "mutex plus semaphore plus waiter count"
  - Good insight!
  - Some worried: "Only *mostly* FIFO"
    - » True, but also true of cvars

Other issues

- "Gratuitous malloc() - see course staff" (please do)
- "Mistakes": lock leak... memory leak... (see papers)

Breakdown

90% = 67.5 6 students (67.0 and up)

80% = 60.0 14 students

70% = 52.5 20 students (52 and up)

60% = 45.0 20 students (44 and up)

50% = 37.5 11 students

<50% 10 students

Comparison

- **Scores are a bit under typical (~3 points)**
- **Beyond that, more “low end” than typical**

Implications

Score 44..52?

- Figure out “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 44?

- Something went *rather* wrong
  - It's important to figure out what!
- Passing the final exam may be a serious challenge
- To pass the class you must demonstrate some proficiency on exams (not just project grades)