15-410

"My other car is a cdr" -- Unknown

Exam #1
Feb. 28, 2011

Dave Eckhardt

Synchronization

Checkpoint schedule

- Wednesday during class time
- Meet in Wean 5207
  - If your group number *ends* with
    - » 0-2 try to arrive 5 minutes early
    - » 3-5 arrive at 10:42:30
    - » 6-9 arrive at 10:59:27
- Preparation
  - Your kernel should be in mygroup/p3ck1
  - It should load one program, enter user space, gettid()
    - » Ideally lprintf() the result of gettid()
  - We will ask you to load & run a test program we will name
  - Explain which parts are “real”, which are “demo quality”

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
- 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

Upcoming events

- 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

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"?

Synchronization

Crash box

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

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

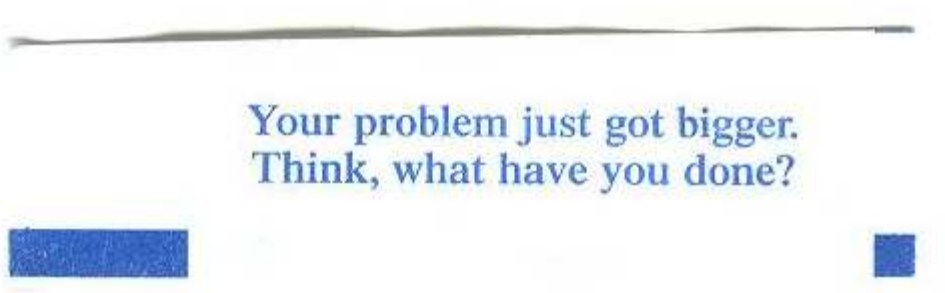

Image credit: Kartik Subramanian

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 – “stack frame”

Key issue

- “stack frame”≠“stack”!!!
- A stack is made out of frames
  - There is a “frame pointer” register, which points to the base of the current frame
  - etc.

What you should tell us

- What's in a frame
- Relationship between one frame and the next

Q1b – “thread-safe”

Key idea

- It's safe for multiple threads to invoke it

Going a little overboard...

- “serializable” is stronger than “thread safe”
- “bounded waiting” is not necessarily ensured (a weaker promise may be ok)

What you should tell us

- Identify some threats to thread safety
- Identify some techniques for ensuring thread safety

Q2 – “Socket locks”

Good news, bad news

- Many people did “pretty well”
- There was a steep drop-off

Frequent problems

- Multiple threads can acquire the lock!! (Generally: “Paradise Lost”, please study that lecture very carefully)
- Possible to “acquire” a lock after it is broken (“broken” checked at the wrong time)
- Some threads waiting for a lock get “stuck” when it is declared broken
- Deadlock
- Lock leak (“return -1” without unlock – then it gets *really* quiet)

Q2 – “Socket locks”

Other problems

- cond_wait() really does need two parameters!
  - Review lecture material and/or your P2 code
- Too many cond_broadcast()
- This problem does *not* require malloc()... if we asked you to “see course staff”, please do.

Q3 – rwlocks

Warning

- This is “not particularly good” rwlock code

Starvation

- You should be able to identify who is starved (and state why/how)
- You should be able to identify who is *not* starved (and state why/how)
  - Many people forgot to explain this

Lock dependency graph

- Most people (80%) who got this part right did well on the rest of the problem
  - The invariant comment hint in rwlock_unlock() is important
  - Complicated problems do benefit from careful analysis

Q3 – rwlocks

Deadlock?

- Almost!
- There is one “deadlock case”, but by definition it can never actually happen
- Careful analysis is required to see the “problem case” and then to explain why it isn't actually a problem

Be careful!

- “Somebody grabs a lock and never unlocks it” conquers every kind of lock, so it proves nothing
- “People might grab a pair of xxx locks in a bad order” does not mean “the xxx-lock code has a deadlock”

Q4 – “process model” (blocking)

Key issue

- Many people lost track of the “runnable” state
- In general, most things that are not running are not-running just because there aren't enough CPU's right now
  - That condition is not “blocked”!

Frequent fuzziness

- “_____ might need some kernel lock, in which case it would block”
  - We graded gently here, but... be very careful to keep in mind “short locks” versus “long locks”
    - » aka “atomic sequences” vs. “voluntary descheduling”
    - » On a multi-processor, “short locks” do *not* generally block threads!
    - » There are more and more multi-processors...

Q4 – “process model” (blocking)

Specific issues

- yield()'s job is fundamentally not blocking
  - We might immediately be picked up by another CPU
  - The person we yield to might get nailed by a timer tick right away, so we could be running again
  - There is no “missing part to our computation” that forbids us from continuing... just a deficiency in the amount of computers
  - We are in the scheduler's queue, ready to go
- sleep() really does block!
  - (Well, except for sleep(0), sleep(-1), etc.)
  - Our computation cannot continue until N ticks have happened
  - We'd better not be in the scheduler's queue!
- make_runnable()... beware “lock implies block” here

Q5 – alloca()

Double-emergency warning

- sizeof() *seriously* cannot be a function
  - There is no magic oracle which can stare at an address and tell when “the object” ends... definitely there is no oracle that can stare at a *value* and do that!
  - If you “called” sizeof(), please “revise and extend” your model of the C language appropriately

Emergency warning

- The problem states that alloca() can't be a regularly-called function... and it really can't.
  - If a() calls b(), and b() changes the size of a()'s frame, that is completely outside the calling convention

Q5 – alloca()

This was a “model” question

- **A check that you understand who the players are and their interaction rules**
- **It will be difficult to debug P3 code if you don't know how it is supposed to work right...**

Breakdown

```
90% = 67.5    7 students
80% = 60.0   14 students
70% = 52.5   15 students (52 and up)
60% = 45.0   18 students
50% = 37.5   19 students
<50%         10 students
```

Comparison

- This exam was a *little* tough... maybe 3-4 points' worth...
- That's not enough to explain the number of *very* low scores

Implications

Score under 50?

- 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 35?

- 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)

Implications

Reminder...

- Final exam will focus more on “design”
  - On this exam, most represented by cvars & rwlocks - if both were trouble for you, be warned!