15-410

"My other car is a cdr" -- Unknown

Exam #1
Oct. 17, 2007

Dave Eckhardt

Roger Dannenberg

Synchronization

Checkpoint 2 –Wednesday, in cluster

- Reminders
  - context switch ≠ mode switch
    - » Identify scenarios with one and not the other
  - context switch ≠ interrupt
    - » Later it will be invoked in other circumstances

Google “Summer of Code”

- http://code.google.com/soc/
- Hack on an open-source project
  - And get paid
  - And probably get recruited

CMU SCS “Coding in the Summer”

Synchronization

Debugging advice

- Monday as I was buying lunch I received a fortune

Synchronization

Debugging advice

- Monday as I was buying lunch I received a fortune

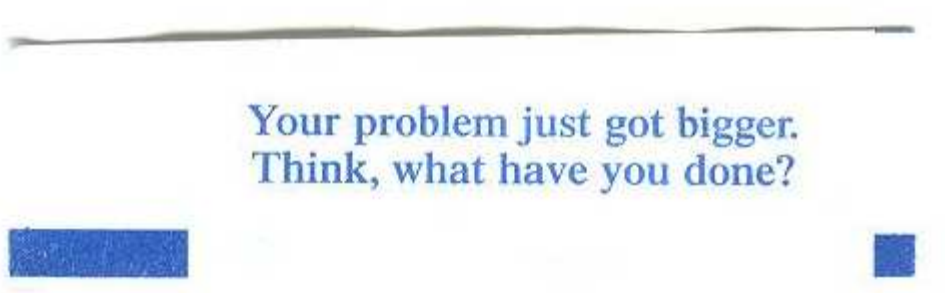

Image credit: Kartik Subramanian

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)

Outline

Question 1

Question 2

Question 3

Question 4

Question 5

Q1 –Short Answer

Write pipe (also known as “write buffer”)

- Key concept: the part of a “modern” computer which makes it “modern”
- Popular but not as relevant to this course
  - The write side of a pipe
  - Some kind of write buffer which isn't a write pipe

Q1 –Short Answer

Interrupt acknowledge

- Best answers covered:
  - What it's for
    - » Sending device back to start of protocol (enabling it to assert another interrupt later)
  - When it happens
    - » When processor has acquired the information necessary to characterize and handle the interrupt
  - How it happens
    - » Processor sends a command (in our world, via an OUTB)

Q2 –Monitor Implementation

Write some macros...

- M_DECL(), M_INIT(), etc.
- ...to support a “monitor style” of programming

Getting started

- `work_setup()` needs to `thr_create()` a worker thread
  - Nobody else can...

Locking issue

- Sometimes we need others to enter the monitor to progress us... `condition_wait()` will make that happen
- Sometimes we need others to *not* enter the monitor just yet... but `condition_wait()` will make that happen

Q2 –Monitor Implementation

Types and returning

- M_RETURN(t,v) –takes a type and a value
- There is a subtle locking problem here
  - What happens when I M_RETURN(int,some_global_int)?
  - M_RETURN() needs to accomplish two things
    - » Neither order will work
    - » So M_RETURN() needs to accomplish *three* things
      - That's what the type parameter is for

Scoping

- A common M_DECL() mistake would mean each program could contain only one monitor.

Q3 –Critical Section Algorithm

The mission

- Evaluate a proposed critical-section algorithm in terms of whether it provides mutual exclusion, progress, and bounded waiting

Terminology to watch out for

- Progress is about the *system*
- Bounded waiting is about a *particular victim*
  - Violating bounded waiting means “we can't write down a bound”
  - It does *not* mean “we can show there exists a small, bounded amount of unfairness” - strict FIFO behavior is *not* required, because it's much too hard

Q3 –Critical Section Algorithm

Mutual exclusion

- Pretty much everybody was able to show this was broken
- Some people lost some points for execution traces that were too terse (the loop is a key part of the story)

Q3 –Critical Section Algorithm

Progress

- **No!**
- **The key problem is that mutual exclusion is broken**
- **Two racing unlockers can leave the lock in a broken state**

Thread 2	Thread 1
T2 is done wanting	
Decide to appoint T1	
	T1 is done wanting
	Lock is available to all
Appoint T1	

Now T1 goes on vacation to Belize...

Q3 –Critical Section Algorithm

Progress

- *Not* progress violations
  - One thread might crash while holding the lock
  - One thread might never unlock the lock
    - » True, but not faults in the algorithm

Another way to show progress isn't assured

- `set()` isn't atomic

Other problems

- Bad execution traces which can't actually happen
- Explaining what the algorithm *wants* to do

Q3 –Critical Section Algorithm

Bounded waiting

- **No!**
  - **Gee, this algorithm isn't so hot, is it?**
- **Key problem: `set()`**

Q4 –Deadlock

Issues with the new cluster

- Description of resources (computers, servers, projector)
- Description of threads (OS, Networks)
- Deadlock? Yes/no/why?

(A) –Can OS students deadlock?

- Observe: this is “Dining Philosophers”!
- Observe: the projector injects a subtle yet important property...

(B) –Can Networks students deadlock?

- Can explain in terms of h&w or graph cycles
- Must state name of property and show it

Q4 –Deadlock

(C) –Can mixture of students deadlock?

- Parts of a complete solution
  - Diagram of sufficient clarity
  - Event trace of sufficient clarity (clear text was accepted)
  - Explanation of why the situation, as diagrammed and traced, is classified the way it is

Q5 –Your Partner's Code

```
char *the_word(int num)
{
    char buf[8];

    switch (num % 4) {
      case 0: snprintf(buf, sizeof(buf), "zero"); break;
      case 1: snprintf(buf, sizeof(buf), "one"); break;
      case 2: snprintf(buf, sizeof(buf), "two"); break;
      case 3: snprintf(buf, sizeof(buf), "three"); break;
    }
    return (buf);
}
```

Q5 – Your Partner's Code

(A) – What's wrong with this picture?

- The “213 answer”: returning a pointer to “automatic storage”

Claims difficult to support

- “Stack memory `disappears' when a function returns”
  - Set to zero...
  - Removed from address space...
  - Will cause a segmentation fault...
  - ...Unfortunately not true
- “`snprintf()` is not up to this job”
- “...the heap...”
- “`sizeof()` is evil”

Q5 – Your Partner's Code

"`sizeof()` is evil"

- There *are* times when sizeof() "doesn't do what you want"

```
void foo(char s[1024]) {
  ... sizeof(s) ... // not 1024
}

void bar(void) {
  char *s;
  s = malloc(512);
  ... sizeof(s) ... // not 512
}
```

Q5 – Your Partner's Code

"`sizeof()` is evil"

- There *are* times when sizeof() "doesn't do what you want"
- ...but it isn't *designed* to be wrong all the time!

The problem isn't actually sizeof()

- The issue is that in C *some* things which look like arrays aren't
- Pointers can be used like arrays, but are pointer-sized
- Function parameters which look like arrays are actually pointers, and are pointer-sized
- Actual arrays (local or global) are actually arrays, and are array-sized

Q5 – Your Partner's Code

(B) What's wrong with the code – in context?

- Two possible answers
- For complete credit, the less-than-obvious one is better
  - There isn't another *thread* out there, but...

Things to avoid

- "Some other thread..." - there are no other threads
- "The kernel..." - this code *is* the kernel
- Generally, avoid mysterious or missing actors

Breakdown

90% = 67.5 0 students

80% = 60.0 19 students

70% = 52.5 25 students (52 and up)

60% = 45.0 4 students

50% = 37.5 9 students

<50% 3 students

Comparison

- **Scores are lower than typical**
  - **Even if we correct for that person who clearly forgot to answer that one question**

Implications

Further analysis will probably suggest a mild scaling

- Maybe something like 3-5 points

Score below 70%?

- Figure out what happened
- Probably plan to do better on the final exam

Warning...

- To pass the class you must demonstrate reasonable proficiency on exams (project grades alone are not sufficient)
- See syllabus