Boot Camp

Dave Eckhardt
de0u@andrew.cmu.edu
Roger Dannenberg
rbd@cs.cmu.edu

This Is a Hard Class

- CS doesn't have “capstone” classes, but similar...
- Traditional hazards
  - 410 letter grade one lower than typical classes
  - All ***other*** classes this semester: one grade lower
- Aim
  - If you aim for a B you might not get one
  - If you aim for a C you might not get one
  - “I'll drop if I can't get an A”
    - You ***must*** discuss this with your partner ***early***

Good News

- Good news...it can be done!
  - Spring 2008
    - Two groups split due to drops
      - One has finished, one is expected soon
    - All other groups turned in working kernels
    - Lots of people graduated
- Remainder of this lecture
  - How to get from here to there

This is a ***Transformative*** Class

- Genuine achievement, available to you
  - What is an OS, ***really?***
  - Mutual exclusion, synchronization, concurrency
  - Deadlock
- Design, planning
- Serious competence in debugging!

Work Flow – You may be used to...

- Assignment handout ⇒ code outline
- Compilation implies correctness
- Graded by a script
- All done!
  - Never use it again
  - Delete it at end of semester
- ***Total opposite of real life***

Work Flow – 410 Additions

- Design
- Divide into parts
- Manage your partner
- Merge
- Debug ***hard*** problems

Surprises

- “Code complete” means ***“I am far behind”***
  - Merge can take ***three days***
  - Then you ***start*** to find bugs (1-2 weeks)
- Code with “the right idea” will ***immediately*** crash
  - If you're lucky!
- This is not a “basic idea is right” class
  - You can't ship “basic ideas” to customers
  - Understand all details–***then*** you have the basic idea

On Debugging

As soon as we started programming, we found to our surprise that it wasn't as easy to get programs right as we had thought. Debugging had to be discovered. I can remember the exact instant when I realized that a large part of my life from then on was going to be spent in finding mistakes in my own programs.

– Maurice Wilkes (1949)

Debugging

- Bugs aren't just last-minute glitches
- They are crucial learning experiences
  - Learning a lot can take a lot of time

What Does A Bug Mean?

- “It tells me 'triple fault' – why??”
  - Research: 20 minutes
  - Think: 20 minutes
  - Debug: 2 hours.
  - ...three times.
- May need to ***write code*** to trap a bad bug
  - Asserts or more-targeted debug module
- Then you will find your design was wrong!
  - Don't be shocked – this is part of 410 / life

“All Done”?

- Finally, when you're done...
  - You will use your code for the next assignment!
  - We will read it (goal: every line)

Interlude

- What is source code “for”?
  - What is done with it?

Interlude

- The purpose of code is for ***people to read***
  - By a reviewer / security auditor
  - By your group
  - By your manager
  - By your successor
  - By you six months later (6 hours later if no sleep)
- Oh, yeah, the compiler reads it too

Confront the Material

- We are doing printf() ***all the way down***
  - Subroutine linkage, how & why
  - Stub routine, IDT entry, trap handler wrapper
  - Output/input-echo interlock
  - Logical cursor vs. physical cursor
  - Video memory (what does scrolling mean?)
- Can't really gloss over ***anything***

On Investing

- A week of coding can sometimes save an hour of thought.
  - Josh Bloch

Confront Debugging

- Real life: you will debug other people's code
  - Any bug could be yours, partner's, ours, or Simics; you need to ***find*** it.
- ***Can't*** debug using only printf()
  - printf() ***changes your code***
  - printf() may be broken by whatever breaks your code
  - Learn the Simics debugger
  - Assertions, consistency checks
  - Debugging code

Confront Debugging

- ½ hour of studying the debugger
  - vs. 2 days of thrashing
- Papering over a problem
  - Re-ordering object files to avoid crash

How to Have Trouble

- How to get an R
  - Arrive unprepared (e.g., barely escape 113, 213)
  - Do everything at the last minute
  - Don't read the book or come to class
  - Hide from course staff no matter what
- How to get a D
  - Don't get the kernel project genuinely working
    - (There are other ways, but this one is popular)

Warning About 15-213

- It's an important class
- We expect you to ***know***
  - Byte, word, register, 1<<2
  - Thread, stack
  - malloc(), free() (when & why)
  - how to translate C ⇔ x86
- Trouble with 213?
  - If you didn't get a B or an A, see me
  - If the malloc() lab didn't go well, see me

Warning to Graduate Students

- This is an undergraduate class
  - There will be “a diversity of grades”
- Getting “average grades on every assignment” ***may well*** mean a C, not a B
- Working really hard and doing everything somewhere between “ok” and “well” may mean a B, not an A.
  - B requires ***repeated solid performance***
  - A requires ***repeated excellence***
  - (“Everything pretty much worked” is C territory)

Doing Well – Embrace the Experience

- Embrace the Unix development experience
  - If you try to keep it at arm's length it will slow you down
- Embrace the Simics debugger
  - If you try to keep it at arm's length it will slow you down
- Embrace source control
  - If you keep it at arm's length ...

Doing Well – Invest in Good Code

- Mentally commit to writing ***good*** code
  - Not just something kinda-ok
  - You will ***depend*** on your code
- Anand Thakker (Fall 2003)
  - Remind yourself that you love yourself
  - So you should write good code for yourself

Doing Well – Start Early

- Starting a week late on a 2-week project will be bad
- Not making "just one" checkpoint can be bad
  - Missing two kernel-project checkpoints...
    - ...may make passing impossible.

Doing Well – Read Partner's Code

- You will ***need*** to read everything your partner wrote
  - (and answer test questions about it)
- Set up a mechanism
  - Daily meeting? Careful reading of merge logs?
- Do “one of each”
  - Partner does N-1 stub routines, you should do the hardest

Doing Well – Time for Design

- “Design” means you may need to think overnight

How to get an A

- Understand ***everything***
  - (consider 2-3 ways to do each thing, pick the best)
- Read ***all of*** your partner's code
- Work ***with*** your partner
  - (not: work alone for 4-5 weeks out of 6, then (fail to) merge)

How to get an A

- Write ***genuinely excellent code***
- Do things which ***help you***
  - asserts, good variable names, source control
- Document ***before*** coding
  - Actual 15-410 students do this!
  - Simple, useful form: write several module .h's before code for any of them
- Be “done” ***days*** early