CS Undergraduate Committee
CS350 CS351 Course Review

Instructor (Michael Lee) Ideas for CS350/351 Redesign:
I've been thinking long and hard about the direction that these two classes are headed in right now, and have a couple of ideas/suggestions.

First of all, though I'd slowly drifted over to the school of thought (in the past few semesters of teaching only CS 350) that a computer organization course that is truly focused on the hardware design aspect of things is an advantage to the students, the first few weeks of this semester have totally changed my perspective. I've identified a few problems with the way the course is currently designed:

1. The lack of digital design prerequisites puts the CS students (for whom the course should target, I presume) at a distinct disadvantage.
2. In order to arrive at a meaningful discussion of datapath/control implementation in a reasonable amount of time, an old ISA (and one that cannot be seen in action, other than on a simulator) is taught --- this leads to the next problem.
3. It is hard to motivate students to learn an assembly language that they simply do not believe is useful (despite the fact that one tries all semester to persuade them that it is an "elegant", "representative" one), and for little purpose besides it being a necessary prerequisite for examining the microprocessor implementation.
4. It is hard to discuss optimization at a compiler level without having looked modern processor execution (e.g. out-of-order execution) --- but so much time and energy is spent on "simple" processor designs, that it is hard to draw student focus back to code optimizations (if modern processor execution is even covered).
5. What really is the benefit of the hardware design aspect of the course? If some students want to explore that facet, that should be what CPE courses are for, or CS 470/471, right?

My feeling is that a computer organization course which focuses, from the start, on the way in which knowledge of architectural details (e.g. data representation, execution style, memory organization, I/O) impacts the way software should be designed, written, and optimized, would benefit CS students a great deal more than the current hardware-centric syllabus does.

That brings me to how I'm currently attempting to teach CS 351. As a systems course, students should be acutely aware of how their software is being interpreted and executed by the underlying architecture --- I devote two weeks to bringing students up to speed on IA32 assembly, and two labs to it --- you can find the first of them on blackboard. I've found students to be far more receptive to these labs (despite their relatively high difficulty level and assembly content) because of their association with "real" compilation, machines, and code (vs. the simulated and artificial environment imposed in CS 350).

The text (http://csapp.cs.cmu.edu/) that I am using for the systems class (and the labs that accompany it) has been a great help --- I've found that the programmer-centric view makes here-to-fore dry topics (e.g. switch statement jump tables, out-of-order execution) interesting and practical again to the student, and satisfying to teach. The only problem now is that I simply do not foresee being able to cover all the material I had set out to cover, as much time is spent bringing students up to speed on the architectural details again.

Instead of condensing the material, my suggestion is to take the programmer-centric architectural portions of the 351 syllabus and move them into CS 350, along with the accompanying labs, thereby making CS 350/351 a two semester architecture/system offering. Not only would this reduce the amount of time spent in each course getting students up to speed, but it would offer

them the chance to become real "experts" on the platform being used (Linux/Intel); I believe this would do wonders for their confidence and abilities as serious programmers.

I've looked over the course overviews for CS 350 and CS 351, and I don't see anything in them that expressly prohibits this. The one problem, of course, is that, if the two courses are to be intrinsically tied together in the future, their management must be tightly monitored and instructors (if different) be in close communication. The prerequisite requirement (350 before 351) must also be strictly imposed (it hasn't been in the past, I believe).

I've come to my conclusions based on the surveys and analysis of at least half a dozen courses, including those offered at UCB, MIT, CMU, Stanford, and UIUC. A useful website I've come across that discusses the case for using Linux in universities is at http://www.kegel.com/linux/edu/. Also, the author's website for the CMU text has a page listing schools that have adopted their CS:APP text in (presumably) their architecture or systems courses. It's at http://csapp.cs.cmu.edu/public/adoptions.html.

Here is a brief summary of my findings about a CS351-Systems Programming course for each school is below:

- MIT --- as usual, they do things their own way. They don't have an exact analog of our systems course, but their 6.033 CSE course covers client-server system design, virtual memory, threads, security, encryption, and is designed around two large (truly massive) projects. The course is a junior-level course and seems to have a large initial focus on operating system topics.
- Stanford combines their operating systems and systems programming courses (CS 140's title is "Operating Systems and Systems Programming") --- the systems programming projects come with a strong OS bent --- threading, virtual memory, file systems, etc. It doesn't seem there's much in the way of higher level systems programming.
- Berkeley follows the trend of putting together systems programming and operating systems --- again, mostly lower level systems assignments.
- CMU has probably the best analog of the course that we should be teaching --- a lower level undergrad systems course that precedes operating systems in the curriculum. This is the course that's based on the text we discussed last week. Good amount of C-based systems programming, albeit with a x86 assembly bent. They also have an interesting operating systems practicum course that I'm attaching the projects/reading webpages for.
- UIUC, as far as I can tell, does not have a "pure" systems programming course in their undergrad curriculum --- while they do have an OS course, it does not have enough of a systems programming component to list it alongside those from other schools. They have a higher-level embedded systems programming course, and a number of systems programming bits spread across their higher level course offerings (e.g. network programming, computer system organization, etc.)
-
- Cal Tech – CS24 - Introduction to Computing Systems. 9 units (3-3-3); third term. Prerequisites: CS 2; and CS 21 or CS/EE/Ma 129 a. Basic introduction to computer systems, including hardware-software interface, computer architecture, and operating systems. Course emphasizes computer system abstractions and the hardware and software techniques necessary to support them, including virtualization (e.g., memory, processing, communication), dynamic resource management, and common-case optimization, isolation, and naming. Instructor: Chandy. CS 134 a. Computing Systems. 9 units (3-3-3); first term. Prerequisite: CS 21 and CS 24 or instructor’s permission. Operating systems, monolithic and microkernels, virtual machines. Naming, memory management, segmentation, paging, and virtual memory. Filesystems and I/O. Threads, processes,

scheduling, locks, semaphores, and mutual exclusion. Security policies, access-control, capabilities, and language-based security. Instructor: Hickey.

- Northwestern - 725-213-0 Introduction to Computer Systems
  This course has four purposes: (1) to learn about the hierarchy of abstractions and implementations that comprise a modern computer system; (2) to demystify the machine and the tools that we use to program it; (3) to come up to speed on systems programming in C in the Unix environment; (4) to prepare students for upper-level systems courses. Prerequisites: 725-211 or 727-230 Course Director: Peter Dinda
  725-343 Operating Systems
  Fundamental overview of operating systems. First Quarter: Operating system structures, processes, process synchronization, deadlocks, cpu scheduling, memory management, file systems, secondary storage management. Requires substantial programming projects. Prerequisites: 725-311 and either 725-213 or ECE 205 and ECE 231
  Prerequisite for: Many 395 offerings Course Director: Fabian Bustamante
- Georgia Tech - CS 2110 - Computer Organiz&Program -Computer Organization and Programming An introduction to basic computer hardware, machine language, assembly language, and C programming.
  CS 2200 - Systems and Networks - Computer Systems and Networks A broad exposure to computer system structure and networking including software abstractions in operating systems for orchestrating the usage of the computing resources. **08-JUL-1998
- UIC - 266 Computer Architecture I: Logic and Computer Structures 4 Hours. Credit is not given for CS 266 if the student has credit in EECS 265 or EECS 365 or EECS 366 or ECE 265 or ECE 267 or ECE 366. Architecture from gate level up. Combinational and sequential logic. Logical minimization. Integer number systems, arithmetic. Datapath design. Finite state machines. Register-based architecture. Memory technologies. Prerequisite: CS 102.
  366 Computer Architecture II: Hardware-Software Interface. 4 Hours. Credit is not given for CS 366 if the student has credit in EECS 265 or EECS 365 or EECS 366 or ECE 265 or ECE 267 or ECE 366. A continuation of CS 266. Control-unit and I/O design; assembly language and machine programming; hardware control and I/O; memory hierarchy and caching. Prerequisite: CS 266

We should also look at the IEEE/ACM Model CS Curriculum
http://www.computer.org/education/cc2001/
http://www.computer.org/education/cc2001/final/index.htm