TEACHING STATEMENT

I have taught 40+ semester-long courses in Boston University's Computer Science Department. Some courses require additional lab time of 2-3 hours per week, although this time is often managed by teaching assistants rather than the primary instructor.

Example courses I have taught include the following:

- **BU CS552 – Operating Systems.** This is predominantly an introductory course in OS concepts, including process and thread management, scheduling, synchronization, memory management, filesystems and I/O device management. I use the source code of my own operating system, called "Quest", to demonstrate key concepts. The course is heavily focused on programming projects. While some assignments optionally leverage my OS, most involve development within the Linux kernel. Example Linux-based assignments include a UNIX-style RAMDISK filesystem, and a streaming video server that requires users to develop their own kernel-level synchronization primitives. Smaller projects include the development of new scheduling policies such as earliest deadline first (EDF), fair queueing (FQ), or those based on my own research (e.g., Dynamic Window Constrained Scheduling, or Virtual CPU Scheduling). Other systems projects taught in CS552 require students to use virtualization tools such as BOCHS, QEMU, VMware Workstation, or Oracle VirtualBox to develop standalone operating systems, targeted primarily at the x86 architecture. Here, students are expected to write or use bootloaders such as GRUB to probe for the availability of physical memory and I/O devices, so that these can be managed by their system.
- **BU CS410 – Advanced Software Systems.** This is a senior undergraduate course that focuses primarily on (UNIX) systems programming. File I/O, libraries, linkers, loaders, system calls, processes, thread management, sockets, IPC, signals, and shell programming are all covered in this class. As with CS552, the course is focused on a "hands on" approach to learning. Example assignments include: (1) a recursive filesystem tree walking algorithm to search for regular expression patterns in files, similar to how a recursive *grep* routine might work, (2) development of shared and static libraries to implement binary utilities for examining various object files (e.g., ELF and COFF), (3) an operating system shell environment, (4) the development of a portable thread library, and (5) the implementation of a simple debugger using process tracing techniques.
- **BU CS210 – Computer Systems.** This second year undergraduate course covers the structure and organization of computer systems, the design and implementation of abstractions that enable humans to use computers efficiently, the basics of assembly programming, how to translate between assembly and machine language, how it is possible to build a machine that executes instructions, and the various interfaces between processors, memory, peripherals, and operating system software. Example projects include the development of an ALU, a simplified MIPS-style processor (using Altera design tools), assembly programming assignments such as the Eight Queens problem, and C-style *malloc* and *free* heap memory management routines.
- **BU CS350 – Fundamentals of Computing Systems.** This course is a required sophomore/junior-level undergraduate Computer Science course. Topics covered in the course include performance analysis and evaluation, scheduling, resource management, concurrency and synchronization. While this course has several programming assignments, it is more mathematically focused than the above courses, covering topics in probability and statistics, queueing theory, and scheduling algorithms and their analysis.
- **BU CS212 – Physical Computing.** This was a pilot undergraduate course, to introduce students to programming. The course centers around the use of Arduinos and other types of embedded programming platforms, such as the ARM-based Beagleboard. Using sensors and actuators, students tackle a series of programming assignments that culminate in team projects using small mobile robots to solve problems such as finding paths through mazes.
- **BU CS553 – Advanced Operating Systems.** This is a graduate-level course, taking the form of a seminar. Students are required to present research papers, and develop independent study projects related to OSes.
- **BU CS697 – Graduation Initiation Seminar.** This is a half-time (post-)graduate-level course intended to help PhD students avoid many of the pitfalls of academic life. The course provides guidance to students on topics such as finding a research advisor, writing and reviewing papers, presenting research work, avoiding academic misconduct, plagiarism and falsifying results. In the US, funding agencies such as the National Science Foundation and National Institutes of Health now require students to take a course that focuses on "responsible conduct in research" (RCR). I have successfully petitioned for CS697 to be an approved RCR course, specifically tailored to computer science PhD students.

Teaching Philosophy

My teaching has focused on a "learning by doing" principle. Most of my courses involve a significant programming component. Driven by my research interests in systems, I tend to focus my courses at the interface between hardware and software. Low-level languages such as C/C++ form a natural match to many of the programming assignments in systems research, as well as assembly programming.

In lectures, I tend to use a combination of electronic slides, board-written notes and interactive examples that I perform live in the classroom. For example, I might walk through the source code to show how to boot a simple operating system that displays available machine memory on the screen. This could be followed by using a virtual machine to run the compiled code.

Courses I Would Like to Teach

I would like to teach courses that leverage my own operating system, called Quest. One of my goals is to perhaps write a book on *Real-Time Operating Systems Design and Implementation*, or similar topic. Most operating system books focus on general purpose OSes, largely structured around the UNIX model. With the advent of multicore processors, embedded and real-time systems are now becoming far more advanced than those targeted at traditional microcontrollers (many of which lack MMUs and MPUs). A course that covers embedded and real-time concepts, with a focus on system design and theory would be very appealing. This could also involve a strong applied focus using Raspberry Pis, the new Intel Galileo Arduinos, or similar platforms. For example, VIA has a pico-ITX single board computer featuring a quad-core x86 with hardware virtualization. Intel has similarly followed with an x86 VT-x Edison board for Arduino-compatible computing. These would make for an exciting platform to teach hardware-software co-design, and embedded systems topics.

Courses covering multi- and many-core processing, and machine virtualization would also be of interest. Similarly, student project courses would be a great way to excite future generations interested in programming and building working systems for tomorrow's world.

Further Information

Details about my teaching experiences can be found on my website:

- http://www.cs.bu.edu/fac/richwest/

In particular, my courses are listed here:

- http://www.cs.bu.edu/fac/richwest/courses.html