Reflective statement about
CS 485 — Systems Programming

1 Overview

CS 485 is intended as a general introduction to computer systems. The
goal is to provide an overall view of the components that make up a com-
puter system and the ways in which programs interact with those com-
ponents. The course touches on a broad range of topics, including hard-
ware, assembly language, compilers, operating systems, networking, and
graphical user interfaces. It presents topics from a programmer's perspec-
tive, hoping to develop a complete understanding of what happens when
a program is executed. The course also introduces useful programming
and debugging tools that can assist you in creating robust and efficient
code. The course provides foundational programming skills required by
many upper level classes.

More specifically, the course touches on hardware instruction sets, data
representations, assembly and machine-level program representations, trans-
lation from C to assembly language, linking and loading, multiprogram-
ning/multitasking, process/thread management, memory management,
virtual memory, file I/O, network programming, and graphical user inter-
faces.

The course is intended to be renumbered as CS270, offered to second-
year students. The demographics of the students were 13% sophomore,
38% junior, 45% senior, and 3% graduate students.
2 Text


This book is a very good reference. I did not prepare my lectures directly from the book, but I did base my lectures on slides prepared by Dr. Griffioen, who has taught this course at UK for several years.

I supplemented the text with my own lecture notes, generally within a few days after each lecture.

I placed all class documentation, including syllabus, lecture notes, hints, and assignments, on the class website: [courses/CS485.html](courses/CS485.html).

3 Lectures

I taught in CB 208, using the very nice, wide, whiteboard, which is my preferred style. This room also has a projector and screen. I used the projection facility only rarely: to display the lab assignments during lab sessions, and to show hands-on examples of using software tools. develop examples more fully, and I cannot write faster than a student can copy. There are very few situations in which slides present material more cogently than the board. However, my use of the board is not always well-organized, because I don’t plan ahead which part of the board to use for what part of the lecture.

The room was quite full, with 80 students initially registered, almost the capacity of the room. Some students prefer to sit at the far sides of the room, which must have reduced the visibility of the board, but I heard no complaints. I encourage students to ask questions, and some students regularly did so.
4 Computer facilities

We provided each student with a personal virtual machine running Ubuntu Linux, version 14.04, on x86_64 hardware. Students were able to connect to their machine either by a web browser or by a secure-shell connection. I provided detailed instructions on connecting to the virtual machine at www.cs portal.edu/vmPrimer.html. The virtual machines ran on facilities provided by the CS Laboratory for Advanced Networking (LAN).

Students were expected (in some cases, required) to use the virtual machines to complete their assignments. Having virtual machines made it harder (but not impossible) for students to copy each other’s code, and several assignments were randomized so each student had a different version, also reducing the possibility of copying. Virtual machines also gave the students an opportunity to act as system administrator for their own “computer”, although it is unclear to what extent students took advantage of this possibility.

Students submitted their assignments via the cportal at www.cs portal.edu/csportal. This facility, built by/CS staff, timestamps submissions and lets the grader provide grades and feedback, and lets the instructor download a CSV (comma-separated values) database of all grades.

5 Assignments

There were five regular assignments:

1. An introduction to systems programming in C.

2. A project to design input that a program, available only in machine language, would accept. This project was designed by the authors of the textbook. It was randomized and required the virtual machine.

3. A project to design input that would cause a program with a buffer-overflow bug to perform certain actions. This project was also designed by the authors of the textbook. It was randomized and required the virtual machine.

4. A simple shell program, to be programmed in C or C++.
5. A simple server-client program pair. The server stores and retrieves files.

These assignments were associated with in-class laboratory exercises. For these exercises, students were required to bring a laptop or equivalent to class. The IT Networking staff were helpful in assuring that there was adequate wireless support in the classroom; they provided guidelines that I summarized in an online guide, at [link to guide]. Lowell (LAN staff) and/or Paul (CS staff, grader) was present during each in-class laboratory to help answer questions and fix problems. The laboratory exercises were mostly on Friday the first few weeks of class to make sure students were ready for the assignments.

1. Getting used to the virtual machine and using it to download, compile and profile a simple program.
2. Learning to use the gdb program debugger.
3. Learning advanced features of gdb and a graphical front end, ddd, and learning to debug programs for which source code is unavailable.
4. Learning about buffer overflows and exploiting an overflow.
5. Learning about Unix process management, primarily the fork() and exec() system calls.

graded the laboratory exercises on a simple integer scale from 0 to 2. He graded assignments on a 100-point scale, first surveying the submissions, then building a grading rubric, communicating with me to agree on the rubric, and then grading the assignments. My goal was that all assignments and laboratory exercises that were submitted on time should be graded and returned within one week. I think we came very close to meeting this goal.

6 Tests

I gave both a midterm and a final, each covering one half of the semester. I intended both tests to be easy. The midterm grades ranged from 34% to
96%, with an average of 67%. The final grades ranged from 16% to 98%, with an average of 64%. The questions mostly tested basic understanding of the concepts of the course.

7 Teaching evaluation

Sophomore were highly over-represented in the teaching evaluations (35%) and seniors under-represented (26%); only 23 of the 60 students finishing the course submitted evaluations, so they should be taken as inconclusive.

The students who responded are generally happy with the course. They rated its overall value (question 20) at 3.63 (out of 4; the University average is 3.24), and they rated my teaching (question 21) at 3.58 (University average: 3.33). Over 60% of the respondents rated both the overall value and my teaching as excellent.

The comments included both positive notes (“Dr. is a wonderful teacher. Best I have had in UK’s CS department. Always available and stimulating, very good with keeping your engaged and listening.” “Loved this course. It was very stimulating and challenging, but only in good ways.”) and negative (“The course did not cover material for Programs 1, 4, and 5.” “This class was tough. Seemed to expect a better understanding of C than previously taught. The extended deadlines for projects may have hurt more because I knew I could put it off some.”).

With regard to the course-specific outcomes:

1. I have the ability to design, implement and evaluate a computer-based system, process, component, or program to meet desired needs: 3.46

2. I have the ability to apply design and development principles in the construction of software systems of varying complexity: 3.42

3. I understand entire software systems including software tools (compilers, linkers/loaders, and debuggers), operating systems, and networks: 3.50

4. I improved my ability to debug programs: 3.58
5. I understand common bugs and the potential for attacks (e.g. buffer overflow attacks): 3.48

6. I understand the various components of an operating system: 3.50

7. I understand memory management: 3.42

8. I understand process management: 3.29

9. I understand file systems: 3.29

10. I understand and apply OS interfaces and system calls for application and shell development: 3.38

11. I understand client/server network programming: 3.29

These evaluations are good, but I would like to see slightly higher values. We missed several class sessions due to snow days, so I was unable to cover all the class material, particularly with regard to process control and file systems.

In the future, I should devote more class time preparing students for the programming assignments, particularly the last one.

8 ABET data

8.1 ABET Item B

I assess ABET item B (An ability to analyze a problem, and identify and define the computing requirements appropriate to its solution) by project 2, which by its nature must be solved independently by each student. This project requires students to use the gdb debugger and the objdump program to discover how a program, available to the students only in object form, works, so that they can design input that prevents the program from reaching a particular state (called “exploding the bomb”). This program is randomized, so each student had a different problem to solve. Full credit required solving six phases; there was also a hidden extra-credit phase. The scores include a 3% daily penalty for late submissions.

Here is a breakdown of the grades:

exceeds expectations Grades 100 or higher: 16/60: 27%
meets expectations  Grades 90–99: 27/60: 45%

almost meets expectations but some elements missing  Grades 70–89: 9/60: 15%

does not meet expectations  Grades 40–69: 4/60: 7%

substantially lacking  Grades 0–39: 4/60: 7%

The number of students who met or exceeded expectations is 43 (72%).
Suggested improvements: Keep using this assignment. It is quite valuable.

8.2 ABET Item I

I assess ABET item I (An ability to use the current techniques, skills, and tools necessary for computing practice) by project 3, which by its nature must be solved independently by each student. This project requires students to design binary input to a program that has a buffer-overflow bug to cause that program do run specific code. This program is randomized, so each student had a different problem to solve. Full credit required solving four increasingly difficulty parts; there was also an extra-credit part. The scores include a 3% daily penalty for late submissions. Here is a breakdown of the grades:

exceeds expectations  Grades 100 or higher: 33/60: 55%

meets expectations  Grades 90–99: 17/60: 28%

almost meets expectations but some elements missing  Grades 70–89: 5/60: 8%

does not meet expectations  Grades 40–69: 1/60: 2%

substantially lacking  Grades 0–39: 4/60: 7%

The number of students who met or exceeded expectations is 50 (83%).
Suggested improvements: Keep using this assignment. It is quite valuable.
9 Summary

I think the course was very useful to students, showing a range of important topics in systems programming. I think the 200 level is appropriate, although some sophomores struggled with programs because they were not proficient with C or C++. 