A Core Course on Systems for a Liberal Arts Curriculum

A. A. Lopez
Computer Science
Division of Science and Mathematics
University of Minnesota, Morris
600 East 4th Street
Morris, MN 56267
alopez@morris.umn.edu

Abstract

This paper is about a one semester core course in systems taught at a liberal arts college. The course covers fundamental concepts of computer organization, assembly language, operating systems and networking. This course has been taught for seven years and we feel that it meets the basic recommendations of the ACM/IEEE Computing Curricula 2001 Project [1]. Topics to be taught in the course had to be carefully selected so as to provide an integrated learning experience compatible with the breadth liberal arts graduation requirements.

A laboratory component has added immensely to the success of the course. The course depends on simulators and emulators to illustrate many of the key concepts taught. About one third of the labs are done in teams of two or three students. The goal of several of the labs is for the students to read documentation on their own and apply that information to the lab exercises.

Introduction

This paper is about a core course in systems taught at a liberal arts college. This course has been taught for seven years and we feel that it meets the recommendations of CC2001 [1]. Due to the typical liberal arts graduation requirements, the course needed to be very innovative on how topics were combined into one instructional unit.

The following sections include a discussion of computer science programs at liberal arts colleges, description of UMM's computer science program, a description of our innovative, integrated core systems course, and how this course fits into the computer science major. A course outline and descriptions of lab assignments are included in the Appendices.

Computer Science programs at liberal arts colleges

Bachelor of Arts degrees in a discipline, typically require that students take about two thirds of the credits required for graduation in disciplines other than their major discipline. This implies that students majoring in computer science at a typical liberal arts college requiring 120 semester credit-hours for graduation will only be allowed to count about 40 credits in computer science toward graduation. This presents a major challenge for a new discipline such as computer science, where knowledge has not been completely synthesized. On the other hand, students are continuously demanding that we offer more and newer courses dealing with the latest topics in the field.

Students frequently feel that they must know all the latest techniques, otherwise they will not be employable. This situation is not unique to computer science, for example on our campus the music majors frequently accumulate 60 credits in music before they graduate. The challenge for computer science, as it is in music, is how to divide or subdivide our discipline so the students get a good sense of the breadth of the field.

Historical Background on UMM's Computer Science Program

The Bachelor of Arts degree in computer science at the University of Minnesota, Morris was approved in 1984. At that time, the institution operated in the quarter system and that allowed us to offer three sets of courses per academic year. In 1999, the University of Minnesota decided to change to a semester system on all of its campuses. The campuses were at liberty to choose their instruction module and our campus chose to adopt a 4-credit module (despite the objection of the science faculty that preferred a 3-credit module). The 4-credit module was chosen to minimize the number of preparations a faculty member would have. With a nominal 20-semester credit teaching load per year, that implied five preps per year.

Once the 4-credit module decision was made, the computer science discipline set out to develop a curriculum that would include the breadth that we thought our students deserved while living within the 4-credit module requirement. We came up with a proposal that included three foundations courses- CS/1, CS/2 and Data Structures; three core courses in Systems, Theory and Programming; a two credit seminar course that encouraged our students to develop their communication skills and a series of elective courses with their corresponding core courses as a prerequisite. The foundation, core and several of the elective courses were proposed as 4-credit courses. We soon realized that if we were to strictly stick with the 4-credit module for our electives, we would be offering many fewer electives than we had been offering under the quarter system. That brought us to consider offering half-courses as elective. These courses would award two semester credits and would deal with more 'researchy' topics that would interest the students and they would be more involved in the learning process.

We have more or less followed this paradigm for the past seven years. After a couple of offerings of the Software Design and Development core course, we realized that a lab was essential to that course and it was added. In 2002 when more reasonable lab facilities became available to our discipline, we added lab components to our Data Structure course and to all the core courses, and increased the number of credits for each of these courses to five credits (so much for the 4-credit module). Along the way we also realized that the seminar course would be more meaningful to our students if it was divided into two one-credit courses. The first course forces the students to talk and write about ethical issues in computing. The second seminar course is more of a capstone experience where the students read, write and talk about recent research papers that they have studied. For the latter course, the student chooses a faculty advisor, the student and faculty advisor choose a topic to be researched, the student does a literature search, writes a paper about her/his findings and presents the results of their research at a mini-conference held at the end of the semester. Proceedings of the conference are prepared and distributed and students present their results orally to faculty, fellow students and anyone else on campus that cares to attend.

The major drawback of the current curriculum is that it is almost as much work to teach a 2-credit elective course as it is to teach a 4-credit elective course. The anticipated involvement of the students in the teaching of these 2-credit has not developed to the faculty expectations. The two credit courses are hard on both the students and the faculty, so we are looking for ways to reduce the number of 2-credit elective courses that we offer every semester.

Systems graduation requirements

All students must currently take the five-credit Systems core course called Models of Computing Systems. In addition, students must take either two or four credits of elective courses in the Systems area before graduation. The students must take 10 credits of electives beyond the core courses and these 10 credits must be distributed among the three core areas in a 2-4-4 fashion. The students get to choose which area they will take the 2-credit elective and then they must take 4 credits in each of the other two areas. In reality, many of our graduates take more than the minimum number of electives.

Systems core course

The Models of Computing Systems course has now been taught seven times. Six of those times by the author. The course has been quite challenging at times for both students and faculty. The course attempts to give the students some principles of computer organization, a little understanding of machine/assembly language, principles of operating systems and an introduction to networking. A serious limitation for this course is that there is no one or two textbooks that cover the water front. Recently, I have resorted to requiring an Operating System textbook [2] and putting other materials on reserve in the library to deal with the computer organization, assembly language and networking.

After experimenting with several systems textbooks for the computer organization and assembly language, I have found that Warford's [3] book, now in its third edition, does an excellent job of treating these topics in a concise and clear way. I have found his Pep emulators to be an ideal way to teach simple assembly language concepts without getting the students overwhelmed with all the details of a real assembly language. This book is very well written and it is relatively easy to skip over sections of the book without losing the students. Despite the fact that I do not require the students to buy this book, the word has gotten around campus that this is a valuable book and I have noticed that a couple of the students bought their own copy of it this year.

The biggest portion of the course is to cover the principles of operating systems. Again, through trial and error, I have found that for our students, a fairly traditional and concise textbook such as Silberschatz [2] works well.

Typically, we will cover about a third of the book including the chapters on processes, inter-process communication, threads, scheduling, synchronization, deadlocks, memory management, virtual memory and security.

The last segment of the course deals with networking principles. Initially I felt that the students would need a reference in order to grasp the principles of networking. I have recommended a couple of simple networking books to the students in the past, but they do not seem to need them. This is my area of research, so this material comes relatively easy to me and the students seem to take good notes in class and master all of the important principles.

In Appendix A, you will find a typical class schedule for the semester. The numbers on the left, on each column, refer to the class period during the semester. The numbers on the right are the pages that the students should read from Silberschatz for that day.

Appendix B outlines some of the lab assignments given during the semester. We are fortunate to have three sets of laboratory hardware that we can use in support of these lab exercises. We have access to a public lab with Windows computers that we use for the Pep assembly language exercises. The Pep emulator is also available to run under Linux, but that version does not seem as well developed. Labs 5 and 6 are designed to familiarize the students with Linux and many of the commands relating to process and resource management. The students are given free rein to explore any of these commands using the man pages. Lab 7 uses a simulator to study various scheduling algorithms developed by Professor Robbins at the University of Texas at San Antonio. While the students are continuously finding bugs in this software, nevertheless it gives them experience with downloading and installing software in a Linux system, it forces them to think critically about the simulations that they run, and the software has many forms of output built into the emulator that saves time in plotting the results, etc. Lab 8 uses simulators from Professor Robbins to study various process synchronization techniques for the Dining Philosophers problem. In Lab 9, the students get to do some Unix shell scripting. Labs 10 through 14 are done in a dedicated network of computers where students can experiment with the configuring of Unix systems without affecting the campus network. These exercises are done in teams of two or three students. We have attempted to do these exercises using older computers that were no longer viable for other work, with VMtware, and most recently, using the Knoppix system that allows each workstation to be booted from a CD and configured without having to save things to disk. Through these exercises the students get to install Unix, configure the network connection for their workstation, configure Domain Name Service (DNS), e-mail and Web server and they also are able to explore other Unix commands that require superuser privileges, such as deleting a process, shutting down a system, etc.

Evaluation of Systems Core Course and Elective Systems Courses

Prior to the establishment of the Models of Computing course, our students graduated with a variety of inconsistent systems background. This course has helped provide a more uniform systems background to our students. This is particularly noticeable when the students take some of the elective systems courses.

Since adding the laboratory component, more of the lectures have had to be geared toward preparing the students for that week's lab exercises. This has made it difficult to cover some basic topics from computer organization for the past two years.

Another challenge is the lack of a textbook that would cover the variety of topics that we cover in this course. The author was hopeful that with the advent of electronic publishing offered by some vendors, this problem could be addressed, but the number of publishers participating in such projects is still limited and no such solution has been achieved yet. Even if we were able to obtain an electronic textbook through such a methodology, challenges would remain since different authors have different writing styles and they will assume different backgrounds on the part of the students.

When the course was offered without a lab, the students had a very hard time conceptualizing the concepts being described during the lecture. While these lab exercises are not perfect, they give the students a better sense of how computers work and some of the issues that they will be confronted with upon graduation.

After taking this course, the students are eligible to take a variety of 2 and 4 credit elective courses in systems. These courses include Computer Networks, Database Systems, Distributed Systems, Parallel Systems, TCP/IP Networks, Network Security, Robotics, Wireless Data Networks and Computer Forensics. The author has also taught a variety of these courses before and, after the Models of Computing course was introduced, there is no doubt that the students are now better prepared to enter these elective courses.

Conclusion

This course has helped us established some uniformity to the systems background of our graduates. The graduates now have a more balanced and better sense for topics such as assembly language, computer organization, operating systems, networking and how to configure a Unix system.

Acknowledgement

I would like to acknowledge the contribution of the UMM computer science faculty and student representatives who since 1998 have worked very hard on adapting our quarter-based computer science curriculum to our semester-based curriculum. I also like to acknowledge the contribution of my colleague, Professor Dian Lopez, who assisted me in the editing of this paper.

References

1. Computing curricula 2001, Journal on Educational Resources in Computing (JERIC), Volume 1, Issue 3, Article 1, 240 pages, ACM, Fall 2001
2. A. Silberschatz, P. Galvin and G. Gagne, Operating System Concepts, 7th Ed, John Wiley & Sons, 2005
3. J. Warford, Computer Systems, 3rd Ed, Jones and Bartlett Publisher, 2005

Appendix A

CSci 3401 Course Outline Spring 2006

The purpose of this course is to introduce the students to computer systems. This course attempts to integrate topics from four areas: Computer Organization, System Programming, Operating Systems and Networks. Each of these topics can be a course in itself. We are hoping to capture the salient characteristics of each of these areas and allow you to learn specifics about each area in our 44xx courses. To assist us in these tasks, I have selected a textbook Operating Systems Concepts by Silberschatz, Galvin and Gagne, Seventh Ed, Wiley, 2005 and several handouts. A brief course outline follows.

1	Machine instructions, Assembly Lang Prog	
2	Assembly Language Programming	
3	Assembly Language-addressing mechanics	
4	Assembly Language-Stack operations	
5	Wrap-up of Assembly Language	
6	Combinational circuits	
7	Sequential circuits	
8	Intro to Operating Systems	3-38
9	Computer-System Structures	39-54
10	OS Structures	55-78
11	Processes	81-95
12	Interprocess communication	96-124
13	Review	
14	***First Hour Exam	
15	Threads	127-151
16	CPU Scheduling	153-172
17	Thread Scheduling, Alg. Evaluation	172-189
18	Process Synchronization	191-200
19	Semaphores	202-209
20	Guest lecture	
	SPRING BREAK	
21	Critical regions, monitors	209-222
22	Wrap-up Synchronization	222-241
23	Deadlocks, models, handling, prevention	245-256
24	Deadlock avoidance, detection	256-265
25	Recovery from Deadlock	266-271
26	Memory Management	275-288
27	Paging, Segmentation, Seg w Paging	288-312
28	Virtual memory	315-363
29	Security, threats, intrusion detection	559-600
30	Computer Networks	
31	Terminology, media, review	
32	***Second Hour exam	
33	Topologies, WAN, LAN	
34	ISO Model	
35	Physical Layer	
36	DLL protocols	
37	Network Layer	
38	Transport Layer	
39	Application Layer	
40	Network devices-routers, bridges,	
41	hubs, switches, NIC, MAU, gateways	
42	Networking protocols	
43	Routing protocols	
44	Faculty evaluation, Review	

Appendix B

Lab 1 Introduction to Assembly Language and the Pep emulator. Download the emulator, run a program from the printed material available.

Lab 2 Simple assembly language program created by the student to satisfy given requirements using direct and immediate addressing mechanisms, input and output functions and giving appropriate documentation.

Lab 3 Assembly language program that demonstrate the use of index addressing and array data structures.

Lab 4 Assembly language program that demonstrate the use of a stack in making recursive calls to a function (e.g computing Fibonnacci numbers)

Lab 5 and 6 Unix based exercises for the students to learn about the 'man' capabilities, investigate topics having to do with processes, resources, etc. Trying to get them to be independent thinkers and not run to the instructor and/or web for all their answers.

Lab 7 Study Scheduling algorithms using simulators developed by Professor Robbins at the University of Texas, San Antonio under an NSF grant.

Lab 8 Study Process Synchronization using simulators developed by Professor Robbins at the University of Texas, San Antonio under an NSF grant.

Lab 9 Unix shell scripting exercise. Build confidence in their ability to do shell scripting.

Lab 10 and 11 Install the Knoppix system on a workstation on a dedicated network, configure the network adapter with IP address, etc. and investigate unix commands that require privileges.

Lab 12 through 14 Using the Knoppix system on a dedicate network configure the workstations such that DNS, e-mail and web services are available on the network. Create servers for the above mentioned services. Test that all services work correctly. Prepare lab reports describing the successes and failures encountered along the way. Done in teams of two to three students.