Course and Objectives

This is a required 4 credit course for Computer Science graduate students going for a Masters or a Ph.D, with most of its content shared with a 3 credit elective for undergraduates. Being an undergraduate major elective, CSI400, with its prerequisite of CSI333, is intended for those prepared and who are willing and able to invest a high level of time, perseverance and commitment to master an intricate subject. If you are not actually a computer science major, you will still need the preparation and motivation expected from CS majors to take this course successfully.

Top Level Objectives: Demonstrated Abilities

Broadly speaking, this course is intended to enable you to do technical work with the internals of complex software systems, especially those with concurrency, multiple resources to be managed, many layers and interfaces, and needs to provide proper performance (i.e., speed) and security in unpredictable environments. Technical work here ranges from advanced usage, maintenance and management (i.e., administration and application development) to analyzing, modifying, implementing and designing of operating systems.

In computer science, “‘knowing how’ matters more than ‘knowing what’” [GianCarlo Rota, http://www.math.tamu.edu/~cyan/Rota/mitless.html], as in exact sciences like core mathematics or physics, or in musical performance, foreign languages, and even sports. Demonstrable skills based on concepts are developed in a sequence in which new ones depend critically on the ability to use previously new ones. The demonstratable skill goals for this course:

1. Explain what a complex software/hardware system does during normal and abnormal operational scenarios, and “drill down” to details of those subsystems that were studied.
   Which one? Modern general purpose computer operating environment, emphasis on Linux/Unix.

2. Do intermediate (1st yr. grad., 4th sem. undergrad. semester) complexity C and some i386 assembler projects that utilize and/or simulate technologies listed in the course descriptions.

3. Solve problems: Analyze scenarios for each topic, including identifying and comparing consequences of alternative choices. Utilize numbers, graphs, formulas, sequence diagrams, etc., together with accurate logical reasoning.
4. Install Linux systems, build Linux kernels and kernel modules, find and annotate code in Linux sources for given OS functions, run, debug and test kernel modifications and modules. Report and interpret OS behavior using system tools and interfaces.

Coverage

This course will cover most of the 3rd edition of Tanenbaum’s *Modern Operating Systems* (MOS).

The emphasis will be on what is entailed by concurrency (multiple communicating processes running at the same time, actually or apparently) and on operating system organization or architecture (process/thread, synchronization, virtual memory, input/output device, file system and other resource sharing and protection functionality, components and their implementation).

“Communicating sequential processes” can mean (1) processes or threads which are runs of programs or are multiple control flows within the memory used by a program; OR (2) any concurrent, probably interfering activities: e.g. walking, breathing and chewing gum.

Abstractions (such as process, virtual memory and CPU, cache, and others) will be used to understand highly complex multicomponent systems piece by piece, where the interfaces are conceptually separated from the implementations.

The general principles from MOS will be supported by study of implementation details from Bovet and Cesati’s book on Linux and from Linux kernel code, written homework questions and problems, and by systems programming project experience guided by some other reference material, including Haviland’s book and others.

Other projects will give you experience with concurrent programming POSIX threads and with simulation of some operating system components. Both required and supplemental readings will be assigned from other material published on the Web.

**What’s the difference between 400 and 402?**

Both require major programming projects in C, under Unix. CSI400/500 will utilize a modern Gnu/Linux environment provided in the Computer Science Dept. “Cheese Box” Lab. There will be (mandatory) weekly 2 hour lab sessions to be scheduled beginning next week. The same lab machines are accessible via ssh Internet connections for programming homework and projects. (CSI402 uses ssh-only access to UAlbany’s ITS Solaris servers.) Having and administering your own Gnu/Linux system by repartitioning or adding a disk to your PC or laptop, or buying a cheap, extra one, is strongly recommended although not absolutely required.

Aside from 400 being an elective and not being required, 400 emphasizes the internal structure and function (architecture) of operating systems whereas 402 emphasizes application programming with knowledgable use of services provided by the operating system. This is not an “information technology” style course about using, comparing, installing or maintaining various “operating systems” (various Windows flavors, Unix, MacOS, etc.), although it will subsume some of this.

Learning the internal architecture plus doing the projects require a greater degree of abstract formulations and thinking plus a larger variety of new concepts than 402. On the other hand, the projects may be less tedious to code but require more time to run and analyze your project programs’ behavior or results. In short, 400 will be harder than 402. So, if you have not taken 402 yet, it may be wise to take 402 this (Spring) semester and defer 400 to another year.
Survival and Success

In order to succeed CSI400/500 you must take an active part in learning the course material. You must do the readings on time, attend the class consistently, and do your assignments with diligence and patience. This is especially important in a major/graduate course serving Ph.D. candidates because of the faster pace. If you do not do these things, you will almost certainly get a weak grade or even fail the course. However, a hint, personal discussion, an explanation in different words, and working in a group work wonders after you learned “what” from reading and listening, and then spent a limited amount of time patiently trying hard to practice “how” to apply it successfully. The key to success is to stay current. That is, stay up to date in the readings and assignments. If you attend the class, do the preparation and followup readings on time, and do the exercises when assigned, you should have relatively few problems. Do not wait until just before a test or assignment is due to try and cram several weeks worth of material in one night. It will not work there are just too many different dependent concepts to master together at once. If you do find that you are having trouble with this course seek help, SOON!. The longer you wait, the tougher it will be to get back on track.

Catalog Descriptions

Graduate: 4 cr. Intro. to OS. Topics incl. processes, concurrency, synchronization, deadlock, memory management, segmentation, paging, replacement policies, caching, interprocess communication, file systems, and protection. Heavy emphasis on abstractions, mechanisms, policies, and design. Prereq: Csi 333, 310, Mat 367, & 1 of Csi 400, 402, or 404.

Undergraduate: 3 cr. Historical overview; OS services; mass storage file organization; memory management in multiprogrammed systems; virtual memory; resource alloc.; concurrent processes; deadlock detection and prevention; security; the design of contemporary OS such as UNIX. Prereq: Csi 333.

Many terms in the above two descriptions are synonymous. We will cover everything above, with a bit more emphasis on “abstractions, mechanisms, policies, and design” than would be in a pure 400-level course.

Prerequisites

Albany CS courses through Programming at the Hardware/Software Interface (CSI 333) or equivalent [3 semesters of introductory computer science C and assembly language programming courses covering C types, statements, and functions (in C and assembly language) including arrays, linked lists, stacks, queues, trees, heaps (data structures), addresses, registers, machine and assembly language instructions, binary (and hexadecimal) number systems; Unix software building skills (separate compilation of modules with header files to define their interfaces, use of make, debugging, etc.) algorithms for sorting, searching and data structure traversal, recursion, function call linkage via the stack, etc.; skilled use of WWW.]

The graduate version’s prerequisites include a junior/senior systems course in software or hardware, and it has more credits. Therefore, the programming projects required from graduates will have higher complexity and depth. However, the probability prerequisite is not required for this
version of the course although it would foster better understanding of performance measurement and queuing theory.

All enrollees who do not have the systems prerequisites on their record will be de-registered except if they demonstrate equivalent preparation to the satisfaction of the instructor!

Required texts:

See the course “Resources” page for links to electronic text editions and many other pointers.


3. Readings from Linux Kernel code, handouts and Web references. These include the Single Unix/Posix specification, Pthreads references, and others.


Recommended Texts including those from prereq. courses


4. Your course C or some C++ textbooks you or others have found useful. (Deitel and Deitel’s books,

5. Materials from CSI333 on assembly language programming, the MIPS architecture, perhaps the SPIM simulator, etc. But, we will use the i386 (Pentium) instruction set architecture.

The books marked (*) above are recommended for you to keep for use in more advanced courses, research and professional work.
Grading

Midterm – Oct. 7, 2011 (Fri.) : In class 25%
Final – Dec. 16, 2011 (Fri.) : 10:30 AM to 12:30 PM 30%
Problem Solving Homework and Projects 25%
In-class: Quizzes, In-lab reports, some team activities 20%

Final letter grades will be based on cutoffs applied to your score computed as above and modified according to the following policy:

40% on each of at least three of the 6 or so programming projects, a 40% total score AND a final or average exam grade of at least 40% is the minimum for passing the undergraduate course (with a D-).

The midterm and final exams will be closed book and no other material besides one 8 1/2 by 11 inch sheet of paper with notes.

C. Policy on Cheating:

1. Cheating in an exam will result in an E grade for the course. Further, the students involved will be referred to the University Judicial System.

2. The code and any written answers for every individual assignment and exercise must be written by yourself. You are welcome to discuss the class material, the problems and ideas for solutions; but each person is expected to write the code and answers he or she submits independently, without copying.

Cheating in an individual assignment or lab exercise will result in a ZERO for that requirement for all the students involved. Students who cheat in two or more problem solving or programming assignments will receive an E grade for the course.

A report of the every cheating incident will also be made to the Office of Undergraduate Studies or the Graduate School Dean in accordance with the University regulations concerning “Penalties and Procedures for Violations of Academic Integrity” in the Undergraduate or Graduate Bulletins.

3. Violations of University IT Policy posted on http://www.albany.edu/policies/computer_usage/ are regarded as academic integrity violations, like cheating. So are Department specific policy violations and disregard of directions of the system administrator or instructor concerning lab security or integrity. (Think of what happens if you blow up the chem lab purposely.)

I Grades

A grade of I will only be given for genuine extenuating circumstances that are beyond your control after the midterm point. Both of the following conditions must be met:

1. Your work must be in good standing as of October 7, 2011, the day of midterm exam; that is, you must have an average of at least 50% in each score component. Further, your midterm grade must also be equivalent to at least a C. Therefore, if you miss the midterm or have performed poorly on assignments, you are not eligible for an I grade.
2. Written documentation must, upon request, be supplied about the extenuating circumstance either by you or the University administration (see the undergraduate or graduate dean’s office for help).

Disabilities

Accommodations will be made for clients of the Office of Disabled Student Services upon adequate prior notice and according to that office’s policies.

Students with genuine continuing hardship situations, or any disability related problems with computer usage should confer with the professor before October.

Team Activities

Some of the lab activities will be done as teams, with each team making a report of its findings to the whole class during a class period later.

Some topics in this course will be introduced using a team-based and active learning methodology. That means some classes will have one or more short quizzes or other activities to assess student preparation and achievement; and some classes will have graded exercises to be done collaboratively. During the second class, teams will be formed for the collaborative class exercises. Steps of team-based learning:

1. Preparation study homework: Textbook or other reading, survey or longer questions or problems.

2. Individual “Readiness Assessment Test” (IRAT) test based on preparation and introducing a new problem.

3. Team solution to the same RAT and a new problem done and reported in class.

4. Individual test in class or problem extension for homework.

This approach is known to be effective in engineering and computer science subjects. The instruction will include the following also.

- Individual and/or team quizzes and/or exercises in most every class, with assigned preparation.

- Written homework questions and problems.

- Lab exercises, some individual, some as teams. All students are required to attend a weekly 2 hour lab. They will generally be lead-in instruction and experience to systems and kernel programming and experimentation project homework, to be done individually, that will take substantial time beyond the lab. This lead-in work will be common to graduates and undergraduates. The programming projects will then have more requirements for graduate students. The labs will also introduce technology skills needed for OS research and advanced system administration.

- “Reverse engineering” of i386 assembly language codes and kernel C codes.
• Other projects with concurrent programming with POSIX threads, OS kernel programming, and simulation of some operating system components.

• Finding and using technical information from the Web.

Tentative Approximate Topic/Lecture Hour Breakdown

There will be a few Readiness Assessment Tests (RATs) given during the lectures – one to be associated with each unit. (The same RATs will be given to individuals and teams and you will get two separate grades. The team may then solve a followup problem and the exercise will end with an individual quiz, to be added to the individual and team grades.) The tentative topics for such team exercises are marked below with *. These structured team-based learning exercises will be in addition to short in-class quizzes and response solicitations.

The midterm and final will be standard individual exams. Some questions will be different for graduates and undergraduates.

Projects will be started as teams in the lab meetings, where the chief team objective is to have everyone understand the project goals and the infrastructure under which they are to be achieved. Individuals will be graded with a project preparation quiz and the team will receive an in-lab report score. The project will then be completed or extended individually, with an individual project score. The completion or extension requirements will be greater for the graduate students. The tentative topics for projects or major lab exercises are indicated below in boldface.

1. (3 hrs.) Programming.
   • C review and writing a unix shell, system calls, Assembly, Compiling
   • Execution stack, frames, registers
   • Functions, system calls and Traps
   • Assembly code analysis, personalized Linux kernel, Linux kernel modules,
   • OS background
     History, hardware, concepts, structure, abstractions
   • *Performance (i.e., speed, etc.)

2. (9 hrs.) Processes and threads.
   • Processes
     Fork, exec, multiprocess application, observing
   • Threads
   • *Analysis of Races
   • Interprocess communication
     pthreads application
   • *Scheduling
   • IPC analysis, problems, solutions
   • Deadlocks
3. (5 hrs.) Memory Management
   - Virtual memory
     Abstraction, reasons, segmentation, implementation, segmentation
   - *Page replacement algorithms and design issues
   - Implementation issues
   - Kernel modules and memory
   - **Paging simulator**

4. (4 hrs.) File Systems
   - Files and directories
   - Implementation
   - *Issues and examples
   - **New Linux filesystem**

5. (6 hrs.) Input/Output
   - Device features
   - OS support architecture and **drivers**
   - *Multimedia Requirements and Performance*