CS 214 Systems Programming

Objective

The aim of CS214 is to introduce the student to the process of writing low-level programs that interact directly with a computer's operating system and hardware, as well as to develop the student's ability to build large applications in a team environment. Upon completion of this course, the successful student should be able to design, write, test, and analyze moderately complicated programs using the C programming language and UNIX/Linux operating systems.

Prerequisite Knowledge

- Structured programming in a high-level language (such as Java).
- Standard data structures (lists, trees, graphs, hash tables).

Textbook

The textbooks do not cover all material discussed in class, and are not a substitute for attending lectures.

The following texts are available online, free of charge.

- *C Programming*
(Wikibook)
HTML or PDF.

- *The C Book* Mike Banahan, Declan Brady and Mark Doran
HTML or PDF.

For those of you not already familiar with the C language, here is a handy reference card in PDF format.

The textbooks do not cover all material discussed in class, and are not a substitute for attending lectures.

Topics Covered in CS214

The following list is organized by topic, not by chronological order of coverage in the course.

0. Programming in C
    - Basic syntax, standard I/O, data manipulation, flow control
    - Pointer manipulation, dynamic memory management
    - Error handling and debugging
1. Programming under UNIX
    - Advanced debugging
    - File I/O
    - Networking
    - Signal handling
    - Linux terminal commands
2. Large-scale development
    - Modules, headers, linking, makefiles
    - Version control
3. Concurrent programming
    - Process creation and communication
    - Multithreaded programming, synchronization

General Schedule of Topics

-=This schedule may change as needed=-

Intro to C, object files, linking and processes
C program structure, C functions, formatted I/O
Linux commands and services, Dynamic memory management
C preprocessor; Libraries (system-provided, static, dynamic) -- Bryant and O'Halloran pp 681-683
C preprocessor; macros and definitions, makefiles
Multi-file projects, Library archives
File I/O, Directory I/O, File descriptor semantic, i-nodes
Multiprogramming -- Bryant and O'Halloran pp 718-736 (fork, exec)
Threads -- Bryant and O'Halloran pp 947-989
Thread synchronization (mutex locks, semaphores)
Thread synchronization (condition variables), thread patterns
Thread patterns (producer-consumer, etc)
Signals and event-based programming -- Bryant and O'Halloran pp 736-758
Signals and processes
Signals and threads
Networking; sockets, file descriptor semantic
Networking; concurrency, threading and client/server model
Basic Distributed Systems

-=This schedule may change as needed=-

Projects

WARNING: This is a project-heavy course. This course should give you more than a passing knowledge of what writing working programs entails, but it will require significant effort on your part. The projects will be a major undertaking. Assess your commitment to this course realistically. If you don't have the time or the inclination to work hard on the projects, you would be better off not taking the course, or taking it later. You will have to learn how to build and debug reasonably-sized C programs and make them robust to outside errors. You will also have to describe how your program works in a written document.

Up to 2 students can work as a project group and you can change group members per project. Students are required to complete the projects by the scheduled deadlines. Failure to turn in the project by the deadline through Sakai will result in a zero for all team members.

There are many different operating systems and variants of C out there and we cannot test your program on all of them. So all program assignments must run on the local iLab Linux machines. We will be grading your assignments on those machines as well.

The programming part of the projects are typically graded on how well they operate and if they mirror the requirements. The written portion is graded on how well the TAs can figure out how your project is constructed only from the written description. Be sure to test all your code. You are not graded for design or time spent coding, but how well your code functions. You can write an incredible massive tower of code over a period of 300 hours, but if it breaks on a simple test you will have a low score.

Be sure to test your submissions in a blank, fresh directory. If your submissions do not decompress, compile or run, they do not work and will be given zeros.

Communication

The instructional staff want to help you, but due to the size of the course and complexity of some issues, you need to be careful in your communication. If you have a question about project details, first search Piazza and if your question is not there, post. We will answer it as soon as possible. Emailing questions individually causes the staff to spend much more time answering and repeating answers, giving you lower-quality responses and delaying responses. If you still have a coding issue or assignment question after posting on Piazza see your TA, preferably in person. If you must email always prepend "[CS214]" to your subject and keep it polite and precise. Include test data and your best idea about what is going on. If you have a question your TA can not answer or a course policy question, then see your Professor in person. Do not email the Professor except for scheduling, course administration or in emergency circumstances.

Working Together and Academic Honesty

Cheating on projects or exams will not be tolerated. We want to protect the fairness and integrity of the class, so we run code similarity detectors on the projects and scrutinize exams for copying. Both parties in the exchange are liable; e.g. if you give away solutions to friends, you're putting yourself at risk too. Keep your code safe and set your GitHub to be private, if you use it. Making your code available to the universe is tantamount to handing a copy to everyone in the class.

If you get caught, it's a nasty process that will result in *at least* a 0 on the assignment in question and potentially your expulsion from Rutgers. You're better off asking for help, or at worst, dropping the course and trying it again. The Office of Academic Integrity's policy can be found at: http://www.cs.rutgers.edu/policies/academicintegrity/. You now need to click explicitly on a link when first login to our computing facilities, use handin, etc., that says you acknowledge being aware of the policy (which you can read through the login screen).

Grading

Midterm: 20%
Final: 30%
Projects: 50%

Assessment grades are summed as weighted percentages at the end of the course. Grades are not calculated per assessment. In general, the final mean is a "C" and each standard deviation is one letter grade. TAs grade all programming assignments and some exam segments, but have all agreed to the same criteria for grading each programming assignment.

Academic Honesty and the Collaboration Line

Your classmates are a great resource, as is the Internet. You should talk to your classmates about your work and look at the Internet. Obviously you shouldn't ever copy code directly, give anyone code, or tell someone precisely why their code is broken. What is the line between honest discussion and dishonest collaboration? If you are discussing a problem with a classmate or looked something up online; first discard any notes or test code written, do something entirely different or mind-numbing for half an hour, then attmpt to reproduce the solution on your own. This is to assure that the work is original, your own, and that you fully understand how and why it works. It does you no good to get too much help on the projects if you do not learn the material you will not do well on the exams.