CS 214 Systems Programming

Objective

The aim of CS214 is to introduce the student to the process of writing lower-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 and error control coding
    - 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 error and broken input. You will also have to describe your program workings and parts of your development strategy in written documents.

Group work:
Due to the remote nature of this semester students are allowed to work in pairs. Both members of a group must submit their group affiliation by the announced sign-up deadline, or you are not working in a group and must each complete your own version of the projects individually.

Due dates:
Due to the remote nature of this semester there will be a six-hour 'grace period' after each project deadline during which a project may be submitted for 50% credit. If you want to be sure to receive full credit, it is your responsibility to submit before the deadline. Emergencies and unavoidable obligations must be reported to a SAS Dean as soon as possible for review.

Code Submission:
Code must be submitted on Sakai before the due date. External files are not considered. Time is as assessed on the Sakai host machine. Sakai is a web service; it is not magic. Do not expect that you can press the 'submit' button one second before the due date and have it work while 300 other people are doing the same thing. It is best to submit your code *at least* 24 hours before the due date.

Testing:
Your code must compile and run correctly on the iLab machines specified or it will receive a zero. If your code doesn't compile, it can not be evaluated and will receive a zero. If your code can not be decompressed, it can not be evaluated and will receive a zero. If your code requires compilation switches we do not use/allow or libraries or external files we do not use/allow, it can not be evaluated and will receive a zero. 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.

Grading:
The programming part of the projects are typically graded on how well they operate, how robust they are and how well they hold to the requirements. The written portion(s) is(are) graded on exactness and completeness. You are not graded for time spent, 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.

Communication

The instructional staff wants 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 synchronously. 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 synchronously. Do not email the Professor except for scheduling, course administration or in emergency circumstances.

Piazza is for posting precise, factual questions related to course content. It is not there for you to complain. If you feel the need to complain to the Internet, take it elsewhere. If Piazza is co-opted for use as a general social networking platform, it will be disabled.

Working Together and Academic Honesty

Cheating on coursework and assessments will not be tolerated. We want to protect the fairness and integrity of the class, so we run code similarity detectors on submitted work and scrutinize exams. If an issue arises, 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 (or other online code control service) to be private, if you use it. Making your code available to the universe is tantamount to taping it to a public wall.

If you get caught, it's a nasty process that will result in *at least* a 0 on the assessment 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

Homeworks: 25%
Final: 35%
Projects: 40%

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 proximate to a "C". In order to achieve pleasant outcomes, try to:

- be above the coursewide mean
- have >75% of the available points.

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? The more detailed the information you trade is, the less likely it is permissable. Keep it general on the order of strategy and maybe design. If you help a fellow student with a minor code issue, do not give them exact solutions but ask leading questions ("did you check all the cases?", "are you 100% sure this line always works?").

If you are discussing a problem, strategy or code with a classmate, or if you looked up a coding example or information online; first discard any notes or test code written, do something entirely different or mind-numbing for half an hour, then attempt 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 since if you do not learn the material you will not do well on the exams.