Systems I: Programming Abstractions

Course Philosophy: The goal of this course is to help students become facile with foundational concepts in programming, including experience with algorithmic problem solving and problem decomposition while developing good software engineering skills. The use of recursion is also introduced and emphasized as an important programming technique in a variety of contexts. Additionally, students learn about data abstraction through the implementation of various basic data structures (e.g., stacks, queues, linked lists, hash tables, and binary trees), analyzing their performance trade-offs, and understanding their use as generic containers.

Potential sample programming assignments and topical coverage
(The course will likely have 6-7 programming assignments. A few ideas for assignment possibilities are listed below.)

- Data manager to store information about students
  Concepts covered:
  - Programming methodology (engineering, modularity, documentation)
  - Algorithmic thinking and problem solving
  - Searching and sorting

- The game Boggle, including a computer player that finds words using recursive search
  Concepts covered:
  - Recursion
  - Libraries and interfaces

- Priority queues (implementing various underlying structures, discussing trade-offs)
  Concepts covered:
  - Data abstractions and generics/templates
  - Basic algorithm analysis (big-Oh) and comparison

- Editor buffer (implemented using doubly-linked chunk lists)
  Concepts covered:
  - Queues
  - Linked lists
  - Basic algorithm analysis (big-Oh) and comparison

- Huffman codes (construction of code trees, encoding, decoding)
  Concepts covered:
  - Binary trees
  - Procedural recursion

General Topics

- Programming methodology (engineering, modularity, documentation)
- Algorithmic thinking and problem solving
- Data abstractions
    - Stacks
    - Queues
    - Linked lists
    - Hash tables
    - Binary trees
    - Generics/templates
- Recursion
    - Procedural recursion
    - Recursive backtracking
- Searching and sorting
- Basic algorithm analysis (big-Oh) and comparison

Potential instructors: Julie Zelenski and Mehran Sahami

Notes: This course is a classic CS2 course covering algorithmic problem solving, fundamental data structures, and recursion. This course is our current CS106B/X course, which will continue in its current form. We have discussed the notion of potentially changing the language in this course from C++ to Java at some point in the future. For the time being, the current format of the class is quite workable for the Systems Core.

Systems II: Computer Organization and Systems

Course Philosophy: The goal of this course is to cover computer systems from the hardware level (e.g., registers, ALUs, memory, etc.) up the source code level. This "hands-on" tour includes concepts such as hardware organization, memory models, and fundamentals of compilation. Programming projects are used extensively to give students working knowledge of the various levels of computer system abstraction and how the transformation from one level to another is performed (e.g., compiler code generation). The course will also stress development in C as a vehicle for working with these concepts.

Potential sample assignments and topical coverage
(The course will likely have 4-5 assignments. A few ideas for assignment possibilities are listed below.)

- Implementing a memory manager
  Concepts covered:
  - Pointers
  - Low-level polymorphism and runtime type identification
  - Data representation
  - Heap management, garbage collection
  - Facility with C programming as part of topical coverage

- Memory hierarchy probing/optimization (Optimizing a memory system using simulator, exploring issues such as access locality, prefetching, latency at various levels of memory hierarchy, etc.)
  Concepts covered:
  - Machine architecture (registers, RAM, I/O)
  - Caching, pipelining
  - Basic assembly language

- Implementing an intermediate code generator
  Concepts covered:
  - Compilation
  - Function call mechanics and stack frames
  - Semantic analysis
  - Code generation
  - Facility with C programming as part of topical coverage

- Implementing a concurrent queuing simulation
  Concepts covered:
  - Basic concurrency usage
  - Threading
  - Synchronization, locks and semaphores

General Topics

- Machine architecture
  - Registers, ALU, CPU, RAM, I/O
  - Caching, pipelining
  - Basic assembly language
- Memory model
  - Pointers
  - Low-level polymorphism and runtime type identification
  - Data representation
  - Heap management, garbage collection
  - Facility with C programming as part of topical coverage
- Compilation
  - Function call mechanics and stack frames
  - Semantic analysis
  - Simple (intermediate) code generation
- Basic concurrency usage
  - Threading
  - Synchronization, locks and semaphores

Potential instructors: Jerry Cain and Nick Parlante

Notes: This course is a significantly modified version of our current CS107 course; material on programming paradigms (which had been reduced over time) is now eliminated from the course and replaced by more coverage of compilers and machine organization. An adjunct course (CS1U) may be introduced to provide students with facility in the UNIX operating system, which is the computing platform for most of the programming projects in this course.

Systems III: Principles of Computer Systems

Course Philosophy: The goal of this course is to provide students with an understanding of the fundamental principles of computer systems (processes, file systems, concurrency) as well as elements of networking and distributed systems. The course aims to make students facile in leveraging the facilities of modern operating systems through rigorous programming projects involving multiple OS concepts. The course also has the dual goal of having students develop designs, abstractions, and appropriate modularity in the larger pieces of software they develop. Modern software engineering practices (e.g., source control, unit testing, etc.) are stressed in project development.

Potential sample assignments and topical coverage
(The course will likely have 2-3 large-scale assignments. A few ideas for assignment possibilities are listed below.)

- Implement a web server (e.g., Apache lite), including ability to run CGI programs
  Concepts covered:
  - Concurrent processes
  - Sockets
  - File systems
  - Modular design (URL handling, security, caching, etc.)

- Implement a networked file system
  Concepts covered:
  - Concurrent processes
  - Networking
  - File systems
  - Distributed systems

- Implement a basic C shell (including pipes and file system access)
  Concepts covered:
  - Process mechanics
  - Forking and interprocess communication
  - File systems

General Topics

- Processes
  - Concurrency mechanics on a single processor
  - Context switching, interrupts and exceptions
  - Forking processes
  - Process mechanics and management
  - Interprocess communication
  - Threading
- Storage and file management
  - File systems
  - Virtual memory
  - Paging
- Networking
  - Sockets
  - Blocking vs. non-blocking strategies
  - Transport layer: TCP/IP
  - Network layer: names, routing
- Understanding of distributed systems

Potential instructors: Mendel Rosenblum and Nick Parlante

Notes: This course is an entirely new course that is **not** meant as a replacement for either CS140 or CS144/244. It is meant to provide a foundation in operating systems and networking concepts that we believe all undergraduates should be exposed to, and which can be further expanded in CS140 and CS144/244 for students specifically interested in implementing Systems. At a high level, this course is aimed at giving students experience with building larger software projects and developing their skills with modern software engineering in the context of working with systems principles.

Students in the Systems Track will certainly be required to take subsequent courses that will focus on the implementation of a wide breadth of OS facilities at a low level (e.g., CS140, CS244A and beyond). For non-Systems students (concentrations, for example, in theory, computational biology, HCI, etc.), the course offers as understanding of OS facilities through their usage in designing and building reasonably sized pieces of software. It also helps students further mature as software engineers, building general skills which will be useful to them in a wide variety of concentration areas.