Proposal for CMSC 216 - Introduction to Computer Systems

Alan Sussman and Neil Spring

1 Motivation and background

An alternate course title could be *How computer systems work*.

The goal of the course is to convey the fundamental concepts that enable programs to execute on real hardware. Those concepts include how the operating system virtualizes the hardware to provide basic services and abstractions to enable a user program to effectively use the available hardware resources. The course also addresses how different programming constructs and idioms work.

The basic abstraction of a program running as one or more threads of control in a single flat address space (a Unix *process*) is the key to the course. Emphasizing that abstraction as the underlying model for understanding how a program works, from both the user program and hardware perspective (with the OS in between), run as a theme through all topics in the course. Examples include C pointers (to data and functions), function calls and runtime stack management, dynamic memory management in the heap, and the fork/exec system calls.

Overall goals of the new course:

- Provide a coherent view of computer systems, from both a software and hardware perspective
- Decrease the length of the introductory sequence, to reduce time to students starting to take upper level CS courses
- Eliminate redundancy between current 212 and 311 topics

2 CMSC 216 topics

2.1 Topics

- Intro - Unix memory model - statics, stack, heap, etc. (1 week) - includes memory hierarchy - registers → cache → main memory → virtual memory
- Moving from Java to C - to support systems programming - arrays, memory management, I/O, machine representation of types, etc. (3 weeks; not necessarily contiguous)
• C pointers and dynamic data structures (2 weeks)
• C I/O, standard libraries (1 week)
• Testing (0.5 weeks)
• Assembly language programming (1 week) - especially mapping between C and assembly/machine code - includes how function calls are implemented by the compiler (runtime stack management)
• Process control (0.5 week) - includes fork() and exec()
• Systems programming (1 week) - includes other system calls, and pipes
• Program measurement and optimization (1 week) - includes software optimization, by compiler or by hand - loop optimizations, array organization, etc.
• Multithreaded programming with pthreads (1 week)
• Libraries and linking (1.5 weeks)
• Implementing dynamic memory management (0.5 weeks) - heap management
• Programming tools (spread through semester, as needed for other topics)
  – Basic programming tools - including gcc, make, gdb (ddd - gdb GUI), valgrind/memcheck - early in semester
  – Advanced programming tools - including gcov, gprof, shared libraries, debugging (sentinel) - later in semester

Additional topics (if time permits):

• Network programming - sockets
• Hardware design, to go fast - pipelining, branch prediction, etc.
• Advanced C literacy - unions, ternary assignment, etc.

3 CMSC 216 details

3.1 Prerequisites, Co-requisites, etc.

Prerequisite: CMSC 132 (C or better) - same as 212

Co-requisite: CMSC 250 - same as 212

CMSC 216 is a prerequisite for all 400 level systems courses, and any other 400 level courses that require systems knowledge - similar to current 311
may also want it to be a pre-requisite or co-requisite for CMSC 330
4 credits (3 hours lecture, 2 hours discussion section) - same as 212

3.2 Textbook

Required


Parts of textbook currently used in both 212 and 311. A 2nd edition is coming out January 2010.

Recommended


May be worthwhile to find a newer text, to cover changes to C standard since 1998.

4 Transition plan

The plan to transition from 212 and 311 to 216 involves transition semesters in which 216 and 311 are offered concurrently (this semester and Spring 2010). After the transition semesters, only 216 would be offered.

Typical students either will have passed 212 and take the subsequent transition semester 311 immediately, or will not have taken 212 and will take 216 instead.

Students who failed 212, so have not attempted 311, will take 216 instead.

Students who failed 311 will have an opportunity to retake 311 during the transition semesters, which will allow them to avoid repeated content. (Should students fail 311 twice, or choose not to retake 311 immediately, perhaps it is better for them to take 216.)

A Current 212 syllabus

The course introduces many of the concepts that lie behind software systems, such as hardware, memory layout, memory management, and operating systems. It explains how these concepts affect the design of software systems. This course provides a transition from the Java environment of the preceding two courses to programming in C.

The objective of the course is to develop a good working knowledge of how to program in the C programming language, to understand how to write programs with explicit memory allocation and deallocation,
to understand the UNIX and C memory model of a program as well as what happens when a program is running, and to introduce how to write systems programs.

A.1 Topics

- Introduction: moving from Java to C (3 weeks)
- Pointers, memory management, and dynamic data structures in C (4 weeks)
- I/O, standard libraries (1 1/2 weeks)
- Testing (1/2 week)
- Data representation (1 week)
- Libraries and linking (1 week)
- Process control (1 week)
- Program measurement and optimization (1 week)
- Data representation and implementation of functions (2 weeks)
- Sockets (1 week)

B Current 311 syllabus

The course is an introduction to computer system organization, mainly targeting modern processor and memory system design and implementation. IA-32 assembly language, and corresponding C language constructs are taught. Some software issues related to optimizing higher level code to run faster are also covered.

B.1 Topics

- Programming tools - gcc, nasm, gdb, UNIX (interspersed throughout the semester)
- Bits and Bytes - endian-ness, alignment, etc.
- Integer and Floating Point number representation
- Assembly Language Programming, and integrating with C - IA32 assembly
- Processor design basics and pipelining
- Program Optimization - compiler and hand code optimizations
- Cache Memory - cache organization details
• Exceptional Control Flow - exceptions, process control
• Virtual Memory - management details
• Hardware - digital circuitry