

**CMPS 311**  
**Introduction to Systems Programming**  
**Required Course in Computer Engineering and Computer Science**

Catalog Data: Introduction to Systems Programming (3).  
Unix system programming in C; organization of programs and data; program analysis and support tools; software standards; common system functions.  
Prerequisite: CMPS 221.

Typical Textbook:

- Advanced Programming in the UNIX Environment, W. Richard Stevens, Stephen A. Rago, 2<sup>nd</sup> Edition, 2005, Addison-Wesley.
- C: A Reference Manual, Samuel P. Harbison III, Guy L. Steele, Jr., 5<sup>th</sup> Edition, 2002, Prentice-Hall.
- An optional Unix user-level introduction should be added.

Course Objectives: In general, students should be able to evaluate design alternatives according to standard practice, specifications, performance analysis, robustness, etc. To concentrate attention, we investigate one system and one programming language in detail, through demonstration programs, short- and long-term programming assignments. The specific system is Unix, a family of operating systems forming a complete standardized programming environment based on the idea of software tools. The specific language is C, which is widely used for operating system implementations, and which forms the basis for the C++ and Java languages studied in the prerequisite courses. This will help students understand operating system services available to application programmers, and provide a firm ground for study of operating systems in general.

Primary Course Outcomes: There are several themes of the course:

- (1) Understand computer systems, especially low-level influences on high-level goals. This includes the machine-level representation of programs and data structures; the memory hierarchy and its impact on performance; access to stored information via file systems, and access to other computer systems via networks.
- (2) Understand existing system software and software standards, especially the UNIX toolset. This includes preparing a program (editors, static analysis, development environments); running a program (compilers and interpreters, assembler, linker, loader, debugger, profiler, tracer); controlling parts of a program (memory management, threads); communication between programs (within one system using signals, between systems using sockets and communication protocols); and combinations of software tools with scripting languages.
- (3) Understand "real code", such as selections from the Linux operating system kernel and GNU utilities and libraries, and through comparative selections from Solaris, Linux, and Mac OS X.
- (4) Understand system performance, including experiments on program performance and optimization techniques.

Relationship to Undergraduate Program Outcomes: Fundamental Skills (Software):

- Outcome 7: Analyze algorithms or computer code for correctness and efficiency.
- Outcome 8: Develop a modest (on the order of a thousand lines of code) software application, using appropriate data structures and algorithms.

Professional Skills:

- Outcome 12: Write clear and effective technical prose.
- Outcome 14: Demonstrate independent learning by using unfamiliar computer systems, test equipment, and software tools to solve technical problems.
- Outcome 15: Be able to discuss major trends in industry and current research activities within the discipline.
- Outcome 17: Be able to state a code of professional ethics and to identify ethical issues in engineering case studies.

Required Topics:

- Introduction to Unix (2 weeks)
- Review of C vs. C++, further topics of C (2 weeks)
- The C Standard and the Posix Standard (2 weeks)
- Organization of a Unix/C program (1 week)
- Process environment, memory layout, environment variables, command-line arguments (1 week)
- Shell scripts (1 week)
- Review of pointers and linked lists (1 week)
- Process control, fork(), exit(), wait(), signals and signal handlers (1 week)
- Introduction to threads (1 week)
- File I/O and the Standard I/O Library (1 week)
- Review of recent and current projects (1 week)

Class Format:

Lecture: Three lectures per week and each lecture is 50 minutes long.  
Laboratory: Open laboratory used to complete programming assignments.  
Recitation: none

Professional Component:

CMPS 311 is the fourth in a sequence of courses in software development, introducing systems programming. System programming concerns the development of software components and methods for their combination, independent of any particular application. This course will provide information and experience required to understand, design and implement components of large software systems.

Evaluation:

55% Projects  
25% smaller individual projects, completed in one week  
30% larger two-person projects, completed in two or three weeks  
45% Exams  
20% midterm  
25% final

Author: Don Heller  
Last Revised: March 26, 2008