COURSE DESCRIPTION

Dept. Number	**CSC 60**	Course Title	**Introduction to Systems Programming in UNIX**
Semester hours	**3**	Course Coordinator	**Chung-E Wang**
		URL (if any):	**http://gaia.ecs.csus.edu/~wang/**

Catalog Description

Features of the C language commonly used in systems programming and the application of those features to systems programming in a UNIX environment. Topics include C preprocessor macros, I/O and bit-manipulation facilities; timesharing system concepts; file permissions; shell script programming; make files and source code control; basic system calls like fork and exec; pointers and dynamic memory allocation; libraries, relocation and linking concepts including assembler handling of symbol tables. Prior knowledge of a C programming language is presumed. Prerequisite: CSC 020, CSC 35.

Textbook

John S. Gray, Interprocess Communications in Linux: The Nooks & Crannies, Pearson/Prentice Hall, 2003.

Brian W. Kernighan and Dennis M. Ritchie, The C Programming Language (Ansi C), Second Edition, Pearson/Prentice Hall, 1988.

Syed M. Sarwar et al., Unix: The Textbook, 2nd Edition, Pearson/Addison-Wesley, 2005.

Course Goals

1. To give students experience in the C language and in traditional procedural programming.
2. To give students a basic understanding of the use of a multi-user, time-sharing operating system.
3. To provide an introduction to systems programming in UNIX.

Prerequisites by Topic

Thorough understanding of:

- Representation of integers.
- Programming experience in Java.
- Familiarity with basic concepts of data structures and algorithms.

Basic understanding of:

- Linear lists using linked representation.
- Elements of computer architecture.

Major Topics Covered in the Course

1. UNIX file system (directories, hierarchical structure, types, access control, links and shell commands), I/O, I/O redirection and pipes, utilities, editors, the shell and shell programming (12 hours).
2. C functions, variables (declaration, initialization, scoping, storage classes), operators and expressions (precedence and associativity in evaluation), data types (fundamental and derived, type conversion, type quantifiers), control structures, and bit-manipulation (3 hours).
3. Pointers (compatibility, addressing, dereferencing, address arithmetic), pointer formal parameters and arguments (call-by-reference), pointer type void *, functions returning pointers, pointers to functions, how to define functional parameters and pass functions as arguments (4 hours).
4. Standard IO and system-level IO, string handling (6 hours).
5. Dynamic memory management (3 hours).
6. C preprocessor, header files and directives, macros, conditional compilation (1 hour).
7. UNIX support for C programming, program maintenance through make facility, system calls, error handling and recovery, debugger, and source code control (9 hours).
8. Signals and interrupts (1 hour).
9. Multiprogramming in UNIX, process creation and termination, inter-process communications such as pipes, message queues, semaphores, shared memory, sockets and remote procedure calls. (14 hours).

Outcomes

Thorough understanding of:

- The language C and procedural programming.
- Macros and C preprocessor.
- Files and standard IO.
- Pointers and dynamic variables.
- UNIX shell and shell commands.

Basic understanding of:

- UNIX programming environment and programming tools.
- UNIX shell script programming.
- System-level IO.
- Signals and interrupts.
- Process management and inter-process communications.

Exposure to:

- Threads and concurrent programming.
- Remote procedure calls.

Laboratory Projects

Seven programming assignments each taking approximately two weeks.

Estimated Curriculum Category Content (Semester hours)

Area	*Core*	*Advanced*	*Area*	*Core*	*Advanced*
Algorithms			Data Structures	.5	
Software Design			Prog. Languages	1	
Comp. Arch.					

Oral and Written Communications

No significant component.

Social and Ethical Issues

No significant component.

Theoretical Content

No significant component.

Problem Analysis

Programming assignments are designed so that students need to show either the design for the solution to a given problem using a specified problem-solving technique, or the analysis result of some given algorithm for a problem.

Solution Design

Programming assignments are designed so that students need to show either the design for the solution to a given problem using a specified problem solving technique, or the analysis result of some given algorithm for a problem.

/sj