School of Computing
UNIVERSITY OF GEORGIA

Course Information Sheet
CSCI 1730
Systems Programming

Brief Course Description (50-words or less)	Programs and programming techniques used in systems programming in Unix environments. Focus on Unix system call interfaces and the interface between the Unix kernel and application software running in Unix environments.
Extended Course Description / Comments	Students will learn the basics of Unix systems programming, including file and directory structures, basic and advanced file I/O, process creation, and inter-process communication.
Pre-Requisites	Prerequisite: CSCI 1302
Required, Elective or Selected Elective	Required Course
Approved Textbook	Author: W. Richard Stevens and Stephen A. Rago Title: Advanced Programming in the UNIX Environment Edition: 3rd Edition ISBN-13: 978-0321637734
Specific Learning Outcomes (Performance Indicators)	1. Navigate a Unix based system and use tools and commands for system programming. 2. Trace, design, and implement software solutions to non-trivial problems using the C programming language. 3. Trace, design, and implement software solutions to non-trivial problems using the C++ programming language. 4. Trace, design, and implement programs that spawn multiple processes or multiple threads and utilize inter-process communication. 5. Understand binary representation of integers in memory and use bitwise operators to solve problems. 6. Compare and contrast various programming paradigms.
ABET Learning Outcomes	A. Graduates of the program will have an ability to: Analyze a complex computing problem and to apply principles of computing and other relevant disciplines to identify solutions. B. Design, implement, and evaluate a computing-based solution to meet a given set of computing requirements in the context of the program's discipline. C. Communicate effectively in a variety of professional contexts. D. Recognize professional responsibilities and make informed judgments in computing practice based on legal and ethical principles. E. Function effectively as a member or leader of a team engaged in activities appropriate to the program's discipline. F. Apply computer science theory and software development fundamentals to produce computing-based solutions. NOTE: In the construction of the student learning outcomes for this

course, the instructors interpreted “computing requirements” in (B) as the functional requirements for a software solution and not as specific hardware requirements for the target platform; likewise, the phrase “apply computer science theory” in (F) was interpreted as using computer science principles.

Relationship Between Student Outcomes and Learning Outcomes

	ABET Learning Outcomes						
Specific Learning Outcomes		A	B	C	D	E	F
	1	●					
	2	●	●				●
	3	●	●				●
	4	●	●				●
	5	●	●				●
	6			●			●

Major Topics Covered

1. Systems Programming Tools (Knowledge level: Usage)

   a) Use emacs or vi to write programs in a Unix environment.
   b) Use basic Unix commands including (but not limited to): cd, pwd, mkdir, ls, grep, man, apropos, ps, kill, top, less, more, chmod, cp, mv, rm, diff, sort, wc, and find.
   c) Use the command line in a Unix environment to run processes and to pipe or redirect input or output.
   d) Demonstrate an understanding of the Unix file system by changing file permissions to allow programs to open, close, read, and write files.
   e) Use a Makefile to compile and run a program with more than one header file and more than one source code file.
   f) Use GDB to debug programs with various errors.
   g) Use valgrind to find memory leaks in programs. Implement programs that use dynamic memory allocation and deallocation and have no memory leaks.

2. C Programming (Knowledge level: Usage)

   a) Compile, link, and run programs via the command line.
   b) Design and implement header files that use preprocessor directives.
   c) Design and implement programs that use basic programming and flow of control structures such as (but not limited to) variables, operators, expressions, decisions statements, loops, pointers, and functions (prototyping, calling, and passing arguments to functions).
   d) Demonstrate knowledge of simple data structures (arrays, strings, structs, enums, unions) by writing out their memory maps and tracing through the output of source code.
   e) Demonstrate knowledge of pointers and arrays by tracing through the output of source code containing pointers to 1D arrays, 2D

arrays, and 3D arrays.
f) Design and implement programs that uses command-line arguments.
g) Design and implement programs in a Unix environment that open, close, read, and write files.
h) Design and implement a program that uses pointers and dynamic memory allocation and deallocation.

3. C++ Programming (Knowledge level: Usage)

   a) Design and implement basic object oriented and structured programming concepts in C++.
   b) Use simple data structures and classes in C++ such as string, vector, and list.
   c) Design and implement a program that consists of more than one class, a constructor, a destructor, a pointer, and more than one object.

4. Process and Inter-process Communication (Knowledge level: Usage)

   a) Design and implement programs that use system calls and signal handling (fork, exec, wait, etc.) and run concurrently.
   b) Design and implement programs that use pipe and dup2 for input/output redirection and inter-process communication.
   c) Design and implement a multithreaded program.
   d) Design and implement client and server programs that communicate via a socket in a Unix environment.

5. Binary Representation (Knowledge level: Usage)

   a) Understand how unsigned integers are represented and stored in memory by converting unsigned integers to and from base-10 to binary.
   b) Understand how signed integers are represented and stored in memory using two's complement representation by converting signed integers to and from base-10 to binary.
   c) Design and implement a program that uses bitwise operators on unsigned integers.

6. Programming Paradigms (Knowledge level: Familiarity)

   a) Explain some key differences between Java, C++, and C such as (but not limited to): object oriented vs. procedural programming languages, lower level vs. higher level programming languages, dynamic memory allocation and deallocation, and pointers.
   b) Compare and contrast generic models in C, C++, and Java (e.g. void pointers vs templates/generics, templates/generics vs macros).

Knowledge Levels

The following is the ACM's categorization of different levels of mastery: Assessment, Usage, and Familiarity. Note that Assessment encompasses both Usage and Familiarity, and Usage encompasses Familiarity.

Familiarity: The student understands what a concept is or what it means.

This level of mastery concerns a basic awareness of a concept as opposed to expecting real facility with its application. It provides an answer to the question “What do you know about this?”

Usage: The student is able to use or apply a concept in a concrete way. Using a concept may include, for example, appropriately using a specific concept in a program, using a particular proof technique, or performing a particular analysis. It provides an answer to the question “What do you know how to do?”

Assessment: The student is able to consider a concept from multiple viewpoints and/or justify the selection of a particular approach to solve a problem. This level of mastery implies more than using a concept; it involves the ability to select an appropriate approach from understood alternatives. It provides an answer to the question “Why would you do that?”

Modified	09/21/2022
Approved	Not yet