UNIVERSITY OF NEBRASKA AT OMAHA
COURSE SYLLABUS/DESCRIPTION

Department and Course Number	CSCI 4510
Course Title	System Programming
Course Coordinator	Stanley Wileman
Total Credits	3
Repeat for Credit?	No
Date of Last Revision	June 16, 2015

1.0 Course Description Information

1.1 Catalog description

This course covers material essential to the successful creation and maintenance of computer systems. Application programming aims to produce systems for a particular problem domain, while system programming aims to produce systems and tools used to develop application programs. While the course focuses on software, some discussion of hardware systems is provided to familiarize students with the types and characteristics of modern hardware. The primary goal of the course is for students to understand enduring concepts common to all modern computer systems and to learn how these affect the correctness, performance, and utility of such systems.

1.2 Prerequisites of the course:

CSCI 4350 (Computer Architecture) and CSCI 4500 (Operating Systems).

1.3 Overview of content and purpose of the course:

This course is concerned with the design, implementation and use of software development tools other than operating systems, compilers, and language interpreters (which are discussed in other courses). It does not address typical application programs, but instead focuses on those programs used by software developers (e.g. linkers, text editors, object file tools, documentation preparation and formatting tools, and archive utilities). The necessary background to consider these tools is also included in the topics covered in the course.

1.4 Unusual circumstances of the course:

None.

2.0 Course Justification Information

2.1 Anticipated audience / demand

This course is intended for students interesting in low-level system programming, which includes those interested in computer architecture, operating systems, programming language compilers and interpreters, and the large number of software tools used by software (and hardware) developers. It is expected that there will be demand for this course from computer science majors (certainly), but also from students in a variety of other disciplines (in particular, computer engineering). The course is not required for any degree program.

2.2 Indicate how often this course will be offered and the anticipated enrollment:

This course will be offered at most once each academic year. The anticipated enrollment is 20 students.

2.3 If it is a significant change to an existing course, please explain why it is needed.

This is a new course.

3.0 List of performance objectives stated in learning outcomes in a student's perspective:

1. Describe a program's execution environment. Include discussion of exceptional condition handling, memory management, dynamic loading/linking, system-level input/output, essentials of network communication, concurrent programming (processes, threads, and synchronization), time, localization, and security essentials.
2. Explain various approaches to measuring and optimizing program performance (profiling, timers, processor cycle counters).
3. Discuss tools commonly used in software production, including (but not necessarily limited to) text editors and GUI layout tools, integrated development environments (IDEs), source code management systems, preprocessors, source and object code librarians (to manipulate UNIX, zip, jar archives), system build tools (e.g. make and ant), and object code linkers (e.g. UNIX ld). Demonstrate the use of a large number of these tools.
4. Discuss common file formats, particularly the format of object and executable files (e.g. PE, ELF, MACH-O), and the significance of these formats on common software tools like compilers, assemblers, and linkers. Explore the run-time memory organization of executable programs, including static and dynamic linking, and overlays.
5. Describe how debuggers (e.g. gdb, debug, WinDbg) can be used to examine and control a program during its execution, and understand typical operating system services that facilitate the operation of debuggers. Also explain how disassemblers can be useful in debugging activities.
6. Demonstrate familiarity with typical tools/libraries used to manipulate files and file systems and some implementation approaches (e.g. macro processors, regular expression tools, file system searching/checking/listing utilities).
7. Describe various approaches used to document systems, system tools and software, and consider the tools used to produce, distribute, and view such documentation (e.g. UNIX man pages, nroff/troff, TEX, docbook, perlpod, javadoc)
8. Discuss the components and configuration of typical hardware systems (e.g. memory, processor(s), external and internal buses, secondary storage, backup devices, network interfaces, video adapters).
9. Explain commonly-used approaches used to limit the use of software to licensees.

10. Explain various open-source licensing agreements.
11. Describe techniques used to package, distribute, and install systems. Demonstrate the use of one or more of these techniques.

4.0 Content and Organization Information

- 4.1 Introduction (0.5 hours)
- 4.2 Extending the execution environment (3.5 hours)
  - 4.2.1 Dealing with exceptional conditions
  - 4.2.2 Memory management (dynamic allocation, exceptions, access faults)
  - 4.2.3 Dynamic linking
  - 4.2.4 Use of system services for input/output
    - 4.2.4.1 When is such activity appropriate?
    - 4.2.4.2 Methods for using system input/output services
  - 4.2.5 Network communication (simple socket-based programming)
  - 4.2.6 Concurrent programs (introduction to POSIX threads, processes, synchronization)
  - 4.2.7 Time (time sources, getting/setting the time, dealing with time zones, formatting dates and times)
  - 4.2.8 Program localization (e.g. message libraries, number formatting, locales)
  - 4.2.9 Security (e.g. passwords, encryption)
- 4.3 Program performance (1 hour)
  - 4.3.1 Performance metrics (memory use, kernel/user mode CPU use)
  - 4.3.2 Selecting and producing test data
  - 4.3.3 Profiling program execution
  - 4.3.4 Interpreting program profiles
  - 4.3.5 Approaches to improving program performance
- 4.4 Tools for software production (5 hours)
  - 4.4.1 Text editors
  - 4.4.2 GUI layout tools
  - 4.4.3 Integrated development environments (IDEs)
  - 4.4.4 Source code management systems
  - 4.4.5 Preprocessors
  - 4.4.6 Object code librarians (ar, ranlib, etc.)
  - 4.4.7 Build tools (make, ant)
  - 4.4.8 Preprocessors
  - 4.4.9 Object code linkers (introduction)
- 4.5 Object files (6 hours)
  - 4.5.1 The function of object files
    - 4.5.1.1 Hold the output of assemblers, linkers, librarians
    - 4.5.1.2 Hold dynamically-linkable modules
    - 4.5.1.3 Hold partially linked collections of object modules
    - 4.5.1.4 Hold executable files
  - 4.5.2 Components of object files
    - 4.5.2.1 Text
    - 4.5.2.2 Data
    - 4.5.2.3 Symbol tables
    - 4.5.2.4 Relocation information
    - 4.5.2.5 Debug information

- 4.5.3 Object file formats
    - 4.5.3.1 Why different object file formats?
    - 4.5.3.2 Object file format and processor relationships
    - 4.5.3.3 Common object file formats
- 4.5.4 Combining (linking) object files
    - 4.5.4.1 Explicit input modules
    - 4.5.4.2 Implied input (library searches)
    - 4.5.4.3 Dynamic linking
    - 4.5.4.4 Overlays
    - 4.5.4.5 Linker scripts
    - 4.5.4.6 Output format
    - 4.5.4.7 Linker symbol tables
    - 4.5.4.8 Relocation issues and approaches
    - 4.5.4.9 Adding explicit/implicit output symbols

4.6 Debuggers (1 hour)
- 4.6.1 How debuggers work
- 4.6.2 Approaches to using debuggers
- 4.6.3 System services used by debuggers
- 4.6.4 Introduction to disassemblers

4.7 File manipulation tools (6 hours)
- 4.7.1 Macro processors (e.g. m4)
- 4.7.2 Regular expression tools (e.g. grep)
- 4.7.3 File system searching (e.g. find)
- 4.7.4 File system checking (e.g. fsck)
- 4.7.5 File system listing (e.g. ls)
- 4.7.6 Other tools

4.8 Documentation (1.5 hours)
- 4.8.1 Documentation formats
- 4.8.2 Documentation location (inline, separate files)
- 4.8.3 Documentation tools (e.g. nroff/troff, man macros, $\TeX$, docbook)

4.9 Hardware systems (2 hours)
- 4.9.1 Processors
- 4.9.2 Memory (ROM, RAM, Flash)
- 4.9.3 Internal buses
- 4.9.4 External buses (e.g. Firewire, USB, SCSI)
- 4.9.5 Secondary storage (disk, CD, DVD, BD, RAID configurations)
- 4.9.6 Backup (removable disks, magnetic tape, NAS)
- 4.9.7 Network interfaces (wired/wireless Ethernet, Bluetooth, token ring, fiber)
- 4.9.8 Video adapters

4.10 Software licensing (0.5 hours)

4.11 System packaging, distribution, installation, configuration (1.5 hours)

5.0 Teaching Methodology

5.1 Methods to be used

Teaching methods will include in-class lectures, hands-on lab exercises, in-class quizzes, homework assignments, and case studies.

5.2 Student role in the course

Students are expected to attend all lectures and labs, participate in class discussions, and complete assigned homework and examinations.

6.0 Evaluation

6.1 Types of student projects

- 6.1.1 Construct programs illustrating various program execution environment features. (6 weeks)
- 6.1.2 Demonstrate understanding of object file formats and object file manipulation tools (3 weeks)
- 6.1.3 Demonstrate program profiling to identify where program optimization would be most effective (1.5 weeks)
- 6.1.4 Demonstrate the use of a program debugger (1.5 weeks)
- 6.1.5 Illustrate the use of several file and file system manipulation tools (3 weeks)

6.2 Basis for determining the final grade

Homework assignments/laboratories: 60%
In-class quizzes: 10%
Midterm exam: 10%
Final examination: 20%

6.3 Grading type

Letter grades will be determine using the weighted average of the various items used to evaluate students. A typical grade mapping is illustrated below.

Percent	**Grade**	**Percent**	**Grade**
97 – 100	A+	77 – 79	C+
94 – 96	A	70 – 76	C
90 – 93	A–	70 – 73	C–
87 – 89	B+	67 – 69	D+
84 – 86	B	64 – 66	D
80 – 83	B–	60 – 63	D–
		0—59	F

7.0 Resource Material

7.1 Textbooks and/or other required readings used in course.

- 7.1.1 Levine, *Linkers and Loaders*, Morgan Kaufmann, (2000)
- 7.1.2 Englander, *The Architecture of Computer Hardware and Systems Software: An Information Technology Approach*, Wiley (2003)

7.2 Other suggested reading materials, if any.

- 7.2.1 Biggerstaff, *Systems Software Tools*, Prentice-Hall (1986)
- 7.2.2 Ecker, Müller, and Dömer, *Hardware-dependent Software: Principles and Practice*, Springer (2009)
- 7.2.3 Gauthier and Ponto, *Designing Systems Programs*, Prentice-Hall (1970)

7.2.4 Grötker, Holtmann, Keding, and Wloka, *The Developer's Guide to Debugging*, Springer (2008)

7.2.5 Robbins and Robbins, *UNIX Systems Programming: Communication, Concurrency and Threads*, Prentice-Hall (2003)

7.2.6 Rosenberg, *How Debuggers Work: Algorithms, Data Structures, and Architecture*, Wiley (1996)

7.2.7 Stallman and Pesch, *Debugging with GDB: The GNU Source-Level Debugger*, Free Software Foundation (1993)

7.2.8 Stevens and Rago, *Advanced Programming in the UNIX Environment*, Addison-Wesley (2008)

7.2.9 Vesperman, *Essential CVS*, O'Reilly (2003)

7.3 Other sources of information.

7.3.1 Documentation of system software tools for a variety of systems (UNIX, Linux, QNX, Windows, Mac OS X, etc.)

7.3.2 http://www.cs.utk.edu/~plank/plank/classes/cs360/

7.3.3 http://www.cs.vu.nl/~gpierre/courses/sysprog/

7.4 Current bibliography of resources for student's information

None

8.0 Other Information

8.1 Accommodations statement:

8.2 Other:

8.3 Author(s): Stanley Wileman

9.0 Computer Science Accreditation Board (CSAB) Category Content (class time in hours):

CSAB Category	*Core*	*Advanced*
Data structures	0	9
Computer organization and architecture	2	9
Algorithms and software design	2	10
Concepts of programming languages	0	1

10.0 Oral and Written Communications

Every student is required to submit at least ___1__ written reports (not including exams, tests, quizzes, or commented programs) to typically _10____ pages and to make __0___ oral presentations of typically __0___ minutes duration. Include only material that is graded for grammar, spelling, style, and so forth, as well as for technical content, completeness, and accuracy.

11.0 Social and Ethical Issues

The potential for security flaws in system software to expose sensitive data is treated (0.5 hours). Issues associated with software licenses and methods to prevent users from violating the terms of software licenses are considered.

12.0 Theoretical content

Please list the types of theoretical material covered, and estimate the time devoted to such coverage.

1. Concurrent processes, threads, and synchronization (at most 2 hours)
2. Memory organization, static and dynamic use of memory resources during program execution (at most 2 hours)
3. Representation of executable and object files, and the effect of processor type and memory hierarchies on the type and organization of such files (2 hours)
4. Elements of lexical analysis, parsing, and regular expressions as used in software tools (at most 2 hours)

13.0 Problem analysis

Please describe the analysis experiences common to all course sections.
The problems associated with system software development tools, and the use of those tools by application programmers, are considered. Students analyze such problems to identify appropriate tools for their solution.

14.0 Solution design

Please describe the design experiences common to all course sections.
Students taking the course are expected to design and use small system software tools (or parts of tools) in the course laboratory assignments. Students also make design decisions when considering which tools to use in solving a problem, and in ancillary activities (such as licensing, configuration, installation, and distribution).

CHANGE HISTORY

Date	***Change***	***By whom***	***Comments***
4/15/2009	Initial version	Wileman	
6/16/2015	Revised the syllabus format	Wileman	
6/26/2016	Numbered list of performance objectives	Wileman	