COURSE DESCRIPTION

Dept., Number  CSC 159  Course Title  Operating System Pragmatics
Semester hours  3  Course Coordinator  Weide Chang
URL (if any):  http://gaia.csus.edu/~changw/

Catalog Description

The application of operating system principles to the design and implementation of a
multitasking operating system. Students will write an operating system for a computer platform.
Topics include: scheduling of processes, control and allocation of computer resources, and user
interfacing. Prerequisite: At least a C- in CSC 139 and full CSC or CPE major status. Cross-listed
as CPE 159; only one may be counted for credit.

Textbook


References

and Programmers (Embedded Technology), Elsevier, 2005.
Richard M. Stallman, Roland H. Pesch, Stan Shebs, Debugging with GDB, Free Software
Graham Glass and King Ables, UNIX for Programmers and Users, 3rd edition, Prentice-Hall,
2003.
Pentium Processor Register and Data Structure Summary, Intel Corporation.

Course Goals

By coding a working kernel and other operating system components, to give students hands-on
practical experience in the application of operating system principles covered in CSC 139, so
they will have a solid and thorough understanding of the design and implementation issues
surrounding the development of contemporary operating systems.

Prerequisites by Topic

Thorough understanding of:

-  Programming proficiency in a high-level language, preferably C or C++, and in assembly
   languages.
-  Data structures – lists, trees, graphs.
Searching and sorting techniques.
Digital logic – combinatorial and sequential circuits, finite state machines, state transition graphs.
Computer organization – CPU structures, memory organization and alternatives, bus structures, instruction cycles, interrupt vectors and interrupt service routines, DMA principles.
I/O programming – status bits and registers, polling.
Operating system fundamentals – processes, scheduling, inter-process synchronization techniques, classical synchronization problems (producer/consumer, readers/writers), memory management problems and alternatives.
Experience using an interactive timesharing computer system (e.g., UNIX).

Basic understanding of:
Programming proficiency in a high-level language, preferably C or C++, and in assembly languages.
Data structures – lists, trees, graphs.
Searching and sorting techniques.
Digital logic – combinatorial and sequential circuits, finite state machines, state transition graphs.
Computer organization – CPU structures, memory organization and alternatives, bus structures, instruction cycles, interrupt vectors and interrupt service routines, DMA principles.
I/O programming – status bits and registers, polling.
Operating system fundamentals – processes, scheduling, inter-process synchronization techniques, classical synchronization problems (producer/consumer, readers/writers), memory management problems and alternatives.
Experience using an interactive timesharing computer system (e.g., UNIX).

Exposure to:
Programming proficiency in a high-level language, preferably C or C++, and in assembly languages.
Data structures – lists, trees, graphs.
Searching and sorting techniques.
Digital logic – combinatorial and sequential circuits, finite state machines, state transition graphs.
Computer organization – CPU structures, memory organization and alternatives, bus structures, instruction cycles, interrupt vectors and interrupt service routines, DMA principles.
I/O programming – status bits and registers, polling.
Operating system fundamentals – processes, scheduling, inter-process synchronization techniques, classical synchronization problems (producer/consumer, readers/writers), memory management problems and alternatives.
Experience using an interactive timesharing computer system (e.g., UNIX).
Major Topics Covered in the Course

1. Software development in a cross-compiled target environment (2 hrs).
2. Target machine architecture and variations; effects on OS development (3 hrs).
3. OS organizational models – monolithic, layered, microkernel based (2 hrs).
5. Implementation mechanisms for kernel services (3 hrs).
6. Kernel service implementations: sleep, semaphores, message queues, miscellaneous hardware access/control (6 hrs).
8. User interface – interactive life cycles, shells, command execution (1 hr).
10. Memory management – implementation of virtual memory, paging hardware (6 hrs).
11. File system implementation – directory structures, I/O requests, buffering (2 hrs).
12. Asynchronous notification mechanisms (2 hrs).
13. Multiprocessor issues (1 hr).
14. Commercial examples (1 hr).

Laboratory Projects

1. Cross-compilation and downloading (1 week).
2. Source code control (1 week).
3. Cross-platform debugging (1 week).
4. Implement a timer interrupt service routine on the remote target machine (1 week).
5. Implement a simple micro-kernel to dispatch processes (2 weeks).
6. Implement semaphore service for process synchronization (2 weeks).
7. Implement message services for processes’ inter-process communication (2 weeks).
8. Implement a user shell with a small set of built-in commands, running several reentrant threads on multiple terminals (2 weeks).
9. Implement several external commands for user shells, and kernel services to support dynamic process creation (2 weeks).
10. Implement virtual memory for dynamic processes (2 weeks).

Estimated Curriculum Category Content (Semester hours)

<table>
<thead>
<tr>
<th>Area</th>
<th>Core</th>
<th>Advanced</th>
<th>Area</th>
<th>Core</th>
<th>Advanced</th>
</tr>
</thead>
<tbody>
<tr>
<td>Algorithms</td>
<td></td>
<td>0.5</td>
<td>Data Structures</td>
<td></td>
<td>0.5</td>
</tr>
<tr>
<td>Software Design</td>
<td>1.0</td>
<td></td>
<td>Prog. Languages</td>
<td>0.5</td>
<td></td>
</tr>
<tr>
<td>Comp. Arch.</td>
<td>0.5</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Oral and Written Communications

Students are required to submit team project reports. Inline document of their source code is also a requirement for team-based software project development.

Social and Ethical Issues

No significant component.

Theoretical Content

No significant component.

Problem Analysis

Students are required to analyze the various options available for organization of the code structure of the microkernel of an OS, and tackle the various issues of inter-process communication, concurrency control, management of processes, memory, I/O, and files. Security and user interface issues are also analyzed.

Solution Design

Students are required to choose an appropriate design for implementation. They are also required to develop methods to implement correct and effective kernel functions including security and user interfaces, and to design and implement a virtual memory system.