Course Description
CMPS 360
Operating Systems and Systems Software

Catalog Description:
A study of the introductory concepts in operating systems: historical development of batch, multiprogrammed, and interactive systems; file, memory, device, process and thread management; interrupt and trap handlers, abstraction layer, message passing; kernel tasks and kernel design issues; signals and interprocess communication; synchronization, concurrency, and deadlock problems.

Prerequisite:
CMPS 223

Units:
5

Instructor:
Marc Thomas

Goals:
(OS1) Overview of operating systems and history: understand the basic structures, primitives, and functional layers of microkernel, monolithic and virtual machine operating systems.
(OS2) Operating system principles: Understand the hardware issues, software abstractions and data structures required for operating system design (e.g. CPU protection, hardware traps/interrupts, call and interrupt gates, etc.).
(OS3) Concurrency: Understand process/thread structure, creation (fork().. execute()) priorities, states, shared data structures and process/thread synchronization.
(OS4) Scheduling and dispatch: Understand the algorithms involved in scheduling, deadlock control, and queueing theory.
(OS5) Memory management: Understand all the issues involved in memory system design especially virtual memory systems (e.g. TLB cache, paging, copy on write, etc.)
Understand the tradeoffs involved in operating system design (e.g. performance vs. security).
(Laboratory) Become proficient in writing basic systems programs utilizing signals, process creation, interprocess communication, multiple threads, and the Unix Run Time Interface; work with a batch/multiprogrammed operating system simulator.

Texts:
Recommended: Kernighan and Ritchie, The ANSI C Programming Language, Chapter 8, the Unix System Interface.
Topics:
Review of basic computer hardware; the CPU, operating system, and hardware interface: device controllers, interrupts, traps, and exceptions; DMA, system and user memory.
What is an operating system and what should it do; historical development: batch systems, multi-programmed systems, and interactive time-sharing systems.
Operating system data structures; the operating system as a resource manager of processes, memory, file store, I/O, and network communications; services provided to processes via system calls; operating system designs: monolithic, microkernel, and virtual machines.
Process management in detail, including process state, process control blocks (PCB), descriptor tables, threads of execution, scheduling, context switches; interprocess communication; process creation: \texttt{fork()} \ldots \texttt{execve()} versus \texttt{spawnve()}.
Relationship between user threads and light-weight processes (LWP),
Process and thread synchronization: critical sections of code, semaphores, and deadlocks.
The advantages of virtual memory and the complications which must be handled by the system software.
Deadlock control and resource allocation graphs.
CPU scheduling strategies.
Kernel structure in a monolithic design.

Laboratory:
The laboratory session will parallel the lecture, illustrating the principles and familiarizing the student with system software constructs. Lab assignments will be in C and will cover the use of the Unix Run-Time Interface, POSIX signal handling, the System V Inter-Process Communications (IPC) suite, and the POSIX \texttt{pthread} package to write portions of operating system code, do simulations, test virtual machines, and experiment with synchronization and deadlock.

Grading:
Two midterms will be given, each worth 25%. I do not give make-up midterms; for an excused absence I count the other grades proportionately higher. One final exam, comprehensive but emphasizing the later material, will be given. It is mandatory and worth 25%. Homework and lab work are together worth the remaining 25%. 