CSCI 3753 / Midterm Study Guide

Notes:

1. While I am making this study guide as detailed and in-depth as possible, and I am attempting to highlight the most relevant material to study, **I want to make clear that ANY material from lecture, programming assignments, recitation, reading assignments, and quizzes is fair game (unless specified otherwise below)**. That said, I will be focusing on the topics listed below.
2. If there are any calculations to be made I will provide the formulas needed. I will NOT provide algorithms.
3. Any pencil coding will NOT be nitpicked to ensure perfect C. In other words, no, I will not deduct a student for forgetting a semicolon.

/Midterm Breakdown

Thursday March 10th – In lecture you will get 30 to 35 multiple choice questions and a scantron sheet. A significant portion of questions will be taken from and inspired by reading quizzes. However, **please expect new questions as well. This will focus mostly on lecture and reading** (you may have a handful of questions on the programming assignments). 50% of the midterm grade.

Friday March 11th – In recitation you will get 3-5 long answer questions. Expect pencil coding, and explaining concepts in detail. This will focus on programming assignments 1-3 as well as things like synchronization problems and concepts, scheduling, and multiprocessing. **Please make sure to review: programming assignments 1-3 (including concepts), synchronization concepts and problems, threading, multi-process APIs like fork and exec, and various schedulers.** 50% of the midterm grade.

/ Topics to Skip

- The monitors lecture, or anything to do with monitors (except where mentioned below)
- Remote procedure calls
- Threads in Java
- Anything in the book about Solaris, Mach, or mobile operating systems
- Anything in the lectures and book about Windows… while this is important material, I will not be testing you over it on the midterm
- Dining-philosophers problem
- Learn what deadlocks are, and what causes them… however deadlock remediation will be covered on the final, NOT ON THE MIDTERM

/ Topics to Review

- The reading quiz questions
- Programming assignments #1, #2, and #3 including all concepts needed to create a solution
- API vs. ABI, what ABI means and what it defines, why can't you run a Linux executable on a Mac?
- systemcalls, what they are, how to call them, passing parameters – advantages and disadvantages of the various methodologies, traps, details about systemcalls in Linux (check PA1 for more details)
- Monolithic vs. micro vs. hybrid kernels and what OS's are what
- Good definition of an OS and what it does, including layers, and components
- Virtual memory, the basics of what it is, how the OS implements it, various address spaces, security and isolation
- Intel processor rings, and the mode bit, what the mode bit does, difference between kernel and user mode
- Interrupts, various kinds, how they're used and their purposes
- Trap table
- Multiprogramming – sequential execution, how to handle I/O blocks
- Multitasking – cooperative vs. preemptive, pro's and con's of both, what OS implements which
- Context switching – what happens, what is backed up, how it works
- Turing machine – what it is, how the theoretical machine works
- BUS – various types, be able to explain a BUS
- Von Neumann architecture – is it a Turing machine? Explain the architecture
- Classes of exceptions and what they're for and their behavior (trap, fault, hardware interrupt, abort)
- The interface for I/O drivers, what functions should a driver support? (see PA1 for more details)
- Block vs. character devices, sequential vs. random access, what kinds of devices are what
- Blocking vs. non-blocking I/O, synchronous vs. asynchronous
- Polling I/O, interrupt driven I/O – what devices are what
- DMA – how it works and what it is, DMA w/ interrupts
- The basic execution of the interrupt handler (slide 34, Lecture 3)
- Interrupt masking
- Port mapped I/O, the basics, how it works, limitations
- Memory mapped I/O, how it works, benefits
- Booting - how it works, POST -> BIOS -> MBR / primitive loader -> Secondary loader (LILO, GRUB) -> OS
- Processes – what is a process? process states, parts of a process: stack, heap, global data, code

- Process scheduling queues
- Process termination scenarios involving parent and child processes (zombies, etc.), cascading termination
- How the stack and heap differ, how they're used, collisions
- Process manager, process control block
- Creating a process in Linux (POSIX OS's) – fork, exec, wait, exit
- Fork copy-on-write
- Threads vs. processes, motivation for threads
- Reentrant code – what makes code reentrant, what is reentrant and what is not, difference vs. thread safe
- User space vs. kernel threads – details about both types of threads, uses
- Inter-process communication – reasoning and methods to implement
- Shared memory – details, how this is implemented in the OS, limitations, shmget, shmat
- UNIX socket vs. Internet domain socket – differences in using bind
- Pipes – how used (read, write, send [async], receive [sync]), ordinary pipes are one way, two way pipes, how pipes are used with fork (slide 17, lecture 7)
- Named pipes – how different from ordinary pipes
- Signals – how they are used, OS notifies process of events with a numerical code, use of signal to register a handler (slide 26, lecture 7)
- kill() – don't kill anyone per se, but know about this command, what it does, and how it works with signals
- Race conditions, need for synchronization
- Spinlocks
- Critical sections
- Disabling interrupts as a strategy, weaknesses of approach
- Locks
- TestAndSet – what it does, the need for it, how to use it to create a lock (slide 20, lecture 9)
- Threads
    - Threading models
    - Thread pools
    - Thread cancellation
    - Pthreads in Linux
- Semaphores, in particular revised semaphores (slide 35, lecture 9)
- Using semaphores to implement mutual exclusion and enforcing critical sections
- Using semaphores to enforce ordering of operations
- In particular, review the code in "Synchronization and threading example code", focus on understanding how this code solves the readers-writers synchronization problem
    - In ThreadReadersWritersSimple.c review the following and their functionality:
        - pthread_create()
        - pthread_mutex_t type

        - pthread_mutex_init()
        - pthread_mutex_destroy()
        - pthread_mutex_lock()
        - pthread_mutex_unlock()
    - In ThreadReadersWritersSemaphores.c review the following and their functionality:
        - sem_t type
        - sem_init()
        - sem_destroy()
        - sem_wait()
        - sem_post()
    - In ThreadReadersWritersMonitor.java, understand the synchronized keyword, and what it does within a class
- Thoroughly understand the “bounded buffer producer-consumer” problem, and an implementation to solve it
- Thoroughly understand the “readers-writers” problem, and an implementation to solve it (you have sample code which does this)
- Switching between processes (context switch), what this entails, time slices
- Wait time, execution time, turnaround time
- Scheduling criteria and goals
- Soft real time system and hard real time systems – characteristics and differences
- Scheduling algorithms - including advantages/disadvantages, preemptive/non-preemptive, characteristics, starvation, etc.:
    - FCFS
    - SJF
    - Round robin / Weighted round robin
    - EDF
    - Multilevel Queue Scheduling – promotion and demotion, etc.
- Linux scheduling algorithms and concepts – how they work, etc.:
    - O(1)
    - O(N) – priorities [0, 140], what’s real time vs. non-real time
    - nice and nice values and what they mean
    - CFS
        - split processor into smaller processors and give each process an equal slice
        - process that gets the cpu accrues a wait time penalty, every other process’s wait time increases
        - give the process that is owed the most time the cpu (largest wait time), use a red-black tree to find largest time deficit
        - task weight = 1024 / (1.25 ^ nice value)
        - runtime = (20 ms * task weight) / (sum of all running process’s weights)
    - Revised CFS

        - vruntime, run time, global fair clock
        - Give process that has the least run time the cpu, use a red-black tree to store vruntime values
        - How does a new process get assigned assigned an initial vruntime? What happens if a new processes vruntime is initially assigned to 0?
    - Real time scheduling, what process priorities?
        - Real time FIFO
        - Real time Round Robin
        - Real time EDF
- Multiprocessor scheduling – asymmetric vs. symmetric, queue concerns
- Deadlocks – understand what they are, and the 4 necessary conditions which must hold for them to happen, resource allocation graph

/ Hints

- Possible programming questions for the long-answer portion of the test:
    - Anything from PA1, PA2, PA3 (I think you know what was important in these three assignments)
    - Using TestAndSet to create a lock
    - Thread safe Reader/Writers or Bounded-Buffer implementation using semaphores or locks
    - Fork a process and replace it with Firefox
    - Create a thread / thread pool
- While I will certainly ask you questions outside of these areas, the most important areas to review in detail are (this is where to start):
    - Reading Quiz questions
    - Programming Assignments
    - Von Neumann architecture
    - Interrupts
    - I/O ports, polling, interrupts, DMA
    - Booting
    - Everything related to syscalls
    - IPC, specifically shared memory, pipes, sockets
    - TestAndSet, locks, semaphores, critical sections
    - Threading and pthreads
    - Process forking, etc.
    - The demonstration code I gave you (shows solutions to Readers/Writers, demonstrates locks and semaphores, and gives you an example of phtreads)
    - Readers/Writer and Bounded-Buffer problems and solutions
    - Process schedulers
    - Kernel process data structures