Spring 2009: CS 241 Final Exam Review Session

***Exam Time**: Thursday, May 14 @ 8:00am @ 1404 SC (A-S) and 1214 SC (T-Z) by last name*
***Exam Duration**: 2 hours and 45 minutes*
***Exam Format**: 60-80 multiple choice questions (bring #2 pencil along with your I-Card)*
This is a guide intended to help you study; there may be topics we missed that are on the final!

Review Topics List:

I. C Programming

1. What is the * operator? What does it do?
2. What is the & operator? What does it do?
3. What's the difference between char c[80] and char *c?
   …what about when they're used in sizeof()?
4. What do common C functions do? How are they called?
   String Functions: strcpy(), strlen(), strcmp(), strcat()
   Binary Functions: memcmp(), memcpy(), memset()
   Memory Functions: malloc(), remalloc(), free()
   I/O Functions: printf(), fgets(), fgetc(), read(), write(), fread(), fwrite(), fopen(), fclose()
   Thread-related Functions: pthread_create(), pthread_join(), pthread_detacth()
   Semaphore-related Functions: sem_init(), sem_destroy(), sem_post(), sem_wait()
   … and others.
5. What is NULL?
6. Understand how to trace and write pointer code.
7. What's the difference between a stack and a heap variable? What about global and static variables?
8. What is stdout? What is stdin? Is stderr even real?

II. Operating Systems

1. What is an operating system?
2. What is the difference between a function call and a system call?
3. What is an example of a C library call that does, eventually, make a system call?
4. What is the difference between a process and a thread?
5. How do a user-thread and a kernel-thread differ?
6. What are the different states that a process may be in within in operating system?
7. What is a zombie thread? What is an orphan process?
8. What are the return values of fork()?
9. Why is fork() nearly always used in conjunction when an exec() call is going to be used?
10. What parts of memory are retained when fork() is called? …when exec() is called?
11. How do you abandon a thread? How do you kill it? How do you wait for it to finish?
12. What does it mean for a function to be thread-safe?

III. Scheduling

1. Why do processes need to be scheduled?
2. How do you schedule with a FIFO policy? FCFS? SJF? Round robin?
3. What does it mean for a scheduling algorithm to be preemptive?

4. How does bounded wait apply to scheduling? What is starvation? What is the convey effect?
5. What is response time? Turn-around time? What other metrics do we use?
6. Which scheduling algorithm minimizes average response time? Waiting time? Turn-around time?
7. How does Round Robin differ in nature when it has a small quantum vs. a large quantum?
8. Why is SJF/PSJF hard to implement in real systems?
9. Why is FIFO not considered a good algorithm in interactive systems?

IV. Synchronization / Semaphores

1. When is synchronization needed?
2. What is the difference between a semaphore and a mutex? Can you use mutexs in place of a semaphore?
3. What is a critical section?
4. What is deadlock? What other properties are there in relation to synchronization?
5. How does semaphores and testandset() differ?
6. What are algorithms learned in class to deal with synchronization?
7. How do you define a POSIX semaphore in C? What are the two main function calls to use it? How to you clean up the memory associated with a semaphore?
8. What is the Dining Philosophers Problem?
9. What is the Producer-Consumer Problem?
10. What is the Reader-Writer Problem?
11. Can you apply concepts in the classic problems to new, unseen problems?

V. Signals

1. What are signals? Why are they needed?
2. What is the signal sent with Ctrl+C at a terminal?
3. Many programmers often use SIGALARM. How is that signal generated?
4. What is a signal mask? What is a signal set?
5. Are the signals that cannot be caught? If so, what is the need for those signals in an OS?
6. It is documented that read() is not signal-safe. Why?
7. What is a standard C function call that is signal-safe?

VI. Deadlocks

1. What does it mean when something is deadlocked?
2. Know the differences between deadlock and starvation.
3. What are the four conditions for deadlock? What if one of them is removed?
4. What is deadlock detection? Prevention? Avoidance?
5. What is the default deadlock solution on many systems (the "ostrich" approach)?
6. What is a resource allocation graph?
7. What is Dijkstra's Banker Algorithm? What does it ensure? Why is it constantly ensuring that there exists a "safe state"? What form of deadlock checking is a "safe state"?
8. What is a wait-for graph?

VII. Queuing Theory

1. What is queuing theory?
2. What is the major assumption made in basic queuing theory that cannot be made in real systems?

3. What is arrival rate? Service rate? Server utilization?
4. How does arrival rate and server utilization relate to service rate?
5. How can you calculate the time an average job will be in the system?
6. How can you calculate the average time a job will spend in the queue?
7. Know how to relate queuing theory problems to practical examples (eg: how long do I have to wait before getting my Jamba Juice? …what if there's two lines?)

VIII. Memory Management

1. What is the principle of locality / locality of reference?
2. What are common page replacement schemes?
3. Why can't an operating system simply implement an optimal page replacement scheme?
4. What is fragmentation? Internal fragmentation? External fragmentation?
5. What form of fragmentation do paging systems essentially eliminate?
6. What is thrashing?
7. What is memory-mapped I/O (mmap)?
8. What are page tables?
9. How do virtual addresses and physical addresses work?
10. What can be implied about a page offset and a virtual address size? How do you calculate it?
11. What are the advantages and disadvantages of multi-level page tables?
12. What is DMA?
13. What are memory storage algorithms?
14. How is first-fit and next-fit different? When would one use worst-fit?

IX. I/O Access and Disk Concepts

1. What are the two commands to open a file in C?
2. What are i-nodes?
3. What information is stored in an i-node? What information isn't stored in an i-node? If the information isn't stored in an i-node, where is it stored?
4. What is a directory file? What does it contain?
5. What are hard links? What are soft links? What are the differences?
6. Understand UNIX access policies and chmod numbers.
7. What are FIFO, SSTF, SCAN, and C-SCAN?
8. What is the total time to access and read a file? How do you calculate it?
9. How do you calculate the maximum size of a given file?

X. Networking

1. What is TCP?
2. What is UDP?
3. When would you use UDP over TCP? When TCP over UDP?
4. What are packet headers? What do they do for the packet?
5. Understand if accept() is a blocking call or not. What does accept() return?
6. What is HTTP?
7. What is a web proxy? How does a proxy make a request on behalf of a client?
8. What are ports?
9. What about networking in C is reused from file I/O in C?
10. What are some services that use TCP on the Internet? UDP?
11. What is peer-to-peer? What is centralized server-client? Give examples of both.
12. What are the benefits of TCP? What do each of those benefits mean?