CS 241 Final Exam Study Guide

Date/Time: Saturday, August 4th, 2012, 1:00pm – 3:00pm
Location: 1404 SC
Details: Closed book, closed notes exam; bring only your i-Card and a #2 pencil.
The exam is cumulative, but only approximately 30-40% of the content will overlap with the midterm.

This review sheet is provided to help you study for the exam. There is no guarantee that this is a complete list of topics, though we tried to make it as complete as possible. Anything covered in lecture or found in the course textbook may appear on the exam.

I. C Programming

1. What is POSIX?
2. What is a library function? What is a system call? What is the difference? Given an example of a pure library function and a pure system call.
3. How does pointer arithmetic work?
4. What is the * operator? What does it do? What is the & operator? What does it do?
5. What is a function pointer? How do you define a function pointer? What functions have you learned about in CS 241 that take a function pointer as a parameter?
6. What is a "C string"? How is a "C string" represented in memory?
7. What is NULL?
8. What is the difference between `strlen()` and `sizeof()`?
9. What's the difference between a stack and a heap variable? What about global and static variables?
10. How do `malloc()` and `free()` work?
11. What's the difference between `char c[80]` and `char *c`? …what about when they're used in `sizeof()`?
12. What is the difference between a string and a string literal?
13. How do `strcpy()`, `strcat()`, `strncpy()`, and `strncat()` work?
14. How do `printf()` and `scanf()` work? What are the common formatting arguments?
15. How do you read a series of lines from a file or `stdin` using `fgets()`?

II. Memory

1. What is the difference between physical and virtual memory?
2. What are common memory allocation algorithms and what are the advantages of each?
3. How are virtual addresses translated to physical addresses in multi-level page tables?
4. How do page size and the number of levels of page tables affect the number of entries in a page table?
5. What is the difference between internal and external fragmentation?
6. What are the different page replacement policies and the advantages of each?
7. Describe how the buddy system works and the run time for its operations.
8. What causes a SEGFAULT and what happens when one occurs?
9. When is a process swapped out to disk?
10. Name three benefits of virtual memory (as opposed to allowing programs to directly access physical memory).
11. Name one advantage of segmentation over paging, and one advantage of paging over segmentation.
12. How is a page table similar to an inode? What is the difference between these structures?

13. Assuming a 32-bit address space and 4 KB pages, what is the virtual page # and offset for virtual address 0xd34f6a5?
14. Give an example of a page fault that is an error, and an example of a page fault that is not an error.
15. Assume LRU page eviction and three pages of physical memory. Describe what happens when the application accesses virtual memory pages in this sequence: 3,4,5,4,1,6,9,3,9,8,4,8,8,2.
16. How many page faults occur in the above example?
17. Why are pages set to read-only in the copy-on-write technique?
18. Suppose we have a 64-bit address space and 16 KB pages. How big is the page table of a single process, if the system uses single-level page tables? What is the problem here? How would multi-level page tables help solve this problem?
19. Which page replacement scheme is better, OPT or LRU? Why?
20. How does the virtual memory subsystem know the exact location where a particular page is stored on disk, if it is swapped out of memory?
21. Compare and contrast (give one benefit and one disadvantage) for: implicit, explicit, segregated, and buddy free lists.

III. Processes and Threads

1. What resources are shared between threads of the same process?
2. Invent some code using `pthread_create()` statements. What could be its output?
3. What are the possible values for X after both threads complete execution? (X is a global variable and initially X = 0.)
4. What happens when a thread calls `exit()`?
5. What happens to a process's resources when it terminates normally?
6. Describe what happens when a process calls `fork()`. Be able to trace through the code.
7. Under what conditions would a process exit normally?
8. Explain the actions needed to perform a process context switch.
9. Explain the actions needed to perform a thread process switch.
10. What are the advantages and disadvantages of kernel-level threads over user-level threads?
11. Compare the use of `fork()` to the use of `pthread_create()`.
12. In a multiprocessor system, what system characteristics will cause other threads of the same process to block?
13. How can a process become orphaned and what does the OS do with it? What's a zombie?
14. Write a piece of code using `fork()` to create a process tree of depth n, where each process (a node in the tree) except for the "leaf" processes has exactly m child processes.
15. Describe how to use the POSIX call `wait()`.
16. Explain what happens when a process calls `exec()`.
17. Explain how thread-safe functions are used in C.
18. What are the maximum number of threads that can be run concurrently? How is this number determined?
19. If a process spawns a number of threads, in what order will these threads run?
20. Explain how to use `pthread_detach()` and `pthread_join()`.
21. Explain how a shell process can execute a different program without using `system()`.
22. Explain how one process can wait on the return value of another process.
23. Describe the transitions between running, ready and blocked in the 5 state model.
24. Understand how `pthread_exit()` differs from `exit()`.

IV. Scheduling

1. What is starvation? Which scheduling policies have the possibility of resulting in starvation?
2. Which scheduling algorithm results the smallest average wait time?
3. What scheduling algorithm has the longest average response time?
4. Define turnaround time, waiting time and response time in the context of scheduling algorithms.
5. Why do processes need to be scheduled?
6. How does bounded wait apply to scheduling?
7. Which scheduling algorithm minimizes average initial response time? Waiting time? Total response time?
8. Why is SJF/PSJF hard to implement in real systems?
9. What does it mean to preempt a process?
10. What does it mean for a scheduling algorithm to be preemptive?
11. Describe Round-Robin scheduling and its performance advantages and disadvantages.
12. Describe the First Come First Serve (FCFS) scheduling algorithm. Explain the performance advantages and disadvantages.
13. Describe the Pre-emptive and Non-preemptive SJF scheduling algorithms. Explain the performance advantages and disadvantages.
14. Describe the Preemptive Priority-based scheduling algorithm. Explain the performance advantages and disadvantages.
15. How does the length of the time quantum affect Round-Robin scheduling?
16. Which scheduling algorithms guarantee progress?
17. A process was switched from running to ready state. Describe the characteristics of the scheduling algorithm being used.
18. Which properties of scheduling algorithms affect the performance of interactive systems?

V. Synchronization

1. What is the readers-writers problem?
2. What is the producers-consumers problem?
3. What is the dining philosopher problem?
4. Recognize a correct solution to the readers-writers problem, the producers-consumers problem, and the dining philosopher. Be able to identify and explain an error in a specific implementation of any of the classic synchronization problems.
5. What happens when readers are prioritized over writers in the classic “readers writer problem”? How about if writers are prioritized over readers?
6. What is required so that deadlock and starvation do not occur in the dining philosopher’s problem? Give examples of solutions.
7. What is the difference between starvation, deadlock, race conditions and critical sections? Describe each.
8. What would happen if a system’s hardware synchronization primitive were replaced with a software function?
9. Which type of variables must be protected against concurrent readers and writers in any combination?
10. Given two threads running example code that contains a critical section, be able to identify if progress and mutual exclusion are ensured.
11. What is a RAG? What does it mean if a cycle exists in a RAG?

VI. Mutexes and Semaphores

1. Understand the common semaphore and mutex functions (`sem_wait()`, `sem_post()`, etc).

2. What is a `test_and_set()` operation?
3. How does the internal counter of a POSIX semaphore work? What does it mean if the value of the semaphore is 1?
4. How can the reader-writer problem be solved using only POSIX mutexes?
5. Using only one mutex, is it possible to create a semaphore? If so, how? If not, why?
6. Understand how to solve the producer-consumer problem using mutexes and semaphores.
7. What is a buffer overflow? What is a buffer underflow? Understand how failures in synchronization could cause buffer over and underflows.
8. What is progress?
9. What is mutual exclusion?
10. What are condition variables? Understand how they can be used in code.
11. Understand how to fix deadlocks and starvation in code involving mutexes, semaphores, and conditional variables.

VII. Processes and Deadlock

1. Define deadlock.
2. Define circular wait, mutual exclusion, hold and wait, and no preemption. How are these related to deadlock?
3. How would the implementation of a web server using threads differ from one using processes?
4. What can happen if synchronization in a multiple-threaded program is not programmed carefully?
5. Why might an operating system use a resource allocation graph?
6. What are the conditions of a deadlock? How could you guarantee that each one of these conditions can be prevented?
7. What does **`waitpid()`** do?
8. What are the approaches for solving deadlock?
9. What is the difference between Deadlock Prevention, Deadlock Detection & Recovery, and Deadlock Avoidance? What deadlock handling mechanism would you use?
10. What are the components of a resource allocation graph?
11. What problem does the Banker's Algorithm solve? Given a set of processes how would you use the Banker's Algorithm?
12. What is a safe state and how can you determine if a system is in a safe state?

VIII. IPC

1. What is the difference between a FIFO and a pipe?
2. How would you redirect standard out to a file?
3. What is the difference between a pipe, a FIFO, and an ordinary file on disk?
4. What happens when two processes read and write to a memory mapped file?
5. Explain how two processes can share memory using **`shmem`**.
6. Explain how a process can set custom signal handlers.
7. How can one process send a signal to another?
8. Describe the purpose of a POSIX signal.
9. Some signals cannot caught or ignored. Which signals are they and why shouldn't they be allowed to be caught?
10. What does "**`kill -<parameter> pid`**" do?
11. How does the function **`alarm()`** work?

IX. Networking

1. When do you use the **`close()`** system call with sockets?
2. Discuss how a multithreaded web server running on a single processor system could be optimized using the process scheduling methods discussed in class. Which do you recommend?
3. How do **`select()`** and **`poll()`** work? What problem do they solve?
4. Describe the POSIX **`accept()`** function.
5. How does HTTP work?
6. Describe the services provided by TCP.
7. How does TCP connection establishment work?
8. Describe the services provided by UDP.
9. Explain the difference between a regular and a connected UDP socket.
10. How do sockets support the client-server model?
11. Which is better, UDP or TCP? Which one would you use?
12. How does the Domain Name System (DNS) work?
13. How does DNS use caching?
14. How is DNS related to IP?

X. File systems and I/O

1. Given a description of the block size and i-node structure, what is the maximum size of a file?
2. How many i-node operations are required to fetch a file at /path/to/file?
3. What information is stored in an i-node? What information isn't?
4. What data structure best describes an i-node?
5. What are the advantages and disadvantages of an i-node based file system?
6. Given the description of an i-node file system, how many i-node accesses are required to read the entire contents of a file of a given size? How many blocks does this file consume on disk?
7. What is an advantage of a soft link over a hard link?
8. When the size of a block changes in an i-node based file system, how does this change the maximum size of a file?
9. How does the page-out process work?
10. Understand how hard-links result in different file names affecting the same i-node.
11. If an i-node based file system has a certain number of direct and single-indirect blocks, how large is the file?
12. Where does **`fstat()`** look to find the information that it returns?
13. How does a file system use caching?