CS 241 Midterm Exam Study Guide

The class midterm examination will be held in **from 7:00 to 9:00 p.m. on Tuesday, October 11th**. Room locations will be announced soon. The exam will begin and end promptly. Please arrive before 7:00 to allow everyone to settle into their seats before the test begins. No extensions will be granted to those who are late, nor will any non-emergency excuse for absence be accepted after Monday, October 3rd.

You may not consult any materials during the exam: no textbooks, no crib sheets, no calculator, etc.

The midterm will contain a set of around 20-25 multiple choice questions and 4-6 long answer questions. About half the total points on the exam will be for each type of problem. Each long answer may consist of multiple parts and will carry approximately equal weight overall, but may break down unevenly amongst the parts. These questions may ask you to reason about or write some code or to determine the behavior of an algorithm. On the long answer questions on the midterm, you must show all work and reasoning, writing both work and solution legibly, and should box all answers. If the course staff cannot read a solution, no credit will be given. ***All questions on the midterm will all be based on the questions on this study guide, in Homework 1, or based on MPs 1-4.***

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 does `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 does `strcpy()`, `strcat()`, `strncpy()`, and `strncat()` work?
14. How does `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. Processes and Threads

1. What resources are shared between threads of the same process?
2. What would be the output of code using `pthread_create()` statements?
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 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. Explain how a process can become orphaned and what the OS does with it. How about zombies?
14. Write a piece of code using `fork()` to create a proc tree of depth n, where each node has exactly m children.
15. Describe how to use the POSIX call `wait()`.
16. Explain what happens when a process calls `exec()`.
17. Explain how re-entrant 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()`.

III. Scheduling

1. What is starvation? Which 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. What are the four necessary conditions for deadlock?
6. Define circular wait, mutual exclusion, hold and wait, and no preemption.
7. What is the convoy effect?
8. Why do processes need to be scheduled?
9. How does bounded wait apply to scheduling?
10. Which scheduling algorithm minimizes average initial response time? Waiting time? Total response time?
11. Why is SJF/PSJF hard to implement in real systems?
12. What does it mean to preempt a process?
13. What does it mean for a scheduling algorithm to be preemptive?
14. Describe the Round-Robin scheduling algorithm. Explain the performance advantages and disadvantages.
15. Describe the First Come First Serve (FCFS) scheduling algorithm. Explain the performance advantages and disadvantages.
16. Describe the Pre-emptive and Non-preemptive SJF scheduling algorithms. Explain the performance advantages and disadvantages.
17. Describe the Preemptive Priority-based scheduling algorithm. Explain the performance advantages and disadvantages.

18. How does the length of the time quantum affect Round-Robin scheduling?
19. Define fairness in terms of scheduling algorithms. What are the fairness properties of each of the scheduling disciplines discussed in class?
20. Which scheduling algorithms guarantee progress?
21. A process was switched from running to ready state. Describe the characteristics of the scheduling algorithm being used.
22. Which properties of scheduling algorithms affect the performance of interactive systems?

IV. 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 a reader or writer are prioritized over the other in the classic "readers writer problem"?
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 is ensured.

V. Mutexes and Semaphores

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