Review all assigned readings from the textbook. This exam covers the following material from SGG: 2.1, 2.2.1, 2.3, 2.4, 2.5, 2.6, 2.7, 3.1, 3.2, 3.3, 3.4, 3.6.1, 3.6.3, 3.7, 4.1, 4.2, 4.3 (except for 4.3.2 and 4.3.3), 4.4, 5.1, 5.2, 5.3, 5.8, 6.1, 6.2, 6.3, 6.4, 6.5, 6.6, 6.7, 6.10.

Go through labs, quizzes, and project assignments. Make sure that you have a solid understanding of the topics they address. The concepts in C systems programming covered in labs and project are important for this exam.

This document doesn’t mean to give an exhaustive coverage of what might appear in the exam, but it will be useful as a self-check list for your preparation.

1. Identify the following concepts.
   - operating system
   - multiprogramming
   - time sharing/multitasking
   - process
   - thread
   - PCB
   - context switch
   - thread-safe library calls
   - scheduling
   - dual mode operation
   - mode bit
   - memory protection
   - starvation
   - CPU burst
   - I/O burst
   - race condition
   - critical region (critical section)
   - mutual exclusion
   - atomicity
   - busy waiting
   - test-and-set (hardware solution)
• swap (hardware solution)
• semaphore (counting semaphore; binary semaphore/mutex)
• inter-process communication (IPC) with pipes, files, and sockets
• message passing
• shared memory
• deadlock

2. Describe the “big picture” purpose of an operating system.

3. In what regards the design decisions that drove the creation of UNIX system calls and library functions:
   • What motivates the “need” for these two levels of service?

4. Explain how each of the following scheduling algorithms works. Be able to discuss how each one affects CPU utilization, job throughput, turnaround time, waiting time, and response time. Discuss how preemption might apply to each of the algorithms and what effects it might produce on the performance of the system.
   • first come, first served
   • shortest job first
   • shortest remaining time first
   • round robin
   • priority
   • multilevel queue

5. How does the use of preemption and priority levels affect the behavior and the performance of the scheduling algorithms above?

6. Given a set of process with specified arrival times and CPU burst lengths, for one or more scheduling algorithms: draw a Gantt chart of the execution of the processes, calculate performance metrics of the scheduling algorithm, apply the performance metrics to reason on how well the scheduler works and propose alternative scheduling for the given scenario.

7. Given an algorithm to solve the critical section problem, identify whether it meets the three fundamental requirements of mutual exclusion, progress, and bounded waiting. (Be ready to explain the concept behind each of these requirements.)
8. Explain how multiprogramming might improve the performance of an operating system.

9. Construct a scenario in which two process that share a common memory location might run into execution problems (hint: race conditions).

10. What is the difference between user mode and kernel mode? How does the system keep track of which mode it's operating in?

11. What information about processes does the operating system need to store in order to implement multiprogramming? Where is this information stored?

12. What are the different types of channels for interprocess communication and how do their operational characteristics affect how they are used by the programmer?

13. What are the steps an operating system takes to handle an interrupt (or exception in general)?

14. List the states that a process may be in. Draw a diagram showing the possible transitions between theses states and identifying what causes state transitions.

15. From the perspective of the operating system, explain what is involved in the creation of a UNIX process (that is, what data structures are used, what is the sequence of operations in the creation process).

16. Contrast UNIX process and POSIX Pthread threads. Discuss similarities and differences. Identify how a programmer might choose between the two mechanisms to implement concurrent programs.

17. Consider the following system calls: fork(), exec(), exit(). Identify their functionality. Describe the effect of one or more of them in the execution of a multi-threaded program.

18. Explain how the pipe() system call works. Show how a programmer creates pipes to enable the communication between two processes.
19. Given the man page of a given system call or library function, identify what it does, what parameters it receives, what return values it produces, and how it indicates errors to the programmer.

20. Contrast the service models of UNIX files, pipes, and TCP sockets.

21. Define short-term and long-term process scheduling. Are there differences in the types of algorithms we can use for these? Why?

22. What are the various criteria we use to study the performance of a CPU scheduling algorithm?

23. What happens during a context switch? Why can frequent context switches be a problem?

24. Distinguish threads from processes. What benefits does using threads provide?

25. Where and how in the operating system can thread scheduling be done? List an advantage and a disadvantage for managing threads in these OS implementation spaces.

26. How can one estimate the length of CPU bursts so that this information can be used in scheduling of processes and/or threads?

27. What are the classic process/thread synchronization problems we discussed in lecture and lab? Describe briefly the differences among those problems and be prepared to outline solutions for them.