# CSCI 315 Operating Systems Design
# Midterm Exam 1 Study Guide

---

- Review all assigned readings from the textbook (chapter reading guides SGG 1, 2, 3, 4, 5, 6, and 7).
- Go through labs, quizzes, and in-class activities. Make sure that you have a solid understanding of the topics they address. The *concepts* in C systems programming from lab are important to review.
- 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
    - starvation
    - CPU burst
    - race condition
    - critical region (critical section)
    - mutual exclusion
    - atomicity
    - busy waiting
    - TestAndSet (hardware solution)
    - CompareAndSwap (hardware solution)
    - semaphore (counting semaphore; binary semaphore/mutex)
    - inter-process communication (IPC) with shared memory (in threads), pipes, files, and sockets
    - dispatching

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. Identify and solve a critical section problem using the synchronization tools studied in class. 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.)

5. Explain how multiprogramming might improve the performance of an operating system and how there are limitations to what it can achieve (Amdahl’s Law).

6. Construct a scenario in which two process that use a shared variable or file might run into execution problems (*hint:* race conditions).

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

8. What are the different kinds of interprocess communication and how do their operational characteristics affect how they are used by the programmer?

9. What are the steps an operating system takes to handle an interrupt (or exception in general)? What is(are) the data structure(s) involved?

10. 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.

11. 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).

12. 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.

13. Consider the following system calls: **fork()**, **exec()**, **exit()**. Explain what each of these calls accomplish. Describe how one or more of them may affect the execution of a *multi-threaded* program.

14. Explain how the **pipe()** system call works. Show how a programmer creates pipes to enable the communication between two processes.

15. Given the manual 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.

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

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

18. Distinguish threads from processes. What benefits does using threads provide? Why might one want to use a thread pool in the design of an application?

19. 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.

20. Compare threading models such as one-to-one, many-to-one, and one-to-one identifying the advantages and disadvantages of each one.

21. Study the classic process/thread synchronization problems we discussed in lecture and lab. Explain the reason for the use of each mutex or semaphore. Explain the behavior of a solution if one makes a mistake in the initialization of a mutex or semaphore.

22. Describe the difference between *scheduler* and *dispatcher.*

23. Given information about a collection of processes (arrival time, length of CPU burst, and priority), calculate the average wait time and the average turnaround time for these processes when they are scheduled with one of the following algorithms: FCFS, SJT, SRTF, and RR.

24. Explain why the *shortest job first* algorithm is not practical and suggest what one might do in order to implement an approximation.

25. How can one change a scheduling algorithm which can lead processes to suffer from *starvation*?