Midterm Exam Study Guide

CS 415 Fall 2025

Note: The midterm is on November 4, 2025

In class, in person, on paper, closed book/notes

How to use this study guide? (We will also review these questions on Thursday.)

1. For **[Concept]** items, you should prepare for the exam by trying to find answers from the lecture slides and the textbook (Hint: AI is a good source too!). You might expect the following question types in the exam:
    a. Asking you to use a few sentences to explain/compare some concepts
    b. Asking you to draw a diagram to illustrate a concept using a simple example
    c. True/False questions
    d. Multiple-choice questions
2. For **[Algorithms]** items, you should get familiar with related algorithms. We will provide a brief explanation of the algorithms on the exam paper to help you, but it would be easier for you to become familiar with them in advance. The question type would be giving input and an algorithm, and we ask you to show the result.
3. For coding-related questions **[Reading/Writing Code]**, you should become familiar with the relevant functions. What do they do? We will provide a simplified version of the manual for those functions, but it would be easier for you to know them in advance. The code provided with the lectures is a good resource for learning about those functions. You will not be asked to fully master the functionality of those functions; we will only need you to know their basic usage and behavior.
4. For **[Reading Code]** items, you won't be asked to write any code for these types of questions, but you need to be able to run the provided code in your mind and write a possible output.
5. For **[Writing Code]** items, you will be asked to write code to complete given programs (e.g., filling in lines or blanks) to complete the program.

Lecture 1: Introduction to Operating Systems

- **[Concept]** What is an Operating System?
- **[Concept]** What is the role of an OS? What happens if we want to run a program but don't have an OS?
- **[Concept]** Where does OS fit in a computer software/hardware system structure?
- **[Concept]** What is the goal of an OS? What does OS do from the user and system perspectives?
- **[Concept]** How does the OS interact with computer hardware? For example, how does the OS work with an I/O device such as a disk?
- **[Concept]** What can cause an interrupt? How does the CPU/OS handle it?
- **[Concept]** What is a trap? Who can cause a trap?
- **[Concept]** What is the purpose of a timer interrupt?
- **[Concept]** What are the differences between multiprogramming and timesharing?

Lecture 2: Operating System Structure

- **[Concept]** What are the things typically managed by the OS? What services does OS provide for both the users and for efficient system operations?
- **[Concept]** Can OS work on its own in a purely software manner? If so, what would be the drawback of that?
- **[Concept]** What are the examples of hardware and OS working together to provide better service and management for the users?
- **[Concept]** What is the benefit of having a DMA in the system? What happens when a system does not have a DMA?
- **[Concept]** What is concurrency? What are the differences between logical and physical concurrency? Does having one type of concurrency in the system also indicate that the system has another type of concurrency?
- **[Concept]** What is the benefit of having logical or physical concurrency?

Lecture 3: System Calls

- **[Concept]** What are system calls? Where do they fit in the layers of OS designs?
- **[Concept]** Can you use an example to explain how a user application uses system calls to read files from a disk? Can you draw a diagram to show what happens?
- **[Concept]** What are the differences between system call (wrapper) and standard C library functions?
- **[Concept]** Can you list a few tasks that system calls can do?
- **[Concept]** What are system programs?
- **[Concept]** What is a file descriptor? In what scenarios are they used?

- **[Concept]** What are the differences between OS policy and mechanism? Examples?

Lecture 4: Processes

- Process concept and management
    - **[Concept]** What is the difference between a program vs. a process?
    - **[Concept]** What makes up a process? Explain what CPU state, process address space, and PCB are? What is the purpose of each of them?
    - **[Concept/Reading Code]** If a program is provided, can you draw a diagram showing what the process address space might be like at a given moment of execution? What roughly might be in the stack and the heap?
    - **[Concept/Reading Code]** If a program in assembly code is provided, can you show how much CPU state needs to be saved to the PCB when the OS decides to perform a context switch after executing a specific instruction?
    - **[Concept]** What might a process state be? How/Why do they convert between?
    - **[Concept]** How does context switching work?
- System calls/POSIX APIs for processes
    - **[Concept/Reading Code]** What happens after a process calls fork()?
    - **[Concept/Reading Code]** What is the purpose of wait/waitpid? Example scenarios where calling them is necessary? What happens if a parent does not call wait?
    - **[Concept/Reading Code]** What happens to a process after calling a function in the exec() family?
    - **[Concept/Reading Code]** What does vfork() do? How is it different than fork()? What should you do/not do after calling vfork()
    - **[Reading Code]** Given a program that calls fork/vfork/wait(pid)/exec(family), can you show the possible output of the program?
    - **[Writing Code]** If a requirement is provided, can you use the right fork/vfork/wait(pid)/exec(family) function to complete the program?

Lecture 5: Inter-process Communication (IPC)

- IPC in general
    - **[Concept]** What is IPC? Why do processes need them? Benefit? What would be the drawback of a system if it does not provide any IPC mechanisms?
    - **[Concept]** What are the two fundamental methods of IPC? How do they work? How is the kernel involved?

    - **[Concept]** What are the main properties (three dimensions) of an IPC implementation? What should we think about? Can you show examples of when we should use what?
- Shared Memory
    - **[Concept]** How does shared memory work? When is it better to use shared memory as IPC? Can you compare named and anonymous shared memory?
    - **[Concept]** What are the differences between POSIX shared memory and pipes?
    - **[Concept]** How does message passing work? How is it different than shared memory? When is it better to use message passing over shared memory?
- Message Passing
    - **[Concept]** How does POSIX message queue work?
    - **[Concept]** How are file descriptors used in IPC such as shared memory, message queue, or socket?
    - **[Concept]** How do remote procedure calls work?
- Code with POSIX IPC
    - **[Reading Code]** Given a program that uses POSIX shared memory, pipes, or POSIX message queue, can you show the possible output of the program?
    - **[Writing Code]** If a requirement is provided, can you use the right IPC function (POSIX shared memory, pipes, or POSIX message queue) to complete the program?

<u>Lecture 6: Threads</u>

- From processes to threads
    - **[Concept]** Why are processes considered heavyweight in terms of creation and context switch?
    - **[Concept]** How is execution state separated from resources? How are threads live inside a process?
    - **[Concept]** What is generally shared between threads in a process? What is private to each thread?
    - **[Concept]** Compare processes with threads. When is a multithreaded approach a better choice than multiprocessing? When is it not?
    - **[Concept]** If given requirements, can you choose the most suitable concurrency model (threads vs. processes)?
    - **[Concept]** Can different threads run different programs like child processes? What happens when a thread calls exec() to load a new program?
- Kernel threads

    - **[Concept]** What are kernel threads? How does it relate to user threads? Why are kernel threads needed?
    - **[Concept]** Does the OS scheduler directly manage user threads or kernel threads? Which type of thread is using OS resources, such as needing the OS to do bookkeeping?
- Thread models
    - **[Concept]** What happens when there are more user threads than kernel threads?
    - **[Concept]** What ways can user threads be mapped to kernel threads?
    - **[Concept]** How does each thread model work? Scheduling?
    - **[Concept]** What are the major benefits/drawbacks of each thread model?
    - **[Concept]** Which thread model does Linux use?
- Code with POSIX thread (pthread)
    - **[Reading Code]** Given a program that uses POSIX threads, can you show the possible output of the program?
    - **[Writing Code]** If a requirement is provided, can you use the right (POSIX threads APIs to complete the program?

Lecture 7: Scheduling

- **[Concept]** What could be the goal of resource scheduling?
- **[Concept]** When does the OS need to make scheduling decisions?
- **[Concept]** What are the benefits/drawbacks of non-preemptive and preemptive algorithms?
- **[Algorithm]** If given processes with their arrival time, burst duration/priority, can you show the scheduling results using FCFS, SJF, SRTF, Priority Scheduling (non-preemptive or preemptive), and Round Robin?
- **[Concept]** Can you calculate the average waiting time for all processes?
- **[Concept]** Can you calculate the turnaround time and waiting time for the individual process?