Final Exam Study Guide

CS 415 Fall 2025

Note: The exam time is 12:30-14:30 December 11, 2025

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

How to use this study guide? (We will also review these questions on Monday's OH.)

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. Asking you to show the solution to a problem if a technique/policy/strategy is applied
    d. True/False questions
    e. 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 final result and/or step by step results.
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.

<u>Lecture 1: Introduction to Operating Systems</u>

- **[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?

<u>Lecture 2: Operating System Structure</u>

- **[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?

<u>Lecture 3: System Calls</u>

- **[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?

Lecture 6: Threads

- 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?
- 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?
- **[Concept]** How can different scheduling algorithms combined together? Can you give an example?
- **[Concept]** How to design a scheduling system that is optimized for both interactive and batch jobs? Can you explain why it is designed this way?
- **[Concept]** What is a light-weight process?
- **[Concept]** In what case scheduling will be needed in user space thread library? In what case scheduling is necessary between kernel threads?

Lecture 8: Synchronization

- Race condition and critical sections
    - **[Concept]** What are race conditions? Be able to explain, using a small shared-variable example (like a shared queue index), how instruction interleavings can lead to incorrect results even if each individual instruction is correct.
    - **[Concept]** What does it mean for a sequence of instructions to be atomic? What are the limits of atomicity at the hardware level (e.g., single variable vs. a whole function)?
    - **[Concept]** What is a critical section? What are the three requirements for a correct solution to critical-section problem?
    - **[Reading Code]** Can you identify the critical section in a small code snippet and the shared data that should be protected?
- Critical section problem and solutions
    - **[Reading Code]** Given one solution shown in the lecture, can you identify whether it meets all three requirements to be the correct solution for a critical section problem? If all not requirement are met, can you explain which one does not and why?
    - **[Algorithms]** How does Peterson's algorithm use flag[i] and turn to enforce mutual exclusion, progress, and bounded waiting for 2 processes?
    - **[Concept]** Why can't we just make the whole critical section atomic as a single CPU instruction? Why is that impractical?
    - **[Concept]** What is busy-waiting (spinning)? Why can it waste CPU time, especially under round-robin scheduling or long spin periods?

  - **[Concept]** What is a blocking solution to critical sections? Can you explain it at a high level?
  - **[Concept]** Compare spinning vs. blocking: when might spinning be acceptable vs. when blocking is more efficient
  - **[Reading Code]** Given code that uses test_and_set or compare_and_swap to implement a spinlock, can you trace a few steps to show how mutual exclusion is achieved?
  - **[Concept]** What is a mutex lock? How does acquire() / release() around a critical section enforce mutual exclusion?
  - **[Concept]** What is a semaphore? What is the difference between binary semaphores and counting semaphores?
  - **[Writing Code]** Given a small multi-threaded task, can you insert the correct POSIX synchronization functions to solve the critical section problem?
  - **[Writing Code]** Can you design a shared buffer using semaphore?

<u>Lecture 9: Deadlocks</u>

- Deadlock problem and properties
  - **[Concept]** Understand the system model: resource types R1, R2, ..., Rm, each with some number of instances Wi (e.g., printer queues, CPU cores, files, shared queues). Be able to recognize examples of resource types and instances.
  - **[Concept]** Know the basic resource usage pattern: Request → Use → Release. Be able to explain what it means for a process (or thread) to be blocked waiting for a resource.
  - **[Concept]** Be able to define a deadlock in your own words: a set of processes where each is blocked waiting for resources held by others in the set, so no one can make progress.
  - **[Concept]** Know the four necessary conditions for deadlock: mutual exclusion, hold-and-wait, no preemption, and circular wait. Be able to restate each condition and give a small example scenario for each.
- Handling deadlocks
  - **[Concept]** Given a set of resource and a set of thread/process with resource requests/owning relationships, can you draw a resource-allocation graph (RAG)?
  - **[Concept]** Know the basic facts: if the RAG has no cycle, there is no deadlock. If there is a cycle and each resource type has only one instance, then there is a deadlock. With multiple instances per type, a cycle means a possible deadlock but not necessarily one.
  - **[Concept]** Given a small RAG (or equivalent diagram) showing which processes hold and request which resources, be able to determine whether the system is deadlocked, or whether it might still be able to proceed.
  - **[Concept]** Know the four general strategies for handling deadlocks: deadlock prevention, deadlock avoidance, deadlock detection and recovery,

    and "do nothing" (ignore the problem). Be able to explain the high-level idea of each.
  - **[Concept]** Be able to define a safe state and a safe sequence: in a safe state, there exists some order of finishing the processes such that each can obtain the resources it needs in turn and complete without deadlock.
  - **[Concept]** Understand the difference between safe, unsafe, and deadlocked states.
  - **[Algorithms]** Understand the Resource-Allocation Graph (RAG) Algorithm. When we can use this algorithm? What are claim edges?
  - **[Algorithms]** Know the idea of the Resource-Allocation Graph (RAG) Algorithm. Given a RAG with claim edges and a request, can you use this algorithm to identify if it is safe to grant the request or not?
  - **[Algorithms]** For Banker's algorithm, Understand the purpose of the Safety Algorithm and Resource-Request Algorithm.
  - **[Algorithms]** For Banker's algorithm, what matrices are needed to run the two algorithms?
  - **[Algorithms]** For a small example like the one in the slides, can you run the Safety Algorithm by hand and identify a safe sequence? Can you show this step by step?
  - **[Algorithms]** In addition, given a resource question vector, can you run the Resource-Request Algorithm to determine whether the request can be immediately granted or must be delayed because it would lead to an unsafe state? Can you show this step by step?

<u>Lecture 10: Memory</u>

- General memory management
  - **[Concept]** How does OS present each process a logical (virtual) address space (code, data, heap, stack) that is a contiguous (range 0..max) even though physical memory is shared among processes?
  - **[Concept]** What is address binding? Why it is needed?
  - **[Concept]** Compare compile-time, load-time, and execution-time (run-time) binding, what kind of code is generated for each stage along the path of compile → link → load → run pipeline?
  - **[Concept]** If a program's page needs to be moved at runtime due to swap in/out, what type of binding is most suitable?
- Contiguous memory allocation
  - **[Concept]** What are contiguous memory management and non-contiguous management? Can you give an example layout of physical memory for both cases.
  - **[Concept]** When using contiguous memory management, explain how logical addresses are translated to physical addresses? How is this done for

    - multiple processes? What happens at context switch? What is role of MMU? Why it must be done by hardware?
    - **[Concept]** What are base and limit registers in contiguous memory management?
    - **[Concept]** Given a logical address, base, and limit, can you compute the physical address or determine that an addressing error (trap) occurs?
    - **[Concept]** What are the benefits and drawbacks of contiguous memory management?
    - **[Concept]** What is external fragmentation? Can you show an example?
    - **[Concept]** What is swapping? What is benefit and cost? Which type of scheduler works best with swapping?
- Non-contiguous memory allocation
    - **[Concept]** What are frames and pages? Do they have to have the same size?
    - **[Concept]** What is a page table? Can you use a given page table to translate a logical address to physical address?
    - **[Concept]** Given multiple processes with different number of page requirement and a physical memory, can you assign frames to processes and design page tables for each one of them?
    - **[Concept]** What is internal fragmentation? Can you show an example?
    - **[Concept]** Where are page table stored? What are the Page-Table Base Register (PTBR) and Page-Table Length Register (PTLR)? What happens at context switch?
    - **[Concept]** Given logical address size, page size, and number of pages, can you compute the number of page-table entries and approximate memory needed to store a page table?
    - **[Concept]** What is TLB? What is the benefit of having TLB?
    - **[Concept]** Can you compare a memory access with and without TLB?
    - **[Concept]** What is ASID? What is the benefit of ASID? What happens at context switch with and without ASID?
    - **[Concept]** Can you use the EAT formula, given memory access time M, TLB lookup fraction e, and hit ratio a, to compute the effective access time?
    - **[Concept]** Can you explain the idea of swapping pages? What is it benefit? How the valid/invalid bit help in swapping?
    - **[Concept]** Can you outline the steps the OS takes on a page fault?
    - **[Concept]** What is shared pages? What is the benefit? When can we use it?
    - **[Concept]** Why the page table has to represent all addresses even though only part of the memory space is used? What techniques can help reduce the size taken by page table?
    - **[Concept]** What is inverted page table? Benefit and drawbacks?

<u>Lecture 12: File System</u>

- **[Concept]** Why file management an OS problem?
- **[Concept]** Understand the separation between file system interface and implementation: high-level operations.

- **[Concept]** What is a file? Does it exist in physical disk?
- **[Concept]** What is file structure? Who define it?
- **[Concept]** What is FCB?
- **[Concept]** What are file attributes?
- **[Concept]** When happens when we call open()? How is FCB used? How does system-wide and per-process open-file table work together. When is returned from open()?
- **[Concept]** What extra information need to be stored in system-wide and per-process open-file table?
- **[Concept]** What is sequential access and direct (random) access? Can you use one to simulate another?
- **[Concept]** How a directory link with files? How does directory itself store in filesystem? Does directory have FCB too?
- **[Concept]** If a directory hierarchy is given, can you show how those can be stored in a filesystem?
- **[Concept]** If a series of FCB and data blocks are given, can you show the directory hierarchy that they represent?
- **[Concept]** What are the differences between a flat (single-level) directory organization from a hierarchical tree-structured directories with subdirectories?
- **[Concept]** What are soft/hard symbolic links in hierarchical directories? What is the benefit and potential issues can cause by links?
- **[Concept]** How filesystems typically enable file sharing?
- **[Concept]** How filesystems typically enable access permission control?
- **[Concept]** Given a small access matrix, be able to determine whether a subject has a particular right over an object and reason about secrecy and integrity of example resources (e.g., private keys vs. public keys).

Lecture 13: File System Implementation

- **[Concept]** What is a block and what is a sector?
- **[Concept]** How is file system designed in layer? What does each layer do?
- **[Concept]** What is the in-memory filesystem structure? What does it do?
- **[Concept]** How can a directory be implemented? How to organize the pointer to FCBs?
- **[Concept]** How to allocate data blocks in disks? How does main allocation strategies works? contiguous allocation, extent-based allocation, linked allocation (per-block pointers), linked allocation with a File Allocation Table (FAT), indexed allocation, and the combined indexed + direct pointer approach in Unix inodes.
- **[Concept]** What are the trade-offs between each allocation strategies? Access pattern, storage overhead, (external/internal) fragmentation?
- **[Concept]** Given a file allocation strategy, can you count the number of disk accesses needed for fetching a byte of data?

- **[Concept]** Given a file allocation strategy and N data blocks that need to be stored, can you show how potential those blocks can be stored in disk?
- **[Concept]** How are free space managed by the filesystem/OS?