> *The fourth exam will given in the registrar-assigned timeslot on Monday, May 11.* **Take care of any necessary business before class** *so that you do not need to leave the room during the exam.*
>
> *Note that only your dean can reschedule this exam. If you have three exams on the same day, you may work with the instructor and your dean to reschedule one of them. To reschedule this exam, you must be in touch with me* **by the end of the day Wednesday 5/6**.
>
> *The exam covers material from OSTEP chapters 30–32, 36–37, 40–43, and 45 (condition variables, semaphores, and concurrency bugs; I/O, disks, and disk scheduling; and file systems). It is designed to be similar to the midterm exams in length (approximately an hour), though you will have the full three hour time period to work on it if you wish.*
>
> *The exam is closed book. You may have a single page of notes (8.5x11", one side) which will be handed in with your exam. This page may be handwritten or typed and can contain whatever you would like, but it must be a hardcopy — on a piece of paper, not a laptop, tablet, phone, or other device — and must be personally prepared by you — you may not copy another student's page or hand out copies of yours to others. Creating your own notes is an essential part of the learning process — deciding what to include requires engagement with the material which reinforces understanding and improves long-term retention of the material, provides an opportunity for review in order to identify gaps in your knowledge in time to ask questions before the exam, increases confidence in what you do know, and encourages taking ownership of your own learning.*

You should be familiar with the material covered in class and in OSTEP chapters 30–32, 36–37, 40–43, and 45. Key terms and ideas include:

- condition variables
  - what a condition variable is, why locks alone are insufficient
  - what condition variables are used for
  - core operations (`wait`, `signal`, `broadcast`) and their semantics
  - canonical usage pattern (lock+while+wait), why `while` not `if`)
  - classic patterns (e.g. producer-consumer)
- semaphores
  - what a semaphore is
  - core operations (`wait`, `post`) and their semantics
  - role of the initial value (0, 1, N)
  - binary vs counting semaphores (what, when to use each)
  - what semaphores are used for
  - classic patterns (e.g. producer-consumer, thread ordering)

- concurrency bugs
  - types of bugs: atomicity violations, order violations
  - deadlock: conditions for deadlock, prevention strategies
  - terms: livelock, starvation
- I/O devices and hard drives
  - components of an I/O device
  - ways the OS interacts with devices, tradeoffs
  - device abstraction and role of device drivers
  - performance concepts (latency vs throughput, polling vs interrupts)
  - hard disks: physical structure, components of access time, sequential vs random access
  - disk scheduling: goals, specific algorithms (FCFS, SSTF, SCAN, C-SCAN) and tradeoffs
  - logical (OS) view vs physical disk layout
- files and directories
  - file and directory abstractions, basic operations
  - key abstractions: inode, file descriptor, open file table
  - file offset
  - naming: hierarchical directory structure, absolute vs relative paths, `.` and `..`
  - links: hard links vs symbolic links, reference counting and deletion semantics
- file system implementation
  - disk layout: superblock, inode table, data blocks, bitmaps
  - inode structure and pointer types (direct, indirect, double indirect, etc)
  - purpose of indirection, why the imbalanced tree structure
  - directory implementation
  - key processes: path traversal, file access
  - performance: goals, techniques
  - FFS
    - motivation
    - key idea
    - implementation: disk layout, placement policies, large files
- crash consistency
  - the problem (what can go wrong and why)
  - types of inconsistencies
  - approaches: fsck, journaling (write-ahead logging)
    - goal and key idea
    - how it works
    - advantages, limitations, tradeoffs

- data integrity and protection
  - goal
  - sources of errors
  - terms: latent errors
  - error detection vs correction
  - checksums: what is a checksum, how are checksums used, implementation, overhead, limitations
  - correcting errors: techniques

Specific implementation or computation things you should be able to do include:

- concurrency
  - identify potential concurrency problems in C code and explain what the problem is
  - fix/prevent concurrency problems using locks, condition variables, and/or semaphores (at the conceptual level e.g. when to wait or signal; you don't need to know the specific `pthreads` API)

You should also have done homeworks 9–11 and be familiar with the concepts explored, the inputs analyzed, and the output of specific tools run — you won't be asked about specific command lines or to do the disk access and other computations done by the simulators, but you may be given similar inputs or outputs and/or be asked about concepts from those problems.