Exam review

- shell, fork, and exec
- Posix system calls
- kernel structure: monolithic, microkernel, layered
- processes, threads, scheduling, context switch
- virtual machines
- IPC, races, locks, deadlocks, monitors, pipes, and message passing
- Minix
- seL4

shell, fork, and exec

- a shell runs as user code
- using Posix API to execute user programs
- shell scripts evolved into scripting languages
- fork: an exact copy of the process
  - signals only delivered to the parent
  - copy the stack and heap, not the code (unless necessary)
  - parent and child have same open file descriptors
  - changing the standard file descriptors (0, 1, 2) allows for pipes and redirects
  - wait allows (requires!) the parent to find out when the child completes, and allows background processing
- exec: the process is replaced by the execution of a different program
  - replace the text (code) segment
  - re-initialize the stack and heap
  - set the PC (IP) to the entry point of the program

system calls

- system call implementation in hardware
- system call implementation in Minix
- the three basic system calls in seL4

Posix system calls

- see list (under January 28th)
- be familiar with the main process management calls: `fork`, `exec`, `wait`, `exit`
- be familiar with the main file management calls: `open`, `creat`, `unlink`, `mknod`, `read`, `write`, `close`, `stat`, `pipe`, `mount`, `umount`, `chroot`

kernel structure

- monolithic
- microkernel
- layered
- understand the consequences of each choice
- in the case of microkernel, understand some of the choices available for how the microkernel coordinates the operation of the different components

processes and scheduling

- a process as a virtual machine
- memory management: virtual memory, isolation of one process from another
- context switches: what is needed, implementation in Minix
- Minix scheduling: round robin with priorities
- real-time scheduling: earliest deadline first (EDF), rate monotonic for periodic processes

threads and synchronization

- a process is a thread of execution together with an address space
- threads have independent execution within the same address space
- a mutex can only be acquired by one thread at a time
- a signal can wake up a waiting thread

- most threading systems are implemented by an operating system, but thread switching is not a privileged operation, and so thread systems could be implemented entirely in user space

virtual machines

- VM mechanisms: interpreter, hardware support, what still must be implemented in software
- host OS, guest OS
- protection and limitations of the protection

IPC, races, locks, deadlocks, monitors, pipes, message passing

- InterProcess Communication
- message passing, Minix message passing
- pipes
- race conditions
- locks (and semaphores)
- deadlock, starvation
- monitors

Minix

- overall architecture:
  - message-passing microkernel
  - device drivers are tasks
    - interrupts generate messages to these tasks
  - Posix system calls are implemented by specific server tasks
    - hardware system calls generate messages to these servers
- textbook: chapters 1, 2.2, 2.4, 2.5, 2.6

seL4

- overall architecture:
  - minimal kernel, not a complete OS
    - OS such as Linux can be run as a user process
  - user processes manage all resources
  - all permissions represented as capabilities
    - e.g. a capability to a certain amount of memory can be used to derive a capability to a fraction of that memory
    - the derived capability can be given as the memory for a new process
- chapters 2 and 3 of the reference manual