Exam Review

- booting
- I/O hardware, DMA, I/O software
- device drivers
- memory (i.e. address space) management
- virtual memory

booting

- hardware is configured to execute a program in Read-Only Memory (ROM) or flash memory:
  - the BIOS, basic I/O system
  - UEFI is the current equivalent
- BIOS knows how to access all the disk drives, chooses one to boot (perhaps with user assistance), loads the first sector (512 bytes) into memory, and starts to execute it (*jmp*)
  - first sector often includes a partition table

I/O hardware and DMA

- electronics, and sometimes (disks, printers) moving parts
- *status* registers and *control* registers read and set by CPU software
  - registers can directly control hardware, or be read and set by the device controller
- device controller can be instructed to do Direct Memory Access to transfer data to and from the CPU's memory
  - DMA typically uses physical addresses

Structure of I/O software

- user programs request I/O: read/write, send/recv, etc
  - daemons and servers work autonomously
- device-independent software converts the request to a device-dependent operation, and also handles requests from device drivers
  - e.g file systems and protocol stacks
  - e.g. *server*s in Minix
- one device driver may manage multiple devices
  - and handles interrupts
- buffer management required!

Device Drivers

- configure the device or device controller
  - i.e. must know specifics about the hardware
- respond to I/O requests from higher-level software: read, write, ioctl
- respond to interrupts, usually by reading status registers, writing to control registers, and transferring data (either via DMA, or by reading and writing *data registers*)

Memory Management

- linear array of memory locations
- memory is either divided into fixed-sized units (e.g. pages) or variable-sized units (segments)
- management of the two is very different!

- everything that applies to managing physical memory also applies to managing virtual memory: really, we are managing addresses

Memory Allocation for fixed-sized units

- keep a linked list of free units
  - each unit must be large enough to store a pointer to the next available unit (or NULL)
- keep a bitmap of free units
  - for units of n bits, the bitmap takes up 1/n of the memory
    - e.g. for 4K-byte units, takes up 1/32K = 2^{-15} of the space
  - useful if you might need to allocate contiguous units
- virtual memory pages and disk sectors are both fixed-sized units

Memory Allocation for variable-sized units

- keep a linked list of free units
  - each unit must be large enough to store a pointer to the next available unit (or NULL)
- keep the linked list sorted by address
- when freeing memory, insert at the correct location in the linked list
- to allocate memory, use a variety of algorithms:
  - first fit or next fit
  - best fit
  - free list of equal-sized elements of common sizes
- segments and malloc/free manage variable-sized units

Virtual Memory

- paging and segmentation
  - usually paging, but Minix uses segmentation
- offer each process a view of memory that is independent of the physical memory
  - some of the storage may be on disk
- also adds protection
  - when protection is violated or page is not mapped, leads to a *page fault* (or *segmentation fault*), which transfers control to the kernel
  - can implement Copy-on-Write, Not-Recently-Used

Paged Virtual Memory

- each virtual address is split into page number and page offset
  - if page size is a power of two. some bits of the address are the page offset, the rest page number
  - so page size is always a power of two!
- the page table indicates what frame number corresponds to the given page number
  - multi-level and inverted page tables address some of the drawbacks of single-level page tables, particularly the sparsity of address space actually used by a program
- TLB caches recent accesses to the page table

Segmentation

- each access must reference a segment register, which is added to the virtual address to give the physical address
  - addition is more time-consuming than bit replacement, but still faster than memory access
- the segment register may have more bits than the virtual address
- depending on the architecture, segment registers may be per-process, or per type of memory reference (instruction, data, stack)