Today's plan

- Minix terminal driver
- Memory management

Minix Terminal Driver

- very complex: supports memory-mapped keyboard and displays, RS-232 serial terminals, and network-based logins (Pseudo-ttys, or PTYs, on other systems)
- character devices, but with two-dimensional positioning capabilities (screens)
- screens can be character-based or pixel-based (minix only supports character-based)
- buffering can be reserved per terminal (as is the case in Minix) or shared among all terminals (central buffer pool)
- terminal driver must perform **line editing functions** to honor erase characters and possibly change newlines into CR-LF or viceversa (or other combinations)
  - modern systems support more advanced line editing

Terminal modes

- editors will do their own screen redrawing, can handle erase characters, so should be given the **raw** stream of characters the user enters
- most programs take line input and would prefer to have the operating system take care of editing: **canonical** or **cooked** mode
- special characters can control freezing (^S) or restarting (^Q) the output, mark end of file (^D), end of line, etc
- in non-canonical mode, Minix allows the specification of a minimum number of characters to read and of a timeout for terminal reads -- if either is satisfied, the read call completes

Reading from the terminal

- book, figure 3-33, p. 319

1. user process sends message to file system
2. file system sends message to TTY task, which may directly go to (6), but most likely
3. replies asking the file system to suspend the process
4. when a key is pressed, the generic interrupt handler notifies the TTY task
5. the TTY task reads the I/O ports to determine which key was pressed, and adds the character to a queue
6. at the top of the TTY task, the task copies available data directly to the user space using physical memory copying
7. the TTY task tells the file server (whenever it is ready to receive the message) that it may wake up the user process
8. the FS server wakes up the user process

- this complex technique gives acceptable performance for large bursts of characters from the serial port on slow hardware, since the user process and the file system are only involved when enough characters (or a timeout) have been received
- serial line controllers may be configured to only interrupt after receiving several characters, rather than once per character

Bitmapped Displays

- each pixel (usually 1, 8, 16, 24, or 32 bits) represents one dot on the display: one scan sweep position or one pixel on an LCD display
- more resolution requires more memory (1280x1024x24 requires 4MB for each display, without counting virtual desktop space)
- data can be arranged in various fashions in memory, but usually such that adjacent elements of a row are adjacent in memory
- basic display operations include moving a block (**bitblt**), drawing a point or a line, or filling in a rectangle (can also be done with bitblt)
- the device controller may include a fast data-parallel (SIMD) computer to operate on many bits at once -- the **GPU**
- a **window manager**, which may be part of the OS, must create windows, associate them with processes, and support opening, closing, moving, iconifying, etc
- the **X-window system** is a user-level program (an **X server**) that supports basic window operations for (possibly remote) client programs
- one of these client programs is the X window manager
- other client programs show the time, check for mail, allow for user command-line input (by running shells, perhaps remote shells), support surfing the web, etc.

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

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

Memory allocation algorithms

- first fit: first free segment that fits is split and used. Fast, fairly good with regard to fragmentation, may waste memory
- next fit: same as first fit, but start from the end of the last search, and wrap around. Almost the same performance as first fit
- best fit: search the entire list to find the smallest hole that will fit. Works great if exact matches are found (i.e. if allocations are a few different sizes), otherwise leaves unusably small holes.
- worst fit: use the biggest free block. makes it hard to ever allocate large blocks.
- quick fit: keep free lists for commonly requested sizes -- but merging after deallocation is expensive

Minix Memory Management

- segments, including swapping, but no paging
- program manager allocates memory using first fit, never moves it or changes it
- program manager keeps track of available memory and assigns it to processes (policy), kernel does the actual memory mapping (mechanism)
- program manager allocates memory for fork (copy of the parent is the child) and for exec (memory is returned, new memory is allocated)
- two modes (program-dependent):
  - combined I and D, meaning the code and data are in the same segment, or
  - shared text, meaning the code is in a separate segment that can be used by multiple processes (e.g. parents and children)
- for the second case, exec searches to see if the required text is already loaded and can be shared

Linux Memory Management

- Linux virtual memory management is described in detail at https://www.tldp.org/LDP/tlk/mm/memory.html
- Linux internal memory management is described in detail (but from 2007!) at https://www.kernel.org/doc/gorman/pdf/understand.pdf
- more current information (for sysadmins) at:
  - https://linux-mm.org/LinuxMMDocumentation
  - https://linux-mm.org/LinuxMMInternals