Discussion 1/18/19

Today

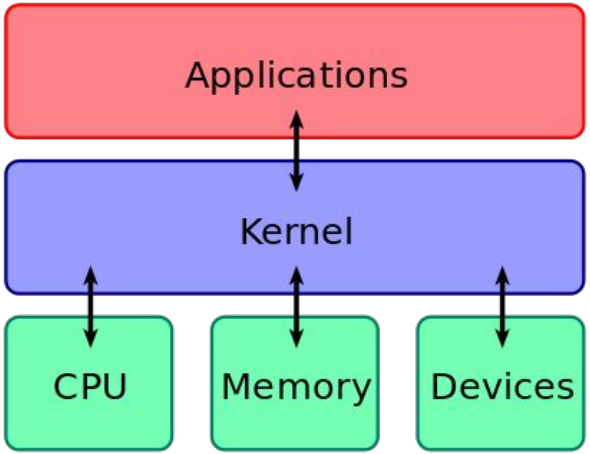
- We have to be out by 10:15 (probably won't go this long)
- Quick review of operating system / kernel
- Project 1
- Office Hours

Discussion materials will be uploaded at:

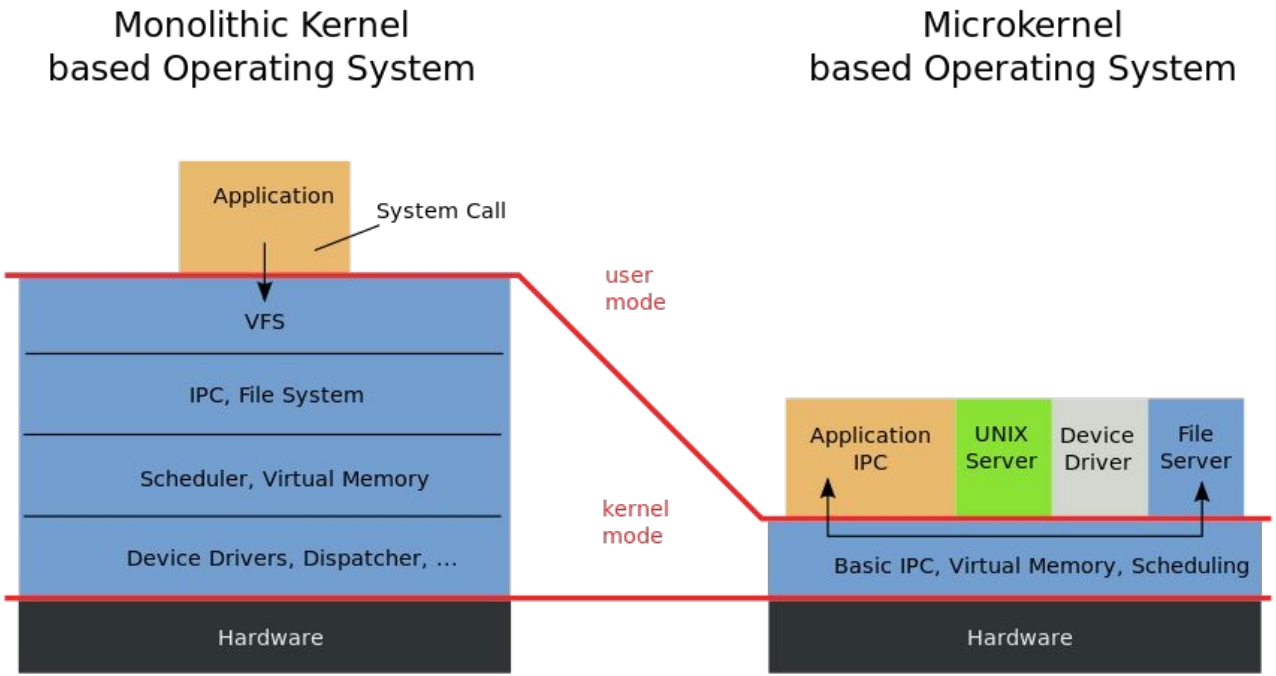
https://github.com/dpdicken/cs452-spring19-discussion

Kernel

- Kernel is the core program within an operating system
- Protects sensitive code and procedures in a protected memory space
- An application interfaces with the kernel via a system call
- System calls are interrupts, interrupts are not system calls!



Micro Kernel vs Monolithic Kernel



Project 1: Graphics Library

- In this project, you'll be writing a small graphics library that can set a pixel to a particular color, draw some basic shapes, and read keypresses.
- Will create **library.c**, **graphics.h**, and a **utilizing program**
- You will implement several library functions to allow a user to draw on the screen. Each library function must be implemented using only the Linux system calls. You may **not** use any C standard library functions in your library anywhere.
- In the version of Linux we are using, the kernel has been built to allow you direct access to the framebuffer via **/dev/fb0**
- To set a pixel on a screen, we can just modify this file.
- To modify the file, we could use **open**, **read**, **seek**, etc.
- To make your life much easier, use **mmap** instead.

Setting up environment

- Download the qemu-arm.zip file from the website and extract the files into a folder.
- Install QEMU
  - On windows, run with .bat file
  - On MAC/Linux, run with start.sh
- Run devtools.sh to install necessary development tools
- VM’s like to misbehave, consistently make copies of **disk.qcow2** to make sure you have a good reset point
  - Additionally, backup your code on lectura or a **private** github repo

C Review

- Typedef
- Macro
- Bit manipulation
- Pointer arithmetic
- Compiling multiple files

How can we use a system call in C?

- To the code!

Mmap

void *mmap(void *addr, size_t length, int prot, int flags, int fd, off_t offset);

- **addr -** The address of your memory you want to map, NULL if you don’t care.
- **length -** The amount of memory you want to map in bytes.
- **prot -** The protections you want on the memory you are going to map (R/W/X)
- **flags -** Additional arguments you want to make, for example MAP_SHARED
- **fd -** The file descriptor that represents the file you are mapping from
- **offset -** The offset to the location of memory you want to map to inside the file
- **returns -** A pointer to an “array” of memory that lazily reads and writes to the file

Iotcl

Int ioctl(int fd, unsigned long request, …);

- **fd -** File descriptor or the device (Remember: Everything is a file in unix!)
- **request -** The code for the request you want to make
- **… -** This is a pointer to the resulting struct defined by the first arguments
- **returns -** an int, usually 0, means success. Non-zero results depend on the request.

Select

- Better than **read()**. Why?

int select(int nfds, fd_set *readfds, fd_set *writefds, fd_set *exceptfds, struct timeval *timeout);

- **nfds -** Highest-numbered file descriptor in any of the three sets, plus 1
- **readfds -** A fd_set of file descriptors you want to read from
- **writefds -** An fds_set of file descriptors you want to write to
- **exceptfds -** An fd_set of files that might cause exceptional conditions
- **timeout** - How long to block and wait for something to happen, 0 to not block
- **Returns -** The number of file descriptors that are ready

Bresenham’s Algorithm

- Won't get into it here, but I recommend:
  - https://www.cs.helsinki.fi/group/goa/mallinnus/lines/bresenh.html
  - https://www.geeksforgeeks.org/bresenhams-line-generation-algorithm/
  - https://www.youtube.com/watch?v=zytBpLISHms (Starts being useful about 3 minutes in)

Double Buffering

- In computer graphics, **double buffering** is a technique for drawing graphics that shows no (or less) stutter, tearing, and other artifacts.
- Good reference: https://www.youtube.com/watch?v=7cRRxlWRl8g