CSC 452 Review 2

Today

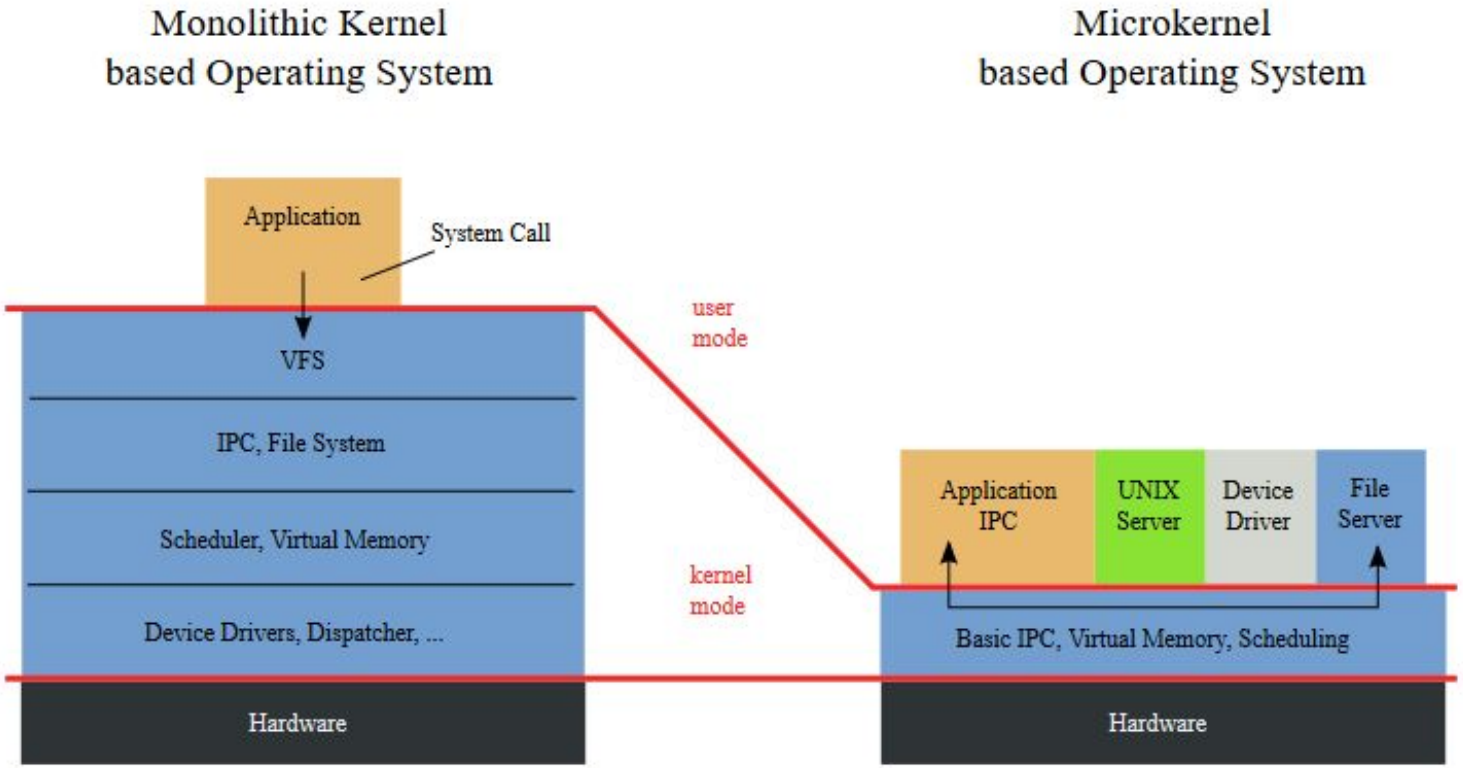
1. Logistics
2. Quick Review of Monolithic and Micro Operating Systems
3. Review Assignment 1
4. Open Office Hours

Discussion Resources

Resources such as slides will now be hosted on https://github.com/Jon-Davis/CSC452

If anything is missing feel free to email me at davisj4@email.arizona.edu

Monolithic Kernels and Microkernels



Project 1: Graphics Library

Objective: 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 key presses.

In the version of Linux we are using (Raspbian), the kernel has been built to allow you direct access to the framebuffer.

This framebuffer is exposed to us as /dev/fb0, Since it appears to be a file, it can be opened using the open() syscall.

To set a pixel, we only need to do basic file operations. For example Read/Write. However there is a better way.

Project: Initialize the Graphics Library

Without Initializing the Graphics Library we could still write to the framebuffer.

Every time we want to write a pixel we simply need to open the framebuffer, write to the frame buffer, and close the frame buffer. However we are NOT going to do this.

Instead what we are going to do is open the file once. Then will create a mapping between a portion of our virtual memory and the framebuffer.

We will also want to know how large this framebuffer is, so that we can map the right amount of memory. For this we will need to know the Width, Height, and Pixel size.

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

IOCTL

int ioctl(int fd, unsigned long request, ...);

fd - File Descriptor or the Device. In Unix Everything is a File*

request - The code for the request you want to make

… - This is a pointer to the resulting struct defined by the first two arguments

Returns - an int, usually 0 means success however non-zero results depend on the request.

Project: I/O

What we want to be able to do, in order to make interactive applications, is to accept user input. We can do this using the read system call.

But there is a problem with this approach.

Select

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 - An fd_set of file descriptors you want to read from
writefds - An fd_set of file descriptors you want to write too
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 - A the number of File Descriptors that are ready