Project One

Ryan Sheffler
Ryan.Sheffler1@Marist.edu

September 8, 2020

1 Lab One

1.1

What are the advantages and disadvantages of using the same system call interface for manipulating both files and devices?

By using both through the same interface, users are more easily able to create device drivers using the system call interface. More device drivers means that more devices are easily accessible. Later users will find it easier to manipulate the devices through the drivers and APIs, as everything the kernel manipulates is treated the same way. On the other hand, this can make it much more difficult for certain devices to access their "true potential." Using the same interface that is used for file manipulation may limit exactly what can be efficiently coded into a device driver.

1.2

Would it be possible for the user to develop a new command interpreter using the system call interface provide by the operating system? How?

The system call interface allows a user to start, manage, and end processes as well as manipulate files and how those processes interact with them, albeit indirectly. Though there is another layer between the user and the hardware (i.e., the user is interacting through another "safety net" line of code in the operating system), the user should still have the necessary amount of control over the system through the system call interface to code a new command interpreter on top of the already existing operating system.

2 Lab Two

2.1

How is your console like the ancient TTY subsystem in Unix as described on Linus Akesson's website?

Just like the primitive TTY system, our operating system uses a simple extra layer of programming to manage it. Most of this code is simply run in *console.ts*, which handles primarily the drawing of elements to the canvas area, but also keeps track of a text buffer for the current line. This is how our operating system console provides the same simple text editing options he mentions under the Line editing section. In the future, this should also allow us to administratively manage or end processes, just as mentioned in the Session management section. Our keyboard driver also is capable of operating asynchronously, passively waiting on standby for a keypress, at which point it will throw up an interrupt signal to the kernel. Our kernel also uses a very similar method of flow control to what he mentions. This can be observer in real time by simply holding a key down. Since our "CPU" runs so slowly (at least by today's standards), it cannot keep up with the constantly spammed keypress and will lag behind. When you let go of the key, it will keep drawing that letter until it has emptied the buffer.