CS4375-17800 Fall 2022 Lab Report

Eric Smith. Submitted on Sep 11, 2022

essmith1@miners.utep.edu

**Lab 1: Introduction to xv6**

Task 1. Boot xv6 and explore utilities

For this lab, I used Ubuntu LTS on a VirtualBox vm environment. I was able to show XV6 working correctly as shown in the following photo.

Text

Description automatically generated

The process of getting XV6 went relatively smoothly. The main difficulty I had was with getting VirtualBox running. I run a mac with an m1 chip, so I have to use another computer to get any virtual machine working. It was possible to get a virtual machine to work on m1, but the resulting VM lacked a lot of features and was generally buggy. Setting up a virtual machine environment on Windows went much more smoothly.

Mkdir – Mkdir is a command which will generate a directory within the operating system. By default, this directory will be nested within the current working directory the user is working within the terminal. There are flags such as -p which will further specify the nested structure that the created directory will be within.

Kill – kill is a command which will allows the user to terminate a given process. Generally, typing kill with a flag of l will allow the user to see the processes which they are allowed to kill. Kill is useful when the user wishes to end a process which may be slowing the system down, or even causing crashes. Internally, the process will be terminated and a new process can be created in it’s place by using fork on the parent process.

Sh – sh is a command which will allow the user to execute a given command. We made use of this command in HW1 in which we were required to execute an sh command to perform and ls. Generally, the command can be given from the terminal, a standard input, or a given file. This command is useful when creating code in which a system call is supposed to be executed.

Task 2.

Implementing sleep was a useful exercise which allowed me to practice implementing a system call with relative ease. I was guided through the process of how the system call would be initialized, added to the list of existing system calls, and compiled down to machine language. Overall, the process of implementing sleep was relatively easy, as there are existing system calls which implement similar functionality. This was a good warmup exercise for the next task, which was significantly harder.

Task 3.

PS gave me a lot of difficulties. One problem I had was the procinfo() does 2 things, it returns the amount of processes running in the system, and it modifies the fields of the uproc structure which was passed using copyout. The fact that this function did 2 things was not intuitive. The other problem I had was copying the structure over to user space. I know that I had to use the copyout function to transfer the variables of the proc structure to the uproc structure, but I was not able to pass the destination address of uproc to the procinfo function. First, I tried passing in the pointer itself, but the address of the pointer changes when I do this. I believe this may be because there is some translation going on between user and kernel space.

Another solution I tried to fix this problem was casting the address of the pointer to an integer and passing that along. The main issue I had with this is that the cast was not perfect, and there would be variation between the value of the integer and the actual location itself. I am unsure of how exactly the address of a pointer is cast to a uint64, and I was not able find any documentation on how exactly this was done.

A picture containing text

Description automatically generated

Task 4.

Once PS was implemented getting PSTREE is relatively simple. PSTree just requires the use of the proc info method similarly to how PS applies procinfo to a uproc array. Once this is done, the algorithm for printing the processes in a tree is relatively simple. All you need to do is create a recursive algorithm which detects PPID, and recursively calls itself once this has been done.

**Discussion Question:**

This lab was helpful in allowing me to learn how system calls are made in XV6. I am still confused about how memory gets modified when translating from user space to kernel space. This translation was inconsistent for me, and I still wonder how it works.