ELEC 377 Lab 1 Documentation

Emma Chan – 20206641
Jake Moffat – 20212243

Problem Statement

The purpose of this lab is to examine extra functionality of the Linux Kernel. Through creating a proc file, the process information about the currently running process would have it stored in the Process Control Block. From there, the Linux module would be able to return that process information to allow further understanding of the internal data structures in the Linux kernel. The process information includes name, PID, PPID, State, Real UID, Effective UID, Saved UID, Real GID, Effective GID, and Saved GID.

Description of Lab Chain

The way this module is able to achieve this starts with building a kernel module. This is done through the make command. Then inside the file titled “lab1mod.c”, the final three lines of the file are used to provide the “/proc” file system to use the functionality from the module. The “module_init” function calls the “lab1_init” function because that is what is used to create the lab1 file. Similarly, when the “lab1_exit” function is called, it is what removes the file from the “/proc” directory.

When “lab1_init” is called on line 94, it moves to line 82 to the “lab1_init” function. The function calls the kernel routine “proc_create” found on line 83. This routine contains a pointer to the “proc_ops” struct. This is important because as seen on lines 66-70 this struct contains the default functions “seq_read”, “seq_lseek” and “single_release” that were assigned to the “.proc_read”, “.proc_lseek” and “.proc_release” fields of the structure. The “lab1_open” function is called on line 61 and returns the struct “lab1_show” on line 17. From there, the function contains all the variables of the struct fields on line 21-28, then takes all the data from the Process Control Block and outputs the process information including the state types seen on lines 39-47.

Read Function

The “lab1_show” function accesses the data from the Process Control Block about the current running process. It first initializes all the required variables as ints including, PID, PPID, State, Real UID, Effective UID, Saved UID, Real GID, Effective GID, and Saved GID, Real GID, Effective GID, and Saved GID. The function then extracts the values for each variable from “task_struct”. Then it uses a pointer to access the information about the current running process. The variables PID, PPID and the state types possess their own fields. The state is interpreted from the state field in the “task_struct” and outputs accordingly. The other variables use the “cred” struct, it is a nested structure whose implementation can be seen through the initializations of these variables from lines 23-28, where the pointer goes through the “cred” struct. The variables are all outputted using the “seq_printf” function and returns 0 as a check on the program’s execution.

Differences in Output Files

The differences in the output files between cat and dd are from the way they are called, the way they output data and the output they yield. The cat command is called much simpler as the dd command requires a space if= and the location of the file that will be ran. This is due to dd being an older command. The way they output data is another difference because the name, process id and parent process id change every time because the process counter changes each time a command is typed. Lastly, the output they yield is different because the dd command finishes with extra lines describing the “records in”, the “records out”, and the bytes copied, runtime and kB/s. Whereas the cat command ends by presenting the user with “Saved GID”.