ELEC 377 Operating Systems F22

Lab 1: Problem Statement & Solution Due September 27th,2022 By: Kevin Yu, Raatik Sharma Student Numbers: 20203451, 20120770

Problem Statement

Extra functionality must be added to the Linux kernel to access data in the process control block about the currently running process for two different users. Accessing the data from the process control block will require a building kernel module.

Describing how different parts of the lab link together

1. lab1mod.c file

Using the Makefile tools, lab1mod.c is used to process the kernel module to create the file lab1mod.ko.

2. module_init(lab1_init)

When the “make” command is executed a kernel object is created called “lab1mod.ko”. The module_init() function is used to define which function is executed when the lab1mod.c file is called with insmod to the kernel. When the lab1_init function is called, this creates a proc entry called “lab1” with a pointer to its operations lab1_fops.

3. proc_create(“lab1”, 0, NULL, &lab1_fops)

If the proc_create method succeeds, it will print a message to the Linux kernel log that a proc file has been created. If the proc_create() method fails, it will return -ENOMEM, meaning error no memory from the system could be provided. The following proc_create parameters include

1. Given name of the file in the proc directory
2. The permissions of the file (default value is 0 allowing anyone to read file)
3. Options for building nested directories (this is null since the file is at top level)
4. A pointer to a structure that contains references to the functions that will generate the content of the file.

4. #IFDEF / #ELSE

The “#ifdef” statement represents the conditional initialization of a pointer that contains the references to the functions that will generate the content of the file. Identifiers known as the macro definitions are placed inside the #ifdef and #else statements which control which file operations to use depending on the kernel version. The structure for the operation mapping in the two ifdef statements is dependent on the specified version of the kernel, in version 5.6, this structure uses a “.proc_” association prior a specified field as opposed to what was being used prior to version 5.6, which was a “.” followed by the specified field for example (.open vs .proc_open).

5. lab1_fops

Lab1_fops is structure that maps different defined functions to specific file operations such as

.read to the seq_read, .open to the lab1_open etc. When the “lab1_open” function is called by .open file operation, it calls another function called “single_open()”, and that calls “lab1_show” to start printing out the current PCB Information.

6. lab1_show

The lab1_show function will display information about the PCB to the terminal. When working with the previously mentioned methods/structures, the lab1_show function is called when a process reads the proc entry, which begins by the process opening the proc entry file that starts a call to the function mapped to the .open file operation (which is lab1_open). Now with the file open and beginning to be read, this makes the lab1_show function to execute printing out the current process information using the seq_printf() function. The parameters for the seq_printf() function are similar to the printf() function but includes the parameter: struct seq_file *m to work with the files that is being outputted. Strings that are passed to seq_printf() are passed to the file buffer, which is then printed to the screen using the cat process. The properties being printed out include fields such as ppid, state, process pid and many more in regard to the PCB. The file format that was specified to print out was in the form of a compiled executable file.

7. module_exit(lab1_exit)

In order to ensure that this process entry does not stay loaded forever in the kernel space, the remove_proc_entry function is used to remove the process entry from the /proc directory when rmmod lab1mod is called.

Describing the contents of the read function

When a command like cat is called, lab1_show method displays information to the terminal using seq_printf. The seq_printf is similar to printf, but includes a pointer to a struct that allows access to the current PCB from the proc file that was created.

The values such as **name** and **PID** were found by accessing the pointer to the current struct (which is a kernel variable that points to the current running process in the PCB) and then accessing “com” and “pid” respectively. The name property had a char array data type which was casted to a string for printing. The PID had a pid_t struct data type and was casted to an integer value. The **PPID** was found also in the current struct, under the parent field and then under the “pid” property. PPID had a pid_t struct data type which was casted to an integer datatype. The **Real UID, Effective UID, Saved UID**, **Real GID, Effective GID, and Saved GID** were found under the current struct, then under the struct cred, then the value properties were located in uid.val, euid.val, suid.val, gid.val, egid.val, and sgid.val respectively. The UID and GID properties specified above had a gid_t struct data type and they were casted to a integer value.