**CS 4500 Operating Systems**

**Section 001**

**Project #2**

**Report**

**Chase Bauer, Sean Campbell, Phil Hilt**

**24 September, 2022**

**Statement: “**We have neither given nor received unauthorized assistance on this work.”

# 1 Virtual Machine Information

Directory and Name of Virtual Machine: cbauer VM

Password for VM Account[[1]](#footnote-1): password

Path to Source Code: /home/CS4500/project2/

# 2 Description

### 2.1 How We Solved the Problems

Part 1

### After updating the kernel and installing the kernel modules, the helloworld kernel module was implemented in its own folder called ‘helloworld’. The provided code used function names ‘init\_module’ and ‘cleanup\_module’ that were already defined in a linux header so to make the code the function names were changed to ‘init\_mod’ and ‘cleanup\_mod’. Then the provided commands were run as root to load the model, view its output, and remove the model from the kernel. To view print the messages of the two functions, the printf function should be used.

Part 2

For step two another folder called printself was made to hold the necessary Makefile and print\_self.c files. The target in the make file was changed to print\_self.o. The provided link to the linuxgazette post proved helpful in solving this part. A function print\_self\_init\_module was created which instantiated a task\_struct to the current PID of the process module. Then the process name, ID, and state were printed before moving the task pointer to the parent process. A while loop moved through the process hierarchy until it reached the init process, printing out process info through each iteration. A function state\_string was made to map each process’ state number to its corresponding state string, and a function print\_self\_cleanup\_module was also written to print a message once the process is removed from the kernel. After running make to compile and create batch files. The same commands for part 1 are used to load the module and print its output with the parameters for insmod and rmmod being changed to ‘print\_self.ko’ and ‘print\_self’, respectively. The output shows that the ‘init’ equivalent process for CentOS 7 is called ‘systemd’. From the output the state TASK\_INTERRUPTABLE is observed.

Part 3

For this part, the provided thread and link to The Linux Kernel Module Programming Guide were referenced. The same structure for part 2 were followed with the Makefile and print\_other.c files being written in the folder print\_other. The moduel\_param function from the moduleparam macro was used to get the user provided PID argument. Then instead of setting the task\_struct to the current PID it was set to the user provided PID using ‘pid\_task(find\_vpid(arbitrary\_process)’, PIDTYPE\_PID). The bash command ‘pgrep bash’ was used to find and select a valid PID. The steps to make, load the module, and see its output are the same for the previous steps only when loading the module the user must provide a PID i.e. ‘insmod print\_other.ko arbitrary\_process=PID’.

Part 4

A kernel module is a piece of code that can be loaded and unloaded into the kernel on demand. A system call makes operating system services possible and acts as a “gate” into the kernel by allowing software interrupts to move from user mode into kernel mode. The supplied code failed to compile because there is no macro for syscall.h. It is a good thing that code this old doesn’t work since hardware and architecture specifications may be out of date.

### 2.2 What We learned

### This project was helpful in learning how to write Linux kernel models and how they are managed in the operating system. The Linux kernel uses a circular doubly-linked list to store records of processes. Using standard kernel macros a user can programmatically move through the process tree to get information about processes. The commands insmod and rmmod can be used to load a module and remove a module from the kernel. Once a module is loaded it becomes a process. The command lsmod shows which loadable modules are loaded. The command dmesg -T | tail can be used to see a module’s output.

### The operating system maintains a system call table with pointers to functions necessary to implement system calls inside the kernel. The init process is the mother of all processes and always has the static PID 1. In CentOS 7, the init equivalent is systemd.

### 2.3 How to Run Our Code

### The code is all located in /home/CS4500/project2 on our VM. Each part has its own subfolder (helloworld, printself, print\_other). For each subfolder run the make command then use ‘insmod <module name>’ to load the module (for print\_other provide arbitrary\_process=<PID#> parameter) use rmmod < module name> then <dmesg -T | tail> to see the module’s output.

# 3 Screenshots of Output

### 3.1 Code Output Screenshots

### Part 1. Output shows hello world and goodbye world with different timestamps.

**Text

Description automatically generated**

Part 2. Output shows init as system and shows states as TASK\_INTERRUPTIBLE

**Timeline

Description automatically generated with medium confidence**

Part 3. First process printed used from user defined PID.

**Timeline

Description automatically generated**

Part 4. Fails to compile no macro for syscall.h

**Text

Description automatically generated**

1. Depending on the project requirement, you may need to provide the password of a regular user or the root user. [↑](#footnote-ref-1)