**CS 4500 Operating Systems**
**Section 001**
**Project #4**
**Memory Management**

**Chase Bauer, Sean Campbell, Phil Hilt**
**9 December 2022**

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

# 1 Virtual Machine Information

Directory and Name of Virtual Machine: CS4500/cbauer2/cbauer VM
Password for VM Account[1]: password
Path to Source Code: /home/cbauer/project4

# 2 Description

## 2.1 How We Solved the Problems

**Part 1**

In part 1 we need to write a module that reports the statistics of a process's virtual address space size. The process we used to report statistics on was the bash process, and we found its pid number by running the pgrep command. After storing the pid value in the program, the task, mm, and vm_area structs are declared, as well as the variables for the total size of the space and the virtual memory area count are initialized. The tast_struct is then initialized using the pid_task() function with the input being the pid previously found. The memory map struct is then initialized using the mm variable from the task struct. All the virtual memory areas are stored in the memory maps variable mmap, which is then set the the vm_area struct all_vmas. A loop is then implemented that loops through all the virtual memory areas, using the all_vmas struct, to determine the total size of each virtual memory area. The start of the virtual memory area is found by calling current_vma.vm_start, which is a variable in the vm_area struct and the end of the virtual memory area is similarly found by calling current_vma.vm_end. These variables are then subtracted to find the total size of that virtual memory area and the size is added to the TotalSize variable. After all virtual memory areas are looped through the total virtual address space is printed out in bytes and in kilobytes.

**Part 2**

In part 2 we need to write a module to report the status of a specific virtual address. From gathering information of the pmap of bash, the virtual address where the bash program starts is at 400000. In the va_status_mod_init() function we first started by initializing variables, casting the virtual address string to an unsigned integer and initializing a task struct with the pid of the bash program. After the initializations the page global, upper and middle directories are stored into variables in order to find the page table entry, PTE, of the virtual address using the page middle directory. To make sure entries do not change when accessing the PTE, a spinlock is implemented to lock the current entries. Once the lock is called the function pte_present() is called with the input being the PTE of the virtual address and the page middle directory. If the PTE is present, then a printk will be called notifying that the address is in memory. If it is not present, then that means the address is on the disk and not in memory. After checking if the PTE is present the spinlock is unlocked and the program ends.

## 2.2 What We learned

This project helped solidify previous knowledge on how to build and load Linux kernel modules with arguments. An understanding of how to interact with and utilize a process's virtual address was also gained. Both parts required the use of the linux/mm_types.h module to utilize the

[1] Depending on the project requirement, you may need to provide the password of a regular user or the root user.

memory descriptor data structure, mm_struct. For part 1 we had to learn how to use the memory descriptor to loop through all virtual memory addresses in a process and calculate the total size of a process's vma. For part 2 we learned how to make use of the mm_struct to see the status of a specific virtual memory address.

### 2.3 How to Run Our Code

There are folders va_size and va_status located in /home/cbauer/project4/ for part 1 and part 2 respectively. For each part go to the necessary folder and run Make to compile the C code. For part 1 load the module with a valid PID and look at the module output to see the vma size of the provided process. For part 2 load the module with a valid PID and virtual address (as a string) as arguments.

## 3 Screenshots of Output

### 3.1 Code Output Screenshots

**Part 1**

```
[root@localhost va_space]# pgrep bash
1613
[root@localhost va_space]# insmod va_space.ko pid=1613
[root@localhost va_space]# dmesg -T | tail
[Wed Dec  7 22:57:15 2022] VMA #32 Start Address: 0x7fffa39f3000
[Wed Dec  7 22:57:15 2022] VMA #32 End Address: 0x7fffa3a14000
[Wed Dec  7 22:57:15 2022] VMA #32 Total Size: 0x21000
[Wed Dec  7 22:57:15 2022] ======================
[Wed Dec  7 22:57:15 2022] VMA #33 Start Address: 0x7fffa3ace000
[Wed Dec  7 22:57:15 2022] VMA #33 End Address: 0x7fffa3ad0000
[Wed Dec  7 22:57:15 2022] VMA #33 Total Size: 0x2000
[Wed Dec  7 22:57:15 2022] ======================
[Wed Dec  7 22:57:15 2022] Total VMA Size Bytes: 118456320
[Wed Dec  7 22:57:15 2022] Total VMA Size Kbytes: 115680
[root@localhost va_space]# pmap 1613 | grep total
 total           115684K
```