Project 3 – Memory Management

CS 4500

Daniel Frey, Justin Hale
4-23-2019

1. **Project Members**: Daniel Frey and Justin Hale
   We have neither given nor received unauthorized assistance on this work.

2. **VM**: dfrey-CS4500
   Password: `instructor` account password: zhuang
   ***Note: `instructor` account has sudo privileges without password**

3. **Part 1**: **/home/instructor/Project3**
   Solving Part 1 was straightforward. To solve this problem, we obtained the task_struct of the bash process with '`pgrep bash`.' From there, the memory map struct was obtained. Then the vm_area_struct was used to calculate the size of the VMA. This involved using the starting and ending addresses to calculate a size of a specific memory area. A loop was required so that all the VMAs could be combined to get a total size. This will output the size in both bytes and kilobytes.

   To test this, we used '`pgrep bash`' to get a PID to send to the syscall.

   From Part 1, we learned how to find out the total virtual memory of a process, and how the total virtual memory is a collection of virtual memories. Also, it showed us how much other information is available from the process structs.

   ***The program to run is test_vma_stats**

```
[dfrey@localhost ~]$ ./test_vma_stats
PID sent to vma_stats: 8021
[dfrey@localhost ~]$
Message from syslogd@localhost at Apr 19 21:19:02 ...
 kernel:PID 8021 total size VMA (bytes): 5390336, (KB): 5264
```

   Part 2: **/home/instructor/Project3**
   Solving Part 2 involved doing things in a certain order. It involved obtaining the task_struct from the PID argument. Then the task_struct was used to get the mm_struct. Once that was obtained, then we started getting the page directories. Getting these involved a specific order. First the page global directory was gotten, followed by the page upper directory, then the page middle directory. Once the page middle directory was had, a spinlock was applied to the page table entry pointer. This was done to avoid a kernel oops to make sure that any processes did not change anything. From the pointer, the actual page table entry was gotten, and the present bit was checked to see if the memory address was in memory or on disk. It will display 1 if the address is in memory.

   To test this, we used the command '`pmap $(pgrep bash)`' to get the pmap of the PID from '`pgrep bash`.' The first line was used to get only the PID. Similarly, the second line was used to grab just a memory address. The address and PID were sent into the syscall.
   ***Note:** `pmap` **command is executed with sudo access –no password required. It is written into the test program.**

   From Part 2, we learned how to perform a page walk. It helped us become familiar with how to navigate the page entries.

***The program to run is test_vma_status**

```
[dfrey@localhost ~]$ ./test_vma_status
Sent to vma_status: Address: 3215478784, PID: 8021
[dfrey@localhost ~]$
Message from syslogd@localhost at Apr 19 21:23:01 ...
 kernel:Address: 3215478784, PID: 8021, Present: 0
```

```
[dfrey@localhost ~]$ ./test_vma_status
Sent to vma_status: Address: 134508544, PID: 8021
[dfrey@localhost ~]$
Message from syslogd@localhost at Apr 19 21:25:28 ...
 kernel:Address: 134508544, PID: 8021, Present: 1
```

***Note: Source codes and test files can be found in Project 3 folder**