Kernel Module to translate virtual to physical address mapping by page table walk

Objective

You will learn how to develop and use kernel modules. You will also learn how the Linux OS maintains virtual to physical address mappings in per-process page tables.

Problem Description: Write a kernel module to translate virtual addresses to physical addresses for a target process

Create a kernel module that takes the process-ID of an active process as an argument (module parameter) and outputs all the virtual to physical address mappings of the target process. Print the virtual address and the corresponding physical addresses to the system log using printk(). Show that your code works for any arbitrary target process.

Below we describe all the resources you will need to learn and/or use to completethis assignment. Start early and ask us for help if you get stuck.

Accessing your Virtual Machines

This assignment will be carried out on a Linux virtual machine. Each student has been assigned a VM with a unique IP address. The login credentials have been emailed to you. You can either use the VM assigned to you or you can use your own Linux VM on your personal computer.

Instructions to access your virtual machine remotely

Learn how to write and execute a Linux kernel module.

Kernel modules allow the code to be dynamically added to the kernel, dynamically. Try compiling and inserting the kernel modules hello.c and hellon.c from kernel module slides.

1. Save the Makefile, hello.c, and hellon,c in your home directory. The files hello.c has initialization and cleanup functions that are invoked upon loading and unloading the kernel module. The second file hellon.c is similar, except that it takes command-line arguments with insmod command to print a custom message.
2. Compile the kernel module:

   ```
   make
   ```

3. Load the kernel module:

   ```
   sudo insmod hello.ko
   ```

   If the module was successfully inserted, you will see "mymodule: Hello World!" message in kernel log. Use commands "dmesg" or "cat /var/log/kern.log" to see the kernel log messages.

4. Unload the kernel module:

   ```
   sudo rmmod hello
   ```

Upon removing the kernel module, use the "dmesg" command to check if kernel log prints the message "mymodule: Goodbye, cruel world!!"

5. For hellon,c, module insertion command is `sudo insmod hellon.ko whom=class howmany=10` (or any other argument values you want to give). Removal is the same as for hello.c.

How to browse Linux source code

There are a couple of ways you can browse linux source code.

- Use the Linux cross reference website
- Download the entire linux source code to your VM (or any other Linux computer) and use the `cscope` tool to browse. (slightly faster as you get familiar with cscope).
  - `wget https://cdn.kernel.org/pub/linux/kernel/v4.x/linux-4.19.80.tar.xz` (replace with whatever kernel verssion you are developing your module against)
  - `sudo apt install cscope`
  - Cscope tutorial

Understand how to look up the virtual address to physical address mapping in the Linux kernel.

1. Slides to 5-level page table
2. Get Page Global Directory (PGD) from mm_struct. mm_struct structure holds the process virtual memory information. CR3 register is loaded with **mm_struct->pgd** which points to the process page table or base physical address of page global directory.
3. Each entry in the PGD points to a page frame of next level page directory called p4d directory. Use **pgd_offset()** to get the offset in PGD page frame that points to the next level p4d page directory.
4. Each entry in the P4D points to a page frame of next level page directory called page upper directory (PUD). Use **p4d_offset()** to get the offset in P4D page frame that points to the next level pud.
5. Each entry in the PUD points to a page frame of next level page directory called page middle directory (PMD). Use **pud_offset()** to get the offset in PUD page frame that points to the next levelpmd.
6. Each entry in the PMD points to a page frame of next level page directory that contains page table entries. Use **pmd_offset()** to get the offset in PMD page frame that points page frame with page table entries.
7. Use **pte_offset_map()** to get the offset in the page directory that contains a list of page table entries.
8. Finally, check if the page table entry points to page using **pte_present**. If a page table entry is present, get the page frame number or PFN for the given virtual address using **pte_pfn().**
9. Test applications: computation_intensive.c, memory_intensive.c

Grading guideline

20 - Part A: Printing virtual memory areas of any arbitrary process.
50 - Part B: Demonstrating that your code prints the virtual address to page frame number mappings for any arbitrary process.
20 - Part C: Report explaining behavior of two given applications.
10 - Handle all major error conditions. Clean, modular, and commented code. Clean readable output. No "giant" functions.
Total = 100