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

Objective

You will learn how the Linux OS maintains virtual to physical address mappings in per-process page tables. You will also learn how to use the `ioctl()` interface to interact with your character device.

Part A: 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.

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

Part B: Create an ioctl interface retrieve a virtual page number to physical page number mapping.

1. Learn how to use an `ioctl()` interface using this example. Download it, compile it, and run it on your VM.
2. Next, modify your kernel module from Part A to implement a character device with the following behavior.
   - The character device should provide an `ioctl()` interface to a user-space application.
   - Implement an `ioctl()` command called `IOCTL_GET_PFN`.
   - The `IOCTL_GET_PFN` command should accept a virtual page number of the target process (specified earlier as the module argument) as the `ioctl` parameter. This is the last `ioctl_param` argument to the `device_ioctl()` function in the above example.

- It should return the physical page number mapped to the given virtual page number, as the return value of the `ioctl()` call.

3. Write a user-space program that invokes the `ioctl()` command on your character device to convert virtual page numbers to physical page numbers for the target process.

Grading guideline

50 - Part A
50 - Part B
Total = 100