PLAGIARISM STATEMENT <Include it in your report>

We certify that this assignment/report is our own work, based on our personal study and/or research and that we have acknowledged all material and sources used in its preparation, whether they be books, articles, reports, lecture notes, and any other kind of document, electronic or personal communication. We also certify that this assignment/report has not previously been submitted for assessment in any other course, except where specific permission has been granted from all course instructors involved, or at any other time in this course, and that we have not copied in part or whole or otherwise plagiarised the work of other students and/or persons. We pledge to uphold the principles of honesty and responsibility at CSE@IITH. In addition, We understand our responsibility to report honour violations by other students if we become aware of it.

Names: J Sai Nishith, NJ Swaroop

Date: 21-06-20

Signatures: JSN, NJS.

Aim: The aim of this assignment is to understand how virtual memory paging happens in Linux.

Two files which are edited are

1.mykmod_main.c

2.memutil.cpp

Explanation for those edits is given below.

1.mykmod_main.c:

This is a C program that generates an object file when we invoke ‘make’ as mentioned in the readme file which serves the purpose of a kernel module. We initialize few structures like mykmod_virt_data, mykmod_dev_info which keep track of the per device info which contains char* data , unsigned int *pageindex..

mykmod_init_module(): Called when we load the module and the pseudo device is initialized.

mykmod_open(): This is called when we create a device special file using mkmod and open it in user space. For the first time we check if the inode private data is 'NULL' because initially nothing is stored in it.

We store the device info in a 'mykmod_dev_info' structure and a pointer to 'mykmod_dev_info' structure is stored in inode pointers i_private and file pointers file_private slots.

mykmod_close(): Gets invoked whenever a file is closed is user space and it exits by printing the corresponding information.

mykmod_mmap(): This function is used to map the device file data into kernel space. Demand paging or the prefetching decided by the mmap based on flags it gets. For instance we send only MAP_SHARED flag for demand paging, whereas in case of prefetching we send both MAP_SHARED,MAP_POPULATE flags. Other vma flags and mykmod_ops are intialized accordingly in this method. We use vma_track to store device info and number of page faults which in turn is stored in vma's private data and the page faults are also set to zero.

mykmod_vm_fault(): This function is called whenever a page fault incurs, i.e. if we access regions that are not mapped yet. A simple addition of offset is done to these virtual addresses which are later passed to **virt_to_page** function in order to map the physical page frame to this address. We increment the number of page faults(npagefault). **get_page()** is used for incrementing the count of references to this page.

2.memutil.cpp:

There are two possible cases

1. OP_MAPREAD
2. OP_MAPWRITE

In case of write operation ,we need to write the message passed through command line argument in device memory. **mykmod_mmap()** function maps device memory into user space and returns a pointer to the starting location of device memory. Using this we write the message required into that data.

In case of read operation, we check if data written is corrupt. We use **mykmod_mmap()** function mentioned above in the write method and check the data in memory with messages acquired through command line argument. Accordingly EXIT_SUCCESS or EXIT_FAILURE is returned later.

mykmod_vm_fault() function handles the page faults in both the write and read operations.

Memory is unmapped using the **munmap()** function and based on flags program returns EXIT_FAILURE as mentioned above.

SAMPLE OUTPUT:

Demand paging (Read)+TestScript: