Operating Systems II

Programming Assignment 3 - Report

Sandeep Kumar : CS18BTECH11041 | Vedant Singh: CS18BTECH11047

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: Vedant Singh and Sandeep Kumar

Date: 20/6/2020

Signatures: VS, SK

Overview

In this assignment, we attempt to implement the basic underlying code required for the `mmap()` function call, so as to create a mapping for a given device special file. Reads and Writes are done to this device file via both demand and prefetch paging. Two files were crucial for execution of the above tasks; `mykmod_main.c` and `memutil.cpp`. Descriptions for both are given below.

Description about mykmod_main.c:

The major structures in the file are `mykmod_dev_info` and `mykmod_vma_info`. The former contains a string and a variable to store the device file's size, whereas the latter contains a pointer of type `mykmod_dev_info` and also an integer variable to store the number of page faults the vma has experienced so far.

The following are some brief descriptions about the important functions in mykmod_main.c.

1. **mykmod_init_module** : This function is called whenever a kernel driver is loaded. It's main tasks are to register the device module and allocate memory to the already defined device table using `kmalloc(2)`.

2. **mykmod_cleanup_module** : This function is called whenever a kernel driver is unloaded. The purpose of this function is to free up the memory occupied by the device table. To achieve this we have to `kfree(2)` the device table array as well as the individual `mykmod_dev_info` structures.

3. **mykmod_open** : This function takes in two parameters: **file** pointer and **inode** pointer. If the passed inode's `i_private` field is yet to be set, we encapsulate the device info into a `mykmod_dev_info` structure and make `i_private` point to it. `kzalloc(2)` is used to allocate and assign values to the data field whereas the size field is a predefined constant. Now, a for loop finds the first empty slot in the device table to store this structure. The device info is stored in the file's `private_data` as well, which is essential for the working of mmap later on.

4. **mykmod_close** : Denotes the closing of a device special file.

5. **mykmod_mmap** : This function takes in two parameters: the **file** pointer and the **VMA** pointer. Firstly, the flags for the VMA are set. The purpose of this function is quite similar to **mykmod_open** in the sense that its main aim is to initialize the `vm_private_data` field of the newly created `mykmod_vm_info` structure. The `devinfo` field of this structure is populated using the private field of the file pointer passed (which was in turn populated in the `mykmod_open` function).

6. **mykmod_vm_open** : This function takes the VMA's information through its `private_data` field for printing. Since the number of page faults should be 0 at the starting, it does this by assigning 0 to the field `npagefaults`.

7. **mykmod_vm_close** : The VMA's information is extracted like in the above method. After printing the information, the `npagefaults` field is reset to 0.

8. **mykmod_vm_fault** : When a user tries to access a page in the VMA which is not yet in the memory, it results in a page fault. This is where the `mykmod_vm_fault` function comes into play. It uses the page offset fields of the two arguments passed to it i.e `vma` and `vmf`, to determine where the fault occurred in the VMA. It can then easily find the pointer to this missing page using the `get_page` function. The function also makes sure that the `npagefaults` field of the vma is updated accordingly. If the vma hasn't been initialized yet, it results in a segmentation fault.

Description about memutil.cpp

The memutil.cpp file's main purpose is to call the mmap() function and to test writes and reads from the newly mapped memory space. In order to provide a well-rounded test, we are filling up the entire memory space while writing by repeating the same message string over and over again till the end of the allotted 1MB. Even when reading from memory, memutil.cpp expects the message to be looped throughout the memory space.
Depending on the command line arguments given to the file it decides whether it will execute the prefetch paging or demand paging.

Proof of successful execution of test script: