Project by: Sai Varshittha P & K Vamshi Krishna Reddy
Roll no: CS18BTECH11035, CS18BTECH11024
Date: June 21, 2020

This document is generated by LaTeX

Report

PLAGIARISM STATEMENT : I certify that this assignment/report is my own work, based on my personal study and/or research and that I 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. I 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 I have not copied in part or whole or otherwise plagiarised the work of other students and/or persons. I pledge to uphold the principles of honesty and responsibility at CSE@IITH. In addition, I understand my responsibility to report honour violations by other students if I become aware of it.

Name:Sai Varshittha P & K Vamshi Krishna Reddy
Date:21-06-2020
Signature: Sai Varshittha P & K Vamshi Krishna Reddy

- We tried to implement pseudo character device by enhancing the mykmod linux character device driver shared with us and implemented prefetching and demand paging on a device special file by understanding virtual memory paging in linux.

Overview of Code in **mykmod_main.c**

- The struct dev_file_info is the data structure used to store the size of the allocated memory and a pointer to data to be written/read.
- The struct vm_using_struct is the data structure that keeps per VMA info namely npagefaults(number of page faults) and a pointer to device info.
- The variable device_count keeps the count of number of device instances.
- The **device table** is implemented by array of struct pointers declared and it is initialised globally.
- We register the character device driver in **mykmod_init_module** function.
- In **mykmod_cleanup_module** function,we unregister the character device driver and free the pointers in the device table as well as data in them.
- In **mykmod_open** function,we allocate 1MB memory for a device instance file when it is first opened.We Store the device info pointer for the device in the i_private of inodep and also store it in device table.We also store this pointer in private_data of filep struct for future usage.
- In **mykmod_mmap** function,we set the vma flags by vma→vm_flags |= VM_DONTEXPAND | VM_DONTDUMP. We also save the private data of device(i.e.,devinfo and npagefaults) in vm_private_data.
- In **mykmod_vm_open** function,we initialise npagefaults to 0.
- In **mykmod_vm_close** function ,we free the private data in vma.

- In **mykmod_vm_fault** function ,we do the following :

  - We increment the npagefaults by 1 everytime this function is called.
  - Now , to get the next page , we first take the virtual address of the starting address of the allocated 1 MB memory.Secondly,we find the offset from this starting address till the address with first page fault . To get the offset , we first find the offset from this starting address till the start of mmap-ing by vma→vm_pgoff and then add this to vmf→pgoff which is the offset from the beginning of mmap-ing till the first pagefault.
  - Now we get the total offset in terms of number of pages. To convert into bytes,we do left shift by PAGE_SHIFT to get the offset in bytes.
  - We now add this offset to the starting address to get the virtual address where the page fault is encountered.Then, we pass this address to **virt_to_page** function to get the page pointer.We pass this page pointer inside **get_page** function to get the page and then attach this page to vmf→dfpage.

Overview of Code in **mem_util.cpp**

This codes checks the working of device driver.

- We set the mmap flags to MAP_SHARED for demand paging while we set the flags to MAP_POPULATE | MAP_SHARED in case of prefetching.
- In case of OP_MAPREAD,we first memory map the kernel buffer into user-space segment using mmap.If it is failed , we return EXIT_FAILURE.For non-empty message , we compare the characters in user buffer with kernel one and return EXIT_FAILURE if they mismatch.If the length of message is 0,it is not reported as error; we just access each element of the device memory.Lastly, we unmap the device memory by using munmap.
- In case of OP_MAPWRITE, we first memory map the kernel buffer into user-space segment using mmap.If it is failed , we return EXIT_FAILURE.For non-empty message , we write the message to device memory from the message .Lastly, we unmap the device memory by using munmap.

Following is the screenshot of working of programme:

```
[root@cs3523 99_devmmap_paging]# bash runtest.sh
PASS - Test 0 : Module loaded with majorno: 243
PASS - Test 1 : Single process reading using mapping
PASS - Test 2 : Single process writing using mapping
PASS - Test 3 : Multiple process reading using mapping
PASS - Test 4 : Multiple process writing using mapping
PASS - Test 5 : One process writing using mapping and other process reading using mapping
PASS - Test 6 : One process writing to one dev and other process reading from another dev
[root@cs3523 99_devmmap_paging]# cat test0-dmesg.txt
[12297.599189] mykmod loaded
[12297.599198] mykmod initialized at=ffffffffc0669220
[12297.599206] register character device 243
[root@cs3523 99_devmmap_paging]# cat testn-dmesg.txt
[12297.972324] mykmod unloaded
[root@cs3523 99_devmmap_paging]#
```