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.
Name:Sai Balaram K , D Krishna Pawan
Date: 21-06-20
Signature: KSB,DKP

Goal:

The goal of this assignment is to understand the virtual memory mapping and the different paging techniques implementation for the mmap( ) system call. Using it as a device driver and testing the device driver which is most widely used.2 page fault implementation schemes prefetch and demand paging needs to be implemented for the mmap( ) system call.

Implementation:

mykmod_main.c:

1. Declare a two member structure **mykmod_dev_info** to keep track of the per device info which contains **char* data** and **unsigned int* pageindex**.
2. Declare another two member structure **vma_track**. It has **atomic_t refcnt** and **struct mykmod_dev_info *dev_data** and is used to maintain the data for the vma region.
3. **vm_operations_struct** is a structure containing the functions that execute the corresponding task in the virtual memory region .**open = mykmod_vm_open, .close = mykmod_vm_close, .fault = mykmod_vm_fault**.
4. Declare two global variables '**mykmod_major**' - to keep track of the major number of the device driver and '**npagefaults**' - to keep track of the page faults occurred in the virtual memory region.

5. **mykmod_init_module(void)** is where the device is initialised.
6. **mykmod_cleanup_module(void)** is used to unregister the device based on its major number.
7. **mykmod_open(struct inode *inodep, struct file *filep)** is a file operation for the device special file that opens a file and stores the information in the file. Initially the inode private data is NULL then the inode private data is updated with the **mykmod_dev_info** structure which contains the data else the file private data is updated with the inode private data and the information is printed .
8. **mykmod_close** is used to close the device special file and print the corresponding.
9. **mykmod_mmap** is used to map the file data into the kernel region. **vm_area_struct** is a structure used in the virtual memory region and is initialised with virtual memory regions, functions **mykmod_vm_ops** is assigned to **vma->vm_ops** to perform the operations in the virtual memory regions,and the corresponding flags and other variables are initialized in it. The file private data is stored in the vma_track and the vma track is stored in the vma private data.
10. **mykmod_open** and **mykmod_close** print the vma address and the page faults and page faults are set to zero.
11. **mykmod_vm_fault( )** is automatically called when a page fault occurs when mapping the kernel data the structure mykmid_dev_info and vma_track are initialised and the virtual address mapping is done correspondingly to get the age.
    offset = (unsigned long)(vmf->pgoff << PAGE_SHIFT); { the offset to get the correct address to get the correct page }
    vmf->page = virt_to_page((unsigned long)info->data + offset); { which is free }
    get_page(vmf->page);
    And **npagefaults** is increased.
12. This page fault operation calculates the offset which is the address we are at this point and the page we want to get next using **get_page( )**.
13. **mykmod_fops** and **mykmod_vm_ops** are two structures initialised with the operations to be for file operations and the operations in the virtual memory region respectively.

memutil.cpp:

This is the file which is used to test the usage of the driver from the user space.

1. It reads the ‘msg’ from the argument passed.
2. **read_flags** and **write_flags** are 2 boolean variables which are changed according to the operation needed to be performed.
3. It checks if the paging operation needed to be performed is ‘prefetch’ or ‘demand paging’ and based on that **mmap_flag** is used to set the flags for the operation.
4. If the paging is ‘demand’ paging then **mmap_flags = MAP_SHARED**.
5. If the paging is ‘prefetch’ then **mmap_flags = MAP_SHARED | MAP_POPULATE** the flag which tells the kernel to prefetch the pages
6. Based on the message given we will be asked to perform the operation of write and read from the kernel to check the driver utility.
7. If the case is **O_RDONLY** then we should check the check if the data was previously mapped correctly.

dev_mem = (char *) mmap( ) is done by passing the appropriate flags and variables as arguments and a for loop is run to check the correctness of the mapping. The for loop is run for size of 1MB.
If ***(dev_mem + i) != *(msg + i%msg_len)** at any point we keep a flag and break out from the loop. ‘ **i%msglen** ‘ is used because the message is mapped for 1MB size when it was write into the memory map as the message repeats that is used.Next the memory needs to be unmapped using **munmap(dev_mem, MYDEV_LEN)**
And if the flag is changed means that the mapping has failed and the program should return **EXIT_FAILURE**.

8. In the case of **OP_MAPWRITE** the data should be written into the kernel and it needs to be mapped.
   dev_mem = (char *) mmap( ) is done by passing the arguments and the for loop is run for 1MB size such that the message passed can be repeated and mapped. It is mapped such that ***(dev_mem +i) = *(mem + i%msg_len)** is used to write the message repeatedly of 1MB into the kernel and memory mapping it. After that the file pointer can be closed.

Test cases:

Running the test script in my laptop gave the following result

```
[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]#
```