REPORT
HWP Assignment 3
Paging on Device Memory

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: Pothugunta Chaitanya Janakie, Bheemanath Dheekshitha
Date: 21-06-2020
Signatures: chaitanya, dheekshitha.

mykmod_main.c :

1. There are three data structures which are used in this program. They are:
   1. A structure(‘dev_info’) to keep information of each device. The fields in it are a character pointer(char *data) to store the data and ‘size’ to store the size of each device.

   2. A structure (‘vma_info’) to keep information of each VMA. The fields in it are a pointer to struct dev_info and an integer ‘npagefaults’ to keep track of no.of page faults.

   3. An array (‘dev_table‘) of size ‘MYKMOD_MAX_DEVS’ to keep all devices or to store pointer of each device.

2. The pointers info, vma_inf which point to structs dev_info, vma_info respectively are declared globally. Also an integer variable ‘i‘ is declared globally to keep track of no.of devices.

3. The instance of file_operations is to list all the operations which are supported on a device special file. Similarly the instance of vm_operations_struct will list the operations supported in the virtual memory area.

4. **mykmod_init_module():** Whenever a device driver is loaded, then this method is invoked. In this method we register the pseudo character device driver by passing driver name and file_operations structure to register_chrdev(). It returns a major number which is unique to each driver. If this is less than ‘0’, then giving a warning message.

5. **mykmod_open():** Whenever a device special file is opened then this method is invoked. We manually create a device special file using ‘mknod’. In this method, if this file is not present already i.e; inodep->i_private is NULL, then we allocate memory for dev_info pointer (info) and also allocating memory of 1MB for the data, and setting the size to MYDEV_LEN. Storing the dev_info pointer in the device table and in inodep->i_private so that when we want to open the file again we can access it using inodep. Also storing the device info in the file's private data as well.

6. **mykmod_close():** Whenever a device special file is closed, this method gets invoked. In this method, we are just printing some fields.

7. **mykmod_mmap():** This method is invoked whenever mmap() is invoked from user space. In this method we are setting up vma’s flags. Here we allocate memory for the vma_info pointer (vma_inf), then pointing devinfo to device info using filp->private_data. And then saving the private data (devinfo, npagefaults) in vma->vm_private_data. Now we invoke the method ‘mykmod_vm_open’.

8. **mykmod_vm_open():** In this method we just initialize the npagefaults to ‘0’.

9. **mykmod_vm_close():** Whenever munmap() is called in the user space, this method gets invoked. In this method we are just printing the total no.of page faults and some other things. Also we are re-initialising no.of page faults to ‘0’. And then freeing the memory allocated for the pointer ‘vma_inf’.

10. **mykmod_vm_fault():** Whenever a read or write for a file request comes, and if the file is present in secondary memory, then page fault occurs and this method

gets invoked. In this method we increment npagefaults. To build the virt->phys mappings, we calculate the offset. We calculate that by adding both vma and vmf offsets. Now the virtual address will be info->data + this offset. The method virt_to_page() will take a virtual address and return a page pointer. We store this page pointer in vmf->page.

11. **mykmod_cleanup_module():** When we unload the driver, then this method gets invoked. Here we unregister the character device driver and then free all the devices through the device table. First we free the memory allocated to data in dev_info, then the memory allocated to pointer ‘info’.

memutil.cpp:

1. In this file we just set up the mmap flags and write code in case of memory mapped read and memory mapped write.

2. Here in case of demand paging, mmap_flags is equal to MAP_SHARED, whereas in case of prefetching mmap_flags is equal to MAP_SHARED | MAP_POPULATE. MAP_POPULATE is to preload the pages.

3. In case of read operation to a file, firstly we map the memory in the kernel buffer into use-space segment by using mmap() system call. Now if the length of msg is not equal to ‘0’, then we compare the msg with the data read from device memory. If they are not equal then setting return value to 'EXIT_FAILURE'. And if the length of msg is equal to ‘0’ then we simply read the data. Finally unmapping the kernel buffer of device memory.

4. In case of write operation to a file, firstly we map the memory in the kernel buffer into use-space segment by using mmap() system call. Now we just copy the string msg to the entire device memory. Finally unmapping the kernel buffer of device memory.

NOTE: Sample outputs are shown in README file.