CS3523: OS-II - ASSIGNMENT 3

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

Background:

Linux supports mmap() and munmap() system calls on regular files (eg. /etc/os-release), device special files (eg. /dev/zero). The mmap()/munmap() are the most heavily used system calls in Linux.

For example, when we run any program (eg. ls, cat, firefox, etc), the program executable file is loaded into memory using many mmap() calls. Superior performance of mmap() is the key for faster loading of programs and booting of the OS. By deferring loading of program fragments (eg. segments of text, data) from disk, till they are really needed is the primary reason for Linux's fast boot-up. Conceptually this is called demand paging.

Problem:

Goal: The goal of this assignment is to implement prefetch and demand paging on a device special file.

Details:

1. Enhance the "`mykmod`" Linux character device driver shared here in the assignment, to implement pseudo character devices. Each device instance represents a pseudo device of 1MB. The 1MB is kernel memory allocated using `kmalloc()`.
2. The driver already supports `open(2)` and `close(2)` system calls using `file_operations`. Enhance it to support `mmap(2)` syscall on the devices. Whenever `mmap(2)` is done by an application program, the 1MB in kernel memory must be mapped to the process's address space.
3. No need to implement `read(2)` and `write(2)` syscalls.
4. Implement `open`, `close`, `fault` operations in `vm_operations_struct` for mmap of the 1MB to support prefetching and demand paging.
5. The device retains data that was written to it till the driver unloads or system reboots. Once mmap'd write is done on a device, mmap'd reading from the device must return whatever was written.
6. In the prefetching case, page faults for the entire 1MB are generated in the context of `mmap(2)` itself.
7. In the demand paging case, page faults are generated when the application starts reading/writing from/to the memory.
8. When a VM segment is opened, npagefaults (number of page faults) should be initialized to zero.

9. When a VM segment is closed, it must print (using printk) the npagefaults received, and re-initialize it to zero.

memutil.cpp: The program opens a given device special file using open(2). Does mmap(2) system call followed by read/write memory operations. At last the program unmaps memory using munmap(2) system call, and closes the file using close(2). Skeleton of this program is shared here with the assignment.

Syntax:

```
Usage: ./util/memutil [options] <devname>
Options:
--operation <optype> : Where optype can be mapread, mapwrite
--message <message>  : Message to be written/read-compare to/from the device memory
--paging <ptype>    : Where ptype can be prefetch or demand
--help              : Show this help
```

Example usages:

Building & Loading the driver:

```
# cd 99_devmmap_paging
# make
# insmod kernel/mykmod.ko
# grep mykmod /proc/devices
243 mykmod
```

Prefetch (Read) :

```
# mknod /tmp/mydev_pR6 c 243 10
# ./util/memutil /tmp/mydev_pR6 --pt prefetch --op mapread

# dmesg | grep -e mykmod_vm_open -e mykmod_vm_close
[  476.174464] mykmod_vm_open: vma=ffff9971ee1dfaf8 npagefaults:0
[  476.178813] mykmod_vm_close: vma=ffff9971ee1dfaf8 npagefaults:256
```

Demand paging (Read):

```
# mknod /tmp/mydev_JZl c 243 11
# ./util/memutil /tmp/mydev_JZl --pt demand --op mapread

# dmesg | grep -e mykmod_vm_open -e mykmod_vm_close
[  476.193956] mykmod_vm_open: vma=ffff9971ee182288 npagefaults:0
[  476.197009] mykmod_vm_close: vma=ffff9971ee182288 npagefaults:128
```

Prefetch paging (Write & Read):

```
# mknod /tmp/mydev_fBc c 243 20
# ./util/memutil /tmp/mydev_fBc --pt prefetch --op mapwrite --op mapread --mes test2

# dmesg | grep -e mykmod_vm_open -e mykmod_vm_close
[  476.209981] mykmod_vm_open: vma=ffff9971ee1d8ca8 npagefaults:0
[  476.211928] mykmod_vm_close: vma=ffff9971ee1d8ca8 npagefaults:256
[  476.212130] mykmod_vm_open: vma=ffff9971ee1d8ca8 npagefaults:0
[  476.214705] mykmod_vm_close: vma=ffff9971ee1d8ca8 npagefaults:256
```

Demand paging (Write & Read):

```
# mknod /tmp/mydev_Ln5 c 243 21
# ./util/memutil /tmp/mydev_Ln5 --pt demand --op mapwrite --op mapread --mes test2

# dmesg | grep -e mykmod_vm_open -e mykmod_vm_close
[  476.225507] mykmod_vm_open: vma=ffff9971f3559360 npagefaults:0
[  476.226311] mykmod_vm_close: vma=ffff9971f3559360 npagefaults:128
[  476.226408] mykmod_vm_open: vma=ffff9971f3559360 npagefaults:0
[  476.228335] mykmod_vm_close: vma=ffff9971f3559360 npagefaults:128
```

Unloading the driver:

```
# rm -f /tmp/mydev*
# rmmod mykmod
```

Automated Testing

runtest.sh: A test script is shared here with the assignment to help you test your driver code.
The script, exercises the following tests in prefetch and demand paging:

a. Test with a single process doing mmap() on a given device special file, followed by read using memory addresses.
b. Test with a single process doing mmap() on a given device special file, followed by write and read using memory addresses.
c. Test (a), (b) using multiple device special files.
   There shouldn't be any interference/data corruptions between different device special files. Writing to one device file should not change contents of other device files.

~~*Autograder scripts will test your driver, with its own test code.*~~

Notes:

1. Group assignment: 2 students per group.
2. ~~Autograder uses~~ Test script is based on Centos 7 VM. You are suggested to code and test on CentOS VM launched on your laptop in a previous Lab session.
3. Do not print any output other than given in the input/output specification as the ~~Autograder~~ test script can only check strict outputs as given above. Any debug prints can confuse the ~~autograder~~ test script and ~~award 0 marks for~~ fail test cases.
4. Do not write any computation logic in main().
5. If the code does not compile or run, no marks will be given.
6. If there are correctness issues, segmentation faults, kernel panics, marks will be deducted from the component that the bug belongs to.
7. You need to Strictly follow the C coding standard of Linux kernel for completing this assignment. You may use this command to check coding style.

   $ indent -nbad -bap -nbc -bbo -hnl -br -brs -c33 -cd33 -ncdb -ce -ci4 -cli0 -d0 -di1 -nfc1 -i8 -ip0 -l80 -lp -npcs -nprs -npsl -sai -saf -saw -ncs -nsc -sob -nfca -cp33 -ss -ts8 -il1 [input-files]

8. Create a README which should contain instructions on how to compile, run the program and sample outputs from your system. Do not submit object files, assembler files, or executables.
9. Marks of this assignment are divided for CODE files, test results, coding style and documentation (report+README).
10. main() must return either EXIT_SUCCESS or EXIT_FAILURE to shell. if exit value is not success (eg. thread or synchronization primitive creation error.), ~~Autograder~~ Test script will not match stdout, stderr.
11. ~~Please do not overuse the Autograder. Try to limit to three attempts per day. Also do not use the Autograder for unit-testing of your code.~~
12. Marks may be deducted for cooking output just to appease the ~~Autograder~~ test script without real problem solution.

Deliverables:

1. A report describing how you completed above task(s). Make sure that your report is technically sound and readable.
2. README file which helps to know list of files submitted and how to compile and run your program

3. Upload mykmod_main.c, mem_util.cpp, README, report to ~~autograder~~ Google classroom. Dont submit binary files.

Learning Accomplishments:

- strace command.
- open(), mmap(), munmap(), close() system calls on regular and device special files.
- Linux pseudo character device driver.
- Demand paging and prefetching in Linux
- pagefault handlers in Linux kernel.

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:

Date:

Signatures: <keep your initials here>

Ref

- Linux Device Drivers, Memory mapping. https://www.oreilly.com/library/view/linux-device-drivers/0596005903/ch15.html

Marks:

Test	Description	Marks
0 - load	# Test 0 : Load the driver and find major no of driver	5
(prefetch) 1a - mapread (demand) 1b - mapread	# Test 1 : Single process reading using mapping	10
(prefetch)	# Test 2 : Single process writing using mapping	10

2a - mapwrite/read (demand) 2b - mapwrite/read		
(prefetch) 3a - mapread 3b - mapread (demand) 3c - mapread 3d - mapread	# Test 3 : Multiple processes reading using mapping	10
(prefetch) 4a - mapwrite/read 4b - mapwrite/read (demand) 4c - mapwrite/read 4d - mapwrite/read	# Test 4 : Multiple processes writing using mapping	10
(prefetch) 5a - mapwrite 5b - mapread (demand) 5c - mapwrite 5d - mapread	# Test 5 : One process writing using mapping and other process reading using mapping	20
(prefetch) 6a - mapread 6b - mapwrite (demand) 6c - mapread 6d - mapwrite	# Test 6 : One process writing to one dev and other process reading from another dev.	20
n - unload	# Test n : Unload the driver and check sanity	5
	Total	90