OPERATING SYSTEMS II (CS3523)

ASSIGNMENT-3 PAGING ON DEVICE MEMORY

REPORT

CS18BTECH11042

CS18BTECH11050

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: Shreyas Jayant Havaldar (CS18BTECH11042)

Manya Goel (CS18BTECH11050)

Date: 20th June 2020

Signature: S.J.H ; M.G

NOTE:

Both the program files “mykmod_main.c” and “memutil.cpp” follow the C coding standard of Linux kernel.

Checked the coding style by the following command:

```
$ 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]
```

Where input files are “mykmod_main.c” and “memutil.cpp”

DESIGN DECISIONS:

mykmod_main.c

- Defined a data structure “*struct mykmod_dev_info*” in order to keep per device information. Each object of the structure stores the information and size for each device respectively.

```
struct mykmod_dev_info {

  char *data;

  size_t size;

};
```

- In order to store the device table we declared a pointer (*dev_table*) to a pointer of type *‘struct mykmod_dev_info’* which will store all the devices and their respective information. The data structure we determined to be appropriate was an array so size 256, as the maximum size is fixed. This allows us to access the device special files with ease and speed.
- Defined a data structure “*struct mykmod_vma_info*” including a pointer to the struct *“mykmod_dev_info”* and to maintain the number of page faults per device.

```
struct mykmod_vma_info {
  struct mykmod_dev_info *devinfo;
  long unsigned int npagefaults;
};
```

- *static int mykmod_init_module(void)*:
  On loading the module, we initialize the dev_table using kmalloc with a memory sufficient enough to store the information of the maximum devices as defined by MYKMOD_MAX_DEVS. The maximum number of devices is 256.
- dev_table = kmalloc(MYKMOD_MAX_DEVS*sizeof(struct mykmod_dev_info), GFP_KERNEL);

```
dev_table = kmalloc(MYKMOD_MAX_DEVS*sizeof(struct mykmod_dev_info), GFP_KERNEL);
```

- *static int mykmod_open(struct inode *inodep, struct file *filep)*
  - We allocate memory for storing device info using kmalloc of size sizeof(struct mykmod_dev_info).
  - Then we allocate 1 MB memory to store device info using kmalloc for the field data and store the info using memcpy() and initialise inodep->i_private with dev info.
  - Then we store the device information in dev_table provided that it does not exceed the limit defined by MYKMOD_MAX_DEVS and initialise filp->private_data with devinfo.

- *static int mykmod_mmap(struct file *filp, struct vm_area_struct *vma)*

  - Initialised vma->vm_ops with &mykmod_vm_ops of type struct vm_operations_struct as defined.
  - Ensuring that the size and the offset do not exceed the size of the device file (ie 1MB), if exceeds function returns -EINVAL.
  - We tested with offsets like 4096*257 and memory mapping more than MYDEV_LEN and correctly obtained the “Invalid argument” error as the mmap failed, as demonstrated in case 10 of our readme.
  - Set up of vma->vm_flags with VM_DONTDUMP and VM_DONTEXPAND to ensure that it is not included in core dump and cannot be expanded with mremap().
  - Intitialisation of vma_>vm_private_data with filp->private_data using a pointer ‘info’ of type mykmod_vma_info.
  - Call the function open in vm_operation_struct, mykmod_vm_open().
- Implementation of Open, Close, Fault operations In ‘vm_operations_struct’ for mmap
  - ***static void mykmod_vm_open(struct vm_area_struct *vma)***
    On opening of the vm segment, *npagefaults* are initialised to 0 and we print the vma and page faults using printk.
  - ***static void mykmod_vm_close(struct vm_area_struct *vma)***
    On closing of the vm segment, vma and pagefaults are printed using printk and we reinitialise *npagefaults* to 0.
  - ***static int mykmod_vm_fault(struct vm_area_struct *vma, struct vm_fault *vmf)***
    - Building of virtual to physical mappings. We initialise a pointer ‘info_fault’ of type mykmod_vma_info with vma->vm_private_data.
    - If the device info stored is not NULL, physical addresses are translated to struct pages using the macro virt_to_page().

- The macro virt_to_page() from sysdep.h, takes the virtual address, converts it to the physical address with __pa(), converts it into an array index by bit shifting it right PAGE_SHIFT bits and indexing into the mem_map by simply adding them together.
- Here the virtual address is given by:
  info_fault->devinfo->data + ((vmf->pgoff + vma->vm_pgoff) * *PAGE_SIZE)*
  Where info_fault->devinfo->data is device information,
  vmf->pgoff is logical page offset based on vma,
  vma->vm_pgoff is the offset of the area in the file, in pages and
  PAGE_SIZE defined as 4096. (Page size is 4KB, as there is total 1024*1024, ie. 1MB memory and 256 pages.)
- We need to account for the both the offsets as they vmf->pgoff takes care of messages that extend over length of a page, and vma->pgoff ensures when the memory mapping is done on a page with a non zero offset, the virtual to physical mapping maintains the correct relation.
- The robustness of our code with regards to offsets is demonstrated in the last test case shown in the README file.
- Then we return the page generated to *vm->page* and print the necessary information using printk.

- *static void mykmod_cleanup_module(void)*
    - After unloading the module, we free the dev_table entries using kfree() until the dev_table stores NULL.
    - Then free the dev_table structure using kfree.

    - Thus there is no memory leak and we ensure there are no dangling pointers or garbage memory.

memutil.cpp

- The program opens a given device special file using open. It does mmap system call followed by read/write memory operations.
- It sets up the mmap flags for the prefetching and demand paging cases. Bitwise OR is used for determining the flags as is the case usually.
- The message is passed with multiple offsets and the program runs successfully for non zero offset mapping as well.
- Looping was implemented to write and read string to the entire device memory.
- In case of OP_MAPREAD, if there is a message to read (msg!=NULL), we compare the data read from device memory(dev_mem) with msg. If the message read from dev_mem is not identical to the msg, “read error” is encountered. We are ensuring that we are reading from the entire device memory by incrementing offsets and reading each successive segment sequentially.
- In case of OP_MAPWRITE, the message (msg) is written iteratively to device memory such that it occupies all available space on that device, here 1MB. We are ensuring that we are writing to the device memory by incrementing offsets and writing each successive segment sequentially.
- We have ensured the code is flexible for writing and reading the messages at various offsets the user presents.
- At last the program unmaps memory using munmap system call, and closes the file using close.

OBSERVATIONS:

PREFETCH

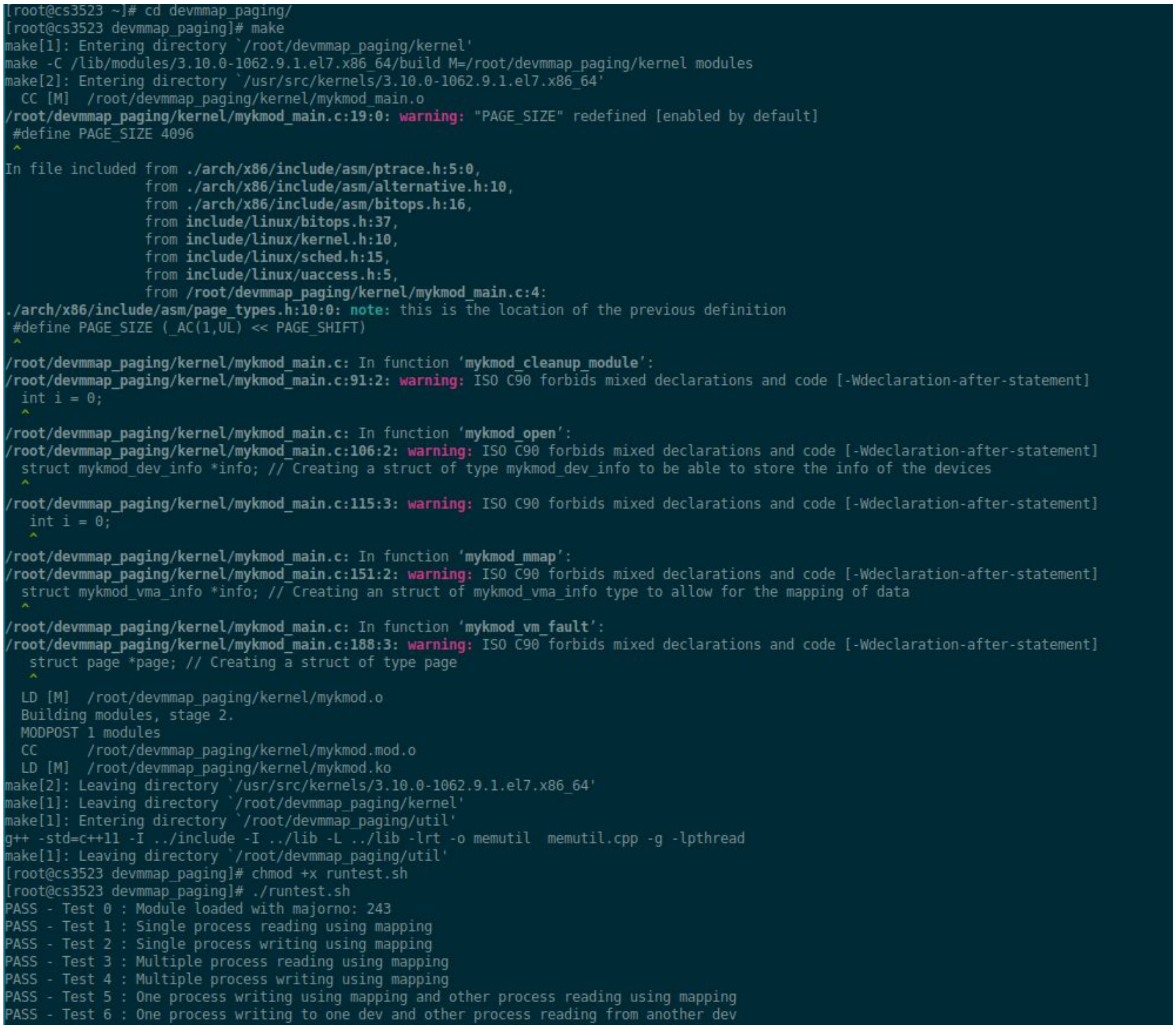
- In this case, page faults for the entire 1MB are generated altogether in the context of mmap itself.
- We set the mmap flags for the mmap() call as:-
  mmap_flags = MAP_SHARED | MAP_POPULATE = 0x01 | 0x08000
- MAP_SHARED flag manages to share this mapping with all other processes that map this object.
- MAP_POPULATE flag populates (prefaults) the page tables for a file mapping, by performing read-ahead on the file. It ensures that later access to the mapping will not be blocked by page faults.
- Irrespective of whether there is a message to be read from the file, the number of page faults is equal to 256.
- In case of non-zero offset, all page faults are still generated together, even if the first page was not mapped, so we always notice 256 page faults.

DEMAND PAGING

- In this case, page faults are generated when the application starts reading/writing from/to the memory.
- We set the standard mmap flag for the mmap() call as:-
  mmap_flags = MAP_SHARED = 0x01
- Where MAP_SHARED flag shares the mapping with all other processes that map this object. Storing to the region and writing to the file are equivalent. The file is not updated until msync or munmap are called.
- When there is no message to read from the file, the number of page faults experienced is 0.

- In case of non-zero offset, all page faults generated do not include the first page as the message was written only from the 2nd page onwards, thus we notice 255 page faults when off = 4096.

TEST SCRIPT RUN:



```
[root@cs3523 ~]# cd devmmap_paging/
[root@cs3523 devmmap_paging]# make
make[1]: Entering directory `/root/devmmap_paging/kernel'
make -C /lib/modules/3.10.0-1062.9.1.el7.x86_64/build M=/root/devmmap_paging/kernel modules
make[2]: Entering directory `/usr/src/kernels/3.10.0-1062.9.1.el7.x86_64'
  CC [M]  /root/devmmap_paging/kernel/mykmod_main.o
/root/devmmap_paging/kernel/mykmod_main.c:19:0: warning: "PAGE_SIZE" redefined [enabled by default]
 #define PAGE_SIZE 4096
 ^
In file included from ./arch/x86/include/asm/ptrace.h:5:0,
                 from ./arch/x86/include/asm/alternative.h:10,
                 from ./arch/x86/include/asm/bitops.h:16,
                 from include/linux/bitops.h:37,
                 from include/linux/kernel.h:10,
                 from include/linux/sched.h:15,
                 from include/linux/uaccess.h:5,
                 from /root/devmmap_paging/kernel/mykmod_main.c:4:
./arch/x86/include/asm/page_types.h:10:0: note: this is the location of the previous definition
 #define PAGE_SIZE (_AC(1,UL) << PAGE_SHIFT)
 ^
/root/devmmap_paging/kernel/mykmod_main.c: In function ‘mykmod_cleanup_module’:
/root/devmmap_paging/kernel/mykmod_main.c:91:2: warning: ISO C90 forbids mixed declarations and code [-Wdeclaration-after-statement]
  int i = 0;
  ^
/root/devmmap_paging/kernel/mykmod_main.c: In function ‘mykmod_open’:
/root/devmmap_paging/kernel/mykmod_main.c:106:2: warning: ISO C90 forbids mixed declarations and code [-Wdeclaration-after-statement]
  struct mykmod_dev_info *info; // Creating a struct of type mykmod_dev_info to be able to store the info of the devices
  ^
/root/devmmap_paging/kernel/mykmod_main.c:115:3: warning: ISO C90 forbids mixed declarations and code [-Wdeclaration-after-statement]
   int i = 0;
   ^
/root/devmmap_paging/kernel/mykmod_main.c: In function ‘mykmod_mmap’:
/root/devmmap_paging/kernel/mykmod_main.c:151:2: warning: ISO C90 forbids mixed declarations and code [-Wdeclaration-after-statement]
  struct mykmod_vma_info *info; // Creating an struct of mykmod_vma_info type to allow for the mapping of data
  ^
/root/devmmap_paging/kernel/mykmod_main.c: In function ‘mykmod_vm_fault’:
/root/devmmap_paging/kernel/mykmod_main.c:188:3: warning: ISO C90 forbids mixed declarations and code [-Wdeclaration-after-statement]
   struct page *page; // Creating a struct of type page
   ^
  LD [M]  /root/devmmap_paging/kernel/mykmod.o
  Building modules, stage 2.
  MODPOST 1 modules
  CC      /root/devmmap_paging/kernel/mykmod.mod.o
  LD [M]  /root/devmmap_paging/kernel/mykmod.ko
make[2]: Leaving directory `/usr/src/kernels/3.10.0-1062.9.1.el7.x86_64'
make[1]: Leaving directory `/root/devmmap_paging/kernel'
make[1]: Entering directory `/root/devmmap_paging/util'
g++ -std=c++11 -I ../include -I ../lib -L ../lib -lrt -o memutil  memutil.cpp -g -lpthread
make[1]: Leaving directory `/root/devmmap_paging/util'
[root@cs3523 devmmap_paging]# chmod +x runtest.sh
[root@cs3523 devmmap_paging]# ./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
```