Berkay Barlas 54512
Ege Onat Özsüer 60210

Project 3 Report

In this report we will be documenting our Project 3 implementation and observations. We have completed all parts of the project.
To compile program use:

make virtmem

To run the program use:

./virtmem.o BACKING_STORE.bin addresses.txt -p 0

Part 1

In part 1, we were given a program that does demand paging from a virtual memory to physical memory. We are asked to modify it so that our memory size is only big enough for 64 pages, and therefore we need to implement a page replacement algorithm.

First, we modified the defined macros, so that the memory size and the number of pages are not both 256 anymore. We kept the page count at 256 but added a new variable PAGE_FRAMES and made it 64. We then defined our memory size to be PAGE_FRAMES * PAGE_SIZE. We also defined a new macro named BACKING_SIZE which is equal to PAGES * PAGE_SIZE. We modified the code accordingly, replacing some of the PAGES variables with PAGE_FRAMES, and MEMORY_SIZE with BACKING_SIZE. These were necessary since the size of the memory and the size of the backing_store, which signifies our disk are not the same anymore.

We then had to implement FIFO and LRU replacement strategies. We started with the FIFO algorithm since it is simpler. For this, we used the already existing free_page variable to keep track of the next page frame we wish to write in. In our implementation free_page variable starts from 0 and goes up to 63, and afterwards it becomes 0 again. It is declared in the main function but updated in the fifoPageSelect function. If we always write to the page frame pointed by this variable FIFO strategy is complete.

Next, we implemented the LRU replacement algorithm. Here, until the memory fills up for the first time, we behave like FIFO, placing pages in the empty page frames. But once the memory is filled, we select pages to evict by looking at which page in our memory has been accessed least recently. To make it work, we used a counting table, recording

the last access time of all the pages we have in our virtual memory. Instead of real time, we treated each address read as one step, and recorded the last step in which we accessed the page. This table is updated every time a page is accessed, in the main function. To find the least accessed page in memory, we iterate over this table, and among the pages that are currently in memory, we find the one with the oldest access time. We then pick this physical address to replace with our new page.

Both of these functions also need a mechanism to update the page table, the given implementation only adds pages to the page table, but does not remove older pages when they are replaced. To encapsulate the job of copying a page from disk to memory and then updating the page table, we wrote another function, called putPageInMemory. This function takes the logical and physical addresses as inputs and copies the page from the proper disk location to the proper location in our memory. Then we check if there are any entries in the page table made invalid by this action, and if so update them accordingly. Finally, we update the page table so that the entry for the logical address given points to the newly allocated physical address.

The last function we added was the cmdline function, which we used to read the -p option, which can be selected as 0 or 1. If p is given as 0, we use FIFO, if it is 1, we use LRU. Since the given code to read the BACKING_STORE.bin and the address files are hard coded to take the first and second command line arguments, our -p option needs to come last, or the program does not run properly.

All these functions are called from the main function, and we think their usages are intuitive and self-explanatory. We also tried to comment our implementation as much as possible. To compile the program, you can use the command “make virtmem” and use our makefile, or just use gcc without any special options.

Part 2

A **file** in the file system is a link to an inode.
A hard link creates another file with a link to the same underlying inode.
When you delete a file it removes one link to the underlying inode. The inode is only deleted when all links to the inode have been deleted.

The **inode** is a data structure in a Unix-style file system which describes a filesystem object such as a file or a directory.

Man Page for unlink()

unlink() deletes a name from the file system. If that name was the last link to a file and no processes have the file open the file is deleted and the space it was using is made available for reuse.

- If the name was the last link to a file but any processes still have the file open the file will remain in existence until the last file descriptor referring to it is closed.\
- If the name referred to a symbolic link the link is removed.
- If the name referred to a socket, fifo or device the name for it is removed but processes which have the object open may continue to use it.

Man Page for rm()

rm() deletes a name from the file system. It calls unlink(2) for files, and rmdir(2) for directories.

- If the removed name was the last link to a file and no processes have the file open, the file is deleted and the space it was using is made available for reuse.
- If the name was the last link to a file, but any processes still have the file open, the file will remain in existence until the last file descriptor referring to it is closed.
- If the name referred to a symbolic link, the link is removed.
- If the name referred to a socket, FIFO, or device, the name is removed, but processes which have the object open may continue to use it.

a) Hard Link with text1.txt and text2.txt

What are the inode values of file1.txt and file2.txt? Are they the same or different?

- Inode values of file1.txt and file2.txt are 3702263 and both them has same value.

Do the two files have the same—or different— contents?

- The contents of file1.txt and file2.txt are same.

Next, edit file2.txt and change its contents. After you have done so, examine the contents of file1.txt . Are the contents of file1.txt and file2.txt the same or different?

- After the contents of file2.txt changed, the contents of file1.txt changed too. Their contents are still same.

After rm file1.txt Does file2.txt still exist as well?

* After removing file1.txt, file2.txt is still exists.

b) Soft Link with text3.txt and text4.txt

Are the inodes the same, or is each unique?

- Inode values of text3.txt and text4.txt are not same. text3.txt has inode value of 3702264. text4.txt has inode value of 3681131.

Next, edit the contents of file4.txt. Have the contents of file3.txt been altered as well?

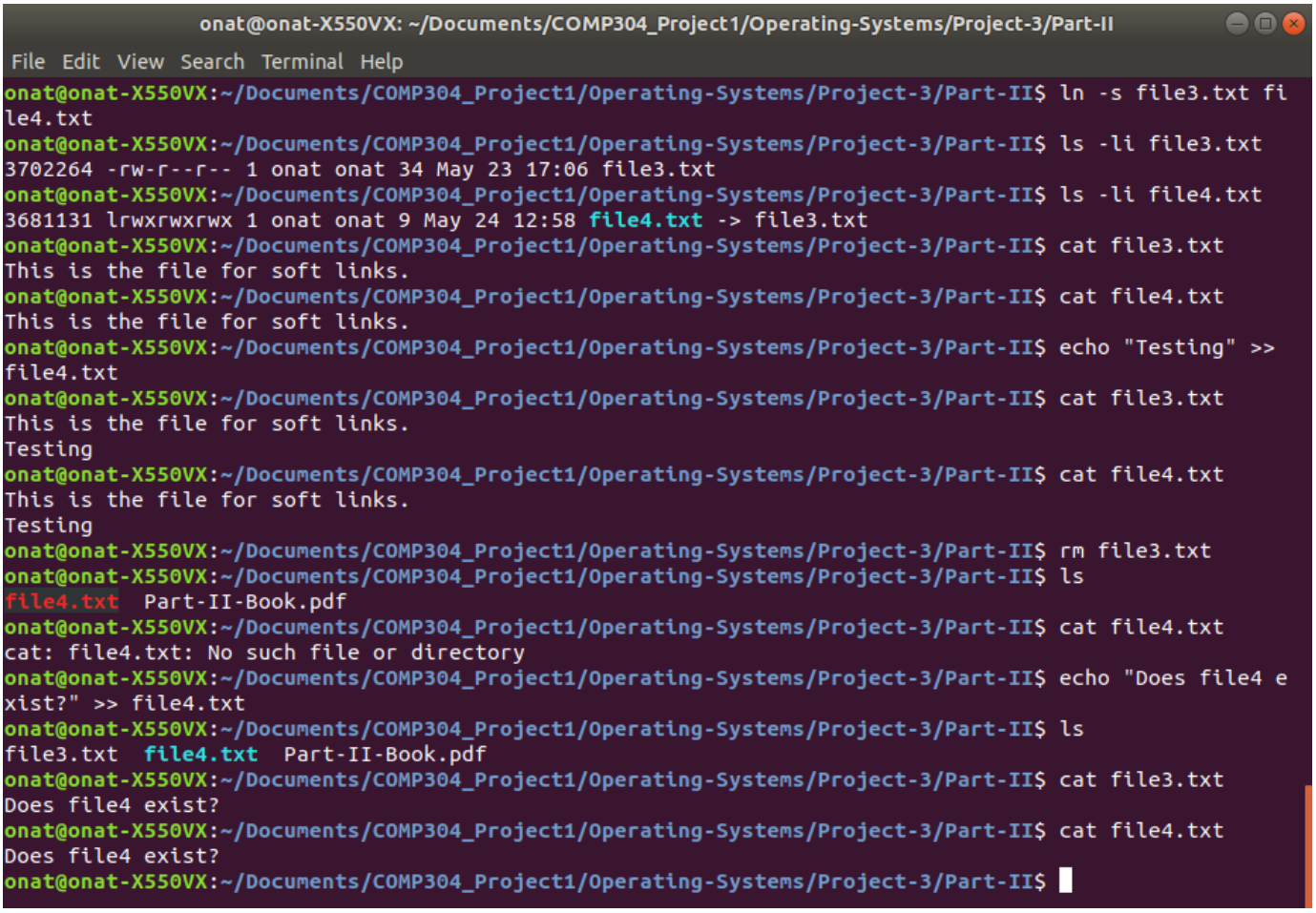
- After the contents of file4.txt changed, the contents of file3.txt changed too. Their contents are still same.

After delete file3.txt what happens when you attempt to edit file4.txt.

- After the deletion of file3.txt, the contents of file4.txt is cleared.
- After editing file4.txt, a file named file3.txt is created with the same content of file3.txt

Screenshots

```
onat@onat-X550VX:~/Documents/COMP304_Project1/Operating-Systems/Project-3/Part-II$ ls -li file1.txt
3702263 -rw-r--r-- 1 onat onat 33 May 23 17:06 file1.txt
onat@onat-X550VX:~/Documents/COMP304_Project1/Operating-Systems/Project-3/Part-II$ ln file1.txt file2.txt
onat@onat-X550VX:~/Documents/COMP304_Project1/Operating-Systems/Project-3/Part-II$ ls -li file2.txt
3702263 -rw-r--r-- 2 onat onat 33 May 23 17:06 file2.txt
onat@onat-X550VX:~/Documents/COMP304_Project1/Operating-Systems/Project-3/Part-II$ rm file1.txt
onat@onat-X550VX:~/Documents/COMP304_Project1/Operating-Systems/Project-3/Part-II$ ls
file2.txt  file3.txt  Part-II-Book.pdf
onat@onat-X550VX:~/Documents/COMP304_Project1/Operating-Systems/Project-3/Part-II$ cat file2.txt
This is the first example file.
onat@onat-X550VX:~/Documents/COMP304_Project1/Operating-Systems/Project-3/Part-II$ strace rm file2.txt
execve("/bin/rm", ["rm", "file2.txt"], 0x7ffd6e261c28 /* 63 vars */) = 0
brk(NULL)                               = 0x556643822000
access("/etc/ld.so.nohwcap", F_OK)      = -1 ENOENT (No such file or directory)
access("/etc/ld.so.preload", R_OK)      = -1 ENOENT (No such file or directory)
openat(AT_FDCWD, "/etc/ld.so.cache", O_RDONLY|O_CLOEXEC) = 3
fstat(3, {st_mode=S_IFREG|0644, st_size=109765, ...}) = 0
mmap(NULL, 109765, PROT_READ, MAP_PRIVATE, 3, 0) = 0x7f3b809eb000
close(3)                                = 0
access("/etc/ld.so.nohwcap", F_OK)      = -1 ENOENT (No such file or directory)
openat(AT_FDCWD, "/lib/x86_64-linux-gnu/libc.so.6", O_RDONLY|O_CLOEXEC) = 3
read(3, "\177ELF\2\1\1\3\0\0\0\0\0\0\0\0\3\0>\0\1\0\0\0\260\34\2\0\0\0\0\0"..., 832) = 832
fstat(3, {st_mode=S_IFREG|0755, st_size=2030544, ...}) = 0
mmap(NULL, 8192, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_ANONYMOUS, -1, 0) = 0x7f3b809e9000
mmap(NULL, 4131552, PROT_READ|PROT_EXEC, MAP_PRIVATE|MAP_DENYWRITE, 3, 0) = 0x7f3b803ee000
mprotect(0x7f3b805d5000, 2097152, PROT_NONE) = 0
mmap(0x7f3b807d5000, 24576, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_FIXED|MAP_DENYWRITE, 3, 0x1e7000) = 0x7f3b807d5000
mmap(0x7f3b807db000, 15072, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_FIXED|MAP_ANONYMOUS, -1, 0) = 0x7f3b807db000
close(3)                                = 0
arch_prctl(ARCH_SET_FS, 0x7f3b809ea540) = 0
mprotect(0x7f3b807d5000, 16384, PROT_READ) = 0
mprotect(0x55664202c000, 4096, PROT_READ) = 0
mprotect(0x7f3b80a06000, 4096, PROT_READ) = 0
munmap(0x7f3b809eb000, 109765)          = 0
brk(NULL)                               = 0x556643822000
brk(0x556643843000)                     = 0x556643843000
openat(AT_FDCWD, "/usr/lib/locale/locale-archive", O_RDONLY|O_CLOEXEC) = 3
fstat(3, {st_mode=S_IFREG|0644, st_size=11731760, ...}) = 0
mmap(NULL, 11731760, PROT_READ, MAP_PRIVATE, 3, 0) = 0x7f3b7f8bd000
close(3)                                = 0
ioctl(0, TCGETS, {B38400 opost isig icanon echo ...}) = 0
newfstatat(AT_FDCWD, "file2.txt", {st_mode=S_IFREG|0644, st_size=33, ...}, AT_SYMLINK_NOFOLLOW) = 0
geteuid()                               = 1000
newfstatat(AT_FDCWD, "file2.txt", {st_mode=S_IFREG|0644, st_size=33, ...}, AT_SYMLINK_NOFOLLOW) = 0
faccessat(AT_FDCWD, "file2.txt", W_OK)  = 0
unlinkat(AT_FDCWD, "file2.txt", 0)      = 0
lseek(0, 0, SEEK_CUR)                   = -1 ESPIPE (Illegal seek)
close(0)                                = 0
close(1)                                = 0
close(2)                                = 0
exit_group(0)                           = ?
onat@onat-X550VX:~/Documents/COMP304_Project1/Operating-Systems/Project-3/Part-II$
```


onat@onat-X550VX: ~/Documents/COMP304_Project1/Operating-Systems/Project-3/Part-II
File Edit View Search Terminal Help
onat@onat-X550VX:~/Documents/COMP304_Project1/Operating-Systems/Project-3/Part-II$ ln -s file3.txt file4.txt
onat@onat-X550VX:~/Documents/COMP304_Project1/Operating-Systems/Project-3/Part-II$ ls -li file3.txt
3702264 -rw-r--r-- 1 onat onat 34 May 23 17:06 file3.txt
onat@onat-X550VX:~/Documents/COMP304_Project1/Operating-Systems/Project-3/Part-II$ ls -li file4.txt
3681131 lrwxrwxrwx 1 onat onat 9 May 24 12:58 file4.txt -> file3.txt
onat@onat-X550VX:~/Documents/COMP304_Project1/Operating-Systems/Project-3/Part-II$ cat file3.txt
This is the file for soft links.
onat@onat-X550VX:~/Documents/COMP304_Project1/Operating-Systems/Project-3/Part-II$ cat file4.txt
This is the file for soft links.
onat@onat-X550VX:~/Documents/COMP304_Project1/Operating-Systems/Project-3/Part-II$ echo "Testing" >> file4.txt
onat@onat-X550VX:~/Documents/COMP304_Project1/Operating-Systems/Project-3/Part-II$ cat file3.txt
This is the file for soft links.
Testing
onat@onat-X550VX:~/Documents/COMP304_Project1/Operating-Systems/Project-3/Part-II$ cat file4.txt
This is the file for soft links.
Testing
onat@onat-X550VX:~/Documents/COMP304_Project1/Operating-Systems/Project-3/Part-II$ rm file3.txt
onat@onat-X550VX:~/Documents/COMP304_Project1/Operating-Systems/Project-3/Part-II$ ls
file4.txt Part-II-Book.pdf
onat@onat-X550VX:~/Documents/COMP304_Project1/Operating-Systems/Project-3/Part-II$ cat file4.txt
cat: file4.txt: No such file or directory
onat@onat-X550VX:~/Documents/COMP304_Project1/Operating-Systems/Project-3/Part-II$ echo "Does file4 exist?" >> file4.txt
onat@onat-X550VX:~/Documents/COMP304_Project1/Operating-Systems/Project-3/Part-II$ ls
file3.txt file4.txt Part-II-Book.pdf
onat@onat-X550VX:~/Documents/COMP304_Project1/Operating-Systems/Project-3/Part-II$ cat file3.txt
Does file4 exist?
onat@onat-X550VX:~/Documents/COMP304_Project1/Operating-Systems/Project-3/Part-II$ cat file4.txt
Does file4 exist?
onat@onat-X550VX:~/Documents/COMP304_Project1/Operating-Systems/Project-3/Part-II$