COMP304 – Operating Systems – Assignment 3
Kaan Turkmen

Problem 1

External fragmentation: If there are free memory blocks in the memory, however, if the new memory requirement could not satisfy because of the places of the free memory blocks that is called external fragmentation.

Contiguous allocation: Contiguous allocation is a memory management strategy to allocate single section in the memory. Since it is single piece of allocation, putting and removing from the memory can create holes and thus we can observe external fragmentation. Code sharing among processes is not possible in this strategy due to the reason of each process have their own memory space.

Paging: In paging, we have additional structure called pages. Pages are fixed size storage that maps virtual memory address to physical memory address locations (frames). Since pages sized are fixed, we can prevent external fragmentation by selecting sizes of pages and the main memory frames. Code sharing among processes are possible since we can use copy of the pages and due to the mapping to the same place.

Segmentation: In segmentation, there is a segment table to keep track of base and limit addresses. When CPU calls it with segment index and offset, it is being checked if it's a valid address and if it is, it is being directed to the physical address by the help of segment table. With the help of segmentation and not knowing which address to start at the first place, we are being sure that no one can access data by predicting their addresses. For example, we know that OS located in beginning addresses, however, with the help of segmentation, we can reduce predictability to the malicious user. External fragmentation does happen because of splitting memory to the smaller chunks. In code sharing among processes, segmentation allows to do that because we are using mapping same as paging and thus, we have ability to use mapping the address spaces to the same place and do code sharing.

Problem 2

32-bit virtual addresses and page size of 8 KB.

Virtual Address Space = 2^32 bytes.

1 Page Size = 8 KB = 2^13 bytes.

1 Table Entry Size = 4 bytes = 2^2 bytes.

Physical Addresses = 2^28 bits = 2^26 bytes.

Page Count = 2^32 / 2^13 = 2^19 pages.

a) Space Needed = 2^2 * 2^19 = 2^21 bytes.

b) Physical address is 28 bits as noted above. Since every bit can either 1 or 0, we can produce 2^28 different memory locations. Each memory locations are 1 byte. So that if we do a conversion, we will obtain physical memory address size as 256 MB.

c) Physical Address Size / 1 Page Size = 2^28 / 2^13 = 2^15.

Problem 3

Reference String: 1, 2, 3, 4, 2, 1, 5, 2, 1, 2, 3.

LRU Replacement: If the frames are full, swap new one with least recent used value.

Adding 1: 1. **PF**

Adding 2: 1, 2. **PF**

Adding 3: 1, 2, 3. **PF**

Adding 4: 4, 2, 3. **PF**

Adding 2: 4, 2, 3. (No Change.)

Adding 1: 4, 2, 1. **PF**

Adding 5: 5, 2, 1. **PF**

Adding 2: 5, 2, 1. (No Change.)

Adding 1: 5, 2, 1. (No Change.)

Adding 2: 5, 2, 1. (No Change.)

Adding 3: 3, 2, 1. **PF**

Total Page Fault in LRU Replacement: 7 times.

FIFO Replacement: If the frames are full, swap new one with first in value.

Adding 1: 1. **PF**

Adding 2: 1, 2. **PF**

Adding 3: 1, 2, 3. **PF**

Adding 4: 4, 2, 3. **PF**

Adding 2: 4, 2, 3. (No Change.)

Adding 1: 4, 1, 3. **PF**

Adding 5: 4, 1, 5. **PF**

Adding 2: 2, 1, 5. **PF**

Adding 1: 2, 1, 5. (No Change.)

Adding 2: 2, 1, 5. (No Change.)

Adding 3: 2, 3, 5. **PF**

Total Page Fault in LRU Replacement: 8 times.

Optimal Replacement: If the frames are full, replace the new value with the value which is not used for the longest time.

Adding 1: 1. **PF**

Adding 2: 1, 2. **PF**

Adding 3: 1, 2, 3. **PF**

Adding 4: 1, 2, 4. **PF**

Adding 2: 1, 2, 4. (No Change.)

Adding 1: 1, 2, 4. (No Change.)

Adding 5: 1, 2, 5. **PF**

Adding 2: 1, 2, 5. (No Change.)

Adding 1: 1, 2, 5. (No Change.)

Adding 2: 1, 2, 5. (No Change.)

Adding 3: 3, 2, 5. **PF**

Total Page Fault in LRU Replacement: 6 times.

Problem 4

P1: 2, 5, 7, 4, 4, 1, 2, 3, 7, 3, 3.

P2: 1, 1, 1, 3, 3, 4, 4, 3, 5, 6, 7.

P3: 4, 9, 3, 2, 9, 8, 7, 7, 7, 1, 2.

$\Delta = 7$.

A)

P1: 2, 5, 7, 4, 4, 1, 2, 3, 7, 3, 3.

W = {1, 2, 3, 4, 7} Size = 5.

P2: 1, 1, 1, 3, 3, 4, 4, 3, 5, 6, 7.

W = {3, 4, 5, 6} Size = 4.

P3: 4, 9, 3, 2, 9, 8, 7, 7, 7, 1, 2.

W = {1, 2, 7, 8, 9} Size = 5.

B)

Let D is the total demand for the frames and m is the total number of available frames.

If D > m, thrashing will occur.

In the question, m is given as 10 and we can find D from the part A.

So that, m = 10 and D = 5 + 4 + 5 = 14.

In conclusion, we found that D > m. That means trashing will occur at the time 9.

C)

P1: 2, 5, 7, 4, 4, 1, 2, 3, 7, 3, 3.

W = {1, 2, 3, 4, 5, 7} Size = 6.

P2: 1, 1, 1, 3, 3, 4, 4, 3, 5, 6, 7.

W = {1, 3, 4} Size = 3.

P3: 4, 9, 3, 2, 9, 8, 7, 7, 7, 1, 2.

W = {2, 3, 7, 8, 9} Size = 5.

So that, D = 6 + 3 + 5 = 14.

If we want system does not suffer from trashing, we should satisfy D <= m.

In conclusion, m should be chosen as 14 <= m. Furthermore, total number of frames available should be greater than or equal to the 14. (m >= 14)

Problem 5

```
kaan@katurkmen:~/Developer/Problem5$ ls -li file1.txt
12714712 -rw-r--r-- 1 kaan kaan 33 May  5 17:11 file1.txt
kaan@katurkmen:~/Developer/Problem5$ ln file1.txt file2.txt
kaan@katurkmen:~/Developer/Problem5$ ls -li file2.txt
12714712 -rw-r--r-- 2 kaan kaan 33 May  5 17:11 file2.txt
kaan@katurkmen:~/Developer/Problem5$ cat file1.txt
This is the first example file.
kaan@katurkmen:~/Developer/Problem5$ cat file2.txt
This is the first example file.
kaan@katurkmen:~/Developer/Problem5$ vim file2.txt
kaan@katurkmen:~/Developer/Problem5$ cat file2.txt
This is the second example file.
kaan@katurkmen:~/Developer/Problem5$ cat file1.txt
This is the second example file.
kaan@katurkmen:~/Developer/Problem5$ rm file1.txt
kaan@katurkmen:~/Developer/Problem5$ ls
Book.pdf  file2.txt  file3.txt
kaan@katurkmen:~/Developer/Problem5$ strace rm file2.txt
execve("/usr/bin/rm", ["rm", "file2.txt"], 0x7ffdf704c548 /* 59 vars */) = 0
brk(NULL)                               = 0x55d5712ca000
arch_prctl(0x3001 /* ARCH_??? */, 0x7fff37b5b4d0) = -1 EINVAL (Invalid argument)
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=104593, ...}) = 0
mmap(NULL, 104593, PROT_READ, MAP_PRIVATE, 3, 0) = 0x7fae1dda8000
close(3)                                = 0
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\360q\2\0\0\0\0\0"..., 832) = 832
pread64(3, "\6\0\0\0\4\0\0\0@\0\0\0\0\0\0\0@\0\0\0\0\0\0\0@\0\0\0\0\0\0\0"..., 784, 64) = 784
pread64(3, "\4\0\0\0\20\0\0\0\5\0\0\0GNU\0\2\0\0\300\4\0\0\0\3\0\0\0\0\0\0\0", 32, 848) = 32
pread64(3, "\4\0\0\0\24\0\0\0\3\0\0\0GNU\0\t\233\222%\274\260\320\31\331\326\10\204\276X>\263"..., 68, 880) = 68
fstat(3, {st_mode=S_IFREG|0755, st_size=2029224, ...}) = 0
mmap(NULL, 8192, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_ANONYMOUS, -1, 0) = 0x7fae1dda6000
pread64(3, "\6\0\0\0\4\0\0\0@\0\0\0\0\0\0\0@\0\0\0\0\0\0\0@\0\0\0\0\0\0\0"..., 784, 64) = 784
pread64(3, "\4\0\0\0\20\0\0\0\5\0\0\0GNU\0\2\0\0\300\4\0\0\0\3\0\0\0\0\0\0\0", 32, 848) = 32
pread64(3, "\4\0\0\0\24\0\0\0\3\0\0\0GNU\0\t\233\222%\274\260\320\31\331\326\10\204\276X>\263"..., 68, 880) = 68
mmap(NULL, 2036952, PROT_READ, MAP_PRIVATE|MAP_DENYWRITE, 3, 0) = 0x7fae1dbb4000
mprotect(0x7fae1dbd9000, 1847296, PROT_NONE) = 0
mmap(0x7fae1dbd9000, 1540096, PROT_READ|PROT_EXEC, MAP_PRIVATE|MAP_FIXED|MAP_DENYWRITE, 3, 0x25000) = 0x7fae1dbd9000
mmap(0x7fae1dd51000, 303104, PROT_READ, MAP_PRIVATE|MAP_FIXED|MAP_DENYWRITE, 3, 0x19d000) = 0x7fae1dd51000
mmap(0x7fae1dd9c000, 24576, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_FIXED|MAP_DENYWRITE, 3, 0x1e7000) = 0x7fae1dd9c000
mmap(0x7fae1dda2000, 13528, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_FIXED|MAP_ANONYMOUS, -1, 0) = 0x7fae1dda2000
close(3)                                = 0
arch_prctl(ARCH_SET_FS, 0x7fae1dda7580) = 0
mprotect(0x7fae1dd9c000, 12288, PROT_READ) = 0
mprotect(0x55d56fb40000, 4096, PROT_READ) = 0
mprotect(0x7fae1ddef000, 4096, PROT_READ) = 0
munmap(0x7fae1dda8000, 104593)          = 0
brk(NULL)                               = 0x55d5712ca000
brk(0x55d5712eb000)                     = 0x55d5712eb000
openat(AT_FDCWD, "/usr/lib/locale/locale-archive", O_RDONLY|O_CLOEXEC) = 3
fstat(3, {st_mode=S_IFREG|0644, st_size=8500896, ...}) = 0
mmap(NULL, 8500896, PROT_READ, MAP_PRIVATE, 3, 0) = 0x7fae1d398000
close(3)                                = 0
ioctl(0, TCGETS, {B38400 opost isig icanon echo ...}) = 0
newfstatat(AT_FDCWD, "file2.txt", {st_mode=S_IFREG|0644, st_size=34, ...}, AT_SYMLINK_NOFOLLOW) = 0
geteuid()                               = 1000
newfstatat(AT_FDCWD, "file2.txt", {st_mode=S_IFREG|0644, st_size=34, ...}, 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)                           = ?
+++ exited with 0 +++
kaan@katurkmen:~/Developer/Problem5$ ln -s file3.txt file4.txt
kaan@katurkmen:~/Developer/Problem5$ ls -li file*.txt
12714711 -rw-r--r-- 1 kaan kaan 34 May  5 17:11 file3.txt
12714712 lrwxrwxrwx 1 kaan kaan  9 May 10 15:15 file4.txt -> file3.txt
kaan@katurkmen:~/Developer/Problem5$ vim file4.txt
kaan@katurkmen:~/Developer/Problem5$ cat file4.txt
This is the second file for soft links.
kaan@katurkmen:~/Developer/Problem5$ cat file3.txt
This is the second file for soft links.
kaan@katurkmen:~/Developer/Problem5$ rm file3.txt
kaan@katurkmen:~/Developer/Problem5$ ls
Book.pdf  file4.txt
kaan@katurkmen:~/Developer/Problem5$ vim file4.txt
kaan@katurkmen:~/Developer/Problem5$ vim file4.txt
kaan@katurkmen:~/Developer/Problem5$ ls
Book.pdf  file3.txt  file4.txt
kaan@katurkmen:~/Developer/Problem5$
```

Inode values of file1.txt and file2.txt are the same. Also, their content is the same. After we edit file2.txt we observed that file1.txt's content is also changed. However, when we deleted file1.txt, file2.txt is not deleted. When we straced the rm file2.txt, we have observed system call unlinkat to remove the file2.txt is used.

Inode values of file3.txt and file4.txt are different. Each one has unique inode number. After editing the file4.txt, file3.txt's content is also changed. After we delete file3.txt, file4.txt became red with highlighted color which can be seen from above screenshot. When we tried to write content to it (edit), it created file3.txt along with itself.