**Problem Set 5 – Memory/mmap Test**

1. **Source Code - mmap.c**

#include <stdio.h>

#include <stdlib.h>

#include <errno.h>

#include <fcntl.h>

#include <string.h>

#include <signal.h>

#include <unistd.h>

#include <setjmp.h>

#include <sys/types.h>

#include <sys/stat.h>

#include <sys/mman.h>

jmp\_buf env;

void signalHandler(int sig){

fprintf(stderr, "Signal #%d, \"%s\" received!\n", sig, strsignal(sig));

if(sig!=SIGCHLD && sig!=SIGCONT && sig!=SIGWINCH) exit(sig);

else fprintf(stderr, "Non-critical Signal Ignored\n");

}

void errorPrompt(){

fprintf(stderr, "Usage: mmap [123456] (test\_no)\n");

exit(255);

}

int createFile(size\_t length){

char \*execstring;

size\_t sz;

sz = snprintf(NULL, 0, "dd if=/dev/urandom of=random\_text.txt bs=%zu count=1", length);

if((execstring = (char\*) malloc(sz+1)) == NULL){

fprintf(stderr, "Error malloc failed: %s\n", strerror(errno));

exit(255);

}

snprintf(execstring, sz+1, "dd if=/dev/urandom of=random\_text.txt bs=%zu count=1", length);

if(system(execstring) == 01){

fprintf(stderr, "Failed to create random text file: %s\n",strerror(errno));

exit(255);

}

int fd = open("random\_text.txt", O\_RDWR, 0666);

if(fd == -1){

fprintf(stderr, "Failed to open random text file: %s\n", strerror(errno));

exit(255);

}

return fd;

}

int test1(){

char \*map;

size\_t length = 8195;

int fd = createFile(length);

if((map = mmap(NULL, length, PROT\_READ, MAP\_SHARED, fd, 0)) == MAP\_FAILED){

fprintf(stderr, "Error mapping file to memory: %s\n", strerror(errno));

exit(255);

}

fprintf(stderr,"map[3]==\'%c\'\n", map[3]);

fprintf(stderr,"writing a \'%c\'\n", map[4]);

if(setjmp(env) == 0){

map[0] = '0';

}

if(munmap(map,length) == -1){

fprintf(stderr, "Error unmapping file to memory: %s\n", strerror(errno));

exit(255);

}

if(close(fd) == -1){

fprintf(stderr, "Error closing file descriptor: %s\n", strerror(errno));

exit(255);

}

exit(0);

}

int test23(int flags){

char \*map, \*buffer;

size\_t length = 8195;

int fd = createFile(length);

if((map = mmap(NULL, length, PROT\_READ|PROT\_WRITE, flags, fd, 0)) == MAP\_FAILED){

fprintf(stderr, "Error mapping file to memory: %s\n", strerror(errno));

exit(255);

}

char \*test\_str = "CCCCC";

int i = 0, offset = 25, wbytes = strlen(test\_str);

fprintf(stderr,"Altering %d bytes to offset %d from %p stored in memory map.\n", wbytes, offset, map);

for(i = 0; i < wbytes; i++) map[offset+i] = test\_str[i];

if(lseek(fd,offset,SEEK\_SET) == -1){

fprintf(stderr, "Error lseek on %d: %s\n", fd, strerror(errno));

exit(255);

}

if((buffer = malloc(wbytes+1)) == NULL){

fprintf(stderr, "Error in malloc: %s\n", strerror(errno));

exit(255);

}

if(read(fd, buffer, wbytes) == -1){

fprintf(stderr, "Error reading from %d: %s\n", fd, strerror(errno));

exit(255);

}

int ans = strncmp(buffer, test\_str, wbytes);

printf("Update to the mapped memory with %s %s\n", flags==MAP\_SHARED?"MAP\_SHARED":"MAP\_PRIVATE", ans? "not visible": "is visible.");

if(munmap(map,length) == -1){

fprintf(stderr, "Error unmapping file to memory: %s\n", strerror(errno));

exit(255);

}

if(close(fd) == -1){

fprintf(stderr, "Error closing file descriptor: %s\n", strerror(errno));

exit(255);

}

ans? exit(0):exit(1);

}

int test45(int testno){

char \*map;

size\_t length = 6144;

int fd = createFile(length);

int result;

struct stat sb;

if(fstat(fd, &sb) == -1){

fprintf(stderr, "Error stat on file %d: %s\n", fd, strerror(errno));

exit(255);

}

int orig\_size = sb.st\_size;

if((map = mmap(NULL, length, PROT\_READ|PROT\_WRITE, MAP\_SHARED, fd, 0)) == MAP\_FAILED){

fprintf(stderr, "Error mapping file to memory: %s\n", strerror(errno));

exit(255);

}

char \*test\_str4 = "XYZ";

int i =0, offset = length, wbytes = strlen(test\_str4);

for(i=0; i < wbytes; i++) map[offset+i] = test\_str4[i];

if(fstat(fd, &sb) == -1){

fprintf(stderr, "Error stat on file %d: %s\n", fd, strerror(errno));

exit(255);

}

int new\_size = sb.st\_size;

result = new\_size != orig\_size;

if(testno == 5){

char \*buffer, \*test\_str5 = "ABC";

int i = 0, bytes5 = 16, offset = length, wbytes = strlen(test\_str5);

lseek(fd,bytes5, SEEK\_END);

if(write(fd, test\_str5, strlen(test\_str5)) == -1){

fprintf(stderr, "Error writing %s to file %d: %s\n", test\_str5, fd, strerror(errno));

exit(255);

}

if(lseek(fd,offset,SEEK\_SET) == -1){

fprintf(stderr, "Error lseek on %d: %s'n", fd, strerror(errno));

exit(255);

}

if((buffer = malloc(bytes5 + wbytes + 1)) == NULL){

fprintf(stderr, "Error malloc failed: %s\n", strerror(errno));

exit(255);

}

if(read(fd, buffer, bytes5+wbytes\*2) == -1){

fprintf(stderr, "Error reading from %d: %s\n", fd, strerror(errno));

exit(255);

}

result = !strncmp(buffer, test\_str4, strlen(test\_str4));

}

if(munmap(map, length) == -1){

fprintf(stderr, "Error unmapping file to memory: %s\n", strerror(errno));

exit(255);

}

if(close(fd) == -1){

fprintf(stderr, "Error closing file: %s\n", strerror(errno));

exit(255);

}

if(testno == 4){

if(result) {

printf("Size of the file from stat changed\n");

exit(0);

}else{

printf("Size of the file from stat did not change\n");

exit(1);

}

}else{

if(result){

printf("Byte written is visible\n");

exit(0);

}else{

printf("Byte written is not visible\n");

exit(1);

}

}

}

int test6(){

char \*map;

size\_t length = 20;

int fd = createFile(length);

if((map = mmap(NULL, 8192, PROT\_READ, MAP\_SHARED, fd, 0)) == MAP\_FAILED){

fprintf(stderr, "Error mapping file to memory: %s\n", strerror(errno));

exit(255);

}

fprintf(stderr, "Attempting to access memory from first page.\n");

if(sigsetjmp(env,1) == 0) printf("Reading a byte from offset 2048: %d\n", map[2048]);

fprintf(stderr, "Attempting to access memory from second page.\n");

if(sigsetjmp(env,1) == 0) printf("Reading a byte from offset 4096: %d\n", map[4096]);

if(munmap(map, length) == -1){

fprintf(stderr, "Error unmapping file to memory: %s\n", strerror(errno));

exit(255);

}

if(close(fd) == -1){

fprintf(stderr, "Error closing file: %s\n", strerror(errno));

exit(255);

}

exit(0);

}

int main(int argc, char \*argv[]){

for(int i = 0; i < 32; i++) signal(i, signalHandler);

if(argc != 2) errorPrompt();

int test\_no = atoi(argv[1]);

if(!(test\_no >=1 && test\_no <=6)) errorPrompt();

switch(test\_no){

case 1:

printf("Executing Test #1 wrtie to r/o mmap\n");

test1();

break;

case 2:

printf("Executing Test #2 write to MAP\_SHARED\n");

test23(MAP\_SHARED);

break;

case 3:

printf("Executing Test #3 write to MAP\_PRIVATE\n");

test23(MAP\_PRIVATE);

break;

case 4:

printf("Executing Test #4 writing beyond the edge\n");

test45(4);

break;

case 5:

printf("Executing Test #5 writing into a hole\n");

test45(5);

break;

case 6:

printf("Executing Test #6 reading beyond the edge\n");

test6();

break;

default:

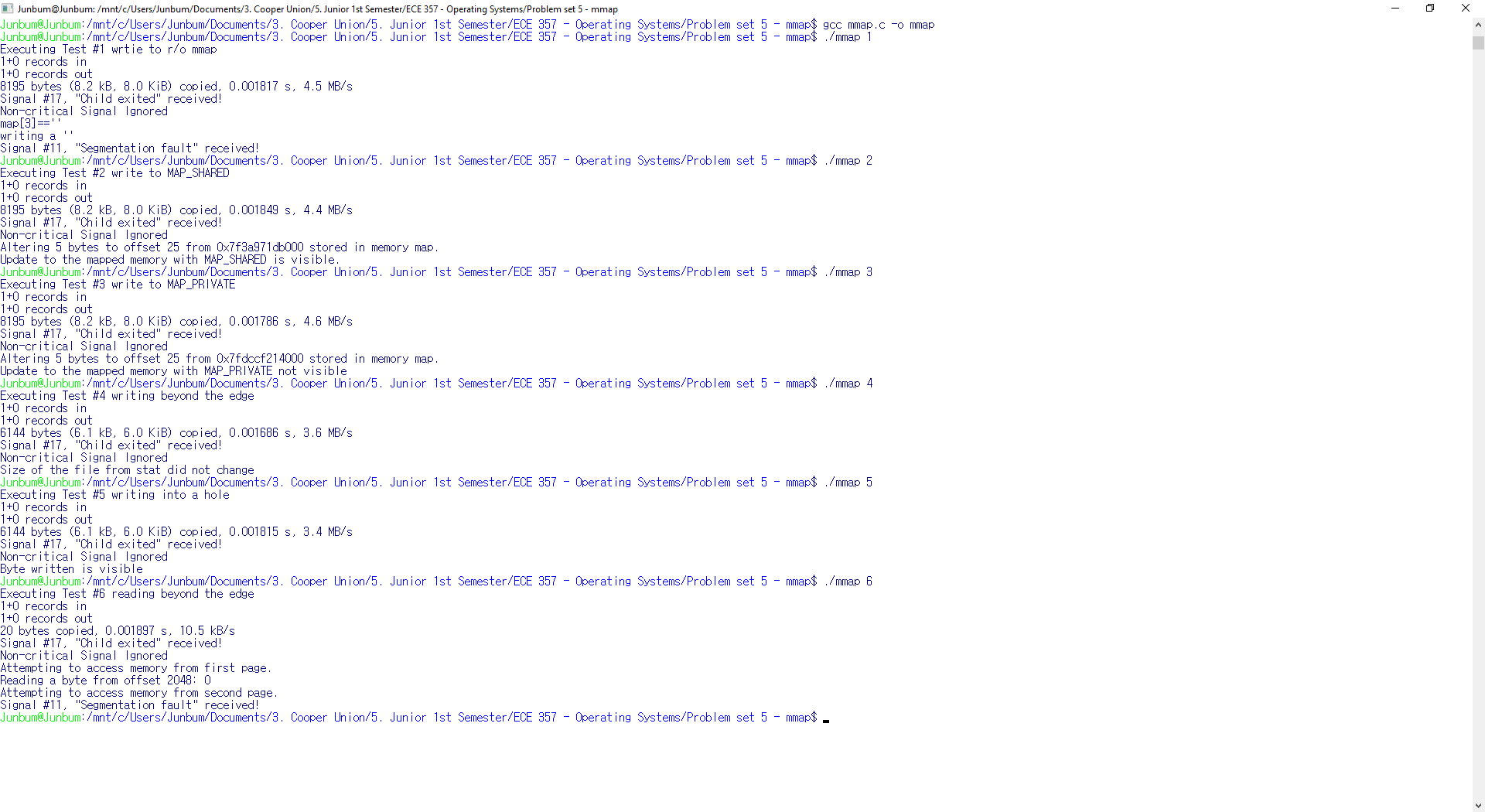
errorPrompt();

break;

}

}

1. **Screenshot of a Sample Run**



Junbum@Junbum:/mnt/c/Users/Junbum/Documents/3. Cooper Union/5. Junior 1st Semester/ECE 357 - Operating Systems/Problem set 5 - mmap$ **gcc mmap.c -o mmap**

Junbum@Junbum:/mnt/c/Users/Junbum/Documents/3. Cooper Union/5. Junior 1st Semester/ECE 357 - Operating Systems/Problem set 5 - mmap$ **./mmap 1**

Executing Test #1 wrtie to r/o mmap

1+0 records in

1+0 records out

8195 bytes (8.2 kB, 8.0 KiB) copied, 0.001817 s, 4.5 MB/s

Signal #17, "Child exited" received!

Non-critical Signal Ignored

map[3]=='?'

writing a 'v'

**Signal #11, "Segmentation fault" received!**

Junbum@Junbum:/mnt/c/Users/Junbum/Documents/3. Cooper Union/5. Junior 1st Semester/ECE 357 - Operating Systems/Problem set 5 - mmap$ **./mmap 2**

Executing Test #2 write to MAP\_SHARED

1+0 records in

1+0 records out

8195 bytes (8.2 kB, 8.0 KiB) copied, 0.001849 s, 4.4 MB/s

Signal #17, "Child exited" received!

Non-critical Signal Ignored

Altering 5 bytes to offset 25 from 0x7f3a971db000 stored in memory map.

**Update to the mapped memory with MAP\_SHARED is visible.**

Junbum@Junbum:/mnt/c/Users/Junbum/Documents/3. Cooper Union/5. Junior 1st Semester/ECE 357 - Operating Systems/Problem set 5 - mmap$ **./mmap 3**

Executing Test #3 write to MAP\_PRIVATE

1+0 records in

1+0 records out

8195 bytes (8.2 kB, 8.0 KiB) copied, 0.001786 s, 4.6 MB/s

Signal #17, "Child exited" received!

Non-critical Signal Ignored

Altering 5 bytes to offset 25 from 0x7fdccf214000 stored in memory map.

**Update to the mapped memory with MAP\_PRIVATE not visible**

Junbum@Junbum:/mnt/c/Users/Junbum/Documents/3. Cooper Union/5. Junior 1st Semester/ECE 357 - Operating Systems/Problem set 5 - mmap$ **./mmap 4**

Executing Test #4 writing beyond the edge

1+0 records in

1+0 records out

6144 bytes (6.1 kB, 6.0 KiB) copied, 0.001686 s, 3.6 MB/s

Signal #17, "Child exited" received!

Non-critical Signal Ignored

**Size of the file from stat did not change**

Junbum@Junbum:/mnt/c/Users/Junbum/Documents/3. Cooper Union/5. Junior 1st Semester/ECE 357 - Operating Systems/Problem set 5 - mmap$ **./mmap 5**

Executing Test #5 writing into a hole

1+0 records in

1+0 records out

6144 bytes (6.1 kB, 6.0 KiB) copied, 0.001815 s, 3.4 MB/s

Signal #17, "Child exited" received!

Non-critical Signal Ignored

**Byte written is visible**

Junbum@Junbum:/mnt/c/Users/Junbum/Documents/3. Cooper Union/5. Junior 1st Semester/ECE 357 - Operating Systems/Problem set 5 - mmap$ **./mmap 6**

Executing Test #6 reading beyond the edge

1+0 records in

1+0 records out

20 bytes copied, 0.001897 s, 10.5 kB/s

Signal #17, "Child exited" received!

Non-critical Signal Ignored

Attempting to access memory from first page.

Reading a byte from offset 2048: 0

Attempting to access memory from second page.

**Signal #11, "Segmentation fault" received!**

1. **Narrative for Test Programs**

TEST4 – In the case, the size of the program does not change because although we have written to memory, we have not set any measure to change the size of the file. The file system does not recognize that the size of the file has changed, and therefore remains the same.

TEST5 – By calling the syscall lseek, we have now increased the size of the file to incorporate the bytes that we have previously written. Now the filesystem recognizes the what we have previously written to memory.

TEST6 – If a program wrote 20 bytes to memory, the program can access up to 4096th byte because a page was allocated. In this case memory that was uninitialized will read a value of 0. Test 6 returns a segmentation fault, because the second page was never allocated to the program. The program does not have permission to access the second page, and therefore the will return an SIGSEGV.