**Source Code**

//Camille Chow

//ECE357 PSET 5

#include <stdio.h>

#include <stdlib.h>

#include <sys/mman.h>

#include <sys/stat.h>

#include <sys/types.h>

#include <string.h>

#include <signal.h>

#include <errno.h>

#include <unistd.h>

int openfile(char\* name, int size)

{

FILE\* fs;

int fd;

//open test file

if ((fs = fopen(name, "w+")) < 0)

{

fprintf(stderr, "Could not open test file for reading and writing: %s\n", strerror(errno));

exit(255);

}

//get file descriptor

if ((fd = fileno(fs)) < 0)

{

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

exit(255);

}

//write to file

char buf[size];

for (int i = 0; i < size; i++)

buf[i] = 'A';

int n;

while ((n = write(fd,buf,size)) < size)

{

if (n < 0)

{

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

exit(255);

}

lseek(fd, 0, SEEK\_SET);

}

return fd;

}

void closemap(char\* map, int size, int fd)

{

//unmap

if (munmap(map, size) < 0)

{

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

exit(255);

}

//close file descriptor

if (close(fd) < 0)

{

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

exit(255);

}

}

void handle(int sig)

{

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

exit(sig);

}

void test1()

{

int fd;

int file\_size = 5000;

char\* map;

//set signal handlers for signals 1 through 31

for (int i = 1; i < 32; i++)

signal(i,handle);

//open test file and get stats

fd = openfile("test1.txt", file\_size);

//call mmap

if ((map = mmap(NULL, file\_size, PROT\_READ, MAP\_SHARED, fd, 0)) < 0)

{

fprintf(stderr, "Error mmap-ing test file: %s\n", strerror(errno));

exit(255);

}

//attempt to write

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

fprintf(stderr, "writing to map[3]\n");

map[3] = 'A';

closemap(map,file\_size,fd);

exit(0);

}

void test2and3(int testno)

{

int fd;

int file\_size = 5000;

char\* map;

char\* name = "test2.txt";

int flag = MAP\_SHARED;

if (testno == 3)

{

name = "test3.txt";

flag = MAP\_PRIVATE;

}

//open test file and get stats

fd = openfile(name, file\_size);

//call mmap

if ((map = mmap(NULL, file\_size, PROT\_READ|PROT\_WRITE, flag, fd, 0)) < 0)

{

fprintf(stderr, "Error mmap-ing test file: %s\n", strerror(errno));

exit(255);

}

//write to memory

fprintf(stderr, "writing 'B' to map[50]\n");

map[50] = 'B';

//check memory

char buf[1];

if (lseek(fd, 50, SEEK\_SET) < 0)

{

fprintf(stderr, "Error lseek-ing test file: %s\n", strerror(errno));

exit(255);

}

if (read(fd,buf,1) < 0)

{

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

exit(255);

}

closemap(map,file\_size,fd);

if (buf[0] == 'B')

{

fprintf(stderr, "success! (byte changed)\n");

exit(0);

}

fprintf(stderr, "fail (byte stayed the same) :(\n");

exit(1);

}

void test4()

{

int fd;

int file\_size = 5000;

char\* map;

struct stat buf;

//open test file and get stats

fd = openfile("test4.txt", file\_size);

//call mmap

if ((map = mmap(NULL, file\_size, PROT\_READ|PROT\_WRITE, MAP\_SHARED, fd, 0)) < 0)

{

fprintf(stderr, "Error mmap-ing test file: %s\n", strerror(errno));

exit(255);

}

//find initial size

if (fstat(fd,&buf) < 0)

{

fprintf(stderr, "error stat-ing test file: %s\n", strerror(errno));

exit(255);

}

off\_t old = buf.st\_size;

fprintf(stderr, "file size: %lld\n", old);

//write beyond last byte

fprintf(stderr, "writing 'B' beyond last byte\n");

map[file\_size+1] = 'B';

//find new size

if (fstat(fd,&buf) < 0)

{

fprintf(stderr, "error stat-ing test file: %s\n", strerror(errno));

exit(255);

}

off\_t new = buf.st\_size;

fprintf(stderr, "new file size: %lld\n", new);

closemap(map,file\_size,fd);

if (old == new)

{

fprintf(stderr, "success! (file size stayed the same)\n");

exit(1);

}

fprintf(stderr, "fail (file size changed) :(\n");

exit(0);

}

void test5()

{

int fd;

int file\_size = 5000;

char\* map;

//open test file and get stats

fd = openfile("test5.txt", file\_size);

//call mmap

if ((map = mmap(NULL, file\_size, PROT\_READ|PROT\_WRITE, MAP\_SHARED, fd, 0)) < 0)

{

fprintf(stderr, "Error mmap-ing test file: %s\n", strerror(errno));

exit(255);

}

//write beyond last byte

fprintf(stderr, "writing 'B' beyond last byte\n");

map[file\_size+1] = 'X';

//write 16 bytes past eof

if (lseek(fd,16,SEEK\_END) < 0)

{

fprintf(stderr, "Error lseek-ing test file: %s\n", strerror(errno));

exit(255);

}

char\* buf = "B";

if (write(fd,buf,1) < 0)

{

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

exit(255);

}

//check for byte 'X'

if (lseek(fd, file\_size + 1, SEEK\_SET) < 0)

{

fprintf(stderr, "Error lseek-ing test file: %s\n", strerror(errno));

exit(255);

}

char buf2[1];

if (read(fd,buf2,1) < 0)

{

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

exit(255);

}

closemap(map,file\_size,fd);

if (buf2[0] == 'X')

{

fprintf(stderr, "success! (byte 'X' is visible)\n");

exit(0);

}

fprintf(stderr, "fail (byte 'X' isn't visible) :(\n");

exit(1);

}

void test6()

{

int fd;

int file\_size = 1000;

char\* map;

//set signal handlers for signals 1 through 31

for (int i = 1; i < 32; i++)

signal(i,handle);

//open test file and get stats

fd = openfile("test6.txt", file\_size);

//call mmap

if ((map = mmap(NULL, 8192, PROT\_READ|PROT\_WRITE, MAP\_SHARED, fd, 0)) < 0)

{

fprintf(stderr, "Error mmap-ing test file: %s\n", strerror(errno));

exit(255);

}

//part a)

fprintf(stderr, "reading memory beyond eof in first page\n");

fprintf(stderr, "map[2000]: %d\nread succeeded!\n", map[2000]);

//part b)

fprintf(stderr, "reading memory beyond eof in second page\n");

fprintf(stderr, "map[5000]: %d\nread succeeded!\n", map[5000]);

closemap(map,8192,fd);

exit(0);

}

int main(int argc, char\*\*argv)

{

if (argv[1] == NULL)

{

fprintf(stderr, "Please specify test number\n");

return 0;

}

switch (atoi(argv[1])) {

case 1:

test1();

case 2:

test2and3(2);

case 3:

test2and3(3);

case 4:

test4();

case 5:

test5();

case 6:

test6();

default:

fprintf(stderr, "Test number must be between 1 and 6\n");

return 0;

}

return 0;

}

**Explanation of Problems 4-6**

In Problem 4, the file size does not change because writing to space in memory that does not correspond to part of the file shouldn't change the file. The size of the file can only be changed by writing directly to the file using write(2)/lseek(2). This is why in Problem 5, when we change the size of the file using these methods, the byte that we altered in memory previously becomes visible because it is now a part of the file.

In Problem 6, the read from the first page of memory succeeds, but the read from the second page fails. This is because the file only needs one page of memory, therefore the second page is not included in the PGD. Thus, when the program attempts to access this page, it is attempting to access an invalid address, causing a bus error.

**Sample Output For All Test Cases**

