Yacine Manseur
11/3/15
Operating Systems
Problem Set 5

```
yacine@yacine-N53SV ~/Documents/OS/OS/PS5 $ ./mm2 A
Part A:
The following signal will be generated...
SIGSEGV signal encountered!
Now exiting.
yacine@yacine-N53SV ~/Documents/OS/OS/PS5 $ ./mm2 B
Part B:
Contents of mapped memory before write:
This is a test file.
Now writing to the mapped memory...
Contents of mapped memory after write:
Hello!
The update is visible when accessing the file through
traditional lseek(2)/read(2) system calls.
yacine@yacine-N53SV ~/Documents/OS/OS/PS5 $ ./mm2 C
Part C:
Contents of mapped memory before write:
This is a test file.
Now writing to the mapped memory...
Contents of mapped memory after write:
This is a test file.
The update is not visible when accessing the file
through traditional lseek(2)/read(2) system calls.
yacine@yacine-N53SV ~/Documents/OS/OS/PS5 $ ./mm2 D
Part D:
Size of test.txt before write: 20
Size of test.txt after write: 20
The size of the file through the traditional interface
did not change. This is because the size of the actual
file is smaller than the size of the memory map.
yacine@yacine-N53SV ~/Documents/OS/OS/PS5 $ ./mm2 E
Part E:
Report:
Before the write:
The file size was: 20
The length of the virtual address was: 49
After the first write:
The file size was: 20
The length of the virtual address was: 78
After the second write:
The file size was: 21
The length of the virtual address was: 21
The data that had been written to the hole previously
via the memory-map was saved in the virtual address space.
The data that had been written to the hole previously
via the memory-map is not visible in the file.
yacine@yacine-N53SV ~/Documents/OS/OS/PS5 $ ./mm2 F
Part F:
Successfully established a valid mmap region two pages long.
Attempting to access memory in the first page...
Memory Access successful!
Attempting to access memory in the second page...
SIGBUS signal encountered!
Now exiting.
```

```
// Yacine Manseur
// Cooper Union Fall 2015
// ECE 357: Operating Systems
// Problem Set 5
// memorymap.c

#include <stdlib.h>
#include <stdio.h>
#include <errno.h>
#include <unistd.h>
#include <fcntl.h>
#include <string.h>
#include <sys/mman.h>
#include <sys/stat.h>
#include <sys/types.h>
#include <sys/signal.h>

char * readFile(int fd, int fileLength)
{
	int bytesRead = 0;
	int bufferSize = 1024;
	char *buffer = (char*) malloc(bufferSize);
	int managedToReadSomething = NULL;
	char *str = (char*) malloc(fileLength);
	if (!buffer)
	{
		fprintf(stderr, "Error allocating memory of size %d\n", bufferSize);
	}

	// Loop until end of file
	while ((bytesRead = read(fd, buffer, bufferSize)) > 0)
	{
		managedToReadSomething = 1;
		strcat(str, buffer);
	}

	if(!managedToReadSomething)
	{
		fprintf(stderr, "Error reading file: %s\n", strerror(errno));
		exit(1);
	}

	return str;
}

// Used to avoid Copy/Paste between different scenarios
void openStatFile(int *fd, char *fileName, struct stat *st)
{
	*fd = open(fileName, O_RDWR | O_APPEND);
	if (*fd < 0)
	{
		fprintf(stderr, "Error opening file: %s\n", fileName);
		exit(1);
	}

	if(fstat(*fd, st) < 0)
	{
		fprintf(stderr, "Error getting file information: %s\n", strerror(errno));
		exit(1);
	}
}

// Signal Handler
```

```
void sig_handler(int signo)
{
	if (signo == SIGSEGV)
	{
		fprintf(stderr, "SIGSEGV signal encountered!\n");
		fprintf(stderr, "Now exiting.\n");
		exit(1);
	}
	else if (signo == SIGBUS)
	{
		fprintf(stderr, "SIGBUS signal encountered!\n");
		fprintf(stderr, "Now exiting.\n");
		exit(1);
	}
}

// "When one has mapped a file for read-only access, but attempts
// to write to that mapped area, what signal is generated?"
void a()
{
	int fd;
	char *fileName = "test.txt";
	struct stat st;
	char *addr;

	openStatFile(&fd, fileName, &st);

	addr = mmap(0, st.st_size, PROT_READ, MAP_SHARED, fd, 0);
	if (addr == MAP_FAILED)
	{
		fprintf(stderr, "Error mapping file into memory: %s\n", strerror(errno));
		exit(1);
	}

	fprintf(stderr, "Part A:\n");
	fprintf(stderr, "The following signal will be generated...\n");
	sprintf(addr, "Try writing to mapped address.");
	return;
}

// "If one maps a file with MAP_SHARED(part = 0) or MAP_PRIVATE(part = 1)
// and then writes to the mapped memory, is that update visible when
// accessing the file through the traditional lseek(2)/read(2) system calls?"
void bc(int part)
{
	int fd;
	struct stat st;
	int status;
	char *fileName = "test.txt";
	char *addr;

	char *beforeWrite, *afterWrite;

	openStatFile(&fd, fileName, &st);

	switch(part)
	{
		case 0: //Execute Part B
			fprintf(stderr, "Part B:\n");
			addr = mmap(0, st.st_size, PROT_READ | PROT_WRITE, MAP_SHARED, fd, 0);
			break;
		case 1: //Execute Part C
			fprintf(stderr, "Part C:\n");
```

```
                addr = mmap(0, st.st_size, PROT_READ | PROT_WRITE, MAP_PRIVATE, fd, 0);
                break;
        }

        if (addr == MAP_FAILED)
        {
                fprintf(stderr, "Error mapping file into memory: %s\n", strerror(errno));
                exit(1);
        }

        beforeWrite = (char*)malloc(strlen(addr));
        if (!beforeWrite)
        {
                fprintf(stderr, "Error occurred allocating memory.\n");
                exit(1);
        }
        strcpy(beforeWrite, addr);

        fprintf(stderr, "Contents of mapped memory before write:\n%s\n", addr);
        fprintf(stderr, "Now writing to the mapped memory...\n");
        sprintf(addr, "Hello!");
        afterWrite = (char*)malloc(strlen(addr));
        if (!afterWrite)
        {
                fprintf(stderr, "Error occurred allocating memory.\n");
                exit(1);
        }
        fprintf(stderr, "Contents of mapped memory after write:\n");

        strcpy(afterWrite, readFile(fd, strlen(addr)));
        fprintf(stderr, "%s\n", afterWrite);

        if(strcmp(beforeWrite,afterWrite) == 0)
        {
                fprintf(stderr, "The update is not visible when accessing the file\n");
                fprintf(stderr, "through traditional lseek(2)/read(2) system calls.\n");
        }
        else
        {
                fprintf(stderr, "The update is visible when accessing the file through\n");
                fprintf(stderr, "traditional lseek(2)/read(2) system calls.\n");
        }

        if(close(fd) < 0)
        {
                fprintf(stderr, "Error closing file: %s\n", strerror(errno));
                exit(1);
        }

        return;
}

// "Say a pre-existing file of a certain size which is not an exact
// multiple of the page size is mapped with MAP_SHARED and read/write
// access, and one writes to the mapped memory just beyond the byte
// corresponding to the last byte of the existing file. Does the size of the
// file through the traditional interface (e.g. stat(2)) change?"
// Explain your reasoning why this is or is not the case
void d()
{
        fprintf(stderr, "Part D:\n");
        int fd;
        struct stat st;
```

```
	char *addr;
	char *fileName = "test.txt";
	size_t sizeOld, sizeNew;

	openStatFile(&fd, fileName, &st);
	sizeOld = st.st_size;
	fprintf(stderr, "Size of %s before write: %zu\n", fileName, sizeOld);

	addr = mmap(0, sizeOld, PROT_READ | PROT_WRITE, MAP_SHARED, fd, 0);
	if (addr == MAP_FAILED)
	{
		fprintf(stderr, "Error mapping file into memory: %s\n", strerror(errno));
		exit(1);
	}

	// Write to mmap just beyond the last byte
	sprintf(addr + strlen(addr), "Write this beyond the buffer.");

	//Get the new size
	if(fstat(fd, &st) < 0)
	{
		fprintf(stderr, "Error getting file information: %s\n", strerror(errno));
		exit(1);
	}

	sizeNew = st.st_size;

	fprintf(stderr, "Size of %s after write: %zu\n", fileName, sizeNew);

	if((sizeNew - sizeOld) == 0)
	{
		fprintf(stderr, "The size of the file through the traditional interface\n");
		fprintf(stderr, "did not change. This is because the size of the actual\n");
		fprintf(stderr, "file is smaller than the size of the memory map.\n");
	}
	else
	{
		fprintf(stderr, "The size of the file did change, meaning something went
wrong.\n");
		exit(1);
	}
	return;
}

// "Let us say that after performing the aformentioned memory write, one
// then increased the size of the file beyond the written area, without
// over-writing the intervening contents (e.g. by using lseek(2) and then
// write(2) with a size of 1), thus creating a 'hole' in the file. What
// happens to the data that had been written to this hole previously via
// the memory-map? Are they visible in the file?"
void e()
{
	fprintf(stderr, "Part E:\n");
	int fd;
	struct stat st;
	char *fileName = "test.txt";
	char *addr;
	size_t sizeBefore, sizeAfter1, sizeAfter2;

	openStatFile(&fd, fileName, &st);
	sizeBefore = st.st_size;

	addr = mmap(0, st.st_size, PROT_READ | PROT_WRITE, MAP_SHARED, fd, 0);
```

```
	if (addr == MAP_FAILED)
	{
		fprintf(stderr, "Error mapping file into memory: %s\n", strerror(errno));
		exit(1);
	}
	char *beforeWrite = (char*)malloc(strlen(addr));
	strcpy(beforeWrite,addr);
	sprintf(addr + strlen(addr), "Write this beyond the buffer.");

	char *attemptWrite = (char*)malloc(strlen(addr));
	strcpy(attemptWrite,addr);

	//Update stat
	if(fstat(fd, &st) < 0)
	{
		fprintf(stderr, "Error getting file information: %s\n", strerror(errno));
		exit(1);
	}

	sizeAfter1 = st.st_size;

	if(lseek(fd,(int)strlen(addr), SEEK_END) < 0)
	{
		fprintf(stderr, "Error occurred during lseek(2): %s\n", strerror(errno));
		exit(1);
	}

	if(write(fd, "0", 1) != 1)
	{
		fprintf(stderr, "Error occurred during write: %s\n", strerror(errno));
		exit(1);
	}

	if(lseek(fd, 0, SEEK_SET) < 0)
	{
		fprintf(stderr, "Error occurred during lseek(2): %s\n", strerror(errno));
		exit(1);
	}

	char *afterWrite = (char*)malloc(strlen(addr));
	strcpy(afterWrite, readFile(fd, strlen(addr)));

	//Update stat
	if(fstat(fd, &st) < 0)
	{
		fprintf(stderr, "Error getting file information: %s\n", strerror(errno));
		exit(1);
	}

	sizeAfter2 = st.st_size;

	fprintf(stderr, "Report:\n");
	fprintf(stderr, "Before the write:\n");
	fprintf(stderr, "\tThe file size was: %zu\n", sizeBefore);
	fprintf(stderr, "\tThe length of the virtual address was: %zu\n",
strlen(beforeWrite));
	fprintf(stderr, "After the first write:\n");
	fprintf(stderr, "\tThe file size was: %zu\n", sizeAfter1);
	fprintf(stderr, "\tThe length of the virtual address was: %zu\n",
strlen(attemptWrite));
	fprintf(stderr, "After the second write:\n");
	fprintf(stderr, "\tThe file size was: %zu\n", sizeAfter2);
```

```
	fprintf(stderr, "\tThe length of the virtual address was: %zu\n",
strlen(afterWrite));

	if(strlen(attemptWrite) > strlen(beforeWrite))
	{
		fprintf(stderr, "The data that had been written to the hole previously\n");
		fprintf(stderr, "via the memory-map was saved in the virtual address
space.\n");
	}
	else
	{
		fprintf(stderr, "The data that had been written to the hole previously\n");
		fprintf(stderr, "via the memory-map was not saved in the virtual address
space.\n");
	}

	if(strlen(attemptWrite) > strlen(afterWrite))
	{
		fprintf(stderr, "The data that had been written to the hole previously\n");
		fprintf(stderr, "via the memory-map is not visible in the file.\n");
	}
	else
	{
		fprintf(stderr, "The data that had been written to the hole previously\n");
		fprintf(stderr, "via the memory-map is visible in the file.\n");
	}

	//printf("%zu %zu %zu\n", sizeBefore, sizeAfter1, sizeAfter2);
	//printf("%zu %zu %zu\n", strlen(beforeWrite), strlen(attemptWrite),
strlen(afterWrite));

	return;
}

// Let's say there is an existing small file (say 10 bytes). Can you
// establish a valid mmap region two pages (8192 bytes) long? If so, what
// signal is delivered when attempting to access memory in the second page?
// What about the first page? Explain any differences in these outcomes.
void f()
{
	fprintf(stderr, "Part F:\n");
	int fd;
	struct stat st;
	char *fileName = "smalltest.txt";
	char *addr;

	openStatFile(&fd, fileName, &st);

	addr = mmap(0,8192, PROT_READ, MAP_SHARED, fd, 0);
	if (addr == MAP_FAILED)
	{
		fprintf(stderr, "Unsuccessfully established a valid mmap region two pages
long: %s\n", strerror(errno));
		exit(1);
	}
	else
		fprintf(stderr, "Successfully established a valid mmap region two pages
long.\n");

	// Attempt to access memory in the first page
	fprintf(stderr, "Attempting to access memory in the first page...\n");
	char temp1 = addr[1000];
	fprintf(stderr, "Memory Access successful!\n");
```

```
	// Attempt to access memory in the second page
	fprintf(stderr, "Attempting to access memory in the second page...\n");
	char temp2 = addr[7000];
	fprintf(stderr, "Memory Access successful!\n");

	/*
	As you can see from the output of this function, the signal SIGBUS is thrown when
	trying to access memory on the second page even though there was no error from the
	first memory access. This is because the first page has some information where
there
	is nothing found on the second page. Because there is no information about the
memory
	for the second page, we get a bus error.
	*/

	return;
}

int main (int argc, char *argv[])
{
	// Establish signal handlers
	if(signal(SIGSEGV, sig_handler) == SIG_ERR)
	{
		fprintf(stderr, "Cannot catch SIGSEGV: %s\n", strerror(errno));
	}
	if(signal(SIGBUS, sig_handler) == SIG_ERR)
	{
		fprintf(stderr, "Cannot catch SIGBUS: %s\n", strerror(errno));
	}

	// Incorrect number of inputs
	if (argc != 2)
	{
		fprintf(stderr, "Error: Incorrect input\n");
		exit(1);
	}

	char question = *argv[1];

	switch(question)
	{
		case('A'):
			a();
			break;
		case('B'):
			bc(0);
			break;
		case('C'):
			bc(1);
			break;
		case('D'):
			d();
			break;
		case('E'):
			e();
			break;
		case('F'):
			f();
			break;
		default:
			fprintf(stderr, "Error: Incorrect argument.\nAccepted arguments (A-
F)\n");
```

```
                exit(1);
        }
        exit(0);
}
```