**Московский авиационный институт**

**(Национальный исследовательский университет)**

Институт: «Информационные технологии и прикладная математика»

Кафедра: 806 «Вычислительная математика и программирование»

Дисциплина: «Операционные системы»

**Лабораторная работа № 4**

Тема: Отображение файла в память

Студент: Волков

Группа: 80-207

Преподаватель: Миронов Е. С.

Дата:

Оценка:

1. **Постановка задачи**

Составить и отладить программу на языке Си, осуществляющую работу с процессами и взаимодействие между ними в одной из двух операционных систем. В результате работы программа (основной процесс) должен создать для решение задачи один или несколько дочерних процессов. Взаимодействие между процессами осуществляется через системные сигналы/события и/или через отображаемые файлы (memory-mapped files). Необходимо обрабатывать системные ошибки, которые могут возникнуть в результате работы.

*Вариант 1:* Пользователь вводит команды вида: «число число число <endline>». Далее эти числа передаются от родительского процесса в дочерний. Дочерний процесс считает их сумму и выводит её в файл. Числа имеют тип int.

1. **Описание программы**

Пользователь вводит сначала имя файла, который нужно будет записать результат выполнения программы. Далее вводится массив данных до <endline>. Вычисление суммы происходит после ввода массива данных. Этот массив при помощи общей памяти передается в дочерний процесс. После чего дочерний процесс выполняет сложение и записывает ответ в файл.

1. **Наборы и результаты тестов**

*Тест 1:*

enter file name:

> a.txt

Enter sum terms:

>1 2 3 4 5

6 7 8 10

*strace:*

execve("./lab4", ["./lab4"], 0x7ffdf88d0910 /\* 20 vars \*/) = 0

brk(NULL) = 0x5557f141d000

arch\_prctl(0x3001 /\* ARCH\_??? \*/, 0x7ffee52ce4a0) = -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=36270, ...}) = 0

mmap(NULL, 36270, PROT\_READ, MAP\_PRIVATE, 3, 0) = 0x7fc8b13e9000

close(3) = 0

openat(AT\_FDCWD, "/lib/x86\_64-linux-gnu/librt.so.1", O\_RDONLY|O\_CLOEXEC) = 3

read(3, "\177ELF\2\1\1\0\0\0\0\0\0\0\0\0\3\0>\0\1\0\0\0 7\0\0\0\0\0\0"..., 832) = 832

fstat(3, {st\_mode=S\_IFREG|0644, st\_size=40040, ...}) = 0

mmap(NULL, 8192, PROT\_READ|PROT\_WRITE, MAP\_PRIVATE|MAP\_ANONYMOUS, -1, 0) = 0x7fc8b13e7000

mmap(NULL, 44000, PROT\_READ, MAP\_PRIVATE|MAP\_DENYWRITE, 3, 0) = 0x7fc8b13dc000

mprotect(0x7fc8b13df000, 24576, PROT\_NONE) = 0

mmap(0x7fc8b13df000, 16384, PROT\_READ|PROT\_EXEC, MAP\_PRIVATE|MAP\_FIXED|MAP\_DENYWRITE, 3, 0x3000) = 0x7fc8b13df000

mmap(0x7fc8b13e3000, 4096, PROT\_READ, MAP\_PRIVATE|MAP\_FIXED|MAP\_DENYWRITE, 3, 0x7000) = 0x7fc8b13e3000

mmap(0x7fc8b13e5000, 8192, PROT\_READ|PROT\_WRITE, MAP\_PRIVATE|MAP\_FIXED|MAP\_DENYWRITE, 3, 0x8000) = 0x7fc8b13e5000

close(3) = 0

openat(AT\_FDCWD, "/lib/x86\_64-linux-gnu/libpthread.so.0", O\_RDONLY|O\_CLOEXEC) = 3

read(3, "\177ELF\2\1\1\0\0\0\0\0\0\0\0\0\3\0>\0\1\0\0\0\220\201\0\0\0\0\0\0"..., 832) = 832

pread64(3, "\4\0\0\0\24\0\0\0\3\0\0\0GNU\0O\305\3743\364B\2216\244\224\306@\261\23\327o"..., 68, 824) = 68

fstat(3, {st\_mode=S\_IFREG|0755, st\_size=157224, ...}) = 0

pread64(3, "\4\0\0\0\24\0\0\0\3\0\0\0GNU\0O\305\3743\364B\2216\244\224\306@\261\23\327o"..., 68, 824) = 68

mmap(NULL, 140408, PROT\_READ, MAP\_PRIVATE|MAP\_DENYWRITE, 3, 0) = 0x7fc8b13b9000

mmap(0x7fc8b13c0000, 69632, PROT\_READ|PROT\_EXEC, MAP\_PRIVATE|MAP\_FIXED|MAP\_DENYWRITE, 3, 0x7000) = 0x7fc8b13c0000

mmap(0x7fc8b13d1000, 20480, PROT\_READ, MAP\_PRIVATE|MAP\_FIXED|MAP\_DENYWRITE, 3, 0x18000) = 0x7fc8b13d1000

mmap(0x7fc8b13d6000, 8192, PROT\_READ|PROT\_WRITE, MAP\_PRIVATE|MAP\_FIXED|MAP\_DENYWRITE, 3, 0x1c000) = 0x7fc8b13d6000

mmap(0x7fc8b13d8000, 13432, PROT\_READ|PROT\_WRITE, MAP\_PRIVATE|MAP\_FIXED|MAP\_ANONYMOUS, -1, 0) = 0x7fc8b13d8000

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\363\377?\332\200\270\27\304d\245n\355Y\377\t\334"..., 68, 880) = 68

fstat(3, {st\_mode=S\_IFREG|0755, st\_size=2029224, ...}) = 0

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\363\377?\332\200\270\27\304d\245n\355Y\377\t\334"..., 68, 880) = 68

mmap(NULL, 2036952, PROT\_READ, MAP\_PRIVATE|MAP\_DENYWRITE, 3, 0) = 0x7fc8b11c7000

mprotect(0x7fc8b11ec000, 1847296, PROT\_NONE) = 0

mmap(0x7fc8b11ec000, 1540096, PROT\_READ|PROT\_EXEC, MAP\_PRIVATE|MAP\_FIXED|MAP\_DENYWRITE, 3, 0x25000) = 0x7fc8b11ec000

mmap(0x7fc8b1364000, 303104, PROT\_READ, MAP\_PRIVATE|MAP\_FIXED|MAP\_DENYWRITE, 3, 0x19d000) = 0x7fc8b1364000

mmap(0x7fc8b13af000, 24576, PROT\_READ|PROT\_WRITE, MAP\_PRIVATE|MAP\_FIXED|MAP\_DENYWRITE, 3, 0x1e7000) = 0x7fc8b13af000

mmap(0x7fc8b13b5000, 13528, PROT\_READ|PROT\_WRITE, MAP\_PRIVATE|MAP\_FIXED|MAP\_ANONYMOUS, -1, 0) = 0x7fc8b13b5000

close(3) = 0

mmap(NULL, 12288, PROT\_READ|PROT\_WRITE, MAP\_PRIVATE|MAP\_ANONYMOUS, -1, 0) = 0x7fc8b11c4000

arch\_prctl(ARCH\_SET\_FS, 0x7fc8b11c4740) = 0

mprotect(0x7fc8b13af000, 12288, PROT\_READ) = 0

mprotect(0x7fc8b13d6000, 4096, PROT\_READ) = 0

mprotect(0x7fc8b13e5000, 4096, PROT\_READ) = 0

mprotect(0x5557f1351000, 4096, PROT\_READ) = 0

mprotect(0x7fc8b141f000, 4096, PROT\_READ) = 0

munmap(0x7fc8b13e9000, 36270) = 0

set\_tid\_address(0x7fc8b11c4a10) = 25836

set\_robust\_list(0x7fc8b11c4a20, 24) = 0

rt\_sigaction(SIGRTMIN, {sa\_handler=0x7fc8b13c0bf0, sa\_mask=[], sa\_flags=SA\_RESTORER|SA\_SIGINFO, sa\_restorer=0x7fc8b13ce3c0}, NULL, 8) = 0

rt\_sigaction(SIGRT\_1, {sa\_handler=0x7fc8b13c0c90, sa\_mask=[], sa\_flags=SA\_RESTORER|SA\_RESTART|SA\_SIGINFO, sa\_restorer=0x7fc8b13ce3c0}, NULL, 8) = 0

rt\_sigprocmask(SIG\_UNBLOCK, [RTMIN RT\_1], NULL, 8) = 0

prlimit64(0, RLIMIT\_STACK, NULL, {rlim\_cur=8192\*1024, rlim\_max=RLIM64\_INFINITY}) = 0

fstat(1, {st\_mode=S\_IFCHR|0620, st\_rdev=makedev(0x88, 0x3), ...}) = 0

brk(NULL) = 0x5557f141d000

brk(0x5557f143e000) = 0x5557f143e000

write(1, "enter file name:\n", 17enter file name:

) = 17

write(1, "> ", 2> ) = 2

**fstat(0, {st\_mode=S\_IFCHR|0620, st\_rdev=makedev(0x88, 0x3), ...}) = 0**

**read(0, a.txt**

**"a.txt\n", 1024) = 6**

**statfs("/dev/shm/", {f\_type=TMPFS\_MAGIC, f\_bsize=4096, f\_blocks=3279193, f\_bfree=3279193, f\_bavail=3279193, f\_files=3279193, f\_ffree=3279192, f\_fsid={val=[0, 0]}, f\_namelen=255, f\_frsize=4096, f\_flags=ST\_VALID|ST\_NOSUID|ST\_NODEV|ST\_**

**NOATIME}) = 0**

**futex(0x7fc8b13db390, FUTEX\_WAKE\_PRIVATE, 2147483647) = 0**

**openat(AT\_FDCWD, "/dev/shm/pipe", O\_RDWR|O\_CREAT|O\_NOFOLLOW|O\_CLOEXEC, 0700) = 3**

**openat(AT\_FDCWD, "/dev/shm/size", O\_RDWR|O\_CREAT|O\_NOFOLLOW|O\_CLOEXEC, 0700) = 4**

**openat(AT\_FDCWD, "/dev/shm/mutex", O\_RDWR|O\_CREAT|O\_NOFOLLOW|O\_CLOEXEC, 0700) = 5**

**openat(AT\_FDCWD, "/dev/shm/mutex2", O\_RDWR|O\_CREAT|O\_NOFOLLOW|O\_CLOEXEC, 0700) = 6**

**ftruncate(3, 4096) = 0**

**ftruncate(4, 4) = 0**

**ftruncate(5, 40) = 0**

**ftruncate(6, 40) = 0**

**mmap(NULL, 4096, PROT\_READ|PROT\_WRITE, MAP\_SHARED, 3, 0) = 0x7fc8b141e000**

**mmap(NULL, 4096, PROT\_READ|PROT\_WRITE, MAP\_SHARED, 4, 0) = 0x7fc8b13f1000**

**mmap(NULL, 40, PROT\_READ|PROT\_WRITE, MAP\_SHARED, 5, 0) = 0x7fc8b13f0000**

**mmap(NULL, 40, PROT\_READ|PROT\_WRITE, MAP\_SHARED, 6, 0) = 0x7fc8b13ef000**

**clone(child\_stack=NULL, flags=CLONE\_CHILD\_CLEARTID|CLONE\_CHILD\_SETTID|SIGCHLD, child\_tidptr=0x7fc8b11c4a10) = 25837**

**write(1, "Enter sum terms:\n", 17Enter sum terms:**

**) = 17**

**write(1, ">", 1>) = 1**

**read(0, 1 2 3 4 5 6 7 8 9 10 123 41**

**"1 2 3 4 5 6 7 8 9 10 123 41\n", 1024) = 28**

**futex(0x7fc8b13f0000, FUTEX\_WAKE, 1) = 1**

**futex(0x7fc8b13ef000, FUTEX\_WAIT, 2, NULL) = 0**

**read(0, "", 1024) = 0**

**futex(0x7fc8b13f0000, FUTEX\_WAKE, 1) = 1**

**unlink("/dev/shm/pipe") = 0**

**unlink("/dev/shm/size") = 0**

**unlink("/dev/shm/mutex") = 0**

**unlink("/dev/shm/mutex2") = 0**

**exit\_group(0) = ?**

+++ exited with 0 +++

1. **Листинг программы**

*main.c*

#include <stdio.h>  
#include <unistd.h>  
#include <sys/mman.h>  
#include <sys/types.h>  
#include <fcntl.h>  
#include <pthread.h>  
  
int main() {  
 printf("enter file name:\n> ");  
 fflush(stdout);  
 char filename[256];  
 scanf("%s", filename);  
  
 char pipeName[] = "pipe";  
 char mutexName[] = "mutex";  
 char mutex2Name[] = "mutex2";  
 char pipe1SizeName[] = "size";  
  
  
 int pipe1 = shm\_open(pipeName, O\_RDWR | O\_CREAT, S\_IRWXU);  
 int pipe1Size = shm\_open(pipe1SizeName, O\_RDWR | O\_CREAT, S\_IRWXU);  
 int mutex1 = shm\_open(mutexName, O\_RDWR | O\_CREAT, S\_IRWXU);  
 int mutex2 = shm\_open(mutex2Name, O\_RDWR | O\_CREAT, S\_IRWXU);  
  
  
 if (pipe1 == -1 || pipe1Size == -1 || mutex1 == -1 || mutex2 == -1) {  
 perror("shm\_open\n");  
 return -1;  
 }  
 //проверка правильности выполнения системных вызовов  
 if (ftruncate(pipe1, getpagesize()) == -1) {  
 perror("ftruncate error\n");  
 return -1;  
 }  
 if (ftruncate(pipe1Size, sizeof(int)) == -1) {  
 perror("ftruncate error\n");  
 return -1;  
 }  
 if (ftruncate(mutex1, sizeof(pthread\_mutex\_t)) == -1) {  
 perror("ftruncate error\n");  
 return -1;  
 }  
 if (ftruncate(mutex2, sizeof(pthread\_mutex\_t)) == -1) {  
 perror("ftruncate error\n");  
 return -1;  
 }  
  
 int\* mmfData = (int \*) mmap(NULL, getpagesize(), PROT\_READ | PROT\_WRITE, MAP\_SHARED, pipe1, 0);  
 int\* mmfDataSize = (int \*) mmap(NULL, getpagesize(), PROT\_READ | PROT\_WRITE, MAP\_SHARED, pipe1Size, 0);  
 \*mmfDataSize = -1;  
 pthread\_mutex\_t \*mutex = (pthread\_mutex\_t \*) mmap(NULL, sizeof(pthread\_mutex\_t), PROT\_READ | PROT\_WRITE, MAP\_SHARED,  
 mutex1, 0);  
 pthread\_mutex\_t \*secondMutex = (pthread\_mutex\_t \*) mmap(NULL, sizeof(pthread\_mutex\_t), PROT\_READ | PROT\_WRITE,  
 MAP\_SHARED, mutex2, 0);  
  
  
 //------проверка системных вызовов------  
 if (mmfData == MAP\_FAILED || mmfDataSize == MAP\_FAILED || mutex == MAP\_FAILED || mutex2 == MAP\_FAILED) {  
 perror("mmap error\n");  
 return -1;  
 }  
 pthread\_mutexattr\_t mutexAttribute;  
 if (pthread\_mutexattr\_init(&mutexAttribute) != 0) {  
 perror("pthread\_mutexattr\_init error\n");  
 return -1;  
 }  
 if (pthread\_mutexattr\_setpshared(&mutexAttribute, PTHREAD\_PROCESS\_SHARED) != 0) {  
 perror("pthread\_mutexattr\_setpshared error\n");  
 return -1;  
 }  
 if (pthread\_mutex\_init(mutex, &mutexAttribute) != 0) {  
 perror("pthread\_mutex\_init error\n");  
 return -1;  
 }  
 if (pthread\_mutex\_init(secondMutex, &mutexAttribute) != 0) {  
 perror("pthread\_mutex\_init error\n");  
 return -1;  
 }  
  
 if (pthread\_mutex\_lock(secondMutex) != 0) {  
 perror("pthread\_mutex\_lock error\n");  
 return -1;  
 }  
 if (pthread\_mutex\_lock(mutex) != 0) {  
 perror("pthread\_mutex\_lock error\n");  
 return -1;  
 }  
 //----конец проверки системных вызовов----  
  
  
 int id = fork();  
 if (id == -1) {  
 perror("fork error\n");  
 return -1;  
 }  
  
 //child  
 else if (id == 0) {  
 char \*argv[] = {"child", filename, mutexName, mutex2Name, pipeName, pipe1SizeName, (char \*) NULL};  
 if (execv("child.out", argv) == -1) {  
 perror("execl error\n");  
 \_exit(1);  
 }  
 }  
  
 //parent  
 else {  
 printf("Enter sum terms:\n>");  
 fflush(stdout);  
 int currentTerm;  
 char c;  
  
 //счетчик, куда записываем данные  
 int cnt = 0;  
 int space\_flag = 0;  
  
 while (scanf("%d%c", &currentTerm, &c) > 0) {  
 //???????????????тут оказывается не нужен malloc??????????????????  
 mmfData[cnt] = currentTerm;  
 ++cnt;  
 if (c == '\n') {  
 \*mmfDataSize = cnt;  
  
 //-----проверка системных вызовов------  
 //первый mutex нужен, чтобы контролировать ребенка  
 if (pthread\_mutex\_unlock(mutex) != 0) {  
 perror("pthread\_mutex\_unlock error\n");  
 return -1;  
 }  
 //второй, чтобы контролировать батьку  
 if (pthread\_mutex\_lock(secondMutex) != 0) {  
 perror("pthread\_mutex\_lock error\n");  
 return -1;  
 }  
 //---конец проверки системных вызовов---  
  
 cnt = 0;  
 space\_flag = 0;  
 }  
  
 if(c == ' '){  
 space\_flag = 1;  
 }  
 }  
  
 //------защита от последнего пробела------  
 if(space\_flag){  
 \*mmfDataSize = cnt;  
  
 //-----проверка системных вызовов------  
 //первый mutex нужен, чтобы контролировать ребенка  
 if (pthread\_mutex\_unlock(mutex) != 0) {  
 perror("pthread\_mutex\_unlock error\n");  
 return -1;  
 }  
 //второй, чтобы контролировать батьку  
 if (pthread\_mutex\_lock(secondMutex) != 0) {  
 perror("pthread\_mutex\_lock error\n");  
 return -1;  
 }  
 //---конец проверки системных вызовов---  
  
 cnt = 0;  
 }  
 //--------------конец защиты--------------  
  
  
 //так мы говорим ребенку, что все закончилось  
 \*mmfDataSize = -2;  
 //даем добро на дочерний процесс  
 if (pthread\_mutex\_unlock(mutex) != 0) {  
 perror("pthread\_mutex\_unlock error\n");  
 return -1;  
 }  
 }  
  
 //вырубаем все то, что создали  
 if (shm\_unlink(pipeName) != 0) {  
 perror("shm\_unlink error\n");  
 return -1;  
 }  
 if (shm\_unlink(pipe1SizeName) != 0) {  
 perror("shm\_unlink error\n");  
 return -1;  
 }  
 if (shm\_unlink(mutexName) != 0) {  
 perror("shm\_unlink error\n");  
 return -1;  
 }  
 if (shm\_unlink(mutex2Name) != 0) {  
 perror("shm\_unlink error\n");  
 return -1;  
 }  
  
 return 0;  
}

*child.c*

#include <stdio.h>  
#include <unistd.h>  
#include <sys/mman.h>  
#include <sys/types.h>  
#include <fcntl.h>  
#include <pthread.h>  
  
int main(int argc, char \*argv[]) {  
 FILE \*ResFile = fopen(argv[1], "a");  
 if (ResFile == NULL) {  
 perror("fopen error\n");  
 return -1;  
 }  
  
 //инициализация mutex и тд  
 int mutex1 = shm\_open(argv[2], O\_RDWR | O\_CREAT, S\_IRWXU);  
 int mutex2 = shm\_open(argv[3], O\_RDWR | O\_CREAT, S\_IRWXU);  
 int pipe1 = shm\_open(argv[4], O\_RDWR | O\_CREAT, S\_IRWXU);  
 int pipe1Size = shm\_open(argv[5], O\_RDWR | O\_CREAT, S\_IRWXU);  
 if (pipe1 == -1 || pipe1Size == -1 || mutex1 == -1 || mutex2 == -1) {  
 perror("shm\_open\n");  
 return -1;  
 }  
  
 //навешивание mmap  
 pthread\_mutex\_t \*mutex = (pthread\_mutex\_t \*) mmap(NULL, sizeof(pthread\_mutex\_t), PROT\_READ | PROT\_WRITE,  
 MAP\_SHARED,  
 mutex1, 0);  
 pthread\_mutex\_t \*secondMutex = (pthread\_mutex\_t \*) mmap(NULL, sizeof(pthread\_mutex\_t),PROT\_READ | PROT\_WRITE,  
 MAP\_SHARED,  
 mutex2, 0);  
  
 int \*mmfData = (int \*) mmap(NULL, getpagesize(), PROT\_READ | PROT\_WRITE, MAP\_SHARED, pipe1, 0);  
 int \*mmfDataSize = (int \*) mmap(NULL, getpagesize(), PROT\_READ | PROT\_WRITE, MAP\_SHARED, pipe1Size, 0);  
 if (mmfData == MAP\_FAILED || mmfDataSize == MAP\_FAILED || mutex == MAP\_FAILED || secondMutex == MAP\_FAILED) {  
 perror("mmap error\n");  
 return -1;  
 }  
  
 //тут выполняется вся работа  
 while (\*mmfDataSize != -2) {  
 if (pthread\_mutex\_lock(mutex) != 0) {  
 perror("pthread\_mutex\_lock error\n");  
 return -1;  
 }  
 if (\*mmfDataSize == -2) {  
 break;  
 }  
 int result = 0;  
 for (int i = 0; i < \*mmfDataSize; ++i) {  
 fprintf(ResFile, "%d ", mmfData[i]);  
 result += mmfData[i];  
 }  
 fprintf(ResFile, "= %d\n", result);  
 \*mmfDataSize = -1;  
 if (pthread\_mutex\_unlock(secondMutex) != 0) {  
 perror("pthread\_mutex\_unlock error\n");  
 return -1;  
 }  
 }  
 fclose(ResFile);  
 return 0;  
}

1. **Вывод**

В ходе выполнения работы я узнал как какой-либо поток данных, поступаемый в одну программу, сохранить в общей памяти, чтобы использовать информацию в другой программе. Также я узнал как работать с mutex и на практике научился находить и устранять dead lock.

**Список литературы**

1. shm\_open — Linux manual page — man7.org

URL: <https://man7.org/linux/man-pages/man3/shm_open.3.html>

(дата обращения 22.12.2020)

2. mmap — Linux manual page — man7.org

URL: <https://man7.org/linux/man-pages/man2/mmap.2.html>

(дата обращения 22.12.2020)

3. Mutual Exlusion Lock Attributes — Multithreaded Programming Guide

URL: <https://docs.oracle.com/cd/E19683-01/806-6867/sync-83/index.html>

(дата обращения 22.12.2020)