

Московский Авиационный Институт  
(Национальный Исследовательский Университет)  
Факультет информационных технологий и прикладной математики  
Кафедра вычислительной математики и программирования

**Лабораторная работа №4 по курсу**  
**«Операционные системы»**  
**Тема работы**  
**“Межпроцессорное взаимодействие через memory-mapped files”**

Студент: Зубко Дмитрий Валерьевич  
Группа: М8О-208Б-20  
Вариант: 7  
Преподаватель: Миронов Евгений Сергеевич  
Оценка: \_\_\_\_\_  
Дата: \_\_\_\_\_  
Подпись: \_\_\_\_\_

Москва, 2021  
**Содержание**

1. Репозиторий
2. Постановка задачи
3. Общие сведения о программе
4. Общий метод и алгоритм решения
5. Исходный код
6. Демонстрация работы программы
7. Выводы

### Постановка задачи

#### Цель работы

Приобретение практических навыков в:

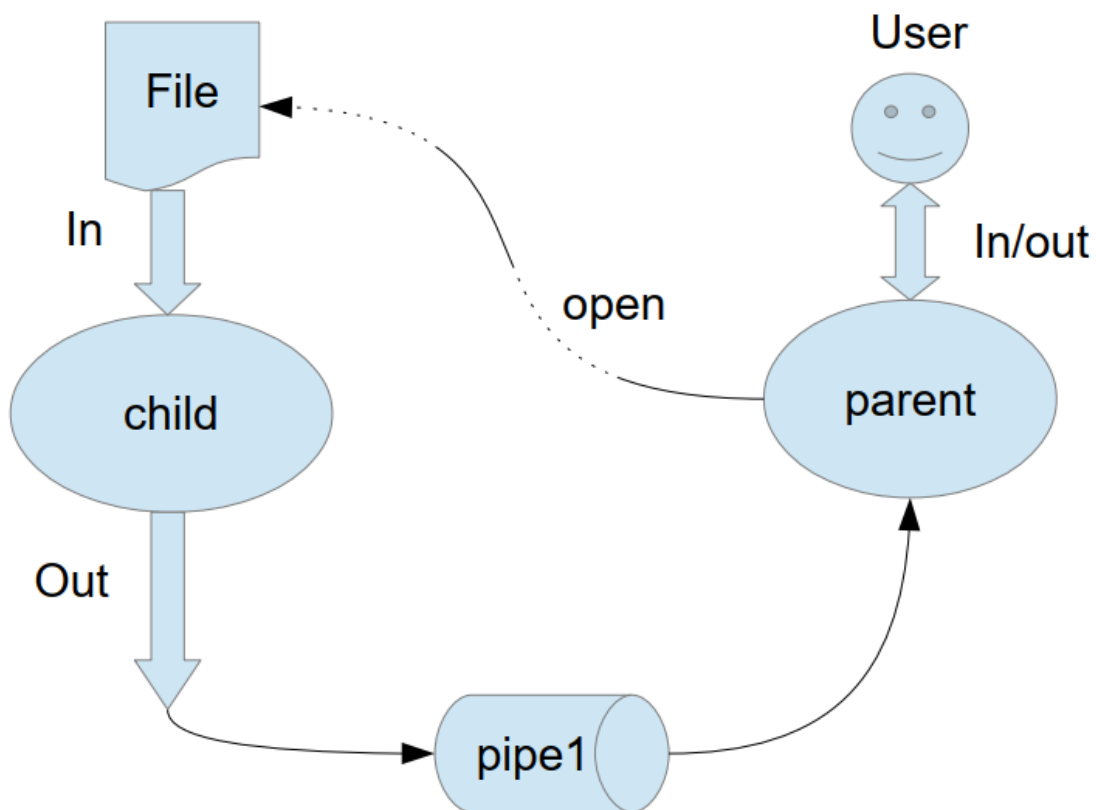
1. Освоение принципов работы с файловыми системами
2. Обеспечение обмена данных между процессами посредством технологии «File mapping»

#### Задание

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.

#### Группа вариантов 2



7 вариант) В файле записаны команды вида: «число число число<newline>». Дочерний процесс считает их сумму и выводит результат в стандартный поток вывода. Числа имеют тип float. Количество чисел может быть произвольным.

### Общие сведения о программе

Программа написана на языке C++ в UNIX-подобной системе. Сборка происходит с помощью make-файла

```
g++ -g -Wall -std=c++17 -pthread main.cpp -lrt -fsanitize=address
```

### Общий метод и алгоритм решения

Программа принимает на вход названия файла с числами. Создаются два семафора для синхронизации родительского и дочернего процессов. Также создаются два файловых дескриптора, которые служат обработчиками для исполнения mmap. Информация из входного файла считывается построчно и передается из родительского процесса через memptr1 к дочернему. Дочерний процесс обрабатывает строку, которую получает из memptr1, и кладёт ее в memptr2, который передаёт информацию из дочернего процесса в родительский.

### Исходный код

```
#include <iostream>
#include <semaphore.h>
#include <sys/stat.h>
#include <fcntl.h>
#include <string>
#include <cstdlib>
#include <fstream>
#include <unistd.h>
#include <sys/mman.h>

using namespace std;

int main() {
    const int map_size = 1024;
    string filename;
    string s = "";
    const char *input_sem_name = "input_sem";
    const char *output_sem_name = "output_sem";
    sem_unlink(input_sem_name);
    sem_unlink(output_sem_name);

    sem_t *input_semaphore = sem_open(input_sem_name, O_CREAT, S_IWUSR | S_IRUSR, 1);
```

```

sem_t *output_semaphore = sem_open(output_sem_name, O_CREAT, S_IWUSR | S_IRUSR, 0);
if(input_semaphore == SEM_FAILED || output_semaphore == SEM_FAILED){
    cout << "Error semaphore create" << endl;
    exit(1);
}

int map_fd1 = shm_open("map_fd1.txt", O_RDWR | O_CREAT, S_IWUSR | S_IRUSR);
int map_fd2 = shm_open("map_fd2.txt", O_RDWR | O_CREAT, S_IWUSR | S_IRUSR);
if (map_fd1 == -1 || map_fd2 == -1) {
    cout << "Error creating file for file mapping" << endl;
    exit(1);
}

char *memptr1 = (char *) mmap(nullptr, map_size, PROT_READ | PROT_WRITE, MAP_SHARED, map_fd1,
0);
char *memptr2 = (char *) mmap(nullptr, map_size, PROT_READ | PROT_WRITE, MAP_SHARED, map_fd2,
0);
if (memptr1 == MAP_FAILED || memptr2 == MAP_FAILED) {
    cout << "Error in file mapping" << endl;
    exit(1);
}

cout << "Enter file name: ";
cin >> filename;
ifstream file1;
file1.open(filename);
if (!file1.is_open()) {
    cout << "File open error" << endl;
    exit(1);
}

int id = fork();
if (id == -1) {
    cout << "Error fork" << endl;
    exit(1);
} else if (id == 0) {
    sem_wait(output_semaphore);
    sem_wait(input_semaphore);
    sem_post(output_semaphore);

    struct stat st;
    if (fstat(map_fd1, &st)) {
        cout << "Error fstat" << endl;
        exit(1);
    }

    int idx = 0;
    int length_1 = 0;
    for(int i = 0; i <= st.st_size; ++i){
        if (memptr1[i] != '\n') {
            s += memptr1[i];
        } else {
            long double res = 0;
            string result = "";
            double number;
            s += " ";

            for (unsigned j = 0; j < s.length(); ++j) {
                if (s[j] != ' ') {
                    result += s[j];
                } else {

```

```

        number = stof(result);
        res += number;
        result = "";
    }
}

s = to_string(res) + "\n";
length_1 += s.length() * sizeof(char);
if (ftruncate(map_fd2, length_1)) {
    cout << "Error ftruncate" << endl;
    exit(1);
}

for (unsigned j = 0; j < s.length(); ++j) {
    memptr2[idx] = s[j];
    idx += 1;
}

s = "";
}
}
sem_post(input_semaphore);
} else {
    sem_wait(input_semaphore);
    sem_post(output_semaphore);
    int idx = 0;
    int length = 0;

    while (!file1.eof()) {
        getline(file1, s);
        if (s != "") {
            s += "\n";

            length += s.length() * sizeof(char);
            if (ftruncate(map_fd1, length)) {
                cout << "Error during ftruncate" << endl;
                exit(1);
            }

            for (unsigned i = 0; i < s.length(); i++) {
                memptr1[idx] = s[i];
                idx += 1;
            }
        }
    }
    s = "";
    sem_post(input_semaphore);
    sem_wait(output_semaphore);
    sem_wait(input_semaphore);

    struct stat st;
    if (fstat(map_fd2, &st)) {
        cout << "Error fstat" << endl;
        exit(1);
    }

    for (int i = 0; i <= st.st_size; ++i) {
        if (memptr2[i] != '\n') {
            s += memptr2[i];
        } else {
            s += "\n";

```

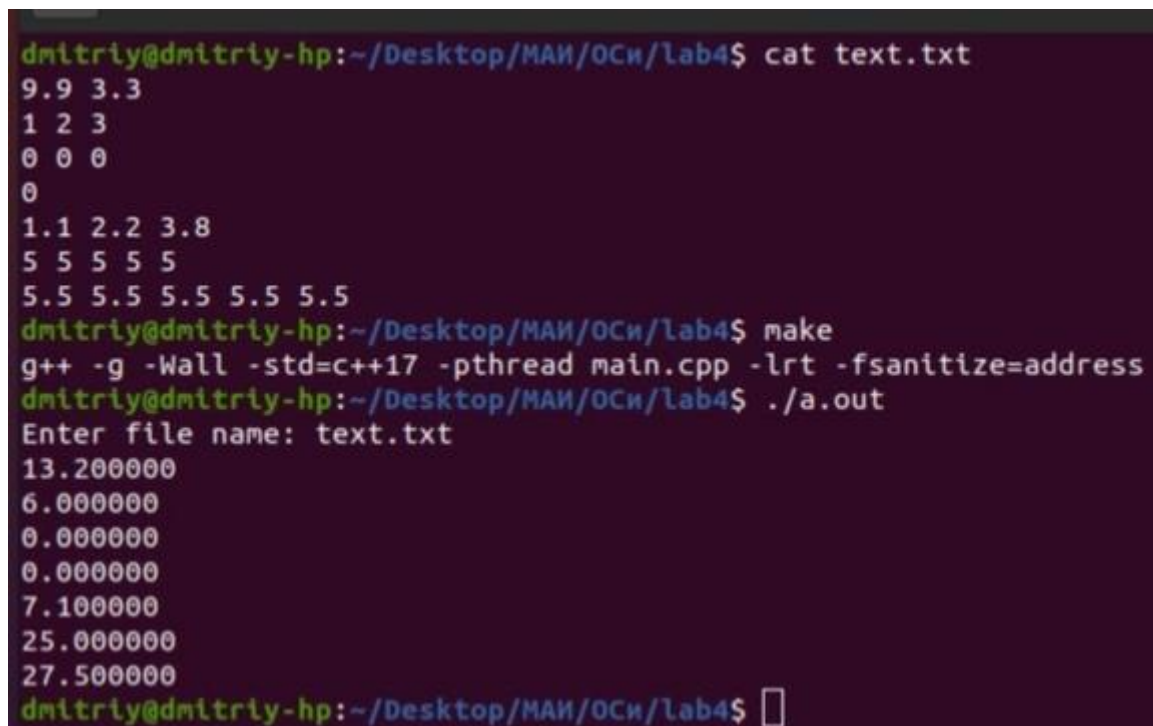
```

        cout << s;
        s = "";
    }
}

munmap(memptr1, map_size);
munmap(memptr2, map_size);
shm_unlink("map_fd1.txt");
shm_unlink("map_fd2.txt");
remove("map_fd1.txt");
remove("map_fd2.txt");
sem_destroy(input_semaphore);
sem_destroy(output_semaphore);
close(map_fd1);
close(map_fd2);
}

```

## Демонстрация работы программы



```

dmitriy@dmitriy-hp:~/Desktop/МАН/ОСк/lab4$ cat text.txt
9.9 3.3
1 2 3
0 0 0
0
1.1 2.2 3.8
5 5 5 5 5
5.5 5.5 5.5 5.5 5.5
dmitriy@dmitriy-hp:~/Desktop/МАН/ОСк/lab4$ make
g++ -g -Wall -std=c++17 -pthread main.cpp -lrt -fsanitize=address
dmitriy@dmitriy-hp:~/Desktop/МАН/ОСк/lab4$ ./a.out
Enter file name: text.txt
13.200000
6.000000
0.000000
0.000000
7.100000
25.000000
27.500000
dmitriy@dmitriy-hp:~/Desktop/МАН/ОСк/lab4$ 

```

## Выводы

Я освоил принципы работы с файловыми системами. Научился пользоваться технологией «File mapping» для обмена данными между процессами. Синхронизировал работу процессов с помощью семафоров.