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

**Репозиторий**

https://github.com/usernameMAI/OS/tree/main/os\_lab4

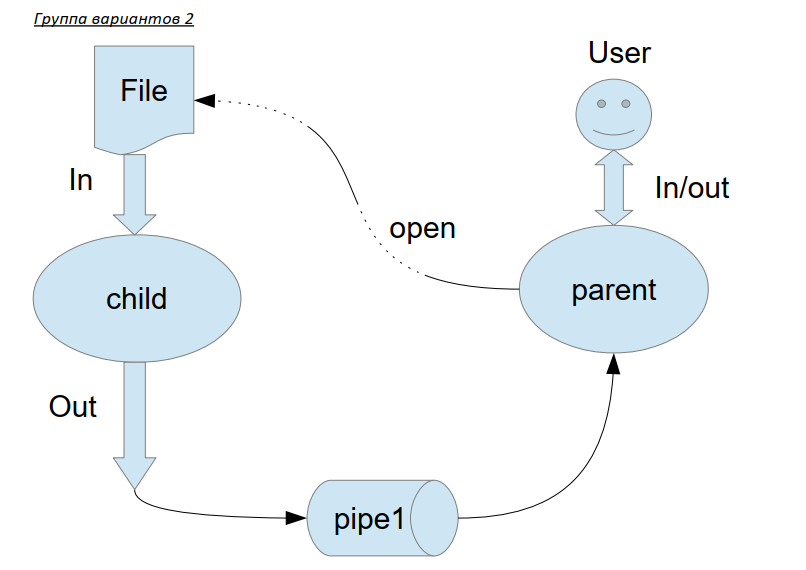
**Постановка задачи**

**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.



7 вариант) В файле записаны команды вида: «число число число<endline>». 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

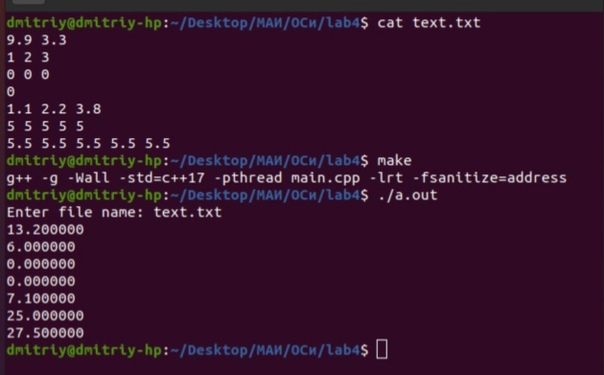
**Общий метод и алгоритм решения**

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**



**Выводы**

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