Московский Авиационный Институт

(Национальный Исследовательский Университет)

Факультет информационных технологий и прикладной математики

Кафедра вычислительной математики и программирования

**Лабораторная работа № 4 по курсу**

**«Операционные системы»**

Студент: Меджидли Исмаил Ибрагим оглы

Группа: М8О-201Б-21

Вариант: 3

Преподаватель: Миронов Евгений Сергеевич

Оценка: \_\_\_\_\_\_\_\_\_\_\_

Дата: \_\_\_\_\_\_\_\_\_\_\_

Подпись: \_\_\_\_\_\_\_\_\_\_\_

Москва, 2023

**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/imedzhidli/Operational-Systems

**Постановка задачи**

Пользователь вводит команды вида: «число число число<endline>». Далее эти числа передаются от родительского процесса в дочерний. Дочерний процесс производит деление первого числа, на последующие, а результат выводит в файл. Если происходит деление на 0, то тогда дочерний и родительский процесс завершают свою работу. Проверка деления на 0 должна осуществляться на стороне дочернего процесса. Числа имеют тип int. Количество чисел может быть произвольным.

**Общие сведения о программе**

CMakeLists.txt - описание процесса сборки проекта  
  
main.cpp - перенаправление потока ввода в функцию ParentRoutine  
  
parent.h - заголовочный файл, в котором описана функция родительского  
  
string\_to\_vector.h - сигнатура функции, которая преобразует строку в вектор float

string\_to\_vector.cpp - реализация функции

parent.cpp - реализация функции родительского процесса

child.cpp - отдельная программа дочернего процесса

lab4\_test.cpp - тесты к лабораторной работе

**Общий метод и алгоритм решения**В родительском процессе создается канал(pipe) и дочерний процесс с помощью системного вызова fork, дочерний процесс получает данные с помощью pipe и запускает программу child.cpp с помощью execl. А уже в child.cpp выполняется задание по варианту и запись в файл.

**Исходный код**

**CMakeLists.txt**

**cmake\_minimum\_required(VERSION 3.16.3)**

**project(lab4 LANGUAGES CXX)**

**find\_package(Threads REQUIRED)**

**set(CMAKE\_THREAD\_LIBS\_INIT "-lpthread")**

**add\_executable(lab4**

**main.cpp**

**include/parent.h src/parent.cpp**

**)**

**target\_link\_libraries(lab4 PRIVATE rt)**

**target\_link\_libraries(lab4 PRIVATE Threads::Threads)**

**target\_include\_directories(lab4 PRIVATE include)**

**add\_executable(child**

**src/child.cpp**

**include/string\_to\_vector.h src/string\_to\_vector.cpp**

**)**

**target\_link\_libraries(child PRIVATE rt)**

**target\_link\_libraries(child PRIVATE Threads::Threads)**

**target\_include\_directories(child PRIVATE include)**

**add\_dependencies(lab4 child)**

**main.cpp**

**#include "parent.h"**

**using namespace std;**

**int main() {**

**ParentRoutine(cin, getenv("PATH\_TO\_CHILD4"));**

**return EXIT\_SUCCESS;**

**}**

**parent.h**

**#ifndef PARENT\_H**

**#define PARENT\_H**

**#include <istream>**

**#include <vector>**

**#include <iostream>**

**#include <string>**

**#include <unistd.h>**

**#include <cstdlib>**

**#include <fstream>**

**#include <cstdio>**

**#include <sys/wait.h>**

**#include <array>**

**#include <iterator>**

**#include <pthread.h>**

**#include <algorithm>**

**#include <sys/types.h>**

**#include <sys/mman.h>**

**#include <fcntl.h>**

**#include <semaphore.h>**

**#include <cstring>**

**using namespace std;**

**void ParentRoutine(istream& stream, const char\* pathToChild);**

**#endif**

**parent.cpp**

**#include "parent.h"**

**#include <algorithm>**

**#include <cstring>**

**#include <semaphore.h>**

**#include <sys/mman.h>**

**#include <sys/types.h>**

**#include <unistd.h>**

**#include <fcntl.h>**

**using namespace std;**

**constexpr auto SHARED\_MEMORY\_OBJECT\_NAME = "shared\_memory";**

**constexpr auto SHARED\_MEMORY\_SEMAPHORE\_INPUT\_NAME = "shared\_semaphore\_input";**

**constexpr auto SHARED\_MEMORY\_SEMAPHORE\_OUTPUT\_NAME = "shared\_semaphore\_output";**

**void ParentRoutine(istream& stream, const char\* pathToChild) {**

**string nameOutputFile;**

**getline(stream, nameOutputFile);**

**/\* shared memory file descriptor \*/**

**int sfd;**

**int semInFd;**

**int semOutFd;**

**/\* create the shared memory object \*/**

**if ((sfd = shm\_open(SHARED\_MEMORY\_OBJECT\_NAME, O\_CREAT | O\_RDWR, S\_IRWXU)) == -1) {**

**cout << "Shm\_open error" << endl;**

**exit(EXIT\_FAILURE);**

**}**

**if ((semInFd = shm\_open(SHARED\_MEMORY\_SEMAPHORE\_INPUT\_NAME, O\_CREAT | O\_RDWR, S\_IRWXU)) == -1) {**

**cout << "Shm\_open error" << endl;**

**exit(EXIT\_FAILURE);**

**}**

**if ((semOutFd = shm\_open(SHARED\_MEMORY\_SEMAPHORE\_OUTPUT\_NAME, O\_CREAT | O\_RDWR, S\_IRWXU)) == -1) {**

**cout << "Shm\_open error" << endl;**

**exit(EXIT\_FAILURE);**

**}**

**/\* configure the size of the shared memory object \*/**

**ftruncate(sfd, getpagesize());**

**ftruncate(semInFd, getpagesize());**

**ftruncate(semOutFd, getpagesize());**

**auto \*semInput = (sem\_t\*)mmap(nullptr, getpagesize(), PROT\_WRITE | PROT\_READ, MAP\_SHARED, semInFd, 0);**

**auto \*semOutput = (sem\_t\*)mmap(nullptr, getpagesize(), PROT\_WRITE | PROT\_READ, MAP\_SHARED, semOutFd, 0);**

**if (semInput == MAP\_FAILED) {**

**cout << "Mmap error" << endl;**

**exit(EXIT\_FAILURE);**

**}**

**if (semOutput == MAP\_FAILED) {**

**cout << "Mmap error" << endl;**

**exit(EXIT\_FAILURE);**

**}**

**sem\_init(semInput, 1, 1);**

**sem\_init(semOutput, 1, 0);**

**int pid = fork();**

**if (pid == -1) {**

**cout << "Error creating process\n";**

**exit(EXIT\_FAILURE);**

**}**

**if (pid != 0) { // родительский процесс**

**/\* memory map the shared memory object \*/**

**char\* ptr = (char\*)mmap(nullptr, getpagesize(), PROT\_WRITE | PROT\_READ, MAP\_SHARED, sfd, 0);**

**if (ptr == MAP\_FAILED) {**

**cout << "Mmap error" << endl;**

**exit(EXIT\_FAILURE);**

**}**

**string stringNumbers;**

**while (getline(stream, stringNumbers)) {**

**sem\_wait(semInput);**

**if (string(ptr) == "Division by zero.") {**

**sem\_post(semInput);**

**break;**

**}**

**stringNumbers += "\n";**

**sprintf((char \*) ptr, "%s", stringNumbers.c\_str());**

**sem\_post(semOutput);**

**}**

**sem\_wait(semInput);**

**sprintf((char \*) ptr, "%s", "");**

**sem\_post(semOutput);**

**wait(nullptr);**

**if (sem\_destroy(semInput) == -1) {**

**cout << "Sem\_destroy error" << endl;**

**exit(EXIT\_FAILURE);**

**}**

**if (sem\_destroy(semOutput) == -1) {**

**cout << "Sem\_destroy error" << endl;**

**exit(EXIT\_FAILURE);**

**}**

**if (munmap(semInput, getpagesize()) == -1) {**

**cout << "Munmap error" << endl;**

**exit(EXIT\_FAILURE);**

**}**

**if (munmap(semOutput, getpagesize()) == -1) {**

**cout << "Munmap error" << endl;**

**exit(EXIT\_FAILURE);**

**}**

**if (munmap(ptr, getpagesize()) == -1) {**

**cout << "Munmap error" << endl;**

**exit(EXIT\_FAILURE);**

**}**

**if (shm\_unlink(SHARED\_MEMORY\_OBJECT\_NAME) == -1) {**

**cout << "Shm\_unlink error" << endl;**

**exit(EXIT\_FAILURE);**

**}**

**if (shm\_unlink(SHARED\_MEMORY\_SEMAPHORE\_INPUT\_NAME) == -1) {**

**cout << "Shm\_unlink error" << endl;**

**exit(EXIT\_FAILURE);**

**}**

**if (shm\_unlink(SHARED\_MEMORY\_SEMAPHORE\_OUTPUT\_NAME) == -1) {**

**cout << "Shm\_unlink error" << endl;**

**exit(EXIT\_FAILURE);**

**}**

**}**

**else { // дочерний процесс**

**if(execl(pathToChild, pathToChild, nameOutputFile.data(),**

**SHARED\_MEMORY\_OBJECT\_NAME, SHARED\_MEMORY\_SEMAPHORE\_INPUT\_NAME,**

**SHARED\_MEMORY\_SEMAPHORE\_OUTPUT\_NAME, nullptr) == -1) {**

**cout << "Failed to exec\n";**

**exit(EXIT\_FAILURE);**

**}**

**}**

**}**

**child.cpp**

**#include <istream>**

**#include <ostream>**

**#include <vector>**

**#include <iostream>**

**#include <string>**

**#include <unistd.h>**

**#include <cstdlib>**

**#include <fstream>**

**#include <cstdio>**

**#include <sys/wait.h>**

**#include <pthread.h>**

**#include <sys/types.h>**

**#include <sys/mman.h>**

**#include <fcntl.h>**

**#include <cstring>**

**#include <semaphore.h>**

**#include "string\_to\_vector.h"**

**using namespace std;**

**int main(int argc, char\* argv[]) {**

**if (argc != 5) {**

**cout << "Invalid arguments.\n";**

**exit(EXIT\_FAILURE);**

**}**

**auto \*nameOutputFile = argv[1];**

**ofstream out(nameOutputFile);**

**int sfd;**

**int semInFd;**

**int semOutFd;**

**if ((sfd = shm\_open(argv[2], O\_RDWR, S\_IRWXU)) == -1) {**

**cout << "shm\_open error" << endl;**

**exit(EXIT\_FAILURE);**

**}**

**if ((semInFd = shm\_open(argv[3], O\_RDWR, S\_IRWXU)) == -1) {**

**cout << "Shm\_open error" << endl;**

**exit(EXIT\_FAILURE);**

**}**

**if ((semOutFd = shm\_open(argv[4], O\_RDWR, S\_IRWXU)) == -1) {**

**cout << "Shm\_open error" << endl;**

**exit(EXIT\_FAILURE);**

**}**

**char\* ptr = (char\*)mmap(nullptr, getpagesize(), PROT\_READ | PROT\_WRITE, MAP\_SHARED, sfd, 0);**

**if (ptr == MAP\_FAILED) {**

**cout << "error mmap func" << endl;**

**exit(EXIT\_FAILURE);**

**}**

**auto \*semInput = (sem\_t\*)mmap(nullptr, getpagesize(), PROT\_WRITE | PROT\_READ, MAP\_SHARED, semInFd, 0);**

**auto \*semOutput = (sem\_t\*)mmap(nullptr, getpagesize(), PROT\_WRITE | PROT\_READ, MAP\_SHARED, semOutFd, 0);**

**if (semInput == MAP\_FAILED) {**

**cout << "Mmap error" << endl;**

**exit(EXIT\_FAILURE);**

**}**

**if (semOutput == MAP\_FAILED) {**

**cout << "Mmap error" << endl;**

**exit(EXIT\_FAILURE);**

**}**

**while (true) {**

**sem\_wait(semOutput);**

**string stringNumbers = ptr;**

**if (stringNumbers.empty()) {**

**sem\_post(semInput);**

**break;**

**}**

**vector <int> numbers = StringToVector(stringNumbers);**

**float firstNumber = numbers[0];**

**for (size\_t i = 1; i < numbers.size(); i++) {**

**if (numbers[i] == 0) {**

**cout << "Division by zero.\n";**

**out << "\n";**

**out.close();**

**sprintf((char \*) ptr, "%s", "Division by zero.");**

**sem\_post(semInput);**

**exit(EXIT\_FAILURE);**

**}**

**firstNumber /= numbers[i];**

**}**

**out << firstNumber << " ";**

**sem\_post(semInput);**

**}**

**out << "\n";**

**out.close();**

**if (munmap(ptr, getpagesize()) == -1) {**

**cout << "Munmap error" << endl;**

**exit(EXIT\_FAILURE);**

**}**

**if (munmap(semInput, getpagesize()) == -1) {**

**cout << "Munmap error" << endl;**

**exit(EXIT\_FAILURE);**

**}**

**if (munmap(semOutput, getpagesize()) == -1) {**

**cout << "Munmap error" << endl;**

**exit(EXIT\_FAILURE);**

**}**

**return EXIT\_SUCCESS;**

**}**

**string\_to\_vector.h**

**#ifndef STRING\_TO\_VECTOR\_H**

**#define STRING\_TO\_VECTOR\_H**

**#include <vector>**

**#include <iostream>**

**#include <string>**

**#include <cstring>**

**#include <cstdlib>**

**#include <algorithm>**

**using namespace std;**

**vector<int> StringToVector(string const& stringNumbers, char separator=' ');**

**#endif//STRING\_TO\_VECTOR\_H**

**string\_to\_vector.cpp**

**#include "string\_to\_vector.h"**

**using namespace std;**

**vector<int> StringToVector(string const& stringNumbers, char separator) {**

**vector<int> results;**

**auto start = stringNumbers.begin();**

**auto end = stringNumbers.end();**

**auto next = find(start, end, separator);**

**while (next != end) {**

**results.push\_back(stof(string(start, next)));**

**start = next + 1;**

**next = find(start, end, separator);**

**}**

**results.push\_back(stof(string(start, next)));**

**return results;**

**}**

**lab2\_test.cpp**

**#include <cstdio>**

**#include <cstdlib>**

**#include <fstream>**

**#include <gtest/gtest.h>**

**#include <string>**

**#include "parent.h"**

**#include "string\_to\_vector.h"**

**using namespace std;**

**TEST(Lab4Test, StringToVectorTest) {**

**vector <vector <int>> expectedVectors = {**

**{1, 2, 3, 4, 5},**

**{200, 4, 5},**

**{10, 0}**

**};**

**vector <string> inputStrings = {**

**"1 2 3 4 5",**

**"200 4 5",**

**"10 0"**

**};**

**long unsigned int countTests = 3;**

**for (long unsigned int i = 0; i < countTests; i++) {**

**vector <int> outputVector = StringToVector(inputStrings[i]);**

**ASSERT\_EQ(expectedVectors[i].size(), outputVector.size());**

**for (long unsigned int j = 0; j < expectedVectors[i].size(); j++) {**

**EXPECT\_EQ(expectedVectors[i][j], outputVector[j]);**

**EXPECT\_EQ(1, 1);**

**}**

**}**

**}**

**TEST(Lab4Test, ParentTest) {**

**vector <string> namesOutputFile = {**

**"checker.txt",**

**"output.txt",**

**"jambo.tea"**

**};**

**vector <string> stringsNumbers = {**

**"200 4 5\n800 8\n1\n90 2"**

**""**

**};**

**vector <string> expectedStrings = {**

**"10 100 1 45 "**

**};**

**long unsigned int countTests = 1;**

**for (long unsigned int i = 0; i < countTests; i++) {**

**{**

**ofstream fOut("input.txt");**

**fOut << namesOutputFile[i] << "\n";**

**fOut << stringsNumbers[i] << "\n";**

**}**

**{**

**ifstream fIn("input.txt");**

**ParentRoutine(fIn, getenv("PATH\_TO\_CHILD4"));**

**}**

**remove("input.txt");**

**{**

**ifstream fInCheckOutput = ifstream(namesOutputFile[i]);**

**ASSERT\_TRUE(fInCheckOutput.good());**

**string outputString;**

**getline(fInCheckOutput, outputString);**

**EXPECT\_EQ(outputString, expectedStrings[i]);**

**fInCheckOutput.clear();**

**}**

**remove(namesOutputFile[i].data());**

**}**

**}**

**Демонстрация работы программы**

imedzhidli@imedzhidli:~/Desktop/OS/4Laba/build$ ls

child CMakeCache.txt CMakeFiles cmake\_install.cmake compile\_commands.json imedy.txt lab4 Makefile test.txt

imedzhidli@imedzhidli:~/Desktop/OS/4Laba/build$ cat test.txt

imedy.txt

1 2 3 4 5

100 2 5

500 2 6 6

10 2 0

imedzhidli@imedzhidli:~/Desktop/OS/4Laba/build$ ./lab4 < test.txt

Division by zero.

imedzhidli@imedzhidli:~/Desktop/OS/4Laba/build$ cat imedy.txt

0.00833333 10 6.94444

imedzhidli@imedzhidli:~/Desktop/OS/4Laba/build$

**Выводы**Приобрел практические навыки в:

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