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: Постнов Александр Вячеславович
Группа: М8О-201Б-21
Вариант: 4
Преподаватель: Миронов Евгений Сергеевич
Оценка: _____
Дата: _____
Подпись: _____

Москва, 2022

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/kappaprideonly/mai-os-labs

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 float. 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

main перенаправляет ввод в родительский процесс, родительский процесс создает дочерний процесс с помощью fork, дочерний процесс запускает отдельно программу. Процессы взаимодействуют с друг другом через файлы, отображаемые в память. Чтобы действие по варианту происходило построчно, использовал примитив синхронизации семафор. 1 семафор на ввод данных, 2 семафор на обработку данных.

Исходный код

CMakeLists.txt

```
add_executable(lab4
        main.cpp
        include/parent.h src/parent.cpp)

target_include_directories(lab4 PRIVATE include)

add_executable(child4
        src/child.cpp
        include/string_to_vector.h src/string_to_vector.cpp)

target_include_directories(child4 PRIVATE include)

add_dependencies(lab4 child4)
```

main.cpp

```
#include "parent.h"


int main() {
   ParentRoutine(std::cin, getenv("PATH_TO_CHILD"));
   return 0;
}
```

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>



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

#endif
```

parent.cpp

```
#include "parent.h"
#include <algorithm>
#include <cstring>
#include <semaphore.h>
#include <sys/mman.h>
#include <unistd.h>
```

```
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(std::istream& stream, const char* pathToChild) {
    // Clear();
    std::string nameOutputFile;
    std::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) {
        std::cout << "Shm_open error" << std::endl;
        exit(EXIT_FAILURE);
    }
    if ((semInFd = shm_open(SHARED_MEMORY_SEMAPHORE_INPUT_NAME, O_CREAT |
O_RDWR, S_IRWXU)) == -1) {
        std::cout << "Shm_open error" << std::endl;
        exit(EXIT_FAILURE);
    }
    if ((semOutFd = shm_open(SHARED_MEMORY_SEMAPHORE_OUTPUT_NAME, O_CREAT |
O_RDWR, S_IRWXU)) == -1) {
        std::cout << "Shm_open error" << std::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) {
        std::cout << "Mmap error" << std::endl;
        exit(EXIT_FAILURE);
```

```
    }

    if (semOutput == MAP_FAILED) {
        std::cout << "Mmap error" << std::endl;
        exit(EXIT_FAILURE);
    }

    sem_init(semInput, 1, 1);
    sem_init(semOutput, 1, 0);


    int pid = fork();
    if (pid == -1) {
        std::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) {
            std::cout << "Mmap error" << std::endl;
            exit(EXIT_FAILURE);
        }

        std::string stringNumbers;
        while (std::getline(stream, stringNumbers)) {
            sem_wait(semInput);
            if (std::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) {
            std::cout << "Sem_destroy error" << std::endl;
            exit(EXIT_FAILURE);
        }
```

```
        if (sem_destroy(semOutput) == -1) {
            std::cout << "Sem_destroy error" << std::endl;
            exit(EXIT_FAILURE);
        }
        if (munmap(semInput, getpagesize()) == -1) {
            std::cout << "Munmap error" << std::endl;
            exit(EXIT_FAILURE);
        }
        if (munmap(semOutput, getpagesize()) == -1) {
            std::cout << "Munmap error" << std::endl;
            exit(EXIT_FAILURE);
        }
        if (munmap(ptr, getpagesize()) == -1) {
            std::cout << "Munmap error" << std::endl;
            exit(EXIT_FAILURE);
        }
        if (shm_unlink(SHARED_MEMORY_OBJECT_NAME) == -1) {
            std::cout << "Shm_unlink error" << std::endl;
            exit(EXIT_FAILURE);
        }
        if (shm_unlink(SHARED_MEMORY_SEMAPHORE_INPUT_NAME) == -1) {
            std::cout << "Shm_unlink error" << std::endl;
            exit(EXIT_FAILURE);
        }
        if (shm_unlink(SHARED_MEMORY_SEMAPHORE_OUTPUT_NAME) == -1) {
            std::cout << "Shm_unlink error" << std::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) {
            std::cout << "Failed to exec\n";
            exit(EXIT_FAILURE);
        }
    }
}
```

child.cpp

```
#include <istream>
```

```cpp
#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"


int main(int argc, char* argv[]) {
    if (argc != 5) {
        std::cout << "Invalid arguments.\n";
        exit(EXIT_FAILURE);
    }

    auto *nameOutputFile = argv[1];
    std::ofstream out(nameOutputFile);
    int sfd;
    int semInFd;
    int semOutFd;
    if ((sfd = shm_open(argv[2], O_RDWR, S_IRWXU)) == -1) {
        std::cout << "shm_open error" << std::endl;
        exit(EXIT_FAILURE);
    }

    if ((semInFd = shm_open(argv[3], O_RDWR, S_IRWXU)) == -1) {
        std::cout << "Shm_open error" << std::endl;
        exit(EXIT_FAILURE);
    }
    if ((semOutFd = shm_open(argv[4], O_RDWR, S_IRWXU)) == -1) {
        std::cout << "Shm_open error" << std::endl;
        exit(EXIT_FAILURE);
    }
```

```
    char* ptr = (char*)mmap(nullptr, getpagesize(), PROT_READ | PROT_WRITE,
MAP_SHARED, sfd, 0);
    if (ptr == MAP_FAILED) {
        std::cout << "error mmap func" << std::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) {
        std::cout << "Mmap error" << std::endl;
        exit(EXIT_FAILURE);
    }

    if (semOutput == MAP_FAILED) {
        std::cout << "Mmap error" << std::endl;
        exit(EXIT_FAILURE);
    }
    while (true) {
        sem_wait(semOutput);
        std::string stringNumbers = ptr;
        if (stringNumbers.empty()) {
            sem_post(semInput);
            break;
        }
        std::vector <float> numbers = StringToVectorFloats(stringNumbers);
        float firstNumber = numbers[0];
        for (size_t i = 1; i < numbers.size(); i++) {
            if (numbers[i] == 0) {
                std::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) {
        std::cout << "Munmap error" << std::endl;
        exit(EXIT_FAILURE);
    }
    if (munmap(semInput, getpagesize()) == -1) {
        std::cout << "Munmap error" << std::endl;
        exit(EXIT_FAILURE);
    }
    if (munmap(semOutput, getpagesize()) == -1) {
        std::cout << "Munmap error" << std::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>

std::vector<float> StringToVectorFloats(std::string const& stringNumbers,
char separator=' ');

#endif//STRING_TO_VECTOR_H
```

string_to_vector.cpp

```
#include "string_to_vector.h"


std::vector<float> StringToVectorFloats(std::string const& stringNumbers,
char separator) {
    std::vector<float> results;
```

```
    auto start = stringNumbers.begin();
    auto end = stringNumbers.end();
    auto next = std::find(start, end, separator);
    while (next != end) {
        results.push_back(stof(std::string(start, next)));
        start = next + 1;
        next = std::find(start, end, separator);
    }
    results.push_back(stof(std::string(start, next)));
    return results;
}
```

lab4_test.cpp

```
#include <cstdio>
#include <cstdlib>
#include <fstream>
#include <gtest/gtest.h>
#include <string>

#include "parent.h"
#include "string_to_vector.h"


TEST(Lab4Test, StringToVectorTest) {
    std::vector <std::vector <float>> expectedVectors = {
            {1.5, 2.5, 3.5},
            {1.5, 2, 3, 4, 5, 0},
            {1}
    };
    std::vector <std::string> inputStrings = {
            "1.5 2.5 3.5",
            "1.5 2 3 4 5 0",
            "1"
    };

    long unsigned int countTests = 3;
    for (long unsigned int i = 0; i < countTests; i++) {
        std::vector <float> outputVector =
StringToVectorFloats(inputStrings[i]);
        ASSERT_EQ(expectedVectors[i].size(), outputVector.size());
```

```
        for (long unsigned int j = 0; i < expectedVectors[i].size(); i++) {
            EXPECT_FLOAT_EQ(expectedVectors[i][j], outputVector[j]);
        }
    }
}


TEST(Lab4Test, ParentTest) {
    std::vector <std::string> namesOutputFile = {
            "checker.txt",
            "output.txt",
            "jambo.tea"
    };
    std::vector <std::string> stringsNumbers = {
            "1 0.5 0.5 0.5\n100 8\n1\n90 2",
            "1 0.5 0.5\n100 0\n1\n90 2\n1 1",
            "1 0 0.5 0.5\n100 0\n1\n90 2"
    };
    std::vector <std::string> expectedStrings = {
            "8 12.5 1 45 ",
            "4 ",
            ""
    };

    long unsigned int countTests = 3;
    for (long unsigned int i = 0; i < countTests; i++) {
        {
            std::ofstream fOut("input.txt");
            fOut << namesOutputFile[i] << "\n";
            fOut << stringsNumbers[i] << "\n";
        }

        {
            std::ifstream fIn("input.txt");
            ParentRoutine(fIn, getenv("PATH_TO_CHILD4"));
        }
        remove("input.txt");

        {
            std::ifstream fInCheckOutput =
std::ifstream(namesOutputFile[i]);
            ASSERT_TRUE(fInCheckOutput.good());
            std::string outputString;
            std::getline(fInCheckOutput, outputString);
            EXPECT_EQ(outputString, expectedStrings[i]);
```

```
        }

        remove(namesOutputFile[i].data());
    }
}
```

Демонстрация работы программы

➜ lab4 git:(main) ls

child4 CMakeFiles lab4 test.txt

cmake_install.cmake CTestTestfile.cmake Makefile

➜ lab4 git:(main) cat test.txt

out.txt

100 2 5

5 5 1

500 5 100

➜ lab4 git:(main) ./lab4 <test.txt

➜ lab4 git:(main) ls

child4 CMakeFiles lab4 out.txt

```
cmake_install.cmake   CTestTestfile.cmake   Makefile   test.txt
➜  lab4 git:(main) cat out.txt
10 1 1
```

Выводы:

Приобрел практические навыки в:

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