Московский Авиационный Институт

(Национальный Исследовательский Университет)

Факультет информационных технологий и прикладной математики

Кафедра вычислительной математики и программирования

**Лабораторная работа № 1 по курсу**

**«Операционные системы»**

Студент: Бурдинский Владислав Дмитриевич

Группа: М8О–203Б–23

Вариант: 5

Преподаватель: Миронов Евгений Сергеевич

Оценка: \_\_\_\_\_\_\_\_\_\_

Дата: \_\_\_\_\_\_\_\_\_\_

Подпись: \_\_\_\_\_\_\_\_\_\_

Москва, 2024.

Цель работы

Приобретение практических навыков в:

Управление процессами в ОС

Обеспечение обмена данных между процессами посредством каналов

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 каналы (pipe).

* Необходимо обрабатывать системные ошибки, которые могут возникнуть в результате работы.

**Код программы**

**child.cpp**

#include <string>

#include <iostream>

#include <sstream>

#include <fstream>

#include <unistd.h>

#include "child.hpp"

int calculation(int a, int b, int c){

return a / b / c;

}

int main(){

std::string input;

std::getline(std::cin, input);

std::stringstream ss(input);

std::string result;

bool flag = true;

int num1, num2, num3;

if (ss >> num1 >> num2 >> num3) {

std::string extra;

if (ss >> extra){

result = "Некорректный ввод";

flag = false;

}

} else{

flag = false;

result = "Некорректный ввод";

}

if (flag == true){

if ((num2 == 0) || (num3 == 0)){

result = "Делить на ноль нельзя!";

} else{

result = "Результат: " + std::to\_string(calculation(num1, num2, num3));

}

}

std::ofstream outfile("result.txt");

if (!outfile) {

std::cerr << "Ошибка при открытии файла для записи" << std::endl;

return 1;

}

outfile << result << std::endl;

outfile.close();

if (write(STDOUT\_FILENO, result.c\_str(), result.size()) == -1){

perror("Не получается отдать результат родителю");

return 1;

}

return 0;

}

parent.cpp

#include <parent.hpp>

void ParentProcess(const char \* pathToChild, std::istream & streamIn, std::ostream & streamOut){

int Parent\_Child[2];

CreatePipe(Parent\_Child);

int Child\_Parent[2];

CreatePipe(Child\_Parent);

pid\_t pid = CreateChild();

//Мы в дочернем процессе

if(pid == 0){

close(Parent\_Child[1]);

close(Child\_Parent[0]);

if (dup2(Parent\_Child[0], STDIN\_FILENO) == -1){

perror("dup2 error");

exit(EXIT\_FAILURE);

}

if (dup2(Child\_Parent[1], STDOUT\_FILENO) == -1){

perror("dup2 error");

exit(EXIT\_FAILURE);

}

close(Parent\_Child[0]);

close(Child\_Parent[1]);

Exec("/Users/vladislavburdinskij/Documents/os\_5-7\_cp/Os\_Labs\_New/lab-1/build/child");

}else{

close(Parent\_Child[0]);

close(Child\_Parent[1]);

std::string line;

std::getline(streamIn, line);

line += "\n";

write(Parent\_Child[1], line.c\_str(), line.size());

close(Parent\_Child[1]);

char buffer[256];

int bytesRead;

while ((bytesRead = read(Child\_Parent[0], buffer, sizeof(buffer))) > 0) {

streamOut.write(buffer, bytesRead);

}

close(Child\_Parent[0]); // Закрываем конец для чтения

// Ожидание завершения дочернего процесса

int status;

wait(&status);

}

}

utils.cpp

#include <utils.hpp>

void CreatePipe(int pipeFd[2]){

if (pipe(pipeFd) < 0){

std::perror("Pipe не создается");

exit(EXIT\_FAILURE);

}

}

pid\_t CreateChild(){

if (pid\_t pid = fork(); pid >= 0){

return pid;

}

std::perror("Дочерний процесс не создан");

exit(EXIT\_FAILURE);

}

void Exec(const char \* pathToChild){

if (execl(pathToChild, pathToChild, nullptr) == -1){

perror("Не исполняется exec");

exit(EXIT\_FAILURE);

}

}

Tests.cpp

// tests/tests.cpp

#include <gtest/gtest.h>

#include <string>

#include <array>

#include <cstdio>

#include <memory>

#include <stdexcept>

// Функция для выполнения команды и получения результата

std::string executeCommand(const std::string& command) {

std::array<char, 128> buffer;

std::string result;

// Открываем поток для чтения вывода команды

std::unique\_ptr<FILE, decltype(&pclose)> pipe(popen(command.c\_str(), "r"), pclose);

if (!pipe) {

throw std::runtime\_error("popen() failed!");

}

// Читаем вывод команды

while (fgets(buffer.data(), buffer.size(), pipe.get()) != nullptr) {

result += buffer.data();

}

return result;

}

// Макрос для упрощения написания тестов

#define RUN\_TEST(input, expected\_output) \

do { \

std::string cmd = "echo \"" input "\" | " "./lab-1"; \

std::string output; \

try { \

output = executeCommand(cmd); \

} catch (const std::exception& e) { \

FAIL() << "Failed to execute command: " << e.what(); \

} \

EXPECT\_NE(output.find(expected\_output), std::string::npos) << "Expected \"" << expected\_output << "\" not found in output."; \

} while(0)

// Тестовый случай 1: корректные входные данные

TEST(Lab1Tests, CorrectInput) {

RUN\_TEST("10 2 5\n", "Результат: 1");

}

// Тестовый случай 2: деление на ноль

TEST(Lab1Tests, DivisionByZero) {

RUN\_TEST("10 0 5\n", "Делить на ноль нельзя!");

}

// Тестовый случай 3: отрицательные числа

TEST(Lab1Tests, NegativeNumbers) {

RUN\_TEST("-10 -2 5\n", "Результат: 1");

}

// Тестовый случай 4: большие числа

TEST(Lab1Tests, LargeNumbers) {

RUN\_TEST("1000 10 10\n", "Результат: 10");

}

// Тестовый случай 5: оба делителя ноль

TEST(Lab1Tests, BothDivisorsZero) {

RUN\_TEST("10 0 0\n", "Делить на ноль нельзя!");

}

// Тестовый случай 6: некорректный ввод

TEST(Lab1Tests, InvalidInput) {

RUN\_TEST("10 a 5\n", "Некорректный ввод"); // Предполагается, что программа выведет "Ошибка" или аналогичное сообщение

}

main.cpp

#include "parent.hpp"

int main(void) {

ParentProcess(getenv("PATH\_TO\_CHILD"), std::cin, std::cout);

exit(EXIT\_SUCCESS);

}

**Пример работы**

**(base) vladislavburdinskij@MacBook-Pro-Vladislav build % ./lab-1**

**100 10 10**

**Результат: 1%**

**(base) vladislavburdinskij@MacBook-Pro-Vladislav build %**

**Вывод**

В ходе работы над лабораторной работой я познакомился с системными вызовами pipe и fork, создал программу которая делит всю работу между двумя процессами и передает данные между ними с помощью pipe.