Московский Авиационный Институт

(Национальный Исследовательский Университет)

Факультет информационных технологий и прикладной математики

Кафедра вычислительной математики и программирования

**Лабораторная работа №3 по курсу**

**«Операционные системы»**

Студент: Бурдинский Владислав Дмитриевич

Группа: М8О–203Б–23

Вариант: 12

Преподаватель: Миронов Евгений Сергеевич

Оценка: \_\_\_\_\_\_\_\_\_\_

Дата: \_\_\_\_\_\_\_\_\_\_

Подпись: \_\_\_\_\_\_\_\_\_\_

Москва, 2024.

**Постановка задачи**

Цель работы

Приобретение навыков по работе с разделяемой памятью.

Задание:

По аналогии с ЛР 1 создать программу которая осуществляет поставленную задачу и предусматривает разделение на несколько отдельных процессов, но вместо pipe использовать file mapping.

**Код программы**

**tests.cpp**

#include <gtest/gtest.h>

#include "parent.hpp"

#include <sstream>

#include <vector>

#include <numeric>

#include <filesystem>

namespace fs = std::filesystem;

void TestParent(const std::string& input, const std::string& expectedOutput, const std::string& pathToChild) {

std::stringstream inFile(input);

std::stringstream outFile;

if (fs::exists(pathToChild)) {

ParentProcess(pathToChild.c\_str(), inFile, outFile);

std::string result = outFile.str();

EXPECT\_EQ(result, expectedOutput);

} else {

std::cerr << "Путь к дочернему процессу не существует: " << pathToChild << std::endl;

FAIL() << "Путь к дочернему процессу не существует";

}

}

const std::string PATH\_TO\_CHILD = getenv("WAY\_TO\_FILE");

TEST(ParentTest, CorrectCalculation) {

std::string input = "100 2 5\nexit\n";

std::string expected\_output = "Результат: 10\n";

TestParent(input, expected\_output, PATH\_TO\_CHILD);

}

TEST(ParentTest, DivisionByZero) {

std::string input = "10 0 5\nexit\n";

std::string expected\_output = "Деление на ноль\n";

TestParent(input, expected\_output, PATH\_TO\_CHILD);

}

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

::testing::InitGoogleTest(&argc, argv);

return RUN\_ALL\_TESTS();

}

**child.hpp**

#ifndef CHILD\_HPP

#define CHILD\_HPP

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

#endif // CHILD\_HPP

parent.hpp

#pragma once

#include <iostream>

#include "utils.hpp"

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

utils.hpp

#pragma once

#include <cstddef>

#include <sys/mman.h>

#include <fcntl.h>

#include <unistd.h>

#include <semaphore.h>

void\* CreateFileMapping(const char\* name, size\_t size);

void CloseFileMapping(const char\* name, void\* addr, size\_t size);

pid\_t CreateChild();

void Exec(const char \* pathToChild);

struct SharedData {

sem\_t sem\_parent;

sem\_t sem\_child;

char fileName[256];

float number;

float result;

bool ready;

};

**child.cpp**

#include "child.hpp"

#include "utils.hpp"

#include <iostream>

#include <string>

#include <sstream>

#include <stdexcept>

#include <semaphore.h>

#include <fcntl.h>

#include <cstring>

int calculation(int num1, int num2, int num3) {

if (num2 == 0 || num3 == 0) throw std::runtime\_error("Деление на ноль");

return num1 / num2 / num3;

}

int main() {

constexpr auto shm\_name = "/shared\_memory";

constexpr size\_t shm\_size = 1024;

char\* shared\_data = static\_cast<char\*>(CreateFileMapping(shm\_name, shm\_size));

if (shared\_data == MAP\_FAILED) {

perror("Child: Ошибка подключения к общей памяти");

exit(EXIT\_FAILURE);

}

sem\_t\* sem\_child = sem\_open("/sem\_child", 0);

sem\_t\* sem\_parent = sem\_open("/sem\_parent", 0);

if (sem\_child == SEM\_FAILED || sem\_parent == SEM\_FAILED) {

perror("Child: Ошибка открытия семафоров");

CloseFileMapping(shm\_name, shared\_data, shm\_size);

exit(EXIT\_FAILURE);

}

while (true) {

sem\_wait(sem\_child);

std::string input(shared\_data);

if (input == "exit") {

sem\_post(sem\_parent);

break;

}

std::stringstream ss(input);

int num1, num2, num3;

std::string result;

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

try {

int res = calculation(num1, num2, num3);

result = "Результат: " + std::to\_string(res);

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

result = e.what();

}

} else {

result = "Некорректный ввод";

}

strncpy(shared\_data, result.c\_str(), shm\_size);

sem\_post(sem\_parent);

}

CloseFileMapping(shm\_name, shared\_data, shm\_size);

sem\_close(sem\_child);

sem\_close(sem\_parent);

return 0;

}

**parent.cpp**

#include "parent.hpp"

#include "utils.hpp"

#include <iostream>

#include <string>

#include <unistd.h>

#include <sys/wait.h>

#include <cstring>

#include <semaphore.h>

#include <fcntl.h>

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

constexpr auto shm\_name = "/shared\_memory";

constexpr size\_t shm\_size = 1024;

char\* shared\_data = static\_cast<char\*>(CreateFileMapping(shm\_name, shm\_size));

if (shared\_data == MAP\_FAILED) {

perror("Parent: Ошибка создания общей памяти");

exit(EXIT\_FAILURE);

}

sem\_t\* sem\_child = sem\_open("/sem\_child", O\_CREAT, 0666, 0);

sem\_t\* sem\_parent = sem\_open("/sem\_parent", O\_CREAT, 0666, 0);

if (sem\_child == SEM\_FAILED || sem\_parent == SEM\_FAILED) {

perror("Parent: Ошибка создания семафоров");

CloseFileMapping(shm\_name, shared\_data, shm\_size);

exit(EXIT\_FAILURE);

}

pid\_t pid = fork();

if (pid == -1) {

perror("Parent: Ошибка fork");

CloseFileMapping(shm\_name, shared\_data, shm\_size);

sem\_close(sem\_child);

sem\_close(sem\_parent);

sem\_unlink("/sem\_child");

sem\_unlink("/sem\_parent");

exit(EXIT\_FAILURE);

} else if (pid == 0) {

execl(pathToChild, pathToChild, nullptr);

perror("Child: Ошибка exec");

exit(EXIT\_FAILURE);

} else {

std::string line;

while (true) {

std::cout << "Введите строку с тремя числами (или 'exit' для выхода):\n";

std::getline(streamIn, line);

strncpy(shared\_data, line.c\_str(), shm\_size);

sem\_post(sem\_child);

if (line == "exit") break;

sem\_wait(sem\_parent);

streamOut << shared\_data << std::endl;

}

waitpid(pid, nullptr, 0);

CloseFileMapping(shm\_name, shared\_data, shm\_size);

sem\_close(sem\_child);

sem\_close(sem\_parent);

sem\_unlink("/sem\_child");

sem\_unlink("/sem\_parent");

}

}

utils.cpp

#include <utils.hpp>

#include <sstream>

#include <sys/wait.h>

#include <iostream>

#include <string>

#include <cstdlib>

#include <unistd.h>

#include <sys/mman.h>

#include <sys/stat.h>

#include <fcntl.h>

void\* CreateFileMapping(const char\* name, size\_t size){

int fd = shm\_open(name, O\_CREAT | O\_RDWR, 0666);

ftruncate(fd, size);

void\* addr = mmap(0, size, PROT\_READ | PROT\_WRITE, MAP\_SHARED, fd, 0);

close(fd);

return addr;

}

void CloseFileMapping(const char\* name, void\* addr, size\_t size){

munmap(addr, size);

shm\_unlink(name);

}

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);

}

}

**main.cpp**

#include "parent.hpp"

#include <iostream>

#include <cstdlib>

int main(void) {

const char\* pathToChild = getenv("WAY\_TO\_FILE");

if (pathToChild == nullptr) {

std::cerr << "Переменная WAY\_TO\_FILE не существует" << std::endl;

exit(EXIT\_FAILURE);

}

ParentProcess(pathToChild, std::cin, std::cout);

exit(EXIT\_SUCCESS);

}

**CMakeLists.txt**

cmake\_minimum\_required(VERSION 3.28)

project(lab1)

set(CMAKE\_CXX\_STANDARD 17)

# Подключение GoogleTest

include(FetchContent)

FetchContent\_Declare(

googletest

GIT\_REPOSITORY https://github.com/google/googletest.git

GIT\_TAG v1.15.2

TLS\_VERIFY false

)

set(gtest\_force\_shared\_crt ON CACHE BOOL "" FORCE)

FetchContent\_MakeAvailable(googletest)

# Исполняемый файл для родительского процесса

add\_executable(lab1 main.cpp src/parent.cpp src/utils.cpp)

target\_include\_directories(lab1 PRIVATE include)

# Исполняемый файл для дочернего процесса

add\_executable(child src/child.cpp src/utils.cpp)

target\_include\_directories(child PRIVATE include)

# Настройка тестов

enable\_testing()

add\_executable(tests tests/tests.cpp src/parent.cpp src/utils.cpp)

target\_include\_directories(tests PRIVATE include)

target\_link\_libraries(tests PRIVATE GTest::gtest\_main pthread)

# Настройка GoogleTest для автоматического обнаружения тестов

include(GoogleTest)

gtest\_discover\_tests(tests)

**Пример работы**

**(base) vladislavburdinskij@MacBook-Pro-Vladislav build % ./lab1**

**Введите строку с тремя числами (или 'exit' для выхода):**

**100 1 1**

**Результат: 100**

**Введите строку с тремя числами (или 'exit' для выхода):**

**20 2 2**

**Результат: 5**

**Введите строку с тремя числами (или 'exit' для выхода):**

**200 50 4**

**Результат: 1**

**Введите строку с тремя числами (или 'exit' для выхода):**

**^C**

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

**Вывод**

В ходе выполнения лабораторной работе я научился работать с разделяемой памятью и работать с дочерними процессами, а также закрепил и улучшил свои знания c++.