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++.