Московский авиационный институт

(национальный исследовательский университет)

# Факультет информационных технологий и прикладной математики

**Кафедра вычислительной математики и программирования**

## Лабораторная работа № 3 по курсу «Операционные системы»

**Динамические библиотеки**

Студент: Кунавин К. В. Преподаватель: Миронов Е. С.

Группа: М8О-203Б-23

Дата: Оценка: Подпись:

**Москва, 2024**

# Условие

Составить и отладить программу на языке С/C++, 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 через отображаемые файлы (memory-mapped files).

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

# Задание

Родительский процесс создает дочерний процесс. Первой строчкой пользователь в консоль родительского процесса пишет имя файла, которое будет передано при создании дочернего процесса. Родительский и дочерний процесс должны быть представлены разными программами. Родительский процесс передает команды пользователя через pipe1, который связан со стандартным входным потоком дочернего процесса. Дочерний процесс при необходимости передает данные в родительский процесс через pipe2. Результаты своей работы дочерний процесс пишет в созданный им файл. Допускается просто открыть файл и писать туда, не перенаправляя стандартный поток вывода.

Пользователь вводит команды вида: «число число число». Далее эти числа передаются от родительского процесса в дочерний. Дочерний процесс считает их сумму и выводит её в файл. Числа имеют тип float. Количество чисел может быть произвольным.

# Метод решения

Были созданы родительский и дочерний процессы, которые совместно используют участки памяти, отобразив их из файла или разделяемой памяти (через mmap, shm\_open и т.д.). Родитель записывал данные (числа, строку и т. п.) в эту отображённую область, а дочерний процесс читал их, выполнял требуемую операцию (например, суммирование), и записывал результат обратно. Для синхронизации применялись семафоры или мьютексы, предотвращая гонки при доступе к общей памяти.

# Код программы

lab3.h

#ifndef PARENT\_H

#define PARENT\_H

#include <iostream>

#include <iomanip>   // setprecision()

#include <fstream>   // ofstream

#include <unistd.h>  // fork(), sleep()

#include <sys/wait.h> // wait()

#include <sys/mman.h> // mmap(), munmap()

#include <sys/stat.h> // fstat()

#include <fcntl.h>    // O\_CREAT, O\_RDWR

#include <cstring>    // memcpy(), strlen()

#include <semaphore.h> // sem\_t, sem\_init(), sem\_wait(), sem\_post()

#include <cstdlib>    // exit()

constexpr auto SHARED\_FILE = "/shared\_memory\_file";

struct SharedData {

    sem\_t sem\_parent; // Семафор для родителя

    sem\_t sem\_child;  // Семафор для ребенка

    char fileName[256];

    float number;

    float sum;

    bool finished;

};

void RunParentProcess(std::istream&);

#endif

child.cpp

#include "lab3.h"

int main() {

    int fd = shm\_open(SHARED\_FILE, O\_RDWR, 0666);

    if (fd == -1) {

        std::cerr << "Ошибка открытия общего файла в дочернем процессе" << std::endl;

        exit(1);

    }

    // Отображаем файл в память

    SharedData\* shared = (SharedData\*)mmap(nullptr, sizeof(SharedData),

                                           PROT\_READ | PROT\_WRITE, MAP\_SHARED, fd, 0);

    if (shared == MAP\_FAILED) {

        std::cerr << "Ошибка mmap в дочернем процессе" << std::endl;

        close(fd);

        shm\_unlink(SHARED\_FILE);

        exit(1);

    }

    close(fd);

    // Ждем сигнал от родительского процесса

    sem\_wait(&shared->sem\_child);

    // Читаем данные

    char fileName[256];

    strcpy(fileName, shared->fileName);

    float sum = 0;

    while (true) {

        // Ждем число

        sem\_wait(&shared->sem\_child);

        if (shared->finished) {

            break;

        }

        sum += shared->number;

        // Можно продолжать

        sem\_post(&shared->sem\_parent);

    }

    // Записываем сумму в файл

    {

        std::ofstream file(fileName);

        if (!file.is\_open()) {

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

            exit(1);

        }

        file << std::fixed << std::setprecision(6);

        file << "Сумма: " << sum << '\n';

    }

    // Записываем результат в общую память

    shared->sum = sum;

    // Сигнализируем родительскому процессу о завершении

    sem\_post(&shared->sem\_parent);

    // Удаляем отображение памяти

    munmap(shared, sizeof(SharedData));

    return 0;

}

parent.cpp

#include "lab3.h"

#include <limits>

void RunParentProcess(std::istream& stream) {

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

    if (fd == -1) {

        std::cerr << "Ошибка открытия общего файла" << std::endl;

        exit(1);

    }

    // Устанавливаем размер файла

    if (ftruncate(fd, sizeof(SharedData)) == -1) {

        std::cerr << "Ошибка ftruncate" << std::endl;

        shm\_unlink(SHARED\_FILE);

        exit(1);

    }

    // Отображаем файл в память

    SharedData\* shared = (SharedData\*)mmap(nullptr, sizeof(SharedData),

                                           PROT\_READ | PROT\_WRITE, MAP\_SHARED, fd, 0);

    if (shared == MAP\_FAILED) {

        std::cerr << "Ошибка mmap" << std::endl;

        close(fd);

        shm\_unlink(SHARED\_FILE);

        exit(1);

    }

    close(fd);

    sem\_init(&shared->sem\_parent, 1, 0);

    sem\_init(&shared->sem\_child, 1, 0);

    pid\_t pid = fork();

    if (pid < 0) {

        std::cerr << "Ошибка fork" << std::endl;

        exit(1);

    }

    if (pid > 0) {

        // Родительский процесс

        // Ввод данных

        std::cout << "Введите имя файла:\n";

        stream.getline(shared->fileName, sizeof(shared->fileName));

        sem\_post(&shared->sem\_child);

        std::cout << "Введите числа (EOF для завершения):\n";

        float num;

        while (stream >> num) {

            shared->finished = false;

            shared->number = num;

            sem\_post(&shared->sem\_child);

            sem\_wait(&shared->sem\_parent);

        }

        shared->finished = true;

        sem\_post(&shared->sem\_child);

        // Очищаем поток ввода

        stream.clear();

        stream.ignore(std::numeric\_limits<std::streamsize>::max(), '\n');

        sem\_wait(&shared->sem\_parent);

        // Читаем результат

        std::cout << "\nРезультат от дочернего процесса: Сумма = " << shared->sum << std::endl;

        // Уничтожаем семафоры

        sem\_destroy(&shared->sem\_parent);

        sem\_destroy(&shared->sem\_child);

        // Удаляем отображение памяти

        munmap(shared, sizeof(SharedData));

        // Удаляем файл

        shm\_unlink(SHARED\_FILE);

        // Ждем завершения дочернего процесса

        wait(nullptr);

    } else {

        // Дочерний процесс

        const char\* pathToChild = std::getenv("PATH\_TO\_CHILD");

        if (!pathToChild) {

            std::cerr << "Переменная окружения PATH\_TO\_CHILD не установлена" << std::endl;

            exit(1);

        }

        execlp(pathToChild, pathToChild, nullptr);

        std::cerr << "Ошибка exec, проверьте корректность PATH\_TO\_CHILD" << std::endl;

        exit(1);

    }

}

main.cpp

#include "lab3.h"

int main() {

    RunParentProcess(std::cin);

    return 0;

}

CMakeLists.txt

add\_executable(lab3 main.cpp src/parent.cpp include/lab3.h)

target\_include\_directories(lab3 PRIVATE include)

add\_executable(lab3\_child src/child.cpp include/lab3.h)

target\_include\_directories(lab3\_child PRIVATE include)

lab3\_test.cpp

#include <gtest/gtest.h>

#include <lab3.h>

class ParentProcessTest : public ::testing::Test {

protected:

    std::string testName;

    std::string outputFileName;

    std::string inputFileName;

    void SetUp() override {

        testName = ::testing::UnitTest::GetInstance()->current\_test\_info()->name();

        // Генерируем имена файлов

        outputFileName = testName + "\_output.txt";

        inputFileName = testName + "\_input.txt";

    }

    void TearDown() override {

        // Удаляем временные файлы после каждого теста

        std::remove(outputFileName.c\_str());

        std::remove(inputFileName.c\_str());

    }

    // Метод для создания входного файла

    void CreateInputFile(const std::string& inputData) {

        std::ofstream testInput(inputFileName);

        testInput << outputFileName << "\n" << inputData;

    }

    // Метод для проверки выходного файла

    void CheckOutputFile(const std::string& expectedLine) {

        std::ifstream resultFile(outputFileName);

        ASSERT\_TRUE(resultFile.is\_open()) << "Файл не был создан";

        std::string line;

        std::getline(resultFile, line);

        EXPECT\_EQ(line, expectedLine) << "Неверный результат в выходном файле";

    }

};

TEST\_F(ParentProcessTest, CheckSumCalculation) {

    // Создаём входной файл

    CreateInputFile("7.6 5.5");

    std::ifstream testFile(inputFileName);

    ASSERT\_TRUE(testFile.is\_open()) << "Не удалось открыть файл ввода";

    // Запуск родительского процесса

    RunParentProcess(testFile);  // Передаем поток для тестирования

    // Проверяем выходной файл

    CheckOutputFile("Сумма: 13.100000");

}

TEST\_F(ParentProcessTest, EmptyInput) {

    // Создаём входной файл (только имя выходного файла)

    CreateInputFile("");

    std::ifstream testFile(inputFileName);

    ASSERT\_TRUE(testFile.is\_open()) << "Не удалось открыть файл ввода";

    // Запуск родительского процесса

    RunParentProcess(testFile);  // Передаем поток для тестирования

    // Проверяем выходной файл

    CheckOutputFile("Сумма: 0.000000");

}

TEST\_F(ParentProcessTest, LargeNumberOfInputs) {

    // Генерируем данные для входного файла

    std::ostringstream inputData;

    for (int i = 1; i <= 100; ++i) {

        inputData << i << " ";

    }

    // Создаём входной файл

    CreateInputFile(inputData.str());

    std::ifstream testFile(inputFileName);

    ASSERT\_TRUE(testFile.is\_open()) << "Не удалось открыть файл ввода";

    // Запуск родительского процесса

    RunParentProcess(testFile);  // Передаем поток для тестирования

    // Проверяем выходной файл

    CheckOutputFile("Сумма: 5050.000000");

}

TEST\_F(ParentProcessTest, InvalidInput) {

    // Создаём входной файл с некорректными данными

    CreateInputFile("10 abc");

    std::ifstream testFile(inputFileName);

    ASSERT\_TRUE(testFile.is\_open()) << "Не удалось открыть файл ввода";

    // Запуск родительского процесса

    RunParentProcess(testFile);  // Передаем поток для тестирования

    // Проверяем выходной файл

    CheckOutputFile("Сумма: 10.000000");

}

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

    testing::InitGoogleTest(&argc, argv);

    return RUN\_ALL\_TESTS();

}

# Выводы

Подход с memory-mapped files позволил эффективный и удобный обмен данными без дополнительных IPC-примитивов (каналов, очередей сообщений). При этом важно грамотно управлять защитой памяти и обеспечивать синхронизацию. Работа продемонстрировала, что mmap даёт высокопроизводительный обмен между процессами, если чётко организовать протокол чтения/записи.