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: Дудовцев Андрей Андреевич
Группа: М8О-208Б-22
Преподаватель: Миронов Евгений Сергеевич
Оценка: _____
Дата: _____
Подпись: _____

Москва, 2024

**Содержание**

## Репозиторий

AndreyDdvts/OS_LABS (github.com)

## Цель работы

Приобретение практических навыков в:

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

## 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) должна создать для решения 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.

## Описание работы программы

Задание аналогично первой лабораторной работе. В ходе выполнения лабораторной работы я использовал следующие системные вызовы:

- fork() - создание нового процесса
- sem_open() - создание/открытие семафора

- sem_post() - увеличивание значения семафора и разблокировка ожидающих потоков
- sem_wait() - уменьшение значения семафора. Если 0, то вызывающий поток блокируется
- sem_close() - закрытие семафора
- shm_open() - создание/открытие разделяемой памяти POSIX
- shm_unlink() - закрытие разделяемой памяти
- ftruncate() - уменьшение длины файла до указанной
- mmap() - отражение файла или устройства в памяти
- munmap() - снятие отражения
- execlp() - запуск файла на исполнение

**Исходный код**

```
========================== parent.hpp ==========================

#pragma once

#include "utils.hpp"

void ParentProcess(const char *pathToChild);

========================== util.hpp ==========================

#pragma once

#include <iostream>
#include <sstream>
#include <string>
#include <vector>
#include <cstring>
#include <stdlib.h>
#include <unistd.h>
```

```
#include <sys/types.h>
#include <sys/wait.h>
#include <sys/mman.h>
#include <ext/stdio_filebuf.h>

#include <fcntl.h>
#include <sys/stat.h>
#include <semaphore.h>

const int MAP_SIZE = 1024;

constexpr const char *SEMAPHORE_NAME_1 = "/semaphore_1";
constexpr const char *SHARED_MEMORY_NAME_1 = "/shared_memory_1";

constexpr const char *SEMAPHORE_NAME_2 = "/semaphore_2";
constexpr const char *SHARED_MEMORY_NAME_2 = "/shared_memory_2";

sem_t* OpenSemaphore(const char *name, int value);
int OpenSharedMemory(const char *name, const int size);
char* MapSharedMemory(const int size, int fd);
pid_t CreateChildProcess();
bool CheckString(const std::string_view str);

========================== child.cpp ==========================

#include "utils.hpp"

int main(int argc, char *argv[]) {
    if (argc != 2) {
        perror("Not enough arguments");
        exit(EXIT_FAILURE);
    }

    const char *fileName = argv[1];
    std::ofstream fout(fileName, std::ios::app);
```

```
    if (!fout.is_open()) {
        perror("Couldn't open the file");
        exit(EXIT_FAILURE);
    }

    sem_t *semptr1 = OpenSemaphore(SEMAPHORE_NAME_1, 0);
    int shared_memory_fd1 = OpenSharedMemory(SHARED_MEMORY_NAME_1,
MAP_SIZE);
    char *memptr1 = MapSharedMemory(MAP_SIZE, shared_memory_fd1);

    sem_t *semptr2 = OpenSemaphore(SEMAPHORE_NAME_2, 0);
    int shared_memory_fd2 = OpenSharedMemory(SHARED_MEMORY_NAME_2,
MAP_SIZE);
    char* memptr2 = MapSharedMemory(MAP_SIZE, shared_memory_fd2);

    while (1) {
        sem_wait(semptr1);
        std::string_view str(memptr1);
        if (str.empty()) {
            sem_post(semptr2);
            break;
        }
        if (CheckString(str)) {
            fout << str << std::endl;
        } else {
            strcpy(memptr2, "ERROR_STRING");
        }
        sem_post(semptr2);
    }

    sem_close(semptr1);
    sem_unlink(SEMAPHORE_NAME_1);
    shm_unlink(SHARED_MEMORY_NAME_1);
    munmap(memptr1, MAP_SIZE);
    close(shared_memory_fd1);
```

```
    sem_close(semptr2);
    sem_unlink(SEMAPHORE_NAME_2);
    shm_unlink(SHARED_MEMORY_NAME_2);
    munmap(memptr2, MAP_SIZE);
    close(shared_memory_fd2);

    exit(EXIT_SUCCESS);
}

========================== parent.cpp ==========================

#include "parent.hpp"
#include "utils.hpp"

void ParentProcess(const char *pathToChild) {

    std::string fileName;
    getline(std::cin, fileName);

    sem_t *semptr1 = OpenSemaphore(SEMAPHORE_NAME_1, 0);
    int shared_memory_fd1 = OpenSharedMemory(SHARED_MEMORY_NAME_1,
MAP_SIZE);
    char *memptr1 = MapSharedMemory(MAP_SIZE, shared_memory_fd1);

    sem_t *semptr2 = OpenSemaphore(SEMAPHORE_NAME_2, 0);
    int shared_memory_fd2 = OpenSharedMemory(SHARED_MEMORY_NAME_2,
MAP_SIZE);
    char* memptr2 = MapSharedMemory(MAP_SIZE, shared_memory_fd2);

    std::string str;
    std::vector<std::string> errorStrings;

    int pid = CreateChildProcess();
    if (pid != 0) { // Parent process
```

```
        while(getline(std::cin, str)) {
            strcpy(memptr1, str.c_str());
            sem_post(semptr1);

            sem_wait(semptr2);
            if (strcmp(memptr2, "ERROR_STRING") == 0) {
                errorStrings.push_back(str);
                strcpy(memptr2, "");
            }
        }
        strcpy(memptr1, "");
        sem_post(semptr1);
    } else { // Child process
        if (execlp(pathToChild, pathToChild, fileName.c_str(),
nullptr) == -1) { // to child.cpp
            perror("Error with execlp");
            exit(EXIT_FAILURE);
        }
    }

    for (const std::string &err : errorStrings) {
        std::cout << "ERROR with string: " << err << std::endl;
    }

    sem_close(semptr1);
    sem_unlink(SEMAPHORE_NAME_1);
    shm_unlink(SHARED_MEMORY_NAME_1);
    munmap(memptr1, MAP_SIZE);
    close(shared_memory_fd1);

    sem_close(semptr2);
    sem_unlink(SEMAPHORE_NAME_2);
    shm_unlink(SHARED_MEMORY_NAME_2);
    munmap(memptr2, MAP_SIZE);
    close(shared_memory_fd2);
```

```
}

========================== utils.cpp ==========================

#include "utils.hpp"

sem_t* OpenSemaphore(const char *name, int value) {
    sem_t *semptr = sem_open(name, O_CREAT, S_IRUSR | S_IWUSR, value);
    if (semptr == SEM_FAILED){
        perror("Couldn't open the semaphore");
        exit(EXIT_FAILURE);
    }
    return semptr;
}

int OpenSharedMemory(const char *name, const int size) {
    int sh_fd = shm_open(name, O_CREAT | O_RDWR, S_IRUSR | S_IWUSR);
    if (sh_fd == -1) {
        perror("Couldn't create memory shared object");
        exit(EXIT_FAILURE);
    }
    if (ftruncate(sh_fd, size) == -1) {
        perror("Couldn't truncate a file");
        exit(EXIT_FAILURE);
    }
    return sh_fd;
}

char* MapSharedMemory(const int size, int fd) {
    char *memptr = (char*)mmap(nullptr, size, PROT_READ | PROT_WRITE,
MAP_SHARED, fd, 0);
    if (memptr == MAP_FAILED) {
        perror("Error with file mapping");
        exit(EXIT_FAILURE);
    }
```

```
    return memptr;
}

pid_t CreateChildProcess() {
    pid_t pid = fork();
    if (pid == -1) {
        perror("Couldn't create child process");
        exit(EXIT_FAILURE);
    }
    return pid;
}

bool CheckString(const std::string_view str) {
    if (str[str.size() - 1] == '.' || str[str.size() - 1] == ';') {
        return true;
    }
    return false;
}

========================== main.cpp ==========================

#include "parent.hpp"

int main() {
    ParentProcess(getenv("PATH_TO_CHILD"));
    // bash: export
PATH_TO_CHILD="/home/qwz/OS_LABS/build/lab3/child3"
    // bash: printenv PATH_TO_CHILD
    exit(EXIT_SUCCESS);
}
```

## Тесты

```
#include <gtest/gtest.h>

#include <filesystem>
#include <memory>
#include <vector>

#include "parent.hpp"

namespace fs = std::filesystem;

void testingProgram(const std::vector<std::string> &input, const
std::vector<std::string> &expectedOutput, const
std::vector<std::string> &expectedFile) {
    const char *fileName = "file.txt";

    std::stringstream inFile;
    inFile << fileName << std::endl;
    for (std::string line : input) {
        inFile << line << std::endl;
    }

    std::streambuf* oldInBuf = std::cin.rdbuf(inFile.rdbuf()); //
выдает старый буфер

    ASSERT_TRUE(fs::exists(getenv("PATH_TO_CHILD")));

    testing::internal::CaptureStdout();
```

```
    ParentProcess(getenv("PATH_TO_CHILD"));
    std::cin.rdbuf(oldInBuf); // чтобы cin вернулся обратно

    std::stringstream
errorOut(testing::internal::GetCapturedStdout());
    for(const std::string &expectation : expectedOutput) {
        std::string result;
        getline(errorOut, result);
        EXPECT_EQ(result, expectation);
    }

    std::ifstream fin(fileName);
    if (!fin.is_open()) {
        perror("Couldn't open the file");
        exit(EXIT_FAILURE);
    }
    for (const std::string &expectation : expectedFile) {
        std::string result;
        getline(fin, result);
        EXPECT_EQ(result, expectation);
    }
    fin.close();
    std::remove(fileName);
}

TEST(thirdLabTests, emptyTest) {
    std::vector<std::string> input = {};

    std::vector<std::string> expectedOutput = {};

    std::vector<std::string> expectedFile = {};

    testingProgram(input, expectedOutput, expectedFile);
}
```

```
TEST(thirdLabTests, simpleTest) {
    std::vector<std::string> input = {
        "No,",
        "you'll never be alone.",
        "When darkness comes;",
        "I'll light the night with stars",
        "Hear my whispers in the dark!"
    };

    std::vector<std::string> expectedOutput = {
        "ERROR with string: No,",
        "ERROR with string: I'll light the night with stars",
        "ERROR with string: Hear my whispers in the dark!"
    };

    std::vector<std::string> expectedFile = {
        "you'll never be alone.",
        "When darkness comes;"
    };

    testingProgram(input, expectedOutput, expectedFile);
}

TEST(thirdLabTests, aQuedaTest) {
    std::vector<std::string> input = {
        "A QUEDA:",
        "E venha ver os deslizes que eu vou cometer;",
        "E venha ver os amigos que eu vou perder;",
        "Não tô cobrando entrada, vem ver o show na faixa.",
        "Hoje tem open bar pra ver minha desgraça."
    };

    std::vector<std::string> expectedOutput = {
```

```
        "ERROR with string: A QUEDA:"
    };

    std::vector<std::string> expectedFile = {
        "E venha ver os deslizes que eu vou cometer;",
        "E venha ver os amigos que eu vou perder;",
        "Não tô cobrando entrada, vem ver o show na faixa.",
        "Hoje tem open bar pra ver minha desgraça."
    };

    testingProgram(input, expectedOutput, expectedFile);
}

TEST(thirdLabTests, anotherTest) {
    std::vector<std::string> input = {
        "But I set fire to the rain.",
        "Watched it pour as- I touched your- face-",
        "Well, it burned while I cried!!!!!!!!",
        "Cause I heard it screamin' out your name;",
        "Your name."
    };

    std::vector<std::string> expectedOutput = {
        "ERROR with string: Watched it pour as- I touched your- face-
",
        "ERROR with string: Well, it burned while I cried!!!!!!!!"
    };

    std::vector<std::string> expectedFile = {
        "But I set fire to the rain.",
        "Cause I heard it screamin' out your name;",
        "Your name."
    };
```

```
    testingProgram(input, expectedOutput, expectedFile);
}

int main(int argc, char *argv[]) {
    std::cout << getenv("PATH_TO_CHILD") << std::endl;

    testing::InitGoogleTest(&argc, argv);
    return RUN_ALL_TESTS();
}
```

## Консоль

```
qwz@qwz-VirtualBox:~/OS_LABS/build/lab1$ ./child file.txt
No,
you’ll never be alone.
When darkness comes;
I’ll light the night with stars
Hear my whispers in the dark!
ERROR with string: I’ll light the night with stars
ERROR with string: No,
ERROR with string: Hear my whispers in the dark!
qwz@qwz-VirtualBox:~/OS_LABS/build/lab1$ ./child file.txt
A QUEDA:
E venha ver os deslizes que eu vou cometer;
E venha ver os amigos que eu vou perder;
N˜ao tˆo cobrando entrada, vem ver o show na faixa.
Hoje tem open bar pra ver minha desgraca.
ERROR with string: A QUEDA:
qwz@qwz-VirtualBox:~/OS_LABS/build/lab1$ ./child file.txt
But I set fire to the rain.
Watched it pour as- I touched your- face
Well, it burned while I cried!!!!!!!!
Cause I heard it screamin’ out your name;
Your name.
ERROR with string: Watched it pour as- I touched your- face
```

```
ERROR with string: Well, it burned while I cried!!!!!!!!
```

## Запуск тестов

```
/home/qwz/OS_LABS/build/lab3/child3
[==========] Running 4 tests from 1 test suite.
[----------] Global test environment set-up.
[----------] 4 tests from thirdLabTests
[ RUN      ] thirdLabTests.emptyTest
[       OK ] thirdLabTests.emptyTest (0 ms)
[ RUN      ] thirdLabTests.simpleTest
[       OK ] thirdLabTests.simpleTest (2 ms)
[ RUN      ] thirdLabTests.aQuedaTest
[       OK ] thirdLabTests.aQuedaTest (3 ms)
[ RUN      ] thirdLabTests.anotherTest
[       OK ] thirdLabTests.anotherTest (2 ms)
[----------] 4 tests from thirdLabTests (9 ms total)

[----------] Global test environment tear-down
[==========] 4 tests from 1 test suite ran. (9 ms total)
[  PASSED  ] 4 tests.
```

## Выводы

В результате выполнения данной лабораторной работы была написана программа на языке C++, осуществляющая работу с процессами и взаимодействие между ними через системные сигналы и отображаемые файлы. Я приобрел практические навыки в освоении принципов работы с файловыми системами и обеспечении обмена данных между процессами посредством технологии «File mapping».