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: Ширяев Никита Алексеевич
Группа: М8О-208Б-22
Вариант: 21
Преподаватель: Миронов Евгений Сергеевич
Оценка: _____
Дата: _____
Подпись: _____

Москва, 2023

Содержание

Репозиторий

https://github.com/resdt/os-labs

Постановка задачи

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 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.

Общие сведения о программе

Программа состоит из двух частей — главная программа, родительский процесс, описанный в lab3.cpp, и программа дочернего процесса, описанная в child.cpp.

Общий метод и алгоритм решения

Родительский процесс создает два дочерних процесса. Первой строкой пользователь в консоль родительского процесса вводит имя файла, которое будет использовано для открытия File с таким именем на запись для child1. Аналогично для второй строки и процесса child2. Родительский и дочерний процесс представлены разными программами. Родительский процесс создает два shared-memory файла, по которым он будет передавать дочерним процессам строки, согласно правилу фильтрации. Процесс child1 и child2 производят работу над строками. Процессы пишут результаты своей работы в стандартный вывод.

Вариант 21) Правило фильтрации: нечетные строки отправляются в pipe1, четные в pipe2. Дочерние процессы инвертируют строки.

Исходный код

lab3.hpp

```
#pragma once

#include "utils.hpp"

int ParentRoutine(const char *pathToChild);
```

utils.hpp

```
#pragma once

#include <algorithm>
#include <iostream>
#include <string>
#include <cstring>
#include <sstream>
#include <stdlib.h>
#include <unistd.h>
#include <sys/types.h>
#include <fstream>
#include <fcntl.h>
#include <sys/wait.h>
#include <semaphore.h>
#include <sys/mman.h>
#include <sys/stat.h>
#include <utility>
#include <thread>
#include <chrono>

const int MAP_SIZE = 1024;

constexpr const char *EVEN_SEMAPHORE_NAME =
"/even_semaphore";
constexpr const char *ODD_SEMAPHORE_NAME = "/odd_semaphore";
constexpr const char *EXIT_SEMAPHORE_NAME =
"/exit_semaphore";
```

```
constexpr const char *SHARED_MEMORY_NAME_1 =
"/shared_memory_1";
constexpr const char *SHARED_MEMORY_NAME_2 =
"/shared_memory_2";


pid_t createChildProcess();
sem_t* OpenSemaphore(const char *name, int value);
int OpenSharedMemory(const char *name, const int size);
char* MapSharedMemory(const int size, int fd);
```

child.cpp

```
#include "utils.hpp"

int main(int argc, char* argv[]) {
    (void)argc;

    sem_t *semptr = OpenSemaphore(argv[1], 0);
    int shared_memory_fd = OpenSharedMemory(argv[2],
MAP_SIZE);
    char *memptr = MapSharedMemory(MAP_SIZE,
shared_memory_fd);
    std::ofstream fout(argv[3],  std::ios::trunc);
    fout.close();

    while (true) {
        sem_wait(semptr);
        std::string_view st(memptr);
        std::string str = {st.begin(), st.end()};
        if (str == "\0") {
            break;
        }
        std::reverse(str.begin(), str.end());
        std::ofstream fout(argv[3],  std::ios::app);
        if (!fout.is_open()) {
```

```
            perror("Couldn't open the file");
            exit(EXIT_FAILURE);
        }
        fout << str << std::endl;
        fout.close();
    }
    sem_close(semptr);
    sem_unlink(argv[1]);
    shm_unlink(argv[2]);
    munmap(memptr, MAP_SIZE);
    close(shared_memory_fd);
    exit(EXIT_SUCCESS);
}
```

lab3.cpp

```
#include "lab3.hpp"

int ParentRoutine(const char *pathToChild) {
    std::string fileName1;
    std::string fileName2;

    getline(std::cin, fileName1);
    getline(std::cin, fileName2);

    sem_t* semptr1 = OpenSemaphore(EVEN_SEMAPHORE_NAME, 0);
    sem_t* semptr2 = OpenSemaphore(ODD_SEMAPHORE_NAME, 0);

    int shared_memory_fd1 =
OpenSharedMemory(SHARED_MEMORY_NAME_1, MAP_SIZE);
    char *memptr1 = MapSharedMemory(MAP_SIZE,
shared_memory_fd1);

    int shared_memory_fd2 =
OpenSharedMemory(SHARED_MEMORY_NAME_2, MAP_SIZE);
    char *memptr2 = MapSharedMemory(MAP_SIZE,
shared_memory_fd2);
```

```
    pid_t pid[2] {-1, -1};
    pid[0] = createChildProcess();
    pid[1] = createChildProcess();

    if (pid[0] == 0) {//child 1
        if (execl(pathToChild, "child", EVEN_SEMAPHORE_NAME,
SHARED_MEMORY_NAME_1, fileName1.c_str(), nullptr) == -1) {
            perror("Error with execl");
            exit(EXIT_FAILURE);
        }

    } else
    if (pid[1] == 0) {//child 2
        if (execl(pathToChild, "child", ODD_SEMAPHORE_NAME,
SHARED_MEMORY_NAME_2, fileName2.c_str(), nullptr) == -1) {
            perror("Error with execl");
            exit(EXIT_FAILURE);
        }

    } else {//parent
        std::string str;
        while (getline(std::cin, str)) {
            std::chrono::milliseconds delay(10);
            std::this_thread::sleep_for(delay);

            if (str.size() % 2 == 0) {
                strcpy(memptr1, str.c_str());
                sem_post(semptr1);
            } else {
                strcpy(memptr2, str.c_str());
                sem_post(semptr2);
            }
            str.clear();
        }
        std::chrono::milliseconds delay(10);
        std::this_thread::sleep_for(delay);
```

```
        strcpy(memptr1, "\0");
        sem_post(semptr1);
        strcpy(memptr2, "\0");
        sem_post(semptr2);

        int status;
        waitpid(pid[0], &status, 0);
        waitpid(pid[1], &status, 0);

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

        sem_close(semptr2);
        sem_unlink(ODD_SEMAPHORE_NAME);
        shm_unlink(SHARED_MEMORY_NAME_2);
        munmap(memptr2, MAP_SIZE);
        close(shared_memory_fd2);
    }
    return 0;
}
```

utils.cpp

```
#include "utils.hpp"

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

```
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;
}
```

main.cpp

```
#include "lab3.hpp"
```

```
int main() {
    const char path[] =
"/home/test/Desktop/os-labs/build/lab3/child";
    ParentRoutine(path);
    exit(EXIT_SUCCESS);
}
```

lab3_test.cpp

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

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

#include <lab3.hpp>

namespace fs = std::filesystem;

void testingProgram(const std::vector<std::string> &input,
const std::vector<std::string> &expectedOutput1, const
std::vector<std::string> &expectedOutput2) {
    const char *fileWithInput = "input.txt";
    const char *fileWithOutput1 = "output1.txt";
    const char *fileWithOutput2 = "output2.txt";

    std::stringstream inFile(fileWithInput);
    inFile << fileWithOutput1 << std::endl;
    inFile << fileWithOutput2 << std::endl;
    for (std::string line : input) {
        inFile << line << std::endl;
    }
    //inFile.close();

    //std::ifstream inFile1(fileWithInput);
/*
```

```
    if (!inFile1.is_open()) {
        perror("Couldn't open the file");
        exit(EXIT_FAILURE);
    }
*/
    std::streambuf* oldInBuf =
std::cin.rdbuf(inFile.rdbuf());


ASSERT_TRUE(fs::exists("/home/test/Desktop/os-labs/build/lab3
/child"));


ParentRoutine("/home/test/Desktop/os-labs/build/lab3/child");
    std::cin.rdbuf(oldInBuf);

    auto outFile1 = std::ifstream(fileWithOutput1);
    auto outFile2 = std::ifstream(fileWithOutput2);
    if (!outFile1.is_open()) {
        perror("Couldn't open the file");
        exit(EXIT_FAILURE);
    }
    for (const std::string &line : expectedOutput1) {
        std::string result;
        getline(outFile1, result);
        EXPECT_EQ(result, line);
    }
    outFile1.close();
    std::remove(fileWithOutput1);

    if (!outFile2.is_open()) {
        perror("Couldn't open the file");
        exit(EXIT_FAILURE);
    }
    for (const std::string &line : expectedOutput2) {
        std::string result;
        getline(outFile2, result);
        EXPECT_EQ(result, line);
```

```
    }
    outFile2.close();
    std::remove(fileWithOutput2);
}

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

    std::vector<std::string> expectedOutput1 = {};

    std::vector<std::string> expectedOutput2 = {};

    testingProgram(input, expectedOutput1, expectedOutput2);
}

TEST(thirdLabTests, firstSimpleTest) {
    std::vector<std::string> input = {
        "01",
        "02",
        "001",
        "002",
        ""
    };

    std::vector<std::string> expectedOutput1 = {
        "10",
        "20"
    };

    std::vector<std::string> expectedOutput2 = {
        "100",
        "200"
    };

    testingProgram(input, expectedOutput1, expectedOutput2);
}
```

```
TEST(thirdLabTests, secondSimpleTest) {
    std::vector<std::string> input = {
            "This test has only",
            "one output file.",
            ""
    };

    std::vector<std::string> expectedOutput1 = {
            "ylno sah tset sihT",
            ".elif tuptuo eno"
    };

    std::vector<std::string> expectedOutput2 = {};


    testingProgram(input, expectedOutput1, expectedOutput2);
}

TEST(thirdLabTests, thirdSimpleTest) {
    std::vector<std::string> input = {
            "The length of this string is even,",
            "but the length of this string isn't even.",
            "There are 37 characters in this line,",
            "but there are already as many as 60 characters
in this line!",
            ""
    };

    std::vector<std::string> expectedOutput1 = {
            ",neve si gnirts siht fo htgnel ehT",
            "!enil siht ni sretcarahc 06 sa ynam sa ydaerla
era ereht tub"
    };

    std::vector<std::string> expectedOutput2 = {
            ".neve t'nsi gnirts siht fo htgnel eht tub",
            ",enil siht ni sretcarahc 73 era erehT"
```

```
    };


    testingProgram(input, expectedOutput1, expectedOutput2);
}

int main(int argc, char *argv[]) {
    testing::InitGoogleTest(&argc, argv);
    return RUN_ALL_TESTS();
}
```

Демонстрация работы программы

```
hacker@warmachine:~/prog/my_os_labs/build/lab3$ ./lab3
file1
file2
hi
hello
good morning
bye
^Z
[3]+ Stopped
./lab3
hacker@warmachine:~/prog/my_os_labs/build/lab3$ cat file1
ihgninrom doog
hacker@warmachine:~/prog/my_os_labs/build/lab3$ cat file2
olleh
eyb
hacker@warmachine:~/prog/my_os_labs/build/lab3$
../tests/lab3_test
[==========] Running 4 tests from 1 test suite.
[----------] Global test environment set-up.
[----------] 4 tests from thirdLabTests
[ RUN ] thirdLabTests.emptyTest
[ OK ] thirdLabTests.emptyTest (21 ms)
[ RUN ] thirdLabTests.secondSimpleTest
[ OK ] thirdLabTests.secondSimpleTest (41 ms)
```

```
[ RUN ] thirdLabTests.firstSimpleTest
[ OK ] thirdLabTests.firstSimpleTest (63 ms)
[ RUN ] thirdLabTests.thirdSimpleTest
[ OK ] thirdLabTests.thirdSimpleTest (62 ms)
[----------] 4 tests from thirdLabTests (190 ms total)
[----------] Global test environment tear-down
[==========] 4 tests from 1 test suite ran. (190 ms total)
[ PASSED ] 4 tests.
hacker@warmachine:~/prog/my_os_labs/build/lab3$
```

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».