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:* Немкова Анастасия Романовна

*Группа:* М8О-208Б-22

*Преподаватель:* Миронов Евгений Сергеевич

*Вариант: 15*

*Оценка:* _____

*Дата:* _____

*Подпись:* _____

Москва, 2023

# Содержание

# Репозиторий

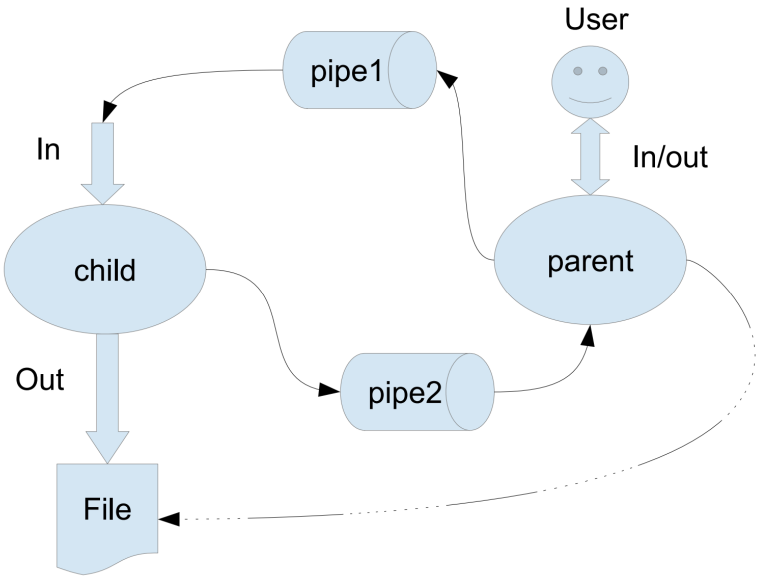
`https://github.com/anastasia-nemkova/OS_labs`

# Постановка задачи

**Цель работы:**

Освоение принципов работы с файловыми системами и обеспечение 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) 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).

Родительский процесс создает дочерний процесс. Первой строкой пользователь в консоль родительского процесса вводит имя файла, которое будет использовано для открытия File с таким именем на запись. Перенаправление стандартных потоков ввода-вывода показано на картинке выше. Родительский и дочерний процесс должны быть представлены разными программами. Родительский процесс принимает от пользователя строки произвольной длины и пересылает их дочернему процессу. Процесс child проверяет строки на валидность правилу. Если строка соответствует правилу, то она выводится в стандартный поток вывода дочернего процесса, иначе в родительский процесс отправляется информация об ошибке. Родительский процесс полученные от child ошибки выводит в стандартный поток вывода

*Вариант 15)* Правило проверки: строка должна начинаться с заглавной буквы

## Общие сведения о программе

Программа компилируется из файлов parent.cpp с основным процессом, child.cpp с дочерним процессом и utils.cpp с вспомагательными функциями. Также имеются заголовочные файлы parent.hpp и utils.hpp и файл с тестами lab3_test.cpp. В программе работы были использованы следующие системные вызовы:

- sem_unlink() - удаление именного семафора
- shm_unlink() - удаление именованного сегмента разделяемой памяти
- shm_open() - создание или открытие объекта разделяемой памяти
- ftruncate() - установка размера разделяемой памяти
- mmap() - отображение файлов в адресное пространство процесса
- sem_open() - создание или открытие именованных семафоров
- fork() - создание нового процесса
- sem_post() - увеличение значения(разблокировка) семафора
- sem_wait() - уменьшение значения(блокировка) семафора
- sem_close() - закрытие именованного семафора
- munmap() - отключение отображения объекта в адресное пространство процесса

## Общий метод и алгоритм решения

Основной процесс считывает строки из стандартного ввода и записывает их в разделяемую память. После записи процесс уведомляет дочерний процесс о наличии новых данных с помощью семафора. Дочерний процесс, получив уведомление, считывает данные из разделяемой памяти, обрабатывает их, если строки соответствуют правилу - записывает из в файл, иначе записывает ошибочную строку(с маленькой буквы) обратно в разделяемую память, после чего уведомляет основной процесс о готовности результата. После обработки всех строк главный процесс отправляет пустую строку в общую память, чтобы дочерний процесс завершил свою работу. После завершения дочернего процесса главный процесс закрывает общие ресурсы и освобождает память.

# Исходный код

### parent.hpp

```
#pragma once

#include "utils.hpp"

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

### parent.cpp

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


void ParentRoutine(const char* pathToChild){
    std::string fileName;
    getline(std::cin, fileName);

    sem_unlink(SEMAPHORE_NAME);
    sem_unlink(RESPONSE_SEMAPHORE_NAME);
    shm_unlink(SHARED_MEMORY_NAME);
    shm_unlink(RESPONSE_MEMORY_NAME);

    int shared_memory_fd = OpenSharedMemory(SHARED_MEMORY_NAME, SIZE);
    int response_fd = OpenSharedMemory(RESPONSE_MEMORY_NAME, SIZE);
    char* shared_memory_ptr = MapSharedMemory(SIZE, shared_memory_fd);
    char* response_memory_ptr = MapSharedMemory(SIZE, response_fd);
    sem_t* semaphore = OpenSemaphore(SEMAPHORE_NAME);
    sem_t* response_semaphore = OpenSemaphore(RESPONSE_SEMAPHORE_NAME);

    std::vector<std::string> lines;
    std::string str;
    while (std::getline(std::cin, str) && !str.empty()) {
        lines.push_back(str);
    }

    pid_t pid = CreateChildProcess();

    if (pid == 0) {
        execlp(pathToChild, pathToChild, fileName.c_str(), nullptr);
        perror("Exec failed");
        exit(EXIT_FAILURE);
    } else {
        for (const std::string& line : lines) {
            strcpy(shared_memory_ptr, line.c_str());
            sem_post(semaphore);

            sem_wait(response_semaphore);
            if (strcmp(response_memory_ptr, "ERROR") == 0) {
                std::cerr << "Error: " << line << std::endl;
                strcpy(response_memory_ptr, "");
            }
        }

        strcpy(shared_memory_ptr, "");
        sem_post(semaphore);
    }

    wait(nullptr);

    sem_close(semaphore);
    sem_close(response_semaphore);
    munmap(shared_memory_ptr, SIZE);
    munmap(response_memory_ptr, SIZE);
    sem_unlink(SEMAPHORE_NAME);
    sem_unlink(RESPONSE_SEMAPHORE_NAME);
    shm_unlink(SHARED_MEMORY_NAME);
    shm_unlink(RESPONSE_MEMORY_NAME);
```

```
}
```

### utils.hpp

```
#pragma once

#include <iostream>
#include <string>
#include <fstream>
#include <stdlib.h>
#include <unistd.h>
#include <sys/types.h>
#include <sys/wait.h>
#include <sys/mman.h>
#include <semaphore.h>
#include <fcntl.h>
#include <cstring>
#include <cctype>
#include <vector>
#include <string_view>

constexpr const char* SHARED_MEMORY_NAME = "/shared_memory";
constexpr const char* RESPONSE_MEMORY_NAME = "/response_memory";
constexpr const char* SEMAPHORE_NAME = "/semaphore";
constexpr const char* RESPONSE_SEMAPHORE_NAME = "/response_semaphore";
constexpr int SIZE = 1024;

pid_t CreateChildProcess();
bool StartsWithCapital(const std::string_view str);
int OpenSharedMemory(const char* sharedMemoryName, int size);
char* MapSharedMemory(int size, int fd);
sem_t* OpenSemaphore(const char* semaphoreName);
```

### utils.cpp

```
#include "utils.hpp"

pid_t CreateChildProcess() {
    pid_t pid = fork();
    if (pid == -1) {
        perror("Error creating process");
        exit(EXIT_FAILURE);
    }
    return pid;
}

bool StartsWithCapital(const std::string_view str) {
    return !str.empty() && isupper(str[0]);
}

int OpenSharedMemory(const char* sharedMemoryName, int size) {
    int shared_memory_fd = shm_open(sharedMemoryName, O_CREAT | O_RDWR, S_IRUSR |
    S_IWUSR);
    if (shared_memory_fd == -1) {
        perror("Can't open shared memory object");
        exit(-1);
    }

    if (ftruncate(shared_memory_fd, size) == -1) {
        perror("Can't resize shared memory object");
        exit(-1);
    }

    return shared_memory_fd;
}

char* MapSharedMemory(int size, int fd) {
    char* shared_memory_ptr = (char*)mmap(nullptr, size, PROT_READ | PROT_WRITE,
    MAP_SHARED, fd, 0);
    if (shared_memory_ptr == MAP_FAILED) {
```

```
34          perror("Can't mmap shared memory object");
35          exit(-1);
36      }
37      return shared_memory_ptr;
38  }
39
40  sem_t* OpenSemaphore(const char* semaphoreName) {
41      sem_t* semaphore = sem_open(semaphoreName, O_CREAT, S_IRUSR | S_IWUSR, 0);
42      if (semaphore == SEM_FAILED) {
43          perror("Can't open semaphore");
44          exit(-1);
45      }
46      return semaphore;
47  }
```

### child.cpp

```
1  #include "utils.hpp"
2
3  int main(const int argc, const char* argv[]) {
4      if (argc != 2) {
5          perror("Necessary arguments were not provided");
6          exit(EXIT_FAILURE);
7      }
8
9      const char* fileName = argv[1];
10     std::ofstream out(fileName, std::ios::app);
11     if(!out.is_open()) {
12         perror("Error open");
13         exit(EXIT_FAILURE);
14     }
15
16     int shared_memory_fd = OpenSharedMemory(SHARED_MEMORY_NAME, SIZE);
17     int response_fd = OpenSharedMemory(RESPONSE_MEMORY_NAME, SIZE);
18
19     char* shared_memory_ptr = MapSharedMemory(SIZE, shared_memory_fd);
20     char* response_memory_ptr = MapSharedMemory(SIZE, response_fd);;
21
22     sem_t* semaphore = OpenSemaphore(SEMAPHORE_NAME);
23     sem_t* response_semaphore = OpenSemaphore(RESPONSE_SEMAPHORE_NAME);
24
25     while (true) {
26         sem_wait(semaphore);
27         std::string_view str(shared_memory_ptr);
28
29         if (str.empty()) {
30             sem_post(response_semaphore);
31             break;
32         }
33
34         if (StartsWithCapital(str)) {
35             out << str << std::endl;
36         } else {
37             strcpy(response_memory_ptr, "ERROR");
38         }
39
40         sem_post(response_semaphore);
41     }
42
43     sem_close(semaphore);
44     sem_close(response_semaphore);
45     munmap(shared_memory_ptr, SIZE);
46     munmap(response_memory_ptr, SIZE);
47     sem_unlink(SEMAPHORE_NAME);
48     sem_unlink(RESPONSE_SEMAPHORE_NAME);
49     shm_unlink(SHARED_MEMORY_NAME);
50     shm_unlink(RESPONSE_MEMORY_NAME);
51
52     exit(EXIT_SUCCESS);
53 }
```

### main.cpp

```cpp
#include "parent.hpp"

int main() {

    ParentRoutine(getenv("PATH_TO_CHILD"));

    exit(EXIT_SUCCESS);
}

// PATH_TO_CHILD="/home/arnemkova/OS_labs/lab3/build/child3"
```

### lab1_test.cpp

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

#include "parent.hpp"

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();
    std::cin.rdbuf(inFile.rdbuf());

    std::stringstream capturedErrorMessages;
    std::streambuf* oldCerrBuf = std::cerr.rdbuf(capturedErrorMessages.rdbuf());

    ParentRoutine(getenv("PATH_TO_CHILD"));

    std::cin.rdbuf(oldInBuf);
    std::cerr.rdbuf(oldCerrBuf);

    std::string capturedErrors = capturedErrorMessages.str();

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

    std::ifstream fin(fileName);
    for (const std::string &expectation : expectedFile) {
        std::string result;
        getline(fin, result);
        EXPECT_EQ(result, expectation);
    }
    fin.close();
    std::remove(fileName);
}


TEST(thirdLabTests, simpleTest) {
    std::vector<std::string> input = {"AAA"};
    std::vector<std::string> expectedOutput = {};
    std::vector<std::string> expectedFile = {"AAA"};
    testingProgram(input, expectedOutput, expectedFile);
}

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

```
55      std::vector<std::string> expectedFile = {};
56      testingProgram(input, expectedOutput, expectedFile);
57  }
58
59  TEST(thirdLabTests, sonneTest) {
60      std::vector<std::string> input = {
61          "Alle warten auf das Licht",
62          "Furchtet euch, furchtet euch nicht.",
63          "die Sonne scheint mir aus den Augen,",
64          "sie wird heute Nacht nicht untergehen",
65          "Und die Welt zahlt laut bis zehn"
66      };
67
68      std::vector<std::string> expectedOutput = {
69          "Error: die Sonne scheint mir aus den Augen,",
70          "Error: sie wird heute Nacht nicht untergehen"
71      };
72
73      std::vector<std::string> expectedFile = {
74          "Alle warten auf das Licht",
75          "Furchtet euch, furchtet euch nicht.",
76          "Und die Welt zahlt laut bis zehn"
77      };
78      testingProgram(input, expectedOutput, expectedFile);
79  }
80
81  TEST(thirdLabTests, deathTest) {
82      std::vector<std::string> input = {
83          "Can death be sleep, when life is but a dream,",
84          "And scenes of bliss pass as a phantom by?",
85          "the transient pleasures as a vision seem,",
86          "And yet we think the greatest pain's to die."
87      };
88
89      std::vector<std::string> expectedOutput = {
90          "Error: the transient pleasures as a vision seem,"
91      };
92
93      std::vector<std::string> expectedFile = {
94          "Can death be sleep, when life is but a dream,",
95          "And scenes of bliss pass as a phantom by?",
96          "And yet we think the greatest pain's to die."
97      };
98      testingProgram(input, expectedOutput, expectedFile);
99  }
100
101
102 int main(int argc, char *argv[]) {
103     testing::InitGoogleTest(&argc, argv);
104     return RUN_ALL_TESTS();
105 }
```

# Демонстрация работы программы

```
arnemkova@LAPTOP-TA2RV74U:~/OS_labs/build$ ./tests/lab3_test
[==========] Running 4 tests from 1 test suite.
[----------] Global test environment set-up.
[----------] 4 tests from thirdLabTests
[ RUN      ] thirdLabTests.simpleTest
[       OK ] thirdLabTests.simpleTest (2 ms)
[ RUN      ] thirdLabTests.emptystrTest
[       OK ] thirdLabTests.emptystrTest (1 ms)
[ RUN      ] thirdLabTests.sonneTest
[       OK ] thirdLabTests.sonneTest (2 ms)
[ RUN      ] thirdLabTests.deathTest
[       OK ] thirdLabTests.deathTest (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.
```

# Вывод

В ходе выполнения данной лабораторной работы я изучила принципы работы с файловыми системами и технологию обмена данными между процессами с использованием файловых отображений в память. Я познакомилась с системными вызовами, такими как open, mmap, munmap, shm_open, ftruncate, sem_open, sem_wait, sem_post. Также изучила работу с семафорами для обеспечения синхронизации доступа к общим ресурсам. При этом я узнала о понятиях таблицы страниц, page hit и page fault, которые играют важную роль при работе с общими сегментами памяти.