

Московский Авиационный Институт  
(Национальный Исследовательский Университет)  
Факультет информационных технологий и прикладной математики  
Кафедра вычислительной математики и программирования

**Лабораторная работа №1 по курсу  
«Операционные системы»**

Студент: Ширяев Никита Алексеевич

Группа: М8О-208Б-22

Вариант: 21

Преподаватель: Миронов Евгений Сергеевич

Оценка: \_\_\_\_\_

Дата: \_\_\_\_\_

Подпись: \_\_\_\_\_

Москва, 2023

## **Содержание**

1. Репозиторий
2. Постановка задачи
3. Общие сведения о программе
4. Общий метод и алгоритм решения
5. Исходный код
6. Демонстрация работы программы
7. Выводы

## 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 каналы (pipe). 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 lab1.cpp, и программа дочернего процесса, описанная в child.cpp. В программе используются следующие системные вызовы:

1. fork()
2. execv()
3. pipe()
4. read()
5. write()
6. open()
7. close()
8. dup2()

### 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 принимает от пользователя строки произвольной длины и пересылает их в pipe1 или в pipe2 в зависимости от правила 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

#### lab1.hpp

```
#pragma once

#include "utils.hpp"

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

#### utils.hpp

```
#pragma once

#include <algorithm>
#include <iostream>
#include <string>
#include <sstream>
#include <stdlib.h>
#include <unistd.h>
#include <sys/types.h>
#include <fstream>
#include <fcntl.h>
#include <sys/wait.h>

void createPipe(int fd[2]);
pid_t createChildProcess();
```

## child.cpp

```
#include "utils.hpp"

int main(void) {
    std::string str;
    while (std::getline(std::cin, str)) {
        std::reverse(str.begin(), str.end());
        printf("%s\n", str.c_str());
    }

    exit(EXIT_SUCCESS);
}
```

## lab1.cpp

```
#include "lab1.hpp"

int ParentRoutine(const char *pathToChild) {

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

    int fd1[2] {-1, -1}; //pipe1
    int fd2[2] {-1, -1}; //pipe2
    int fd[2] {-1, -1};
    createPipe(fd1);
    createPipe(fd2);
    const mode_t MODE = 0644;
    fd[0] = open(fileName1.c_str(), O_CREAT | O_WRONLY |
O_TRUNC, MODE);
    fd[1] = open(fileName2.c_str(), O_CREAT | O_WRONLY |
O_TRUNC, MODE);

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

```

pid[1] = createChildProcess();

if (pid[0] == 0) { //child 1
    close(fd2[0]);
    close(fd2[1]);
    close(fd1[1]);
    close(fd[1]);

    if (dup2(fd1[0], STDIN_FILENO) == -1) {
        perror("Error with dup2");
        exit(EXIT_FAILURE);
    }

    dup2(fd[0], STDOUT_FILENO);
    close(fd[0]);

    close(fd1[0]);
    if (execl(pathToChild, "child", nullptr) == -1) {
        perror("Error with execl");
        exit(EXIT_FAILURE);
    }
} else if (pid[1] == 0) { //child 2
    close(fd1[0]);
    close(fd1[1]);
    close(fd2[1]);
    close(fd[0]);

    if (dup2(fd2[0], STDIN_FILENO) == -1) {
        perror("Error with dup2");
        exit(EXIT_FAILURE);
    }

    dup2(fd[1], STDOUT_FILENO);
    close(fd[1]);

    close(fd2[0]);
    if (execl(pathToChild, "child", nullptr) == -1) {

```

```

        perror("Error with execlp");
        exit(EXIT_FAILURE);
    }

} else { //parent
    close(fd1[0]);
    close(fd2[0]);
    close(fd[0]);
    close(fd[1]);
    std::string str;
    while (getline(std::cin, str)) {
        str += "\n";
        if (str.size() % 2 != 0) {
            write(fd1[1], str.c_str(), str.size());
        } else {
            write(fd2[1], str.c_str(), str.size());
        }
        str.clear();
    }
    close(fd1[1]);
    close(fd2[1]);

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

}
return 0;
}

```

### utils.cpp

```

#include "utils.hpp"

void createPipe(int fd[2]) {
    if (pipe(fd) == -1) {
        perror("Couldn't create pipe");
    }
}

```

```

        exit(EXIT_FAILURE);
    }
}

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

```

### lab1\_test.cpp

```

#include <gtest/gtest.h>

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

#include <lab1.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 *fileWithOutput1 = "output1.txt";
    const char *fileWithOutput2 = "output2.txt";

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

```



```

        std::streambuf* oldInBuf =
std::cin.rdbuf(inFile.rdbuf());

ASSERT_TRUE(fs::exists("/home/test/Desktop/os-labs/build/lab1
/child-lab1"));

ParentRoutine("/home/test/Desktop/os-labs/build/lab1/child-la
b1");
    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();

    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();
}

```

```

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

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

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

    testingProgram(input, expectedOutput1, expectedOutput2);
}

TEST(firstLabTests, 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(firstLabTests, 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(firstLabTests, 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();
}

```

## main.cpp

```
#include "lab1.hpp"

int main() {
    const char path[] =
"/home/test/Desktop/os-labs/build/lab1/child-lab1";
    ParentRoutine(path);
    exit(EXIT_SUCCESS);
}
```

## Демонстрация работы программы

```
hacker@warmachine:~/prog/my_os_labs/build/lab1$ ./lab1
file1
file2
hello
hi
good morning
bye
^Z
[2]+ Stopped ./lab1
hacker@warmachine:~/prog/my_os_labs/build/lab1$ cat file1
ih
gninrom doog
hacker@warmachine:~/prog/my_os_labs/build/lab1$ cat file2
olleh
eyb
```

## Выводы

В ходе выполнения лабораторной работы я получил знания и навыки использования системных вызовов Linux при написании программ. Я узнал о системных вызовах fork, pipe, dup2 и научился их применять. Также были получены знания о структуре размещения процессов в памяти компьютера.