

Московский авиационный институт  
(Национальный исследовательский университет)  
Факультет "Информационные технологии и прикладная математика"  
Кафедра "Вычислительная математика и программирование"

**Лабораторная работа №1 по курсу  
“Операционные системы”**

*Студент:* Сибирцев Роман Денисович

*Группа:* М8О-208Б-22

*Преподаватель:* Миронов Евгений Сергеевич

*Вариант:* 13

*Оценка:* \_\_\_\_\_

*Дата:* \_\_\_\_\_

*Подпись:* \_\_\_\_\_

Москва, 2023

# Содержание

1	Репозиторий . . . . .	3
2	Цель работы . . . . .	3
3	Задание . . . . .	3
4	Описание работы программы . . . . .	3
5	Исходный код . . . . .	4
6	Тесты . . . . .	9
7	Запуск тестов . . . . .	10
8	Выводы . . . . .	11

# 1 Репозиторий

<https://github.com/RomanSibirtsev/MAIOSlabs>

## 2 Цель работы

Приобретение практических навыков в:

- Управлении процессами в ОС
- Обеспечении обмена данных между процессами посредством каналов

## 3 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 каналы (pipe). Необходимо обрабатывать системные ошибки, которые могут возникнуть в результате работы.

## 4 Описание работы программы

Родительский процесс создает два дочерних процесса. Перенаправление стандартных потоков ввода-вывода показано на картинке выше. Child1 и Child2 можно «соединить» между собой дополнительным каналом. Родительский и дочерний процесс должны быть представлены разными программами. Родительский процесс принимает от пользователя строки произвольной длины и пересылает их в pipe1. Процесс child1 и child2 производят работу над строками. Child2 пересылает результат своей работы родительскому процессу. Родительский процесс полученный результат выводит в стандартный поток вывода.

В ходе выполнения лабораторной работы я использовал следующие системные вызовы:

- fork() - создание нового процесса
- pipe() - создание канала
- dup2() - создание копии файлового дескриптора, используя для нового дескриптора самый маленький свободный номер файлового дескриптора.
- execvp() - запуск файла на исполнение

## 5 Исходный код

parent.hpp

```
1 #pragma once
2
3 #include <iostream>
4 #include <fstream>
5 #include <string>
6 #include <sys/wait.h>
7 #include <unistd.h>
8
9 int ParentWork();
10 void CreatePipe(int *fd);
11 int CreateFork();
12 void DoClose(int fd);
```

parent.cpp

```
1 #include "parent.hpp"
2
3 void CreatePipe(int *fd) {
4     int status;
5     status = pipe(fd);
6     if (status == -1) {
7         std::cerr << "Failed pipe()\n";
8         exit(-1);
9     }
10 }
11
12 int CreateFork() {
13     int pid;
14     pid = fork();
15     if (pid == -1) {
16         std::cerr << "Failed fork()\n";
17         exit(-2);
18     }
19     return pid;
20 }
21
22 void DoClose(int fd) {
23     int status;
24     status = close(fd);
25     if (status == -1) {
26         std::cerr << "Failed close()\n";
27         exit(-3);
28     }
29 }
30
31 void DoDup(int a, int b) {
32     int status;
33     status = dup2(a, b);
34     if (status == -1) {
35         std::cerr << "Failed dup2()\n";
36         exit(-4);
37     }
38 }
39
40 int ParentWork() {
41     pid_t pid;
42     int status;
```

```

43     int capacity = 0;
44     int fd1[2];
45     int fd2[2];
46     int connect[2];
47     char *child1;
48     char *child2;
49     std::string line;
50
51     child1 = getenv("PATH_CHILD1");
52     child2 = getenv("PATH_CHILD2");
53
54     CreatePipe(fd1);
55     CreatePipe(fd2);
56     CreatePipe(connect);
57
58     pid = CreateFork();
59     if (pid == 0) { // child 1
60         DoClose(connect[0]);
61         DoClose(fd1[1]);
62         DoDup(connect[1], STDOUT_FILENO);
63         if (execlp(child1, "lower.out", std::to_string(fd1[0]).
c_str(), nullptr) == -1) {
64             std::cerr << "Failed execlp()\n";
65             exit(-5);
66         }
67     }
68     pid = CreateFork();
69     if (pid == 0) { // child 2
70         DoClose(connect[1]);
71         DoClose(fd2[0]);
72         DoDup(connect[0], STDIN_FILENO);
73         if (execlp(child2, "underscore.out", std::to_string(fd2
[1]).c_str(), nullptr) == -1) {
74             std::cerr << "Failed execlp()\n";
75             exit(-5);
76         }
77     }
78
79     if (pid != 0) {
80         DoClose(fd1[0]);
81         DoClose(fd2[1]);
82         char chWrite;
83         char end = '\0';
84         char toOut[256];
85         int readCount;
86         while (std::getline(std::cin, line)) {
87             write(fd1[1], line.c_str(), line.size());
88
89             chWrite = '\n';
90             write(fd1[1], &chWrite, sizeof(chWrite));
91
92             capacity += line.size() + 1;
93         }
94         write(fd1[1], &end, sizeof(end));
95
96         while(capacity > 0) {
97             readCount = read(fd2[0], toOut, sizeof(toOut));
98             if (readCount != -1) {
99                 capacity -= readCount;

```

```

100         }
101         //std::cout << toOut << std::flush;
102         for (int i = 0; i < readCount; ++i) {
103             std::cout << toOut[i] << std::flush;
104         }
105     }
106 }
107
108 waitpid(-1, &status, 0);
109 waitpid(-1, &status, 0);
110 return 0;
111 }

```

#### lower.cpp

```

1 #include <iostream>
2 #include <string>
3 #include <unistd.h>
4 #include <sys/wait.h>
5 #include <sys/stat.h>
6 #include <fcntl.h>
7
8 int main (int argc, char** argv) {
9     if (argc != 2) {
10         std::cerr << "Wrong argc in lower.out\n";
11         exit(-6);
12     }
13     char chLow;
14     int fd = std::atoi(argv[1]);
15     while(true) {
16         read(fd, &chLow, sizeof(chLow));
17         if(chLow == '\0') {
18             break;
19         }
20         chLow = (char)tolower(chLow);
21         std::cout << chLow << std::flush;
22     }
23     std::cout << '\0';
24 }

```

#### underscore.cpp

```

1 #include <iostream>
2 #include <string>
3 #include <unistd.h>
4 #include <sys/wait.h>
5 #include <sys/stat.h>
6 #include <fcntl.h>
7
8 int main (int argc, char** argv) {
9     if (argc != 2) {
10         std::cerr << "Wrong argc in underscore.out\n";
11         exit(-6);
12     }
13     char ch;
14     int fd = std::atoi(argv[1]);
15     (void)argc;
16     while(true) {
17         std::cin.get(ch);
18         if (ch == ',') {
19             ch = '_';

```

```

20         } else if (ch == '\0') {
21             break;
22         }
23         write(fd, &ch, sizeof(ch));
24     }
25 }

```

parent.cpp

```

1 #include "parent.hpp"
2
3 void parentProcess(const char *pathToChild) {
4     std::string fileName;
5     getline(std::cin, fileName);
6
7     int fd1[2], fd2[2];
8     createPipe(fd1);
9     createPipe(fd2);
10
11     int pid = createChildProcess();
12     if (pid != 0) { // Parent process
13         close(fd1[0]);
14         close(fd2[1]);
15
16         std::string str;
17         while (getline(std::cin, str)) {
18             str += "\n";
19             write(fd1[1], str.c_str(), str.length()); // from str
20             to fd1[1]
21         }
22         close(fd1[1]);
23
24         std::stringstream output = readFromPipe(fd2[0]);
25         while (std::getline(output, str)) {
26             std::cout << str << std::endl;
27         }
28         close(fd2[0]);
29     } else { // Child process
30         close(fd1[1]);
31         close(fd2[0]);
32
33         if (dup2(fd1[0], STDIN_FILENO) == -1 || dup2(fd2[1],
34             STDOUT_FILENO) == -1) {
35             perror("Error with dup2");
36             exit(EXIT_FAILURE);
37         }
38
39         if (execlp(pathToChild, pathToChild, fileName.c_str(),
40             nullptr) == -1) { // to child.cpp
41             perror("Error with execlp");
42             exit(EXIT_FAILURE);
43         }
44     }
45 }

```

main.cpp

```

1 #include "parent.hpp"
2
3 int main() {
4     ParentWork();
5 }

```

```
5     return 0;  
6 }
```



## 6 Тесты

```
1 #include <iostream>
2 #include <gtest/gtest.h>
3
4 #include "parent.hpp"
5
6 void TestParent(std::string &src, std::string &res) {
7     std::istringstream srcStream(src);
8
9     std::streambuf* buf = std::cin.rdbuf(srcStream.rdbuf());
10
11     testing::internal::CaptureStdout();
12     ParentWork();
13     ASSERT_EQ(testing::internal::GetCapturedStdout(), res + '\n');
14
15     std::cin.rdbuf(buf);
16 }
17
18
19 TEST(cin_test, ONE) {
20     std::string src = "          ";
21     std::string res = "-----";
22     TestParent(src, res);
23 }
24
25 TEST(cin_test, TWO) {
26     std::string src = "AHAHAHAHAH";
27     std::string res = "ahahahahah";
28     TestParent(src, res);
29 }
30
31 TEST(cin_test, THREE) {
32     std::string src = "    HELLO wOrLd 12 3";
33     std::string res = "___hello_world_12_3";
34     TestParent(src, res);
35 }
```

## 7 Запуск тестов

```
roman@DESKTOP-K3DH39N:~/MAI/MAI_OS_labs/build/tests/lab1_test$ ./lab1_test
Running main() from /home/roman/MAI/MAI_OS_labs/build/_deps/googletest-src/googletest
[=====] Running 3 tests from 1 test suite.
[-----] Global test environment set-up.
[-----] 3 tests from cin_test
[ RUN      ] cin_test.ONE
[       OK ] cin_test.ONE (1 ms)
[ RUN      ] cin_test.TWO
[       OK ] cin_test.TWO (1 ms)
[ RUN      ] cin_test.THREE
[       OK ] cin_test.THREE (1 ms)
[-----] 3 tests from cin_test (3 ms total)

[-----] Global test environment tear-down
[=====] 3 tests from 1 test suite ran. (3 ms total)
[ PASSED  ] 3 tests.
```

## 8 Выводы

В результате выполнения данной лабораторной работы была написана программа на языке C++, осуществляющая работу с процессами и взаимодействие между ними. Я приобрел практические навыки в управлении процессами в ОС и обеспечении обмена данными между процессами посредством каналов.