

МИНОБРНАУКИ РОССИИ  
САНКТ-ПЕТЕРБУРГСКИЙ ГОСУДАРСТВЕННЫЙ  
ЭЛЕКТРОТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ  
«ЛЭТИ» ИМ. В.И. УЛЬЯНОВА (ЛЕНИНА)  
Кафедра Вычислительной техники

ОТЧЁТ  
по лабораторной работе №7  
по дисциплине «Организация процессов и программирования в среде Linux»  
Тема: ОБМЕН ДАННЫМИ ЧЕРЕЗ КАНАЛ

Студент гр. 9308

Преподаватель

\_\_\_\_\_  
\_\_\_\_\_  
\_\_\_\_\_

Соболев М.С.

Разумовский Г.В.

Санкт-Петербург,

2022

## Оглавление

1. Введение.....	3
1.1. Введение.....	3
1.2. Порядок выполнения работы.....	3
1.3. Содержание отчёта.....	4
2. Тексты программы и её двух потомков.....	5
2.1. main.cpp.....	5
2.2. executable_1.cpp.....	8
2.3. executable_2.cpp.....	10
3. Распечатки входных и выходного файлов.....	12
4. Вывод.....	22
5. Список использованных источников.....	23

# 1. Введение

## 1.1. Введение

Тема работы: Обмен данными через канал.

Цель работы: Знакомство с механизмом обмена данными через программный канал и системными вызовами, обеспечивающими такой обмен.

## 1.2. Порядок выполнения работы

1. Написать программу, которая в качестве параметров принимает имена трёх текстовых файлов (2 входных и 1 выходной). Программа должна открыть канал и выходной файл, а затем породить двух потомков, которым передаются дескриптор канала для записи и имя входного файла. Каждый потомок выполняет свою программу, читая построчно текст из входного файла и записывая его в канал. Программа параллельно посимвольно читает данные из канала и записывает их в выходной файл, до тех пор пока оба потомка не закончат свою работу и канал будет пуст.

2. Написать программу, которая обменивается данными через канал с двумя потомками. Программа открывает входной файл, построчно читает из него данные и записывает их в канал. Потомки выполняют свои программы и поочередно читают символы из канала и записывают их в свои выходные файлы: первый потомок — нечётные символы, а второй — чётные. Синхронизация работы потомков должна осуществляться напрямую с использованием сигналов SIGUSR1 и SIGUSR2. Об окончании записи файла в канал программа оповещает потомков сигналом SIGQUIT и ожидает завершения работы потомков. Когда они заканчивают работу, программа закрывает канал.

3. Откомпилировать все программы и запустить их.

Выбранные задания: 2, 3.

### **1.3. Содержание отчёта**

Отчёт по лабораторной работе должен содержать:

1. Цель и задание.
2. Тексты программы и её двух потомков.
3. Распечатки входных и выходного файлов.

## 2. Тексты программы и её двух потомков

### 2.1. main.cpp

```
// start program
// ./main <filename.txt>
// <filename.txt> -- name of the file with .txt extension, which will be read by program
// e.g. "./main lorem_ipsum.txt"

#include <iostream>
#include <fstream>
#include <fcntl.h>
#include <signal.h>
#include <string.h>
#include <unistd.h>
#include <wait.h>

using namespace std;

int main(int argc, char *argv[])
{
    int fildes[2]; // pipe channels handles, fildes[0] -- read from pipe, fildes[1] -- write to pipe
    pid_t pid_1; // child process 1 pid
    pid_t pid_2; // child process 2 pid
    char ch[80]; // char buffer
    char *c = NULL; // file end indicator, if just "char *c;" instead of "char *c = NULL;" program won't end, idk
    why
    sigset_t set; // process' signals set
    FILE* fp = NULL; // file for parent process' read

    fp = fopen(argv[1], "r"); // file opening w/ read flag

    if (fp == NULL)
    {
        cout << "----- FILE HAS NOT BEEN OPENED SUCCESSFULLY -----\\n";
        exit(2);
    }
    else
    {

```

```

        cout << "----- FILE HAS BEEN OPENED SUCCESSFULLY -----\\n";
    }

    // adding sync signals to process set
    sigaddset(&set, SIGQUIT);
    sigaddset(&set, SIGUSR1);
    sigaddset(&set, SIGUSR2);
    sigprocmask(SIG_BLOCK, &set, NULL); // blocking signals in mask

    // creating pipe
    if(pipe2(fildes, O_NONBLOCK) == -1) // if program can't create pipe, then exit w/ error
    {
        cout << "----- PIPE HAS NOT BEEN CREATED SUCCESSFULLY -----\\n"; // message
about creating pipe unsuccessfully/not creating pipe
        exit(1);
    }
    else // if program can create pipe, then continue execution
    {
        cout << "----- PIPE HAS BEEN CREATED SUCCESSFULLY -----\\n"; // message about
creating pipe successfully/not creating pipe
        pid_1 = fork(); // child process 1 creation
        if(pid_1 == 0)
        {
            cout << "----- CHILD PROCESS 1 BEGINS -----\\n";
            close(fildes[1]); // closing pipe to write by child process 1
            execl("executable_1", "executable_1", &fildes[0], &fildes[1], "output_file_1.txt", NULL);
        }
        else
        {
            pid_2 = fork(); // child process 2 creation
            if(pid_2 == 0)
            {
                cout << "----- CHILD PROCESS 2 BEGINS -----\\n";
                close(fildes[1]); // closing pipe to write by child process 2
                execl("executable_2", "executable_2", &fildes[0], &fildes[1], "output_file_2.txt",
NULL);
            }
        }
    }
}

```

```

close(fildes[0]); // closing pipe to read by parent process
cout << "----- PARENT PROCESS BEGINS WRITING DATA TO THE PIPE -----\\n";
c = fgets(ch, 80, fp); // get data from file & check if file is ended
while(c) // writing data from <filename>
{
    write(fildes[1], ch, strlen(ch) - 1); // writing data to the pipe
    c = fgets(ch, 80, fp); // get data from file & check if file is ended
}
cout << "----- PARENT PROCESS ENDS WRITING DATA TO THE PIPE -----\\n";

kill(pid_1, SIGQUIT);
kill(pid_2, SIGQUIT);

waitpid(pid_1, NULL, 0); // waiting child process 1 termination
cout << "----- CHILD PROCESS 1 ENDS -----\\n";
waitpid(pid_2, NULL, 0); // waiting child process 2 termination
cout << "----- CHILD PROCESS 2 ENDS -----\\n";

close(fildes[1]); // closing pipe to write by parent process
}

return 0;
}

```

## 2.2. executable\_1.cpp

```
// WARNING: create file in Linux to have "\n" ending (LF) instead of "\r\n" (CRLF) in Windows file
// LF -- line feed ("\n")
// CRLF -- carriage return line feed ("\r\n")

#include <iostream>
#include <fstream>
#include <unistd.h>
#include <signal.h>

using namespace std;

// false -- writing in pipe is NOT finished
// true -- writing in pipe is finished
bool pipe_write_is_finished = false; // end of writing indication
void LocalHandler (int local_int);

int main(int argc, char *argv[]) // getting output filename as parameter
{
    FILE* output_file_1 = fopen(argv[3], "w"); // file opening w/ write flag
    int sig; // for sigwait
    int pipe_read_is_done = 1; // if > 0, pipe read is NOT done, if < 0, is done
    int fildes[2]; // pipe channels handles, fildes[0] -- read from pipe, fildes[1] -- write to pipe
    char ch; // char buffer
    sigset_t b_set;
    sigset_t set;
    struct sigaction sigact;

    fildes[0] = *argv[1];
    fildes[1] = *argv[2];

    sigaddset(&set, SIGUSR1); // SIGUSR1 signal add to child process 1 set
    sigact.sa_handler = &LocalHandler; // setting new handler
    sigaction(SIGQUIT, &sigact, NULL); // changing function reaction to SIGQUIT
    sigaddset(&b_set, SIGQUIT); // SIGQUIT signal add to set
    sigprocmask(SIG_UNBLOCK, &b_set, NULL); // unblock SIGQUIT w/ set signals reaction
```



```

cout << "----- CHILD PROCESS 1 BEGINS WRITING DATA F/ THE PIPE TO FILE -----\\n";

//pipe_read_is_done = read(fildes[0], &ch, 1); // reading from pipe
// pipe finish condition is in priority -- if pipe writing is not finished, this program MUST wait
while (pipe_write_is_finished == false || (pipe_read_is_done = read(fildes[0], &ch, 1)) > 0) // while pipe
writing by parent process is not finished & pipe not done
{
    if (pipe_read_is_done > 0) // additional check pipe if is done (if previous while there was ||, not &&)
    {
        fputc(ch, output_file_1); // writing to the file
        kill(0, SIGUSR2); // child process 1 give signal to child process 2
        sigwait(&set, &sig); // child process 1 waits child process 2
    }
    //pipe_read_is_done = read(fildes[0], &ch, 1); // reading from pipe
}

kill(0, SIGUSR2); // child process 1 give signal to child process 2 FINAL BEFORE TERMINATION
fclose(output_file_1); // close output file
close(fildes[0]); // close pipe fo read
exit(0);
}

void LocalHandler (int local_int)
{
    pipe_write_is_finished = true; // pipe writing is finished
}

```

## 2.3. executable\_2.cpp

```
// WARNING: create file in Linux to have "\n" ending (LF) instead of "\r\n" (CRLF) in Windows file
// LF -- line feed ("\n")
// CRLF -- carriage return line feed ("\r\n")

#include <iostream>
#include <fstream>
#include <unistd.h>
#include <signal.h>

using namespace std;

// false -- writing in pipe is NOT finished
// true -- writing in pipe is finished
bool pipe_write_is_finished = false; // end of writing indication
void LocalHandler (int local_int);

int main(int argc, char *argv[])
{
    FILE* output_file_2 = fopen(argv[3], "w"); // file opening w/ write flag
    int sig; // for sigwait
    int pipe_read_is_done = 1; // if > 0, pipe read is NOT done, if < 0, is done
    int fildes[2]; // pipe channels handles, fildes[0] -- read from pipe, fildes[1] -- write to pipe
    char ch; // char buffer
    sigset_t b_set;
    sigset_t set;
    struct sigaction sigact;

    fildes[0] = *argv[1];
    fildes[1] = *argv[2];

    sigaddset(&set, SIGUSR2); // SIGUSR2 signal add to child process 2 set
    sigact.sa_handler = &LocalHandler; // setting new handler
    sigaction(SIGQUIT, &sigact, NULL); // changing function reaction to SIGQUIT
    sigaddset(&b_set, SIGQUIT); // SIGQUIT signal add to set
    sigprocmask(SIG_UNBLOCK, &b_set, NULL); // unblock SIGQUIT w/ set signals reaction
```

```

cout << "----- CHILD PROCESS 2 BEGINS WRITING DATA F/ THE PIPE TO FILE -----\\n";

sigwait(&set, &sig); // child process 2 waits child process 1 (1st waiting before reading)

//pipe_read_is_done = read(fildes[0], &ch, 1); // reading from pipe
// pipe finish condition is in priority -- if pipe writing is not finished, this program MUST wait
while (pipe_write_is_finished == false || (pipe_read_is_done = read(fildes[0], &ch, 1)) > 0) // while pipe
writing by parent process is not finished & pipe not done
{
    if (pipe_read_is_done > 0) // additional check pipe if is done (if previous while there was ||, not &&)
    {
        fputc(ch, output_file_2); // writing to the file
        kill(0, SIGUSR1); // child process 2 give signal to child process 1
        sigwait(&set, &sig); // child process 2 waits child process 1
    }
    //pipe_read_is_done = read(fildes[0], &ch, 1); // reading from pipe
}

kill(0, SIGUSR1); // child process 2 give signal to child process 1 FINAL BEFORE TERMINATION
fclose(output_file_2); // close output file
close(fildes[0]); // close pipe fo read
exit(0);
}

void LocalHandler (int local_int)
{
    pipe_write_is_finished = true; // pipe writing is finished
}

```

### 3. Распечатки входных и выходного файлов

Программа запускается с помощью команды «./main <название файла.txt>», где «./main» – это программа, а «<название файла.txt>» – читаемый файл в формате txt.

Программа предусматривает возможность корректного завершения работы при отсутствии необходимого файла.

При создании файла также важно учитывать формат окончания строки, на Unix/Linux это LF («\n»), а на Windows это CRLF («\r\n»). Из-за того, что эти файлы будут читаться по-разному, могут возникнуть ошибки, и программа может работать некорректно. Для этого файлы создавались на Linux.



Рисунок 1. Исходный (читаемый) файл без символов



Рисунок 2. Исходный (читаемый) файл с 1 символом



Рисунок 3. Исходный (читаемый) файл с 2 символами

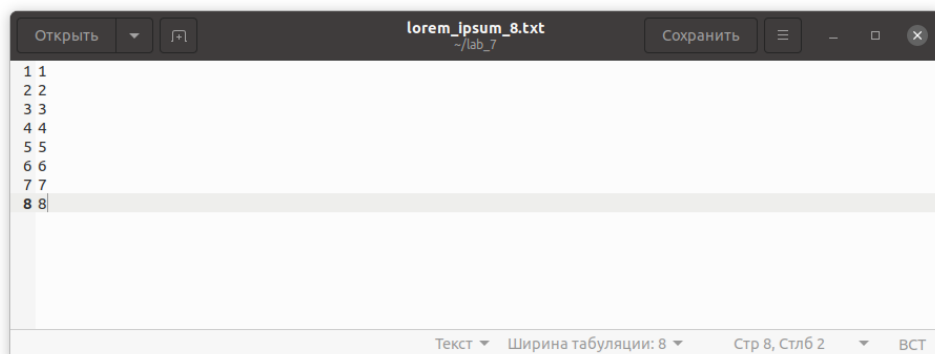
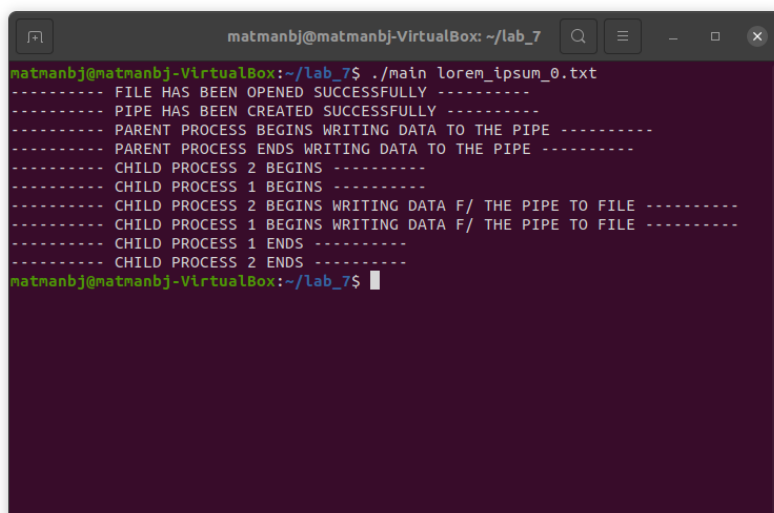


Рисунок 4. Исходный (читаемый) файл с 8 символами



Рисунок 5. Исходный (читаемый) файл с 9 символами



```
matmanbj@matmanbj-VirtualBox: ~/lab_7
matmanbj@matmanbj-VirtualBox:~/lab_7$ ./main lorem_ipsum_0.txt
----- FILE HAS BEEN OPENED SUCCESSFULLY -----
----- PIPE HAS BEEN CREATED SUCCESSFULLY -----
----- PARENT PROCESS BEGINS WRITING DATA TO THE PIPE -----
----- PARENT PROCESS ENDS WRITING DATA TO THE PIPE -----
----- CHILD PROCESS 2 BEGINS -----
----- CHILD PROCESS 1 BEGINS -----
----- CHILD PROCESS 2 BEGINS WRITING DATA F/ THE PIPE TO FILE -----
----- CHILD PROCESS 1 BEGINS WRITING DATA F/ THE PIPE TO FILE -----
----- CHILD PROCESS 1 ENDS -----
----- CHILD PROCESS 2 ENDS -----
matmanbj@matmanbj-VirtualBox:~/lab_7$
```

Рисунок 6. Запуск программы с чтением исходного файла без символов

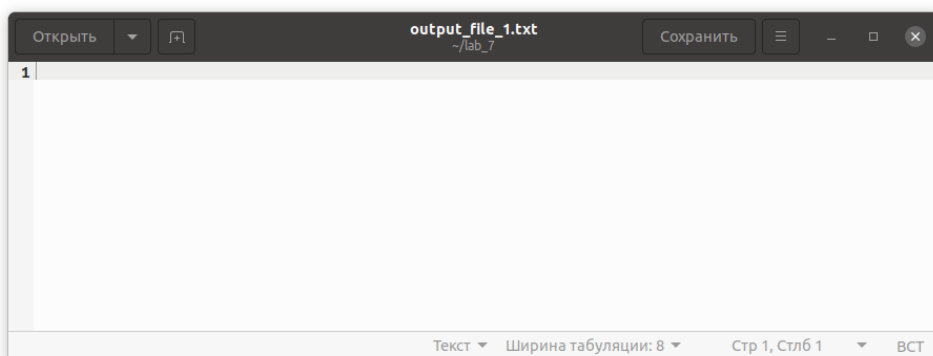


Рисунок 7. Созданный (записываемый) файл для нечётных символов при чтении файла без символов

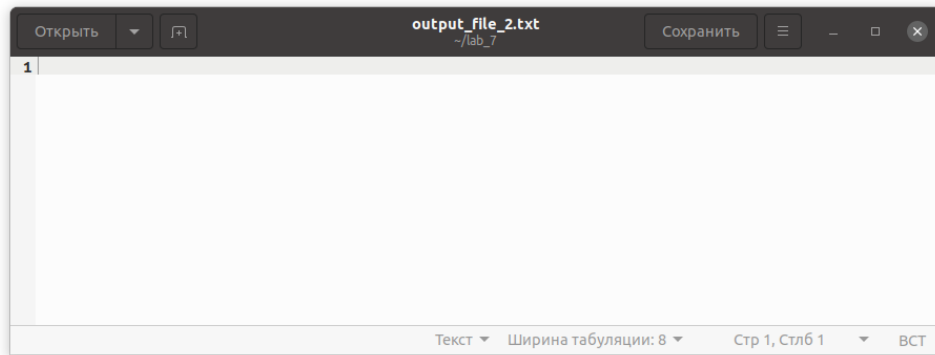


Рисунок 8. Созданный (записываемый) файл для чётных символов при чтении файла без символов

```
matmanbj@matmanbj-VirtualBox: ~/lab_7
matmanbj@matmanbj-VirtualBox:~/lab_7$ ./main lorem_ipsum_1.txt
----- FILE HAS BEEN OPENED SUCCESSFULLY -----
----- PIPE HAS BEEN CREATED SUCCESSFULLY -----
----- PARENT PROCESS BEGINS WRITING DATA TO THE PIPE -----
----- PARENT PROCESS ENDS WRITING DATA TO THE PIPE -----
----- CHILD PROCESS 2 BEGINS -----
----- CHILD PROCESS 1 BEGINS -----
----- CHILD PROCESS 2 BEGINS WRITING DATA F/ THE PIPE TO FILE -----
----- CHILD PROCESS 1 BEGINS WRITING DATA F/ THE PIPE TO FILE -----
----- CHILD PROCESS 1 ENDS -----
----- CHILD PROCESS 2 ENDS -----
matmanbj@matmanbj-VirtualBox:~/lab_7$
```

Рисунок 9. Запуск программы с чтением исходного файла с 1 символом

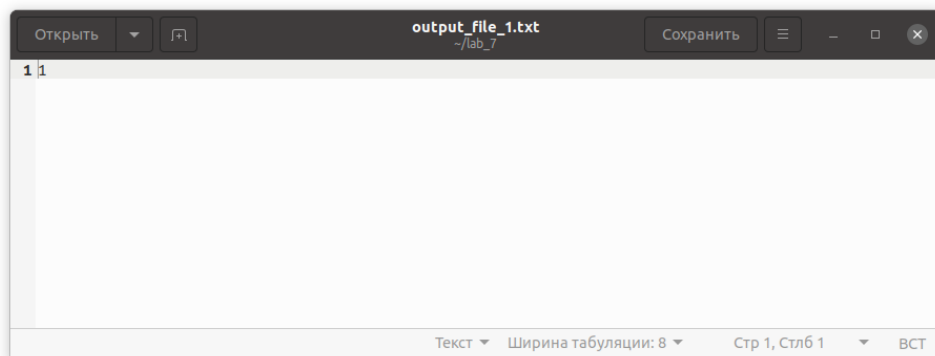


Рисунок 10. Созданный (записываемый) файл для нечётных символов при чтении файла с 1 символом

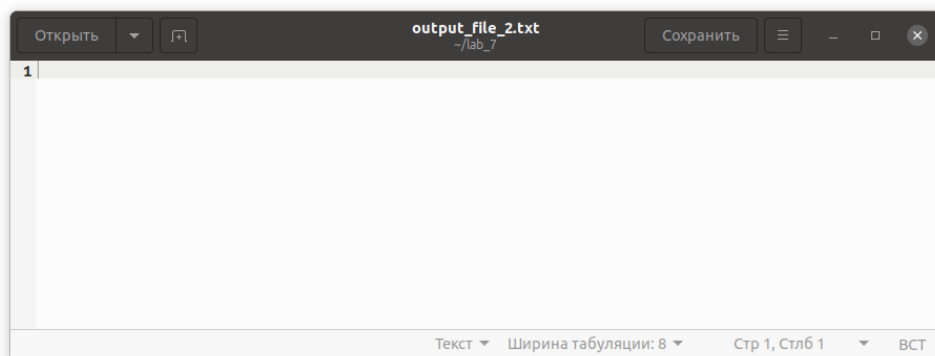
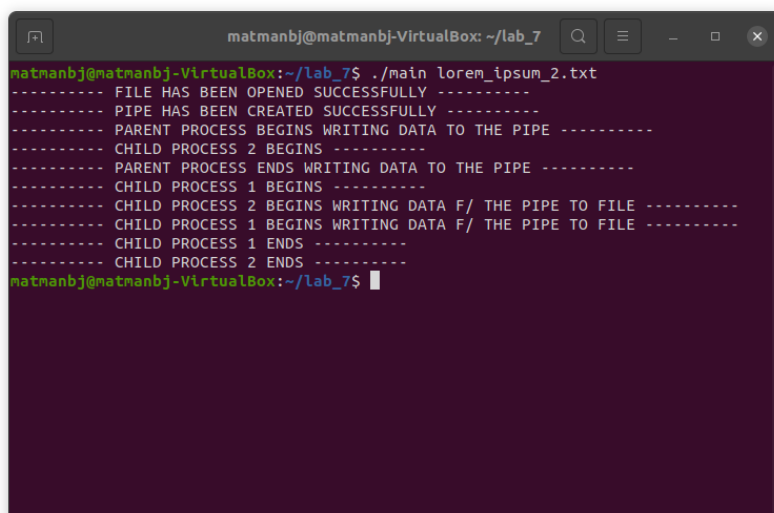


Рисунок 11. Созданный (записываемый) файл для чётных символов при чтении файла с 1 символом





```
matmanbj@matmanbj-VirtualBox: ~/lab_7
matmanbj@matmanbj-VirtualBox:~/lab_7$ ./main lorem_ipsum_2.txt
----- FILE HAS BEEN OPENED SUCCESSFULLY -----
----- PIPE HAS BEEN CREATED SUCCESSFULLY -----
----- PARENT PROCESS BEGINS WRITING DATA TO THE PIPE -----
----- CHILD PROCESS 2 BEGINS -----
----- PARENT PROCESS ENDS WRITING DATA TO THE PIPE -----
----- CHILD PROCESS 1 BEGINS -----
----- CHILD PROCESS 2 BEGINS WRITING DATA F/ THE PIPE TO FILE -----
----- CHILD PROCESS 1 BEGINS WRITING DATA F/ THE PIPE TO FILE -----
----- CHILD PROCESS 1 ENDS -----
----- CHILD PROCESS 2 ENDS -----
matmanbj@matmanbj-VirtualBox:~/lab_7$
```

Рисунок 12. Запуск программы с чтением исходного файла с 2 символами

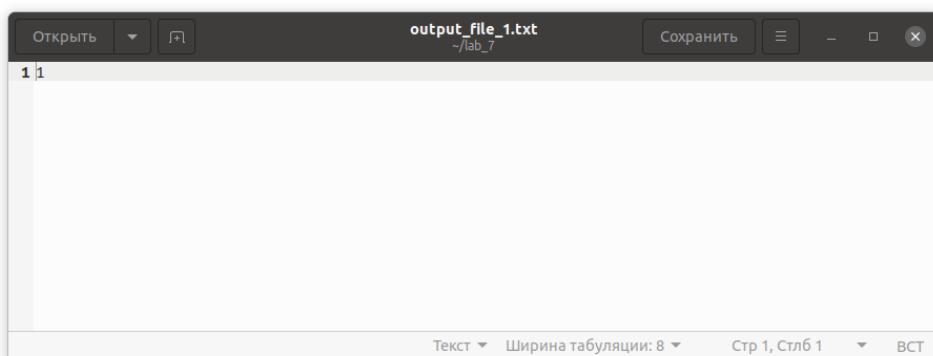


Рисунок 13. Созданный (записываемый) файл для нечётных символов при чтении файла с 2 символами

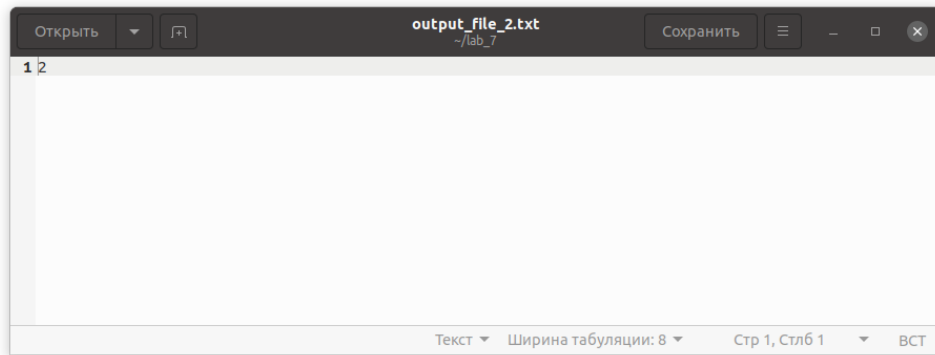


Рисунок 14. Созданный (записываемый) файл для чётных символов при чтении файла с 2 символами

A screenshot of a terminal window with a dark purple background. The prompt is 'matmanbj@matmanbj-VirtualBox: ~/lab\_7'. The command executed is './main lorem\_ipsum\_8.txt'. The output shows a series of status messages: 'FILE HAS BEEN OPENED SUCCESSFULLY', 'PIPE HAS BEEN CREATED SUCCESSFULLY', 'PARENT PROCESS BEGINS WRITING DATA TO THE PIPE', 'PARENT PROCESS ENDS WRITING DATA TO THE PIPE', 'CHILD PROCESS 2 BEGINS', 'CHILD PROCESS 1 BEGINS', 'CHILD PROCESS 1 BEGINS WRITING DATA F/ THE PIPE TO FILE', 'CHILD PROCESS 2 BEGINS WRITING DATA F/ THE PIPE TO FILE', 'CHILD PROCESS 1 ENDS', and 'CHILD PROCESS 2 ENDS'. The prompt returns to 'matmanbj@matmanbj-VirtualBox: ~/lab\_7\$'.

Рисунок 15. Запуск программы с чтением исходного файла с 8 символами

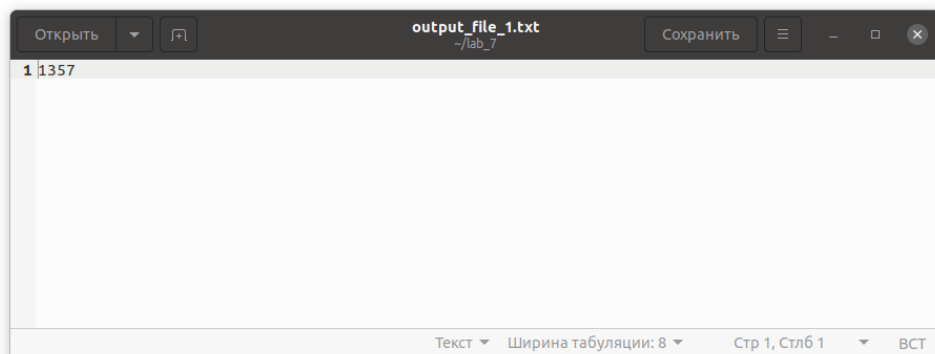


Рисунок 16. Созданный (записываемый) файл для нечётных символов при чтении файла с 8 символами

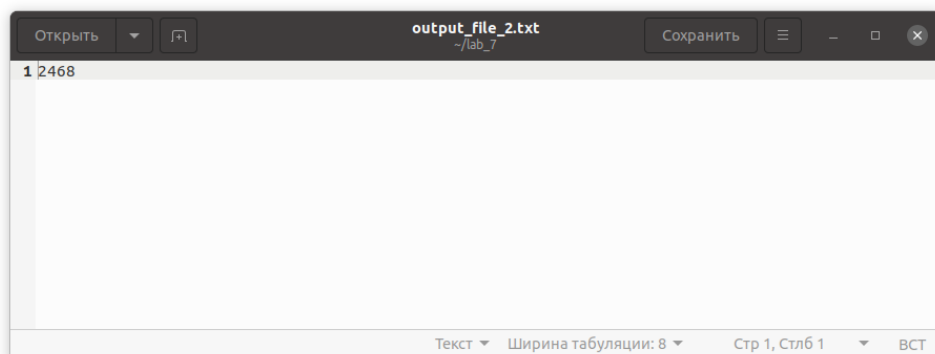
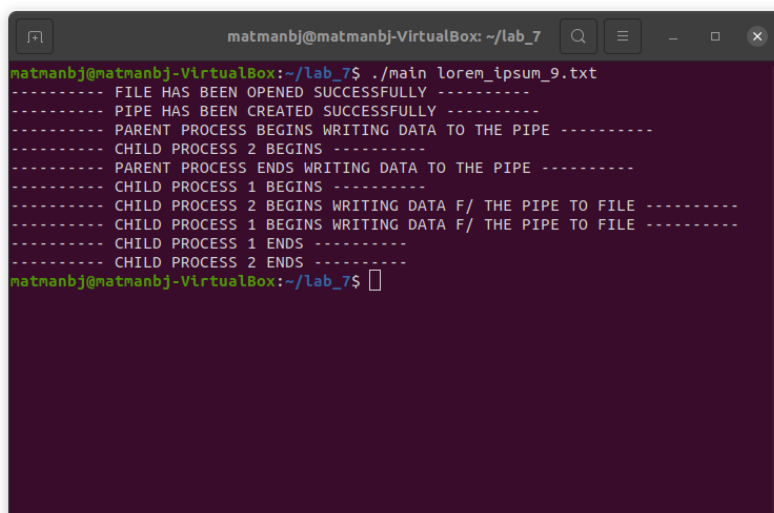


Рисунок 17. Созданный (записываемый) файл для чётных символов при чтении файла с 8 символами



```
matmanbj@matmanbj-VirtualBox: ~/lab_7
matmanbj@matmanbj-VirtualBox:~/lab_7$ ./main lorem_ipsum_9.txt
----- FILE HAS BEEN OPENED SUCCESSFULLY -----
----- PIPE HAS BEEN CREATED SUCCESSFULLY -----
----- PARENT PROCESS BEGINS WRITING DATA TO THE PIPE -----
----- CHILD PROCESS 2 BEGINS -----
----- PARENT PROCESS ENDS WRITING DATA TO THE PIPE -----
----- CHILD PROCESS 1 BEGINS -----
----- CHILD PROCESS 2 BEGINS WRITING DATA F/ THE PIPE TO FILE -----
----- CHILD PROCESS 1 BEGINS WRITING DATA F/ THE PIPE TO FILE -----
----- CHILD PROCESS 1 ENDS -----
----- CHILD PROCESS 2 ENDS -----
matmanbj@matmanbj-VirtualBox:~/lab_7$
```

Рисунок 18. Запуск программы с чтением исходного файла с 9 символами

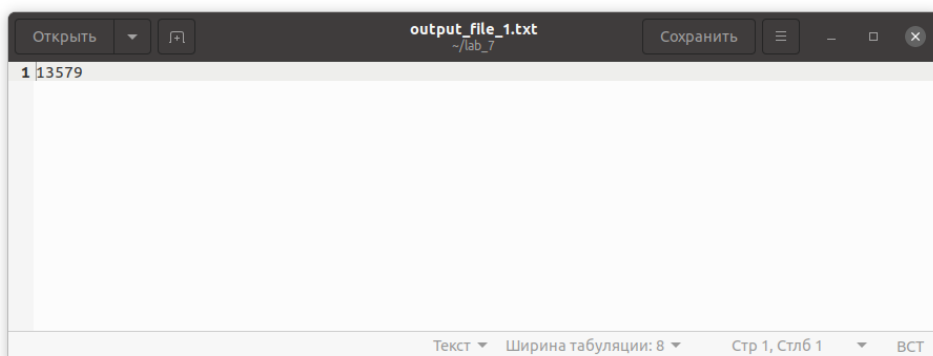


Рисунок 19. Созданный (записываемый) файл для нечётных символов при чтении файла с 9 символами

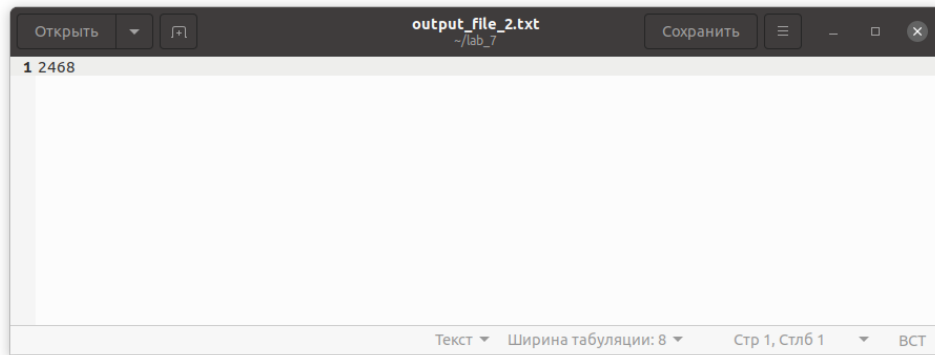


Рисунок 20. Созданный (записываемый) файл для чётных символов при чтении файла с 9 символами

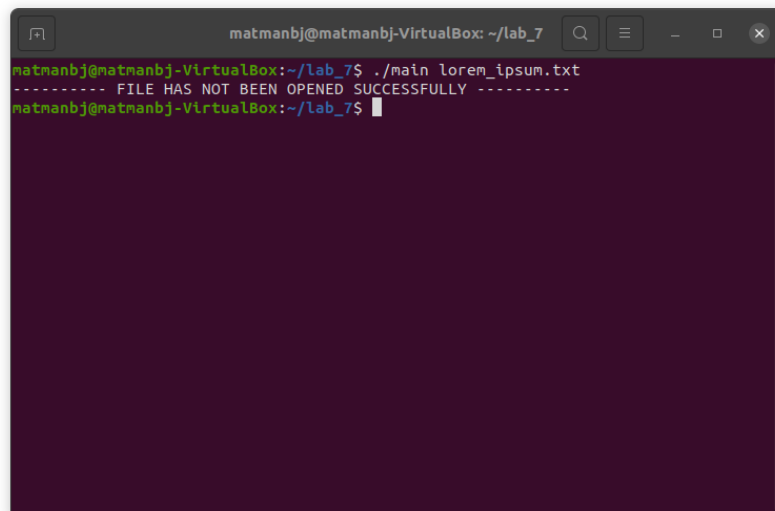


Рисунок 21. Запуск программы с чтением несуществующего файла (программа завершится не аварийно, а корректно, и укажет на отсутствие файла)

## 4. Вывод

В ходе выполнения лабораторной работы №7 «Обмен данными через канал» были изучены системные функции, отвечающие за создание канала (pipe) и за его управление. Было произведено считывание информации из файла, её загрузка в канал (pipe), далее были созданы процессы-потомки, которые считывали информацию из канала и выводили её в файл. Считывание производилось посимвольно, нечётные символы обрабатывались первым потомком, а чётные – вторым. Во время работы процессы синхронизировались специальными сигналами, а по окончании работы канал (pipe) закрывался. Таким образом и было произведено знакомство с механизмом обмена данными через программный канал и системными вызовами, обеспечивающими такой обмен.

## **5. Список использованных источников**

1. Онлайн-курс «Организация процессов и программирование в среде Linux» в LMS Moodle [сайт]. URL: <https://vec.etu.ru/moodle/course/view.php?id=9703>.

2. Разумовский Г.В. Организация процессов и программирование в среде Linux: учебно-методическое пособие. СПб.: Изд-во СПбГЭТУ «ЛЭТИ», 2018. 40с.