

Московский Авиационный Институт  
(Национальный Исследовательский Университет)

Факультет информационных технологий и прикладной математики  
Кафедра вычислительной математики и программирования

**Лабораторная работа №2 по курсу  
«Операционные системы»**

**ДИНАМИЧЕСКИЕ БИБЛИОТЕКИ**

Студент: Злобина Валерия Вадимовна

Группа: М8О–208Б–21

Вариант: 20

Преподаватель: Соколов Андрей Алексеевич

Оценка: \_\_\_\_\_

Дата: \_\_\_\_\_

Подпись: \_\_\_\_\_

Москва, 2022.

## Постановка задачи

### Цель работы

Приобретение практических навыков в:

- Управление процессами в ОС
- Обеспечение обмена данных между процессами посредством каналов

### 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.

Вариант 20

Правило фильтрации: строки длины больше 10 символов отправляются в pipe2, иначе в pipe1. Дочерние процессы инвертируют строки.

### Общие сведения о программе

Программа компилируется из файла main.c. Также используется файл: child.c.

В программе используются следующие системные вызовы:

1. **open(const char \*pathname, int flags, mode\_t mode)** – используется, чтобы преобразовать путь к файлу в описатель файла (небольшое неотрицательно целое число, которое используется с вызовами **read**, **write** и т.п. при последующем вводе-выводе).

Параметры open():

1. O\_RDONLY: открыть только для чтения
2. O\_WRONLY: только запись открыта
3. O\_RDWR: чтение, запись открытая
4. O\_CREAT: Если файл не существует, создать файл
5. O\_APPEND: дополнительная запись

2. **pipe()** – создает однонаправленный канал данных, который может использоваться для межпроцессного взаимодействия. Массив pipefd используется для возврата двух дескрипторов файлов, относящихся к

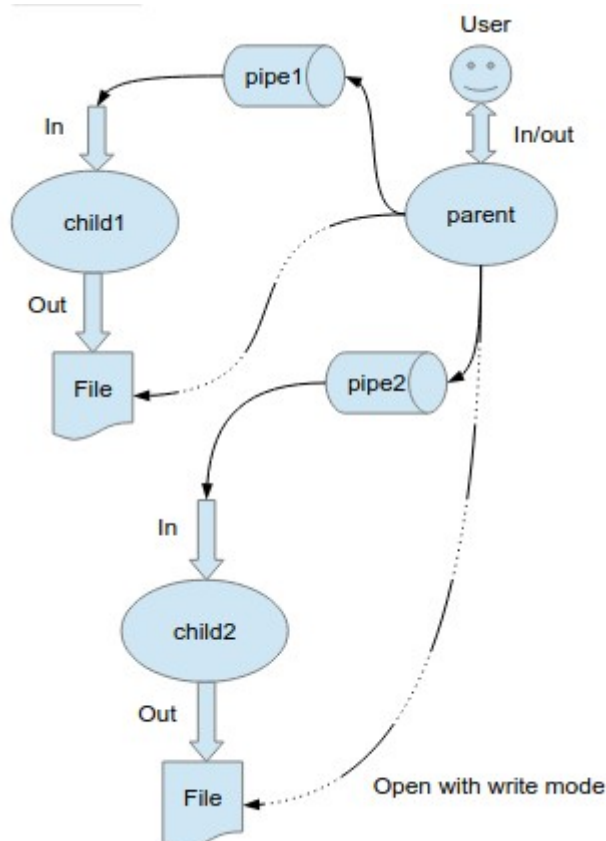
концам pipe. `pipefd[0]` относится к концу трубы для чтения. `pipefd[1]` относится к концу pipe для записи.

3. **read()** - пытается записать *count* байтов файлового описателя *fd* в буфер, адрес которого начинается с *buf*. Если количество *count* равно нулю, то **read()** возвращает это нулевое значение и завершает свою работу.
4. **write** - записывает до *count* байтов из буфера *buf* в файл, на который ссылается файловый описатель *fd*. POSIX указывает на то, что вызов **write()**, произошедший после вызова **read()** возвращает уже новое значение.

### Общий метод и алгоритм решения.

Для реализации поставленной задачи необходимо:

1. Изучить принципы работы `open`, `close`, `pipe`, `read`, `write`.
2. Написать программу, создающую процесс, который в последствии создаст и запустит дочерний процесс.
3. Написать программу `child.c` процесса инвертирования строк.



## Основные файлы программы

### main.c

```
#include <fcntl.h>
#include <sys/stat.h>
#include <sys/types.h>
#include <unistd.h>
#include <stdlib.h>
#include <stdio.h>
#include <string.h>
#include <signal.h>

#define BUFFER_SIZE 128

char* const child_args[] = {"child", NULL};
char* child_env[] = { NULL };

char* read_string(pid_t fd) {
    char* buffer = calloc(sizeof(char), BUFFER_SIZE);
    if (read(fd, buffer, BUFFER_SIZE) <= 0) {
        return NULL;
    }
    char *string = calloc(sizeof(char), strlen(buffer) - 1);
    strncpy(string, buffer, strlen(buffer) - 1);
    free(buffer);
    return string;
}

int main() {
    char* filename1 = read_string(0);
    char* filename2 = read_string(0);

    pid_t fd1[2], fd2[2];
    pipe(fd1);
    pipe(fd2);

    pid_t filed1, filed2;

    if ((filed1 = open(filename1, O_WRONLY)) < 0) {
        perror(filename1);
        exit(1);
    }

    if ((filed2 = open(filename2, O_WRONLY)) < 0) {
        perror(filename2);
        exit(2);
    }

    pid_t P1 = fork();
```

```

if (P1 < 0) {
    perror("fork");
    exit(3);
}

else if (P1 != 0) {
    pid_t P2 = fork();
    if (P2 < 0) {
        perror("fork");
        exit(4);
    }
    else if (P2 != 0) {
        char* string = read_string(0);
        if (string == NULL) {
            kill(P1, SIGKILL);
            kill(P2, SIGKILL);
            return 0;
        }
        while (string != NULL) {
            int n = strlen(string);
            if (n > 10) {
                write(fd2[1], string, n);
            }
            else {
                write(fd1[1], string, n);
            }
            string = read_string(0);
        }
        else {
            if (dup2(fd2[0], 0) < 0) {
                perror("Redirection error");
                exit(9);
            }
            if (dup2(filedes2, 1) < 0) {
                perror("Redirection error");
                exit(10);
            }
            execve(child_args[0], child_args, child_env);
            perror("execve");
            exit(11);
        }
    }

    else {
        if (dup2(fd1[0], 0) < 0) {
            perror("Redirection error");
            exit(6);
        }
        if (dup2(filedes1, 1) < 0) {

```

```

perror("Redirection error");
exit(7);
}
execve(child_args[0], child_args, child_env);
perror("execve");
exit(8);
}
}

```

## **child.c**

```

#include <sys/types.h>

#include <unistd.h>
#include <stdlib.h>
#include <stdio.h>
#include <string.h>

#define BUFFER_SIZE 128

char* read_string(pid_t fd) {
    char* buffer = calloc(sizeof(char), BUFFER_SIZE);
    if (read(fd, buffer, BUFFER_SIZE) <= 0) {
        return NULL;
    }
    char *string = calloc(sizeof(char), strlen(buffer));
    strncpy(string, buffer, strlen(buffer));
    free(buffer);
    return string;
}

char* reverse(char *string, int n) {
    char* string_rev = calloc(sizeof(char), n);
    for (int i = n - 1; i >= 0; i--) {
        string_rev[n - i - 1] = string[i];
    }
    return string_rev;
}

int main() {
    while (1) {
        char* string = read_string(0);

        int n = strlen(string);
        char* string_rev = reverse(string, n);

        write(1, string_rev, n);

        char newline = '\n';
        write(1, &newline, 1);
    }
}

```

}

**test1.txt**

PC

Compik

tram pam pam

pirojok

kakayato strochka

test

**test2.txt**

John Muir

was a naturalist

someone who studies nature

and

a writer who lived

He wrote many books

and

articles about his adventures in nature

Muir fought

hard to preserve forests and mountains

especially

Yosemite

Valley and Sequoia National Park

in California

The reading

for this

section is part

of a letter he wrote

about

saving the Sequoia trees

**test3.txt**

12345678901234

1234

34567890

12345678765432345678

123456789

1234 2345 2345

123

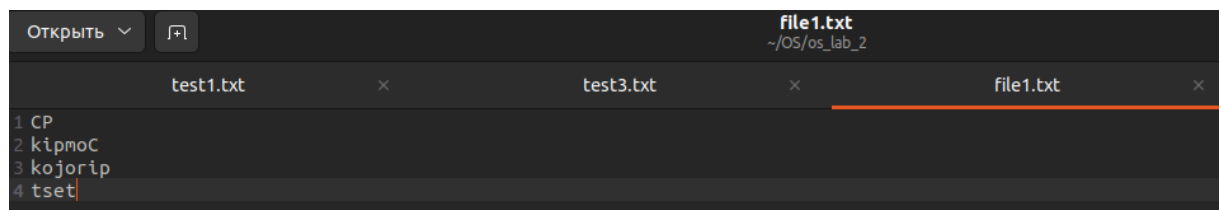
12 34

**Пример работы**

## Запуск теста 1 вручную:

```
valeria@valeria-Lenovo-ideapad-310-15IKB:~/OS/os_lab_2$ ./a.out
file1.txt
file2.txt
PC
Compik
tram pam pam
pirojok
kakayato strochka
test
valeria@valeria-Lenovo-ideapad-310-15IKB:~/OS/os_lab_2$
```

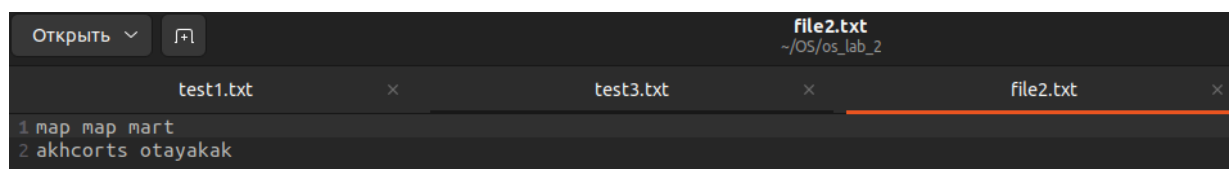
Строчки в файлах после фильтрации и инверсии:



Открыть ▾ [icon] file1.txt  
~/OS/os\_lab\_2

test1.txt × test3.txt × file1.txt ×

```
1 CP
2 k1pmoC
3 kojorip
4 tset
```



Открыть ▾ [icon] file2.txt  
~/OS/os\_lab\_2

test1.txt × test3.txt × file2.txt ×

```
1 map map mart
2 akhcorts otayakak
```

## Запуск теста 2 перенаправлением ввода:

```
valeria@valeria-Lenovo-ideapad-310-15IKB:~/OS/os_lab_2$ ./lab2 < test2.txt
valeria@valeria-Lenovo-ideapad-310-15IKB:~/OS/os_lab_2$ cat file1.txt
riuM nhoJ
dna
dna
yllaicepse
etimesoY
siht rof
tuoba
valeria@valeria-Lenovo-ideapad-310-15IKB:~/OS/os_lab_2$ cat file2.txt
tsilarutan a saw
erutan seiduts ohw enoemos
devil ohw retirw a
skoob ynam etorw eH
erutan ni serutnevda sih tuoba selcitra
thguof riuM
sniatnuom dna stserof evreserp ot drah
kraP lanoitaN aiouqeS dna yellaV
ainrofilaC ni
gnidaer eHT
trap si noitces
etorw eh rettel a fo
dna seert aiouqeS eht gnivas
valeria@valeria-Lenovo-ideapad-310-15IKB:~/OS/os_lab_2$
```



## **Вывод**

В процессе выполнения лабораторной работы, мы познакомились с механизмами низкоуровневого ввода-вывода и изучили системные вызовы `open`, `write`, `read` и `pipe`, а также познакомились с процессами, их идентификаторами и файловыми дескрипторами