Московский Авиационный Институт

(Национальный Исследовательский Университет)

Факультет информационных технологий и прикладной математики

Кафедра вычислительной математики и программирования

**Лабораторная работа №2 по курсу**

**«Операционные системы»**

**Управление процессами в ОС. Обеспечение обмена данных между процессами посредством каналов.**

Студент: Мариничев И. А.

Группа: М8O-208Б-19

Преподаватель: Миронов Е. С.

Дата: 14.12.2020

Оценка: \_\_\_\_\_\_\_\_\_\_\_

Подпись: \_\_\_\_\_\_\_\_\_\_\_

Москва, 2020

1. **Постановка задачи. Вариант 21**

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.

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 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. Родительский и дочерний процесс должны быть представлены разными программами.

Правило фильтрации: нечетные строки отправляются в pipe1, четные в pipe2. Дочерние процессы инвертируют строки.

1. **Общие сведения о программе**

Программа написана на языке Си в UNIX-подобной операционной системе (Ubuntu). В программе создается два дочерних процесса child1 и child2. Каждый дочерний процесс связан с родительским при помощи отдельного канала pipe.

Передача строки в pipe в родительском процессе вынесена в отдельную функцию.

Программа принимает на вход неограниченное количество строк произвольной длины.

Один из двух дочерних процессов выполняет инверсию данной строки и выводит её на экран. Программа для дочерних процессов запускается при помощи функции execlp().

Программа завершает свою работу при нажатии Ctrl+D.

Программа обрабатывает все возможные системные ошибки и выводит соответствующие сообщения в случае их возникновения.

1. **Общий метод и алгоритм решения**

При запуске программы пользователю предлагается ввести имя файла для первого и для второго дочернего процесса. В эти файлы будет записываться вывод соответствующих процессов. Если пользователь ввёл имя несуществующего файла, он будет создан.

После запуска программы создаются два канала fd1 и fd2, затем создаются два дочерних процесса. Родительский процесс считывает строки с консольного ввода при помощи функции GetString(). Данная функция считывает строку произвольной длины из стандартного ввода. Затем при помощи четности/нечетности счетчика counter определяется дочерний процесс, которому отправится эта строка на обработку. Счетчик изначально равен 1 (по номеру первой строки), соответственно и первая строка будет оправлена в pipe1, затем счетчик инкрементируется, поэтому следующая строка будет отпралена в pipe2, так строка с нечетным счетчиком будет передана первому дочернему процессу, в противном случае – второму.

Передача строки в дочерний процесс реализована в виде процедуры WriteToPipe, принимающая в качестве аргумента канал связи и саму строку. Строки передаются посимвольно.

Дочерние процессы закрывают ненужные каналы связи и перенаправляют свой стандартный вывод в созданный файл, а стандартный ввод – через соответствующий pipe. Затем они заменяют свой образ памяти и выполняют программу child, в которой они считывают размер строки, саму строку и выполняют её инверсию.

Инверсия строки производится в процедуре ReverseString(). Она принимает строку и выполняет ее реверс «на месте», используя технику «двух указателей». После обработки новая строка направляется в стандартный вывод.

Если пользователь нажал Ctrl+D, то родительский процесс посылает обоим дочерним процессам сигнал о завершении работы и завершается сам.

1. **Основные файлы программы**

**parent.c**

#include <stdio.h>

#include <stdlib.h>

#include <string.h>

#include <fcntl.h> // open() and O\_XXX flags

#include <unistd.h> // close(), dup2(), execl()

#include <sys/stat.h> // S\_IXXX flags

// Function that writes string with its length to pipe file descriptor(fd)

void WriteToPipe(int\* fd, char\* str) {

    int i = 0;

    char ch;

    do {

        ch = str[i++];

        if (write(fd[1], &ch, sizeof(char)) < 0) {

            perror("ERROR: failed to write to the pipe");

            exit(4);

        }

    } while (ch != '\0');

}

// Function to scan a string with unknown length

char\* GetString() {

    int length = 0, capacity = 10;

    char\* s = (char\*)malloc(10 \* sizeof(char));

    if (s == NULL) {

        perror("ERROR: unable to read string.");

        exit(6);

    }

    char ch;

    while ((ch = getchar()) != '\n') {

        s[length++] = ch;

        if (ch == EOF) {

            break;

        }

        if (length == capacity) {

            capacity \*= 2;

            s = (char\*)realloc(s, capacity \* sizeof(char));

            if (s == NULL) {

                perror("ERROR: unable to read string.");

                exit(6);

            }

        }

    };

    s[length] = '\0'; // Null-terminate the completed string

    return s;

}

int main() {

    // Creating files for child processes

    printf("Enter file's name for child process 1 (gets odd lines): ");

    char\* file1\_name = GetString();

    printf("Enter file's name for child process 2 (gets even lines): ");

    char\* file2\_name = GetString();

    // Flags for open():

    //O\_WRONLY - write only, O\_CREAT - create file if it doesn't exist,

    //S\_IWRITE, S\_IREAD - give user rights to write and read accordingly,

    int file1 = open(file1\_name, O\_WRONLY | O\_CREAT, S\_IWRITE | S\_IREAD); // System call to open file

    int file2 = open(file2\_name, O\_WRONLY | O\_CREAT, S\_IWRITE | S\_IREAD);

    if (file1 < 0 || file2 < 0) {

        perror("ERROR: unable to open file.");

        exit(1);

    }

    // Creating pipes for child processes

    int fd1[2];

    int fd2[2];

    // fd[0] - read, fd[1] - write, fd[2] - error

    if (pipe(fd1) < 0 || pipe(fd2) < 0) {

        perror("ERROR: unable to create pipe."); // pipe() returns '-1'

        exit(2);

    }

    // Creating child processes

    int ID\_1 = fork(); // System call for creating child processes

    if (ID\_1 < 0) { // fork() returns negative value if error occured

        perror("ERROR: unable to create child process.");

        exit(3);

    }

    if (ID\_1 > 0) { // fork() returns default value for parent process

        int ID\_2 = fork();

        if (ID\_2 < 0) {

            perror("ERROR: unable to create child process");

            exit(3);

        }

        if (ID\_2 > 0) { // Does everything parent process related

            // Closes a file descriptor, so that it no longer refers to any file and may be reused

            close(fd1[0]);

            close(fd2[0]);

            int counter = 1; // Counter to differ odd and even lines

            while (1) { // To stop input send EOF (for example using 'Ctrl + D' in \*nix terminal)

                char\* s = GetString();

                if (counter % 2 == 0) { // Writes to pipe1 odd lines

                    counter += 1;

                    WriteToPipe(fd2, s);

                    if (s[0] == EOF) {

                        WriteToPipe(fd2, s);

                        break;

                    }

                }

                else { // Writes to pipe2 even lines

                    counter += 1;

                    WriteToPipe(fd1, s);

                    if (s[0] == EOF) {

                        WriteToPipe(fd1, s);

                        break;

                    }

                }

            }

            close(fd1[1]);

            close(fd2[1]);

        }

        else { // Does everything child2 process related

            // Closes a file descriptor, so that it no longer refers to any file and may be reused

            close(fd1[0]);

            close(fd1[1]);

            close(fd2[1]);

            // dup2 () redirects standart input and output for child processes

            if (dup2(fd2[0], STDIN\_FILENO) < 0) {

                perror("ERROR: unable to redirect stdin for child process.");

                exit(5);

            };

            if (dup2(file2, STDOUT\_FILENO) < 0) {

                perror("ERROR: unable to redirect stdout for child process.");

                exit(5);

            }

            execlp("/mnt/c/Users/Иван/projects/os\_labs/os\_lab2/child", "child", NULL); // System call to execute  file 'child'

            // It won't go here if 'child' executes

            perror("ERROR: unable to execute child process.");

            exit(6);

        }

    }

    else { // Does everything child1 process related

        // Closes a file descriptor, so that it no longer refers to any file and may be reused

        close(fd1[1]);

        close(fd2[0]);

        close(fd2[1]);

        // dup2 () redirects standart input and output for child processes

        if (dup2(fd1[0], STDIN\_FILENO) < 0) {

            perror("ERROR: unable to redirect stdin for child process.");

            exit(5);

        }

        if (dup2(file1, STDOUT\_FILENO) < 0) {

            perror("ERROR: unable to redirect stdout for child process.");

            exit(5);

        }

        execlp("/mnt/c/Users/Иван/projects/os\_labs/os\_lab2/child", "child", NULL); // System call to execute  file 'child'

        // It won't go here if 'child' executes

        perror("ERROR: unable to execute child process.");

        exit(6);

    }

}

**child.c**

#include <stdio.h>

#include <stdlib.h>

#include <string.h>

#include <unistd.h> // close()

// Function that reverses user string

void ReverseString(char \*str) {

    int length = strlen(str);

    char \*front = str;

    char \*back = str + length - 1;

    while (front < back) {

        char tmp = \*front;

        \*front = \*back;

        \*back = tmp;

        ++front;

        --back;

    }

}

// Function to scan a string with unknown length

char\* GetString() {

    int len = 0, capacity = 10;

    char\* s = (char\*)malloc(10 \* sizeof(char));

    if (s == NULL) {

        perror("ERROR: unable to read string.");

        exit(6);

    }

    char c;

    do {

        c = getchar();

        if (c == EOF) {

            close(0);

            exit(0);

        }

        s[len++] = c;

        if (len == capacity) {

            capacity \*= 2;

            s = (char\*)realloc(s, capacity \* sizeof(char));

            if (s == NULL) {

                perror("ERROR: unable to read string.");

                exit(6);

            }

        }

    } while  (c != '\0'); // Null-terminate the completed string

    s[len] = 0;

    return s;

}

int main(int argc, char\* argv[]) {

    while (1) { // To stop input send EOF (for example using 'Ctrl + D' in \*nix terminal)

        char\* str = GetString();

        ReverseString(str);

        printf("%s\n", str);

        fflush(stdout); // makes the OS flush any buffers to the underlying file

    }

}

1. **Демонстрация работы программы**

**ivan@Laptop-IM:/mnt/c/Users/Иван/projects/os\_labs/os\_lab2$** ls

child.c example.c parent.c unistd.h

**ivan@Laptop-IM:/mnt/c/Users/Иван/projects/os\_labs/os\_lab2$** gcc child.c -o child

**ivan@Laptop-IM:/mnt/c/Users/Иван/projects/os\_labs/os\_lab2$** gcc parent.c -o parent

**ivan@Laptop-IM:/mnt/c/Users/Иван/projects/os\_labs/os\_lab2$** ./parent

Enter file's name for child process 1 (gets odd lines): test\_file1

Enter file's name for child process 2 (gets even lines): test\_file2

odd

even

odd

even

odd

even

odd

even

**ivan@Laptop-IM:/mnt/c/Users/Иван/projects/os\_labs/os\_lab2$** cat test\_file1

ddo

ddo

ddo

ddo

**ivan@Laptop-IM:/mnt/c/Users/Иван/projects/os\_labs/os\_lab2$** cat test\_file2

neve

neve

neve

neve

1. **Strace**

**ivan@Laptop-IM:/mnt/c/Users/Иван/projects/os\_labs/os\_lab2$** strace -T -i -f -e trace="%process,read,write,dup2,pipe" -o strace\_log.txt ./parent

Enter file's name for child process 1 (gets odd lines): strace\_file1

Enter file's name for child process 2 (gets even lines): strace\_file2

straceodd

straceeven

straceodd

**ivan@Laptop-IM:/mnt/c/Users/Иван/projects/os\_labs/os\_lab2$** cat strace\_log.txt

81 [00007f7ddfcb4e97] execve("./parent", ["./parent"], 0x7fffea6ff0b8 /\* 19 vars \*/) = 0 <0.002969>

81 [00007fb37c41cdd4] read(3, "\177ELF\2\1\1\3\0\0\0\0\0\0\0\0\3\0>\0\1\0\0\0\260\34\2\0\0\0\0\0"..., 832) = 832 <0.000094>

81 [00007fb37c401024] arch\_prctl(ARCH\_SET\_FS, 0x7fb37c6a14c0) = 0 <0.000010>

81 [00007fb37c110264] write(1, "Enter file's name for child proc"..., 56) = 56 <0.000085>

81 [00007fb37c110191] read(0, "strace\_file1\n", 4096) = 13 <7.269710>

81 [00007fb37c110264] write(1, "Enter file's name for child proc"..., 57) = 57 <0.000273>

81 [00007fb37c110191] read(0, "strace\_file2\n", 4096) = 12 <6.020148>

**81 [00007fb37c110b17] pipe([5, 6]) = 0 <0.000097>**

**81 [00007fb37c110b17] pipe([7, 8]) = 0 <0.000084>**

**81 [00007fb37c0e4b7c] clone(child\_stack=NULL, flags=CLONE\_CHILD\_CLEARTID|CLONE\_CHILD\_SETTID|SIGCHLD, child\_tidptr=0x7fb37c6a1790) = 82 <0.002854>**

**81 [00007fb37c0e4b7c] clone( <unfinished ...>**

**82 [00007fb37c110ab7] dup2(5, 0) = 0 <0.000069>**

**82 [00007fb37c110ab7] dup2(3, 1) = 1 <0.000057>**

**82 [00007fb37c0e4e97] execve("/mnt/c/Users/\320\230\320\262\320\260\320\275/projects/os\_labs/os\_lab2/child", ["child"], 0x7fffdee9b2e8 /\* 19 vars \*/ <unfinished ...>**

**81 [00007fb37c0e4b7c] <... clone resumed> child\_stack=NULL, flags=CLONE\_CHILD\_CLEARTID|CLONE\_CHILD\_SETTID|SIGCHLD, child\_tidptr=0x7fb37c6a1790) = 83 <0.003578>**

81 [00007fb37c110191] read(0, <unfinished ...>

**83 [00007fb37c110ab7] dup2(7, 0) = 0 <0.000036>**

**83 [00007fb37c110ab7] dup2(4, 1) = 1 <0.000035>**

**83 [00007fb37c0e4e97] execve("/mnt/c/Users/\320\230\320\262\320\260\320\275/projects/os\_labs/os\_lab2/child", ["child"], 0x7fffdee9b2e8 /\* 19 vars \*/ <unfinished ...>**

**82 [00007f8062201090] <... execve resumed> ) = 0 <0.004487>**

**83 [00007f09d7601090] <... execve resumed> ) = 0 <0.002149>**

82 [00007f806221cdd4] read(6, "\177ELF\2\1\1\3\0\0\0\0\0\0\0\0\3\0>\0\1\0\0\0\260\34\2\0\0\0\0\0"..., 832) = 832 <0.000079>

82 [00007f8062201024] arch\_prctl(ARCH\_SET\_FS, 0x7f80625e14c0) = 0 <0.000081>

83 [00007f09d761cdd4] read(5, "\177ELF\2\1\1\3\0\0\0\0\0\0\0\0\3\0>\0\1\0\0\0\260\34\2\0\0\0\0\0"..., 832) = 832 <0.000039>

82 [00007f8061f10191] read(0, <unfinished ...>

83 [00007f09d7601024] arch\_prctl(ARCH\_SET\_FS, 0x7f09d79514c0) = 0 <0.000011>

83 [00007f09d7310191] read(0, <unfinished ...>

81 [00007fb37c110191] <... read resumed> "straceodd\n", 4096) = 10 <10.332725>

81 [00007fb37c110264] write(6, "s", 1) = 1 <0.000083>

82 [00007f8061f10191] <... read resumed> "s", 4096) = 1 <10.326293>

81 [00007fb37c110264] write(6, "t", 1 <unfinished ...>

82 [00007f8061f10191] read(0, <unfinished ...>

81 [00007fb37c110264] <... write resumed> ) = 1 <0.000143>

82 [00007f8061f10191] <... read resumed> "t", 4096) = 1 <0.000090>

81 [00007fb37c110264] write(6, "r", 1 <unfinished ...>

82 [00007f8061f10191] read(0, <unfinished ...>

81 [00007fb37c110264] <... write resumed> ) = 1 <0.000181>

82 [00007f8061f10191] <... read resumed> "r", 4096) = 1 <0.000153>

81 [00007fb37c110264] write(6, "a", 1 <unfinished ...>

82 [00007f8061f10191] read(0, <unfinished ...>

81 [00007fb37c110264] <... write resumed> ) = 1 <0.000127>

82 [00007f8061f10191] <... read resumed> "a", 4096) = 1 <0.000092>

82 [00007f8061f10191] read(0, <unfinished ...>

81 [00007fb37c110264] write(6, "c", 1) = 1 <0.000115>

82 [00007f8061f10191] <... read resumed> "c", 4096) = 1 <0.000415>

81 [00007fb37c110264] write(6, "e", 1 <unfinished ...>

82 [00007f8061f10191] read(0, <unfinished ...>

81 [00007fb37c110264] <... write resumed> ) = 1 <0.000149>

82 [00007f8061f10191] <... read resumed> "e", 4096) = 1 <0.000149>

81 [00007fb37c110264] write(6, "o", 1 <unfinished ...>

82 [00007f8061f10191] read(0, <unfinished ...>

81 [00007fb37c110264] <... write resumed> ) = 1 <0.000135>

82 [00007f8061f10191] <... read resumed> "o", 4096) = 1 <0.000115>

81 [00007fb37c110264] write(6, "d", 1 <unfinished ...>

82 [00007f8061f10191] read(0, <unfinished ...>

81 [00007fb37c110264] <... write resumed> ) = 1 <0.000079>

82 [00007f8061f10191] <... read resumed> "d", 4096) = 1 <0.000184>

81 [00007fb37c110264] write(6, "d", 1 <unfinished ...>

82 [00007f8061f10191] read(0, <unfinished ...>

81 [00007fb37c110264] <... write resumed> ) = 1 <0.000122>

82 [00007f8061f10191] <... read resumed> "d", 4096) = 1 <0.000088>

81 [00007fb37c110264] write(6, "\0", 1 <unfinished ...>

82 [00007f8061f10191] read(0, <unfinished ...>

81 [00007fb37c110264] <... write resumed> ) = 1 <0.000109>

82 [00007f8061f10191] <... read resumed> "\0", 4096) = 1 <0.000061>

81 [00007fb37c110191] read(0, <unfinished ...>

82 [00007f8061f10264] write(1, "ddoecarts\n", 10) = 10 <0.000113>

82 [00007f8061f10191] read(0, <unfinished ...>

81 [00007fb37c110191] <... read resumed> "straceeven\n", 4096) = 11 <5.904481>

81 [00007fb37c110264] write(8, "s", 1) = 1 <0.000113>

83 [00007f09d7310191] <... read resumed> "s", 4096) = 1 <16.234433>

81 [00007fb37c110264] write(8, "t", 1 <unfinished ...>

83 [00007f09d7310191] read(0, <unfinished ...>

81 [00007fb37c110264] <... write resumed> ) = 1 <0.000146>

83 [00007f09d7310191] <... read resumed> "t", 4096) = 1 <0.000119>

81 [00007fb37c110264] write(8, "r", 1 <unfinished ...>

83 [00007f09d7310191] read(0, <unfinished ...>

81 [00007fb37c110264] <... write resumed> ) = 1 <0.000091>

83 [00007f09d7310191] <... read resumed> "r", 4096) = 1 <0.000126>

81 [00007fb37c110264] write(8, "a", 1 <unfinished ...>

83 [00007f09d7310191] read(0, <unfinished ...>

81 [00007fb37c110264] <... write resumed> ) = 1 <0.000188>

83 [00007f09d7310191] <... read resumed> "a", 4096) = 1 <0.000071>

81 [00007fb37c110264] write(8, "c", 1 <unfinished ...>

83 [00007f09d7310191] read(0, <unfinished ...>

81 [00007fb37c110264] <... write resumed> ) = 1 <0.000040>

83 [00007f09d7310191] <... read resumed> "c", 4096) = 1 <0.000072>

81 [00007fb37c110264] write(8, "e", 1 <unfinished ...>

83 [00007f09d7310191] read(0, <unfinished ...>

81 [00007fb37c110264] <... write resumed> ) = 1 <0.000055>

83 [00007f09d7310191] <... read resumed> "e", 4096) = 1 <0.000036>

81 [00007fb37c110264] write(8, "e", 1 <unfinished ...>

83 [00007f09d7310191] read(0, <unfinished ...>

81 [00007fb37c110264] <... write resumed> ) = 1 <0.000043>

83 [00007f09d7310191] <... read resumed> "e", 4096) = 1 <0.000043>

81 [00007fb37c110264] write(8, "v", 1) = 1 <0.000014>

83 [00007f09d7310191] read(0, <unfinished ...>

81 [00007fb37c110264] write(8, "e", 1 <unfinished ...>

83 [00007f09d7310191] <... read resumed> "v", 4096) = 1 <0.000147>

81 [00007fb37c110264] <... write resumed> ) = 1 <0.000089>

83 [00007f09d7310191] read(0, <unfinished ...>

81 [00007fb37c110264] write(8, "n", 1 <unfinished ...>

83 [00007f09d7310191] <... read resumed> "e", 4096) = 1 <0.000091>

81 [00007fb37c110264] <... write resumed> ) = 1 <0.000148>

83 [00007f09d7310191] read(0, <unfinished ...>

81 [00007fb37c110264] write(8, "\0", 1 <unfinished ...>

83 [00007f09d7310191] <... read resumed> "n", 4096) = 1 <0.000154>

81 [00007fb37c110264] <... write resumed> ) = 1 <0.000073>

83 [00007f09d7310191] read(0, <unfinished ...>

81 [00007fb37c110191] read(0, <unfinished ...>

83 [00007f09d7310191] <... read resumed> "\0", 4096) = 1 <0.000097>

83 [00007f09d7310264] write(1, "neveecarts\n", 11) = 11 <0.000428>

83 [00007f09d7310191] read(0, <unfinished ...>

81 [00007fb37c110191] <... read resumed> "straceodd\n", 4096) = 10 <4.846124>

81 [00007fb37c110264] write(6, "s", 1) = 1 <0.000077>

82 [00007f8061f10191] <... read resumed> "s", 4096) = 1 <10.754823>

81 [00007fb37c110264] write(6, "t", 1 <unfinished ...>

82 [00007f8061f10191] read(0, <unfinished ...>

81 [00007fb37c110264] <... write resumed> ) = 1 <0.000136>

82 [00007f8061f10191] <... read resumed> "t", 4096) = 1 <0.000093>

81 [00007fb37c110264] write(6, "r", 1 <unfinished ...>

82 [00007f8061f10191] read(0, <unfinished ...>

81 [00007fb37c110264] <... write resumed> ) = 1 <0.000076>

82 [00007f8061f10191] <... read resumed> "r", 4096) = 1 <0.000148>

81 [00007fb37c110264] write(6, "a", 1 <unfinished ...>

82 [00007f8061f10191] read(0, <unfinished ...>

81 [00007fb37c110264] <... write resumed> ) = 1 <0.000031>

82 [00007f8061f10191] <... read resumed> "a", 4096) = 1 <0.000041>

81 [00007fb37c110264] write(6, "c", 1 <unfinished ...>

82 [00007f8061f10191] read(0, <unfinished ...>

81 [00007fb37c110264] <... write resumed> ) = 1 <0.000039>

82 [00007f8061f10191] <... read resumed> "c", 4096) = 1 <0.000042>

81 [00007fb37c110264] write(6, "e", 1 <unfinished ...>

82 [00007f8061f10191] read(0, <unfinished ...>

81 [00007fb37c110264] <... write resumed> ) = 1 <0.000039>

82 [00007f8061f10191] <... read resumed> "e", 4096) = 1 <0.000037>

81 [00007fb37c110264] write(6, "o", 1 <unfinished ...>

82 [00007f8061f10191] read(0, <unfinished ...>

81 [00007fb37c110264] <... write resumed> ) = 1 <0.000037>

82 [00007f8061f10191] <... read resumed> "o", 4096) = 1 <0.000036>

81 [00007fb37c110264] write(6, "d", 1 <unfinished ...>

82 [00007f8061f10191] read(0, <unfinished ...>

81 [00007fb37c110264] <... write resumed> ) = 1 <0.000048>

82 [00007f8061f10191] <... read resumed> "d", 4096) = 1 <0.000149>

81 [00007fb37c110264] write(6, "d", 1 <unfinished ...>

82 [00007f8061f10191] read(0, <unfinished ...>

81 [00007fb37c110264] <... write resumed> ) = 1 <0.000153>

82 [00007f8061f10191] <... read resumed> "d", 4096) = 1 <0.000061>

81 [00007fb37c110264] write(6, "\0", 1 <unfinished ...>

82 [00007f8061f10191] read(0, <unfinished ...>

81 [00007fb37c110264] <... write resumed> ) = 1 <0.000040>

82 [00007f8061f10191] <... read resumed> "\0", 4096) = 1 <0.000042>

81 [00007fb37c110191] read(0, <unfinished ...>

82 [00007f8061f10264] write(1, "ddoecarts\n", 10) = 10 <0.000086>

82 [00007f8061f10191] read(0, <unfinished ...>

81 [00007fb37c110191] <... read resumed> "", 4096) = 0 <1.271390>

81 [00007fb37c110264] write(8, "\377", 1) = 1 <0.000117>

83 [00007f09d7310191] <... read resumed> "\377", 4096) = 1 <6.119869>

81 [00007fb37c110264] write(8, "\0", 1) = 1 <0.000054>

81 [00007fb37c110264] write(8, "\377", 1 <unfinished ...>

83 [00007f09d72e4e66] exit\_group(0 <unfinished ...>

81 [00007fb37c110264] <... write resumed> ) = 1 <0.000126>

83 [00007f09d72e4e66] <... exit\_group resumed>) = ?

81 [00007fb37c110264] write(8, "\0", 1 <unfinished ...>

83 [????????????????] +++ exited with 0 +++

81 [00007fb37c110264] <... write resumed> ) = -1 EPIPE (Broken pipe) <0.000066>

81 [00007fb37c110264] --- SIGPIPE {si\_signo=SIGPIPE, si\_code=SI\_USER, si\_pid=81, si\_uid=1000} ---

82 [00007f8061f10191] <... read resumed> "", 4096) = 0 <1.273573>

81 [????????????????] +++ killed by SIGPIPE +++

82 [00007f8061ee4e66] exit\_group(0) = ?

82 [????????????????] +++ exited with 0 +++

1. **Выводы**

В ходе данной лабораторной работы я получил один из ключевых навыков данного курса, а именно управление процессами. Обычно ОС сама создаёт необходимые для себя и для других программ процессы, но возникают ситуации, когда пользователю требуется вмешаться в работу системы.

Благодаря многопроцессорности внутри программы на языке Си можно создать дополнительный, т. н. дочерний процесс, который продолжит выполнение текущей программы параллельно с родительским процессом. Для этого используется функция fork(), совершающая соответствующий системный вызов. Удобство в том, что при помощи ветвлений в коде программы можно отделить код родительского процесса от кода, предназначенного для ребёнка. А можно заставить ребёнка запустить другую программу. Для этого предназначено семейство функций exec(). Обеспечить связь между процессами можно при помощи канала pipe, запрос на создание которого можно также совершить в языке Си.

Однако не только язык Си способен совершать системные вызовы, связанные с управлением процессами. Похожие библиотеки есть на многих других языках программирования, что говорит о важности и необходимости данного функционала.