МИНИСТЕРСТВО НАУКИ
И ВЫСШЕГО ОБРАЗОВАНИЯ
РОССИЙСКОЙ ФЕДЕРАЦИИ

Федеральное государственное бюджетное
образовательное учреждение высшего образования
«НОВОСИБИРСКИЙ ГОСУДАРСТВЕННЫЙ ТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ»

НГТУ
НЭТИ | Факультет прикладной
математики и информатики

Кафедра прикладной математики

Лабораторная работа № 4

по дисциплине «Управление ресурсами в вычислительных системах»

МОДЕЛИРОВАНИЕ РАБОТЫ ИНТЕРПРЕТАТОРА

Бригада 7	ГРУШЕВ АНДРЕЙ
Группа ПМ-05	БОЛДЫРЕВ СЕРГЕЙ
Вариант 7	ХАБАРОВА АНАСТАСИЯ

Преподаватели	СТАСЫШИН ВЛАДИМИР МИХАЙЛОВИЧ
	СИВАК МАРИЯ АЛЕКСЕЕВНА

Новосибирск, 2023

Условие

Составить программу, моделирующую работу Shell-интерпретатора при обработке командной строки, указанной в варианте. При реализации программы путем выдачи сообщений информировать обо всех этапах ее работы (создан процесс, выполнение команды закончено и т.д.).
Вариант задания: `cat a.txt b.txt c.txt | tr -d "[0-9]" | wc -w.`

Используемые программные средства

`pipe(fd)` – системный вызов, возвращающий два дескриптора файла: для записи данных в канал и для чтения.

`fork(void)` – системный вызов, порождающий новый дочерний процесс.

`close(int fd)` – закрывает файловый дескриптор, все блокировки, находящиеся на соответствующем файле, снимаются.

`dup()` – системный вызов, который обрабатывает свой единственный параметр как пользовательский дескриптор открытого файла. Возвращает целое число, которое может быть использовано как пользовательский дескриптор того же файла.

`execl(const char *path, const char *arg, ...)` – функция, заменяющая текущий образ процесса новым образом процесса. Функция дублирует действия оболочки, относящиеся к поиску исполняемого файла.

`wait(int *status)` – системный вызов, с помощью которого выполняется ожидание завершения процесса-потомка родительским процессом.

`exit(int status)` – системный вызов, предназначенный для завершения функционирования процесса. Аргумент `status` является статусом завершения, который передается родительскому процессу, если он выполнял системный вызов `wait`.

`fprintf(FILE* stream, const char *format, …)` – форматированный вывод в файл на который указывает `stream`.

`printf(const char* format, …)` – форматированный вывод в файл стандартного вывода.

`open(const char *pathname, int flags)` – преобразовывает путь к файлу в описатель файла. Возвращает файловый описатель, который не открыт процессом. Функция с флагом `O_TRUNC` урезает длину файла до нуля, если файл существует, является обычным файлом и режим позволяет запись в этот файл.

`cat` - объединяет файлы и направляет их на стандартный вывод.

`tr` - выполняет символьное преобразование путём подстановки или удаления символов. Опция **`-d`** удаляет все символы, заданные в наборе символов, без преобразования.

`wc` - печатает число строк, слов и байт в файлах. Опция **`-w`** печатает количество слов в заданном файле.

Алгоритм решения

1. Исходный процесс создает программный канал и порождает новый процесс.
2. Дочерний процесс создает второй программный канал и порождает новый процесс.
3. Внучатый процесс помещает подготовленные данные в канал и завершается.
4. Дочерний процесс возобновляется после получения сигнала, считывает данные из канала, помещает подготовленные данные в канал и завершается.
5. Родительский процесс возобновляется после получения сигнала, считывает данные из канала, помещает подготовленные данные в канал и завершается.

Спецификация

Код программы расположен на сервере НГТУ в директории **/home/NSTU/pmi-b0507/upres/lab4**.

Для корректной работы программы необходимо, чтобы в папке с исполняемым файлом находились файлы с наименованием **a.txt, b.txt, c.txt**.
Для получения основного исполняемого файла необходимо, находясь в директории с файлом **main.c,** выполнить команду:
gcc main.c -o [имя_исполняемого_файла],
либо воспользоваться make-файлом при помощи команды:
make main,
которая создаст исполняемый файл **main.o**.

Также можно воспользоваться командой **make all,**
которая создаст исполняемый файл **main.o**.
Запуск программы происходит при помощи команды:

./[имя_исполняемого_файла],
например:
./main.o

Формат вывода результата:

[служебное сообщение]
…
[служебное сообщение]
[результат выполнения команды]

Тестирование программы:

№	Входные данные	Результаты работы программы	Результат работы Shell-интерпретатора
1	./main.o a.txt: testing b.txt: th1s c.txt: program now	P0 is created Pipe(fd1) is created P1 is created Pipe(fd2) is created P2 is created P2 is finished cat executed successfully P1 is finished tr executed successfully 3	cat a.txt b.txt c.txt | tr -d [0-9] | wc -w 3
2	./main.o a.txt: testing b.txt: 565643 c.txt: program now	P0 is created Pipe(fd1) is created P1 is created Pipe(fd2) is created P2 is created P2 is finished cat executed successfully P1 is finished tr executed successfully 2	cat a.txt b.txt c.txt | tr -d [0-9] | wc -w 2
3	./main.o a.txt: 123 b.txt: 565643 c.txt: 447	P0 is created Pipe(fd1) is created P1 is created Pipe(fd2) is created P2 is created P2 is finished cat executed successfully P1 is finished tr executed successfully 0	cat a.txt b.txt c.txt | tr -d [0-9] | wc -w 0

4	./main.o a.txt: testing b.txt: c.txt: 447	P0 is created Pipe(fd1) is created P1 is created Pipe(fd2) is created P2 is created P2 is finished cat executed successfully P1 is finished tr executed successfully 1	cat a.txt b.txt c.txt | tr -d [0-9] | wc -w 1
5	./main.o a.txt: b.txt: c.txt:	P0 is created Pipe(fd1) is created P1 is created Pipe(fd2) is created P2 is created P2 is finished cat executed successfully P1 is finished tr executed successfully 0	cat a.txt b.txt c.txt | tr -d [0-9] | wc -w 0
6	./main.o a.txt: testing b.txt не существует c.txt: now	P0 is created Pipe(fd1) is created P1 is created Pipe(fd2) is created P2 is created cat: b.txt: No such file or directory P2 is finished Error! Cannot execute cat P1 is finished Error! Cannot execute tr	cat a.txt b.txt c.txt | tr -d [0-9] | wc -w cat: b.txt: No such file or directory 1

Make-файлы

Файл **makefile:**

```
main: main.c
    gcc main.c -o main.o

all: main.c
    gcc main.c -o main.o

clean:
    rm main.o
```

Листинг программы

main.c:

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

/* cat a.txt b.txt c.txt | tr -d [0-9] | wc -w */
```

```
int main(int argc, char** argv)
{
    // P0 выполняет "wc -w"
    printf("P0 is created\n");
    int fd1[2], status;
    if (pipe(fd1) == -1) //Канал для общения между P0 и P1
    {
        printf("Cannot create pipe(fd1)\n");
        exit(EXIT_FAILURE);
    }
    else
        printf("Pipe(fd1) is created\n");
    if (fork() == 0)
    {
        // P1 выполняет "tr -d [0-9]"

        printf("P1 is created\n");

        int fd2[2];
        if (pipe(fd2) == -1) //Канал для общения между P1 и P2
        {
            printf("Cannot create pipe(fd2)\n");
            exit(EXIT_FAILURE);
        }
        else
            printf("Pipe(fd2) is created\n");
        if (fork() == 0)
        {
            // P2 выполняет "cat a.txt b.txt c.txt"

            printf("P2 is created\n");

            close(fd1[0]); //
            close(fd1[1]); // Закрытие не используемых декрипторов
            close(fd2[0]); //

            close(1); // Закрытие стандартного вывода
            dup(fd2[1]); // Переназначение стандартного вывода дескриптором записи
канала fd2
            close(fd2[1]); // Закрытие старого дескриптора записи канала fd2

            if (execl("/bin/cat", "cat", "a.txt", "b.txt", "c.txt", NULL) == -1)
                exit(EXIT_FAILURE);
            exit(EXIT_SUCCESS);
        }

        wait(&status);
        printf("P2 is finished\n");
        if (status != EXIT_SUCCESS)
        {
            fprintf(stderr, "Error! Cannot execute cat\n");
```

```
            exit(EXIT_FAILURE);
        }
        else
            printf("cat executed successfully\n");

        close(fd1[0]); //
        close(fd2[1]); // Закрытие не используемых декрипторов

        close(1); // Закрытие стандартного вывода
        dup(fd1[1]); // Переназначение стандартного вывода дескриптором записи ка-
нала fd1
        close(fd1[1]); // Закрытие старого дескриптора записи канала fd1

        close(0); // Закрытие стандартного ввода
        dup(fd2[0]); // Переназначение стандартного ввода дескриптором чтения ка-
нала fd2
        close(fd2[0]); // Закрытие старого дескриптора чтения канала fd2

        if (execl("/bin/tr", "tr", "-d", "[0-9]", NULL) == -1)
            exit(EXIT_FAILURE);
        exit(EXIT_SUCCESS);
    }

    wait(&status);

    printf("P1 is finished\n");

    if (status != EXIT_SUCCESS)
    {
        fprintf(stderr, "Error! Cannot execute tr\n");
        exit(EXIT_FAILURE);
    }
    else
            printf("tr executed successfully\n");

    close(fd1[1]); // Закрытие неиспользуемого дескриптора

    close(0); // Закрытие стандартного ввода
    dup(fd1[0]); // Переназначение стандартного ввода дескриптором чтения канала
fd1
    close(fd1[0]); // Закрытие старого дескриптора чтения канала fd1

    if (execl("/bin/wc", "wc", "-w", NULL) == -1)
    {
        fprintf(stderr, "Error! Cannot execute wc\n");
        exit(EXIT_FAILURE);
    }
    return 0;
}
```