Московский Авиационный Институт

(Национальный Исследовательский Университет)

Факультет информационных технологий и прикладной математики

Кафедра вычислительной математики и программирования

**Лабораторная работа №5 по курсу**

**«Операционные системы»**

**ДИНАМИЧЕСКИЕ БИБЛИОТЕКИ**

Студент: Вержиковская Анастасия Игоревна

Группа: М8О–206Б–20

Вариант: 16

Преподаватель: Соколов Андрей Алексеевич

Оценка: \_\_\_\_\_\_\_\_\_\_

Дата: \_\_\_\_\_\_\_\_\_\_

Подпись: \_\_\_\_\_\_\_\_\_\_

Москва, 2020.

**Постановка задачи**

## Цель работы

Целью является приобретение практических навыков в:

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

## Задание

Родительский процесс создает дочерний процесс. Первой строкой пользователь в консоль родительского процесса вводит имя файла, которое будет использовано для открытия File с таким именем на запись. Перенаправление стандартных потоков ввода-вывода показано на картинке выше. Родительский и дочерний процесс должны быть представлены разными программами. Родительский процесс принимает от пользователя строки произвольной длины и пересылает их в pipe1. Процесс child проверяет строки на валидность правилу. Если строка соответствует правилу, то она выводится в стандартный поток вывода дочернего процесса, иначе в pipe2 выводится информация об ошибке. Родительский процесс полученные от child ошибки выводит в стандартный поток вывода.

**Общие сведения о программе**

Программа компилируется из файла main.c. Также используется заголовочные файлы: stdio.h, stdbool.h, stdlib.h, string.h, unistd.h, errorlib.h. В программе используются следующие системные вызовы:

1. **pid\_t fork()** - создание дочернего процесса, возвращает -1 при ошибке создания дочернего процесса, 0 если процесс является дочерним, и pid если процесс является родительским.
2. **int execlp(const char \*file, const char \*arg, ...)** – заменяет текущий образ процесса новым образом процесса, c аргументами arg.
3. **pid\_t waitpid(pid\_t pid, int \*status, int options)** - Ожидание завершения дочернего процесса
4. **void exit(int status)** - завершения выполнения процесса и возвращение статуса
5. **int pipe(int pipefd[2])** - создание неименованного канала для передачи данных между процессами
6. **int dup2(int oldfd, int newfd)** - переназначение файлового дескриптора
7. **int close(int fd)** - закрыть файл

**Общий метод и алгоритм решения**.

Для реализации поставленной задачи необходимо:

1. Изучить принципы работы fork, pipe, execlp, dup2, waitpid.
2. Написать родительскую программу prog.c для ввода названия файла и строчек
3. Написать дочернюю программу prog\_child.c, которае будет принимать от родительского файла строки и проверять их на валидность правилу
4. Соединить prog.c, prog\_child.c и текстовый файл каналами pipe1 и pipe2.

**Основные файлы программы**

[Исходники; не рекомендуется использовать большой междустрочный интервал и подсветку синтаксиса]

**prog.c :**

#include <stdlib.h>

#include <unistd.h>

#include <fcntl.h>

int main(){

write(1, "enter output file name\n", 24);

char\* out\_file = (char\*)malloc(100 \* sizeof(char)); //путь на название файла

int nleng = read(0, out\_file, 100); //считали имя файла

out\_file[nleng - 1] = 0;

int fd\_out = open(out\_file, O\_WRONLY|O\_APPEND);

write(1, "print str OR print nothing and push enter to stop\n", 38);

int fd1[2], fd2[2];

pipe(fd1);

pipe(fd2);

int id = fork(); //

if(id > 0){

close(fd1[0]);

close(fd2[1]);

char symb, ans;

do{

while(read(0, &symb, 1)){

write(fd1[1], &symb, 1);

if(symb == '\n'){

break;

}

}

read(fd2[0], &ans, 1);

if(ans == '1'){

write(1, "+++invalid line+++\n", 20);

}

}

while(ans != '2');

close(fd1[1]);

close(fd2[0]);

}

else if(id == 0){

close(fd1[1]);

close(fd2[0]);

dup2(fd1[0], 0);

dup2(fd\_out, 1);

dup2(fd2[1], 2);

execl("./prog\_child.out", "prog\_child.out", NULL);

close(fd1[0]);

close(fd2[1]);

}

else{

\_exit(-1);

}

return 0;

}

**prog\_child.c :**

#include <stdlib.h>

#include <unistd.h>

#include <fcntl.h>

int main(int argc, char\* argv[]){

char str[100];//считываемая строкa

int len = 0;//длина строки

char symb, l\_symb = '\n';

while(read(0, &symb, 1)){ // считывает по одному символу из строки

str[len++] = symb;

if(symb == '\n'){

if(l\_symb == '\n'){

write(2, "2", 1);

break;

}

if(l\_symb == '.' || l\_symb == ';'){

write(2, "0", 1);

write(1, &str[0], len);

}

else{

write(2, "1", 1);

}

len = 0;

}

l\_symb = symb;

}

return 0;

}

**Пример работы**

**Ввод:**

Oh, imagine a land, it's ; a faraway place…..

**Вывод:**  
(валидную строку записывает в файл out.txt)

**Ввод:**

Where the caravan camels roam;

**Вывод:**

(валидную строку записывает в файл out.txt)

**Ввод:**

Where you; wander: among. every culture and tongue

**Вывод:**

+++invalid line+++

**Ввод:**

It's chaotic, but, hey, it's home

**Вывод:**  
+++invalid line+++

Ввод:

(enter) → программа завершилась

**Файл out.txt после завершения:**

Oh, imagine a land, it's ; a faraway place…..

Where the caravan camels roam;

**Вывод**

В результате данной лабораторной работы были изучены основные методы работы с процессами в ОС linux. В работе я научилась работать с надором системных вызовов на Linux, работать с родительскими и дочерними процессами, обеспечивать обмен данными между ними с помощью каналов pipe.