Московский Авиационный Институт

(Национальный Исследовательский Университет)

Факультет информационных технологий и прикладной математики

Кафедра вычислительной математики и программирования

**Лабораторная работа №2 по курсу**

**«Операционные системы»**

Студент: Савинова Екатерина

Группа: М8О-207Б-21

Вариант: 15

Преподаватель: Черемисинов Максим

Оценка: \_\_\_\_\_\_\_\_\_\_\_

Дата: \_\_\_\_\_\_\_\_\_\_\_

Подпись: \_\_\_\_\_\_\_\_\_\_\_

Москва, 2022

**Содержание**

1. Репозиторий
2. Постановка задачи
3. Общие сведения о программе
4. Общий метод и алгоритм решения
5. Исходный код
6. Демонстрация работы программы
7. Выводы

**Репозиторий**

<https://github.com/savinova-kati/operating-systems>

**Постановка задачи**

**Цель работы**

Приобретение практических навыков в:

 Управление процессами в ОС

 Обеспечение обмена данных между процессами посредством каналов

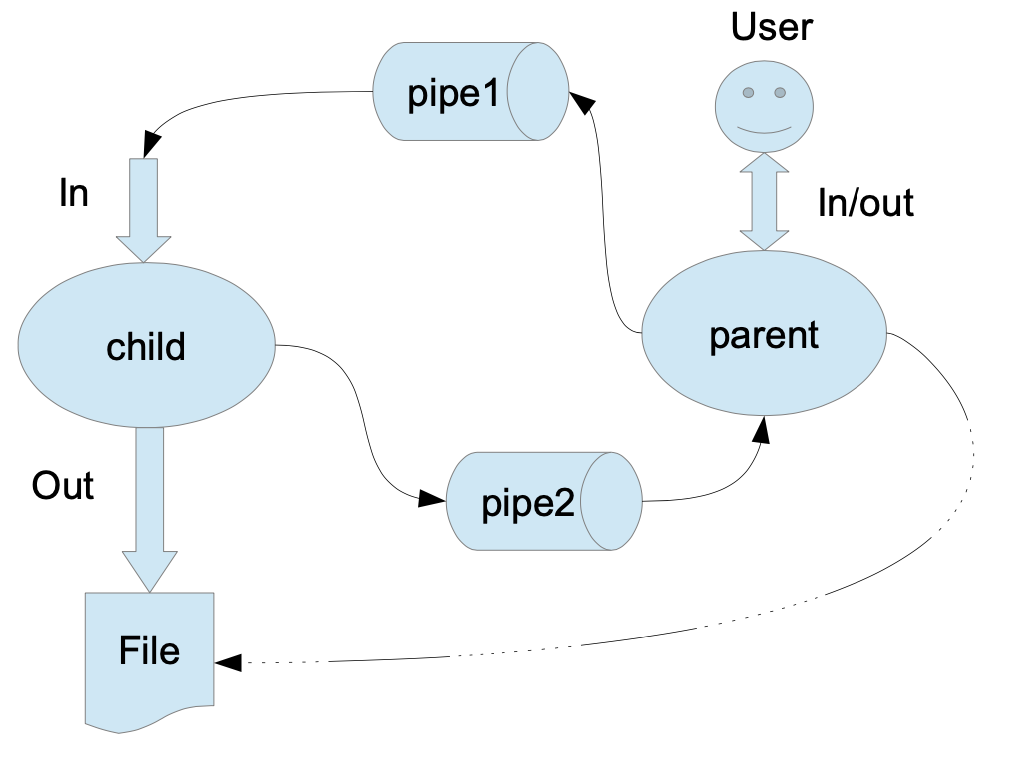
**Задание**

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.

Группа вариантов 4:

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.

Вариант 15) Правило проверки: строка должна начинаться с заглавной буквы



**Общие сведения о программе**

Программа компилируется из файла main.cpp с помощью cmake. Дочерний процесс представлен в child\_.cpp

Системные вызовы:

* pid\_t fork() - создание дочернего процесса
* int execl(const char \*filename, char \*const argv[], char \*const ) - замена образа памяти процесса
* int pipe(int pipefd[2]) - создание неименованного канала для передачи данных между процессами
* int open(const char \*pathname, int flags, mode\_t mode) - открытие\создание файла
* int close(int fd) - закрыть файл

**Общий метод и алгоритм решения**

Родительский процесс принимает от пользователя строки произвольной длины и пересылает их в pipe1. Процесс child проверяет строки. Если строка начинается с заглавной буквы, 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.

**Исходный код**

**main.cpp**

#include <iostream>

#include <string>

#include <cstdlib>

#include <sys/types.h>

#include <unistd.h>

#include <fstream>

#include <errno.h>

#include <signal.h>

#include <sys/wait.h>

using namespace std;

int main(){

fstream file\_1;

string name;

cout<<"Enter filename: "<<endl;

cin >> name;

int truba[2];

int truba\_2[2];

int pipe\_1[2];

int pipe\_2[2];

if (pipe(pipe\_1) == -1){

perror("pipe");

exit(EXIT\_FAILURE);

}

if (pipe(pipe\_2) == -1){

perror("pipe");

exit(EXIT\_FAILURE);

}

string string\_r;

pid\_t id = fork();

if (id == -1){

perror("fork");

cout << "1";

exit(EXIT\_FAILURE);

}

else if (id == 0) {

truba[0] = pipe\_1[0];

truba[1] = pipe\_1[1];

truba\_2[0] = pipe\_2[0];

truba\_2[1] = pipe\_2[1];

execl("./child\_", to\_string(truba[0]).c\_str(), to\_string(truba[1]).c\_str(), to\_string(truba\_2[0]).c\_str(), to\_string(truba\_2[1]).c\_str(), name.c\_str(), NULL);

}

else {

cout<<"Enter amount of strings: ";

int amount;

cin >> amount;

cout << endl;

for (int i = 0; i < amount; ++i) {

cin >> string\_r;

int s\_size = string\_r.size();

char str\_array[s\_size];

for (int k = 0; k < s\_size; ++k) {

str\_array[k] = string\_r[k];

}

write(pipe\_1[1], &s\_size, sizeof(int));

write(pipe\_1[1], str\_array, sizeof(char)\*s\_size);

int flag\_;

read(pipe\_2[0], &flag\_, sizeof(int));

//cout << truba\_2[0] << endl;

if (flag\_ == 1) {

cout << "The string does not fit the rule" << endl;

}

}

}

close(pipe\_1[0]);

close(pipe\_1[1]);

close(pipe\_2[0]);

close(pipe\_2[1]);

return 0;

}

**сhild\_.cpp**

#include <iostream>

#include <string>

#include <cstdlib>

#include <sys/types.h>

#include <unistd.h>

#include <fstream>

#include <errno.h>

#include <signal.h>

#include <sys/wait.h>

using namespace std;

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

string filename = argv[4];

int truba[2];

int truba\_2[2];

int flag1 = 1;

int flag2 = 22;

truba[0] = stoi(argv[0]);

truba[1] = stoi(argv[1]);

truba\_2[0] = stoi(argv[2]);

truba\_2[1] = stoi(argv[3]);

fstream file\_1;

file\_1.open(filename, fstream::in | fstream::out | fstream::app);

while(true) {

int stroka\_size;

read(truba[0], &stroka\_size, sizeof(int));

char stroka[stroka\_size];

read(truba[0], &stroka, sizeof(char)\*stroka\_size);

string result;

for (int i = 0; i < stroka\_size; i++) {

result.push\_back(stroka[i]);

}

if (stroka[0] >= 65 && stroka[0] <= 90) {

file\_1 << result << endl;

cout << "Added string " << result << " to file!" << endl;

write(truba\_2[1], &flag2, sizeof(int));

//cout << truba\_2[1] << endl;

} else {

write(truba\_2[1], &flag1, sizeof(int));

//cout << truba\_2[1] << endl;

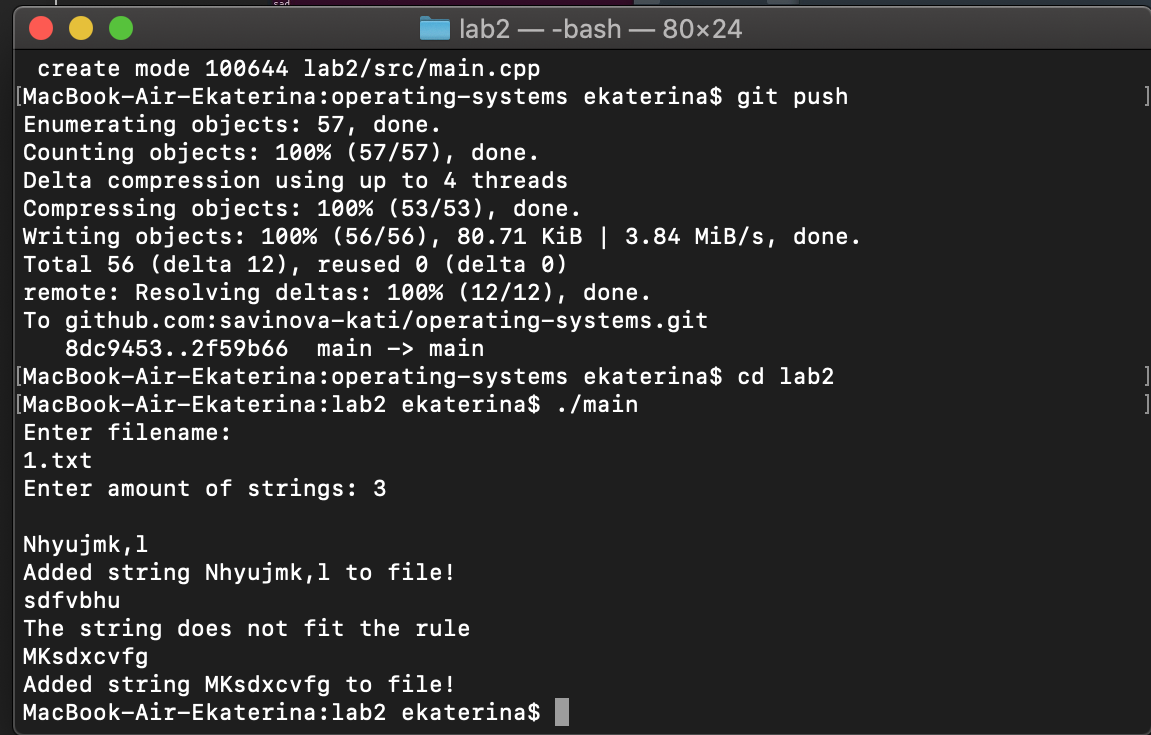
}

}

return 0;

}

**Демонстрация работы программы**



**Выводы**

В заключении хотелось бы сказать, что благодаря данной лабораторной работы я познакомилась с некоторыми системными вызовами в Unix, разобралась с понятием процесс. Я научилась создавать pipe и работать с файловым дескриптором. В целом, теперь у меня есть понимание как управлять процессами и информацией, передающейся между ними.