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: Ткаченко Егор Юрьевич

Группа: М8О-207Б-21

Вариант: 18

Преподаватель: Миронов Евгений Сергеевич

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/Tnirpps/OS\_lab](https://vk.com/away.php?to=https%3A%2F%2Fgithub.com%2FTnirpps%2FOS_lab&cc_key=)

**Постановка задачи**

**Цель работы**

Приобретение практических навыков в:

Управление процессами в ОС

Обеспечение обмена данных между процессами посредством каналов

**Задание**

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 и

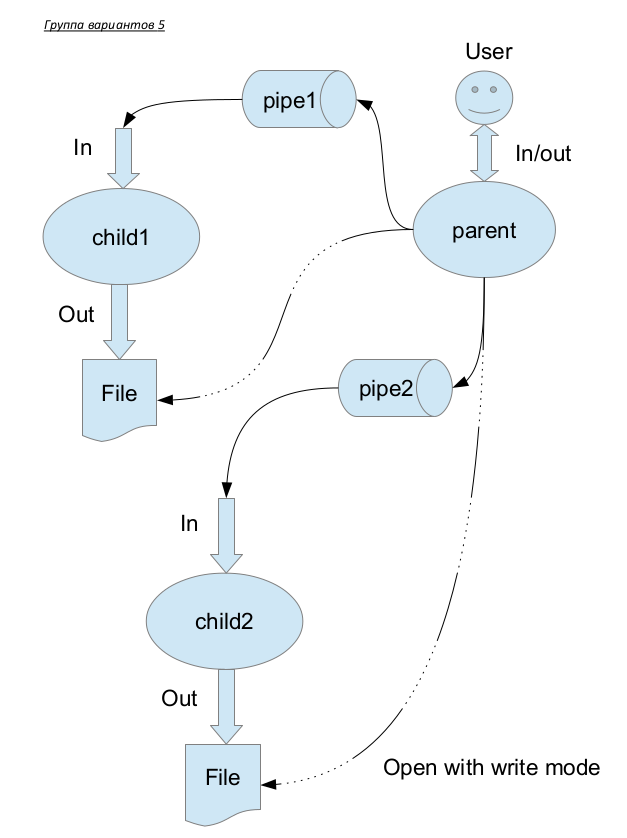
дочерний процесс должны быть представлены разными программами. Родительский процесс принимает от пользователя строки произвольной длины и пересылает их в

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.

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

Дочерние процессы удаляют все гласные из строк.



**Общие сведения о программе**

Программа компилируется из файла main.c, strlib.c. Также используется заголовочные файлы: unistd.h, wys/wait.h, stdio.h. В программе используются следующие системные вызовы:

1. pipe() - существует для передачи информации между различными процессами.
2. fork() - создает новый процесс.
3. execv() - передает процесс на исполнение другой программе.
4. read() - читает данные из файла.
5. write() - записывает данные в файл.
6. close() - закрывает файл.

**Общий метод и алгоритм решения**

Родительский процесс создаёт два дочерних, затем передаёт им данные использую pipes. Дочерние процессы выполняют фильтрацию полученных от родительского процесса данных в соответствии с требованием варианта. Работа заканчивается, когда родительский процесс закрывает каналы с дочерними процессами и их дальнейшее чтение становится невозможным.

**Исходный код**

================================= main.c ===================================

#include <stdio.h>

#include <unistd.h>

#include <sys/wait.h>

#define WRITE\_END 1

#define READ\_END 0

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

char \*Child1\_argv[] = {"child\_1", NULL};

char \*Child2\_argv[] = {"child\_2", NULL};

int fd1[2];

int fd2[2];

int p1 = pipe(fd1);

int p2 = pipe(fd2);

if (p1 == -1 || p2 == -1) {

fprintf(stderr, "%s", "Pipe() error\n");

return 1;

}

int id1 = fork();

if (id1 == -1) {

fprintf(stderr, "%s", "Fork() error\n");

return 1;

} else if (id1 == 0) {

// ================= Child 1 ================= //

if (dup2(fd1[READ\_END], STDIN\_FILENO) == -1) {

fprintf(stderr, "%s", "dup2 error\n");

return 1;

}

if (close(fd1[WRITE\_END]) == -1 ) {

fprintf(stderr, "%s", "Cannot close fd\n");

return 1;

}

if (execv("child\_1", Child1\_argv) == -1) {

fprintf(stderr, "%s", "Cannot call exec child\_1\n");

return 1;

}

} else {

int id2 = fork();

if (id2 == -1) {

fprintf(stderr, "%s", "Fork() error\n");

return 1;

} else if (id2 == 0) {

// ================= Child 2 ================= //

if (dup2(fd2[READ\_END], STDIN\_FILENO) == -1) {

fprintf(stderr, "%s", "dup2 error\n");

return 1;

}

if (close(fd2[WRITE\_END]) == -1) {

fprintf(stderr, "%s", "Cannot close fd\n");

return 1;

}

if (execv("child\_1", Child2\_argv) == -1) {

fprintf(stderr, "%s", "Cannot call exec child\_2\n");

return 1;

}

} else {

// ================= parent ================= //

if (close(fd1[READ\_END]) == -1 || close(fd2[READ\_END]) == -1) {

fprintf(stderr, "%s", "Cannot close fd\n");

return 3;

}

char c;

// write filename to child 1

while ((c = getchar()) != EOF) {

if (write(fd1[WRITE\_END], &c, sizeof (char)) == -1) {

fprintf(stderr, "%s", "Cannot write to fd1\n");

return 3;

} // check error

if (c == '\n') break;

}

// write filename to child 2

while ((c = getchar()) != EOF) {

if (write(fd2[WRITE\_END], &c, sizeof (char)) == -1) {

fprintf(stderr, "%s", "Cannot write to fd1\n");

return 3;

} // check error

if (c == '\n') break;

}

int str\_ind = 0; // even or odd line number flag

while ((c = getchar()) != EOF) {

switch (str\_ind) {

case 0:

if (write(fd1[WRITE\_END], &c, sizeof (char)) == -1) {

fprintf(stderr, "%s", "Cannot write to fd1\n");

return 3;

} // check error

if (c == '\n') {

str\_ind ^= 1;

}

break;

case 1:

if (write(fd2[WRITE\_END], &c, sizeof (char)) == -1) {

fprintf(stderr, "%s", "Cannot write to fd1\n");

return 3;

} // check error

if (c == '\n') {

str\_ind ^= 1;

}

break;

default:

fprintf(stderr, "%s", "I don't know what has happened\n");

return 1;

}

}

// we need to define the end of lines to make read easier

c = '\0';

if (write(fd1[WRITE\_END], &c, sizeof (char)) == -1) {

fprintf(stderr, "%s", "Cannot write to fd1\n");

return 3;

}

if (write(fd2[WRITE\_END], &c, sizeof (char)) == -1) {

fprintf(stderr, "%s", "Cannot write to fd1\n");

return 3;

}

if (close(fd1[WRITE\_END]) == -1) {

fprintf(stderr, "%s", "Cannot close fd\n");

return 1;

}

if (close(fd2[WRITE\_END]) == -1) {

fprintf(stderr, "%s", "Cannot close fd\n");

return 1;

}

// wait for all child process ends

waitpid(-1, NULL, 0);

}

}

return 0;

}

================================= child\_1.c ==================================

//

// Created by hplp739 on 15.09.22.

//

#include "../headers/strlib.h"

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

if (argc < 1) {

fprintf(stderr, "Arguments missing\n");

return 1;

}

char \*filename;

if (read\_line(&filename) <= 1) {

fprintf(stderr, "%s cannot read filename\n", argv[0]);

return 1;

}

FILE \*fp;

fp = fopen(filename, "w");

if (fp == NULL) {

fprintf(stderr,"%s cannot open file: %s\n", argv[0], filename);

return 1;

}

free(filename);

filename = NULL;

char \*line;

int i = 2;

while ((i = read\_line(&line)) > 1) {

for (int j = 0; j < i - 1; ++j) {

if (is\_vowel(line[j])) continue;

putc(line[j], fp);

}

putc('\n', fp);

free(line);

}

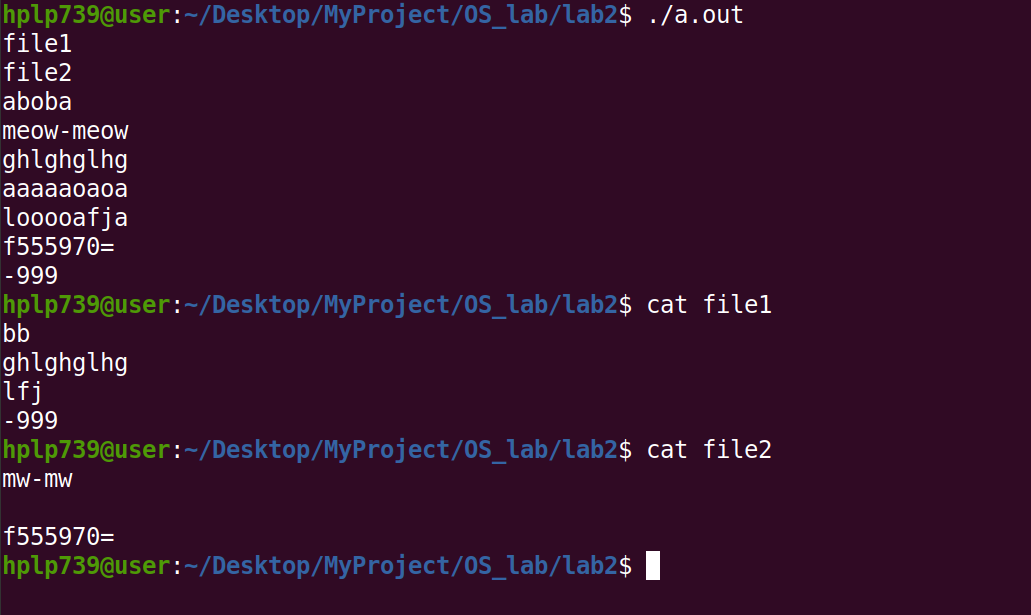
free(line);

fclose(fp);

return 0;

}

**Демонстрация работы программы**



**Выводы**

Данная лабораторная работа оказалась полезной и интересной: она познакомила меня с понятием процесса в операционной системе и системнми вызовами, помогла мне разобраться с тем, как следует работать с неименованными каналами для межпроцессорного взаимодействия, научила переопределять потоки ввода/вывода. Навыки работы с процессами, а также основы межпроцессорного взаимодействия безусловно пригодятся мне в будущем,.