Московский Авиационный Институт

(Национальный Исследовательский Университет)

Факультет информационных технологий и прикладной математики

Кафедра вычислительной математики и программирования

**Лабораторная работа №1 по курсу**

**«Операционные системы»**

Студент: Рамалданов Рустамхан Ражудинович

Группа: М8О-208Б-22

Вариант: 19

Преподаватель: Миронов Евгений Сергеевич

Оценка: \_\_\_\_\_\_\_\_\_\_\_

Дата: \_\_\_\_\_\_\_\_\_\_\_

Подпись: \_\_\_\_\_\_\_\_\_\_\_

Москва, 2023

**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/pepelulka/OS_labs>

**Постановка задачи**

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.

**Общие сведения о программе**

Программа состоит из двух частей – главная программа, родительский процесс, описанный в main.c, и программа дочернего прочесса, описанная в child.c. В программе используются следующие системныe вызовы:

1. fork()
2. execv()
3. pipe()
4. read()
5. write()
6. open()
7. close()
8. dup2()

**Общий метод и алгоритм решения**

Родительский процесс получает на вход имена выходных файлов и создает дочерние процессы, передавая им имена файлов в качестве аргументов командной строки. Перед этим родительский процесс создает два канала, по которым он будет передавать дочерним процессам строки, согласно правилу филтрации. Дочерние процессы удаляют из строк гласные и записывают результат в соответствующие файлы.

**Исходный код**

**========================== parent.h ==========================**

**#ifndef PARENT\_H**

**#define PARENT\_H**

**#include <stdio.h>**

**#include <stdlib.h>**

**#include <string.h>**

**#include <sys/wait.h>**

**#include <unistd.h>**

**#include "utils.h"**

**void ParentRoutine(char\* childProgramPath, FILE\* stream);**

**#endif //PARENT\_H**

**========================== utils.h ==========================**

**#ifndef UTILS\_H**

**#define UTILS\_H**

**#include <assert.h>**

**#include <stdbool.h>**

**#include <stdio.h>**

**#include <stdlib.h>**

**#include <string.h>**

**#include <unistd.h>**

**enum {**

**PIPE\_READ,**

**PIPE\_WRITE**

**};**

**void CreatePipe(int pipeFd[2]);**

**void CreateChildForPipe(char\* fileName, int pipe[2], int pipeToClose[2], char\*\* args);**

**bool IsVowel(char c);**

**bool Probability(int percentage);**

**char\* ReadString(FILE\* stream);**

**char\* ReadStringAndRemoveVowels(FILE\* stream);**

**#endif //UTILS\_H**

**========================== child.c ==========================**

**#include "utils.h"**

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

**if (argc < 2) {**

**printf("Missing arguments!\n");**

**exit(EXIT\_FAILURE);**

**}**

**char\* filename = argv[1];**

**FILE\* file = fopen(filename, "w");**

**if (file == NULL) {**

**printf("Can't open file %s\n", filename);**

**exit(EXIT\_FAILURE);**

**}**

**char\* input;**

**while ((input = ReadStringAndRemoveVowels(stdin)) != NULL) {**

**fprintf(file, "%s", input);**

**fflush(file);**

**free(input);**

**}**

**fclose(file);**

**}**

**========================== parent.c ==========================**

**#include "parent.h"**

**void ParentRoutine(char\* childProgramPath, FILE\* stream) {**

**char fileName1[128], fileName2[128];**

**char \*input1 = ReadString(stream);**

**char \*input2 = ReadString(stream);**

**int lenInput1 = strlen(input1);**

**int lenInput2 = strlen(input2);**

**if (input1 == NULL || input2 == NULL) {**

**printf("Error with input.\n");**

**exit(EXIT\_FAILURE);**

**}**

**strcpy(fileName1, input1);**

**strcpy(fileName2, input2);**

**free(input1);**

**free(input2);**

**fileName1[lenInput1 - 1] = '\0';**

**fileName2[lenInput2 - 1] = '\0';**

**int pipe1[2], pipe2[2];**

**CreatePipe(pipe1);**

**CreatePipe(pipe2);**

**char\* args1[] = {childProgramPath, fileName1, NULL};**

**char\* args2[] = {childProgramPath, fileName2, NULL};**

**CreateChildForPipe(childProgramPath, pipe1, pipe2, args1);**

**CreateChildForPipe(childProgramPath, pipe2, pipe1, args2);**

**close(pipe1[PIPE\_READ]);**

**close(pipe2[PIPE\_READ]);**

**// Now parent's program:**

**char\* input;**

**while ((input = ReadString(stream)) != NULL) {**

**if (Probability(80)) {**

**write(pipe1[PIPE\_WRITE], input, strlen(input));**

**} else {**

**write(pipe2[PIPE\_WRITE], input, strlen(input));**

**}**

**free(input);**

**}**

**close(pipe1[PIPE\_WRITE]);**

**close(pipe2[PIPE\_WRITE]);**

**wait(NULL);**

**wait(NULL);**

**}**

**========================== utils.c ==========================**

**#include "utils.h"**

**void CreatePipe(int pipeFd[2]) {**

**if(pipe(pipeFd) != 0) {**

**printf("Couldn't create pipe\n");**

**exit(EXIT\_FAILURE);**

**}**

**}**

**void CreateChildForPipe(char\* fileName, int pipe[2], int pipeToClose[2], char\*\* args) {**

**int pid = fork();**

**if (pid == -1) {**

**printf("Can't fork!\n");**

**exit(EXIT\_FAILURE);**

**}**

**if (pid == 0) {**

**close(pipe[PIPE\_WRITE]);**

**close(pipeToClose[PIPE\_READ]);**

**close(pipeToClose[PIPE\_WRITE]);**

**dup2(pipe[PIPE\_READ], STDIN\_FILENO);**

**execv(fileName, args);**

**}**

**}**

**bool IsVowel(char c) {**

**return c == 'e' ||**

**c == 'u' ||**

**c == 'i' ||**

**c == 'o' ||**

**c == 'a';**

**}**

**bool Probability(int percentage) {**

**assert(0 <= percentage && percentage <= 100);**

**return (rand() % 100) < percentage;**

**}**

**char\* ReadString(FILE \*stream) {**

**if (feof(stream)) {**

**return NULL;**

**}**

**const size\_t chunkSize = 256;**

**size\_t size = 256;**

**char\* buffer = (char\*)malloc(size \* sizeof(char));**

**size\_t idx = 0;**

**if(!buffer) {**

**printf("Couldn't allocate buffer");**

**exit(EXIT\_FAILURE);**

**}**

**int cur;**

**while ((cur = getc(stream)) != EOF) {**

**buffer[idx++] = (char)cur;**

**if (idx == size) {**

**size += chunkSize;**

**buffer = realloc(buffer, size \* sizeof(char));**

**if(!buffer) {**

**printf("Couldn't allocate buffer");**

**exit(EXIT\_FAILURE);**

**}**

**}**

**if (cur == '\n') {**

**break;**

**}**

**}**

**buffer[idx] = '\0';**

**return buffer;**

**}**

**char\* ReadStringAndRemoveVowels(FILE\* stream) {**

**if (feof(stream)) {**

**return NULL;**

**}**

**const size\_t chunkSize = 256;**

**size\_t size = 256;**

**char \*buffer = (char\*)malloc(size \* sizeof(char));**

**size\_t idx = 0;**

**if(!buffer) {**

**printf("Couldn't allocate buffer");**

**exit(EXIT\_FAILURE);**

**}**

**int cur;**

**while ((cur = getc(stream)) != EOF) {**

**if (!IsVowel(cur)) {**

**buffer[idx++] = cur;**

**}**

**if (idx == size) {**

**size += chunkSize;**

**buffer = realloc(buffer, size \* sizeof(char));**

**if(!buffer) {**

**printf("Couldn't allocate buffer");**

**exit(EXIT\_FAILURE);**

**}**

**}**

**if (cur == '\n') {**

**break;**

**}**

**}**

**buffer[idx] = '\0';**

**return buffer;**

**}**

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

**#include "parent.h"**

**#include <time.h>**

**int main() {**

**srand(time(NULL));**

**char\* childProgramName;**

**if ((childProgramName = getenv("PATH\_TO\_CHILD")) == NULL) {**

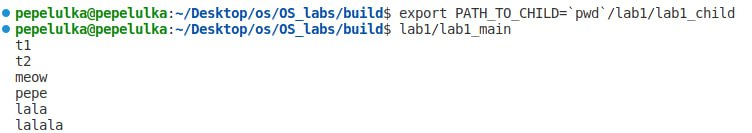
**childProgramName = "./child";**

**}**

**ParentRoutine(childProgramName, stdin);**

**}**

**Демонстрация работы программы**





**Выводы**

В ходе выполнения лабораторной работе я получил знания и навыки использования системных вызовов Linux при написании программ. Я узнал о системных вызовах fork, pipe, dup2 и научился их применять. Также были получены знания о структуре размещения процессов в памяти компьютера.