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: Старцев Иван Романович

Группа: М8О-201Б-21

Вариант: 9

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/IvanTvardovsky/OS-labs

**Постановка задачи**

**Цель работы**

Приобретение практических навыков в:

 Управление процессами в ОС

 Обеспечение обмена данных между процессами посредством каналов

**Задание**

Родительский процесс создает дочерний процесс. Первой строчкой пользователь в консоль родительского процесса вводит имя файла, которое будет использовано для открытия файла с таким именем на чтение. Стандартный поток ввода дочернего процесса переопределяется открытым файлом. Дочерний процесс читает команды из стандартного потока ввода. Стандартный поток вывода дочернего процесса перенаправляется в pipe1. Родительский процесс читает из pipe1 и прочитанное выводит в свой стандартный поток вывода. Родительский и дочерний процесс должны быть представлены разными программами.

9 вариант) В файле записаны команды вида:«число число число<endline>». Дочерний процесс производит деление первого числа команда, на последующие числа в команде, а результат выводит в стандартный поток вывода. Если происходит деление на 0, то тогда дочерний и родительский процесс завершают свою работу. Проверка деления на 0 должна осуществляться на стороне дочернего процесса. Числа имеют тип float. Количество чисел может быть произвольным.

**Общие сведения о программе**

Программа компилируется из файла main.c. Также используются дополнителье файлы с родительской и дочерней программами parent.c и child.c, общие утилиты utils.c

В программе используются следующие системные вызовы:

1. pipe() - cоздает канал для чтения и записи.
2. fork() - создаёт новый процесс посредством копирования вызывающего процесса. Новый процесс считается дочерним процессом. Вызывающий процесс считается родительским процессом.
3. execv() - дублирует действия оболочки, относящиеся к поиску исполняемого файла. (семейство функций exec заменяет текущий образ процесса новым образом процесса.)
4. read() - пытается прочитать заданное число байт из файлового дескриптора в буфер.
5. close() - закрывает файловый дескриптор, который после этого не ссылается ни на один и файл и может быть использован повторно.
6. dup2(oldfd, newfd) - dup2() - закрывает файл, связанный с дескриптором newfd (если он был открыт) и записывает ссылку oldfd в newfd.

**Общий метод и алгоритм решения**

Чтение имени файла из консоли. Открытие файла с заданным именем на чтение и создание файлового дескриптора. Создание пайпа. Создание нового (дочернего) процесса. Передача дочернему процессу файлового дескриптора и пайпа. Считывание в дочернем процессе переданной информации и переопределение потока ввода на открытый файл и переопределение потока вывода на запись в пайп. Построчное считывание и выполнение команд вида :«число число … число<endline>». Запись результатов команд на пайп с дополнительным разграничением “\n” между ответов. В родительском процессе считывание из пайпа переданной информации дочерним процессом и последующим вывод в стандартный поток вывода. Файл, в который направляется вывод определятся второй вводимой строкой.

**Исходный код**

**main.c**

#include "parent.h"

#include <stdio.h>

#include <stdlib.h>

int main() {

ParentRoutine(stdin);

return 0;

}

**child.c**

#include "utils.h"

#include <ctype.h>

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

if (argc != 2) {

printf("Necessary arguments were not provided\n");

exit(EXIT\_FAILURE);

}

FILE\* out = fopen(argv[1], "r");

if (!out) {

printf("Failed to open file\n");

exit(EXIT\_FAILURE);

}

char\* input;

char\* buf;

while ((input = ReadString(stdin)) != NULL) {

int index = 0, inputLen = strlen(input);

buf = ReadNumber(input, index);

index += strlen(buf) + 1;

float result = atof(buf), output;

free(buf);

int flag = 0;

while (index < inputLen) {

if (flag) {

if (atof(buf) == 0) {

exit(EXIT\_FAILURE);

}

result /= atof(buf);

free(buf);

} else {

++flag;

}

buf = ReadNumber(input, index);

index += strlen(buf) + 1;

if (buf == NULL) {

break;

}

}

result /= atof(buf);

free(buf);

free(input);

output = (float)((int)(result \* 100)) / 100;

write(STDOUT\_FILENO, &output, sizeof(output));

}

free(input);

fclose(out);

return 0;

}

**parent.c**

#include "parent.h"

#include "utils.h"

#include <sys/wait.h>

#include <fcntl.h>

void ParentRoutine(FILE\* stream) {

char inputFileName[256];

fscanf(stream, "%s", inputFileName);

char outputFileName[256];

fscanf(stream, "%s", outputFileName);

int inputFile;

inputFile = open(inputFileName, O\_RDONLY);

if (inputFile < 0) {

char message[] = "Can't open input file\n";

write(STDERR\_FILENO, &message, sizeof(message) - 1);

exit(EXIT\_FAILURE);

}

int pipe[2];

CreatePipe(pipe);

int id = fork();

if (id == 0) {

dup2(inputFile, STDIN\_FILENO);

dup2(pipe[1], STDOUT\_FILENO);

dup2(pipe[1], STDERR\_FILENO);

close(inputFile);

close(pipe[0]);

close(pipe[1]);

char\* argv[3];

argv[0] = getenv("child");

argv[1] = inputFileName;

argv[2] = NULL;

if (execv(getenv("child"), argv) == -1) {

printf("Failed to exec\n");

exit(EXIT\_FAILURE);

}

} else if (id > 0) {

close(pipe[1]);

waitpid(id, (int \*)NULL, 0);

FILE \*outputFile;

outputFile = fopen(outputFileName, "w");

float result;

while (read(pipe[0], &result, sizeof(result))) {

fprintf(outputFile, "%f\n", result);

}

close(pipe[0]);

fclose(outputFile);

} else {

printf("Failed to fork\n");

exit(EXIT\_FAILURE);

}

}

**utils.c**

#include "utils.h"

void CreatePipe(int pipeFd[2]) {

if(pipe(pipeFd) != 0) {

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

exit(EXIT\_FAILURE);

}

}

char\* ReadString(FILE\* stream) {

if (feof(stream)) {

return NULL;

}

const int chunkSize = 256;

char\* buffer = (char\*) malloc(chunkSize);

int bufferSize = chunkSize;

if (!buffer) {

printf("Couldn't allocate buffer");

exit(EXIT\_FAILURE);

}

int readChar;

int idx = 0;

while ((readChar = getc(stream)) != EOF) {

buffer[idx++] = readChar;

if (idx == bufferSize) {

buffer = realloc(buffer, bufferSize + chunkSize);

bufferSize += chunkSize;

}

if (readChar == '\n') {

break;

}

}

buffer[idx] = '\0';

return buffer;

}

char\* ReadNumber(char\* string, int idx) {

const int chunkSize = 256;

char\* buffer = (char\*) malloc(chunkSize);

int bufferSize = chunkSize;

if (!buffer) {

printf("Couldn't allocate buffer");

exit(EXIT\_FAILURE);

}

if (string[idx] == ' ' || string[idx] == '\n' || string[idx] == '\0' || string[idx] == EOF) {

free(buffer);

return NULL;

}

int bufInd = 0;

while (string[idx] != ' ' && string[idx] != '\n') {

buffer[bufInd] = string[idx];

++idx;

++bufInd;

if (bufInd == bufferSize) {

buffer = realloc(buffer, bufferSize + chunkSize);

bufferSize += chunkSize;

}

}

if (strlen(buffer) == 0) {

free(buffer);

return NULL;

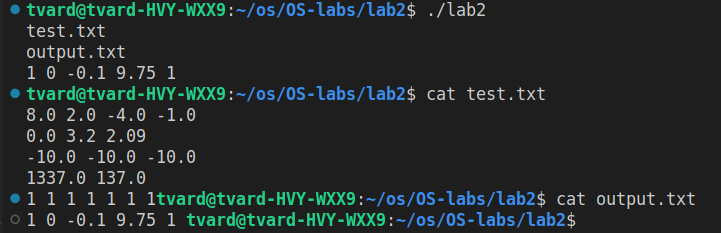
}

buffer[bufInd] = '\0';

return buffer;

}

**Демонстрация работы программы**



**Выводы**

За время выполнения лабораторной работы я научился управлять процессами в ОС, а также разобрался с обеспечением обмена данных между процессов посредством каналов.