МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ

ФЕДЕРАЦИИ

Московский Авиационный Институт

(Национальный Исследовательский Университет)

Институт №8 "Компьютерные науки и прикладная математика"

Кафедра 806 "Вычислительная математика и программирование"

Лабораторная работа №1

По курсу «Операционные системы»

Студент: Степанов Н.Е.

Группа: М8О-208Б-23

Вариант: 2

Преподаватель: Миронов Е. С.

Дата: \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Оценка: \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Подпись: \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Москва, 2024

**Содержание**

1. Репозиторий

2. Постановка задачи

3. Общие сведения о программе

4. Общий метод и алгоритм решения

5. Исходный код

6. Сборка программы

7. Демонстрация работы программы

8. Выводы

**Репозиторий**

<https://github.com/n0w3e/os_labs/tree/lab1>

**Постановка задачи**

**Цель работы**

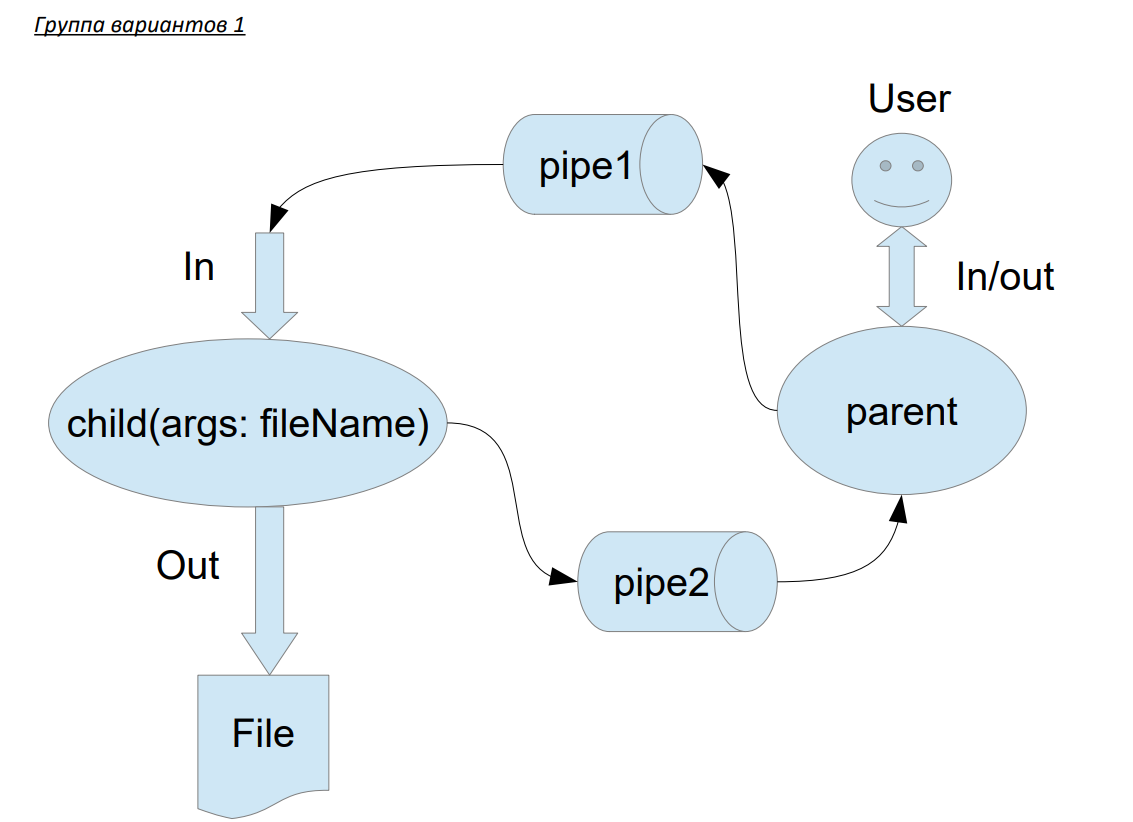
Приобретение практических навыков в:

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.



Родительский процесс создает дочерний процесс. Первой строчкой пользователь в консоль родительского процесса пишет имя файла, которое будет передано при создании дочернего процесса. Родительский и дочерний процесс должны быть представлены разными программами. Родительский процесс передает команды пользователя через pipe1, который связан с стандартным входным потоком дочернего процесса. Дочерний процесс принеобходимости передает данные в родительский процесс через pipe2. Результаты своей работы дочерний процесс пишет в созданный им файл. Допускается просто открыть файл и писать туда, не перенаправляя стандартный поток вывода.

2 вариант) Пользователь вводит команды вида: «число число число». Далее эти числа передаются от родительского процесса в дочерний. Дочерний процесс считает их сумму и выводит её в файл. Числа имеют тип float. Количество чисел может быть произвольным.

**Общие сведения о программе**

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

1. pipe() - существует для передачи информации между различными процессами.

2. fork() - создает новый процесс.

3. execl() - 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**

1. В родительском процессе создаются два канала: **pipe1** для передачи данных от родительского процесса к дочернему, и **pipe2** для передачи данных обратно от дочернего процесса к родительскому.

2. Используем **fork()** для создания дочернего процесса. Если **fork()** возвращает 0, это означает, что текущий процесс является дочерним.

3. В дочернем процессе используем **dup2()** для перенаправления стандартного ввода (**STDIN\_FILENO**) на чтение из **pipe1[0]** и стандартного вывода (**STDOUT\_FILENO**) на запись в **pipe2[1]**.

4. В дочернем процессе используется **execl()** для запуска исполняемого файла дочернего процесса (**lab1**), передавая ему имя файла для записи результатов.

5. Родительский процесс считывает ввод пользователя (строки чисел) и записывает их в **pipe1[1]** для передачи дочернему процессу.

6. Дочерний процесс считывает данные из стандартного ввода (который был перенаправлен на **pipe1[0]**), вычисляет сумму чисел в строке и записывает результат в указанный файл, а также выводит результат в стандартный вывод (который перенаправлен на **pipe2[1]**).

7. Родительский процесс считывает результат из **pipe2[0]** и выводит его на экран.

8. Родительский процесс закрывает каналы и ожидает завершения дочернего процесса с помощью **wait()**.

**Исходный код**

**child.h:**

#ifndef CHILD\_H

#define CHILD\_H

void child\_process(const char\* filename);

#endif

**parent.h:**

#ifndef PARENT\_H

#define PARENT\_H

void parent\_process();

#endif

**child.c:**

#include "child.h"

#include <stdio.h>

#include <stdlib.h>

#include <string.h>

void child\_process(const char\* filename) {

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

if (!file) {

perror("fopen");

exit(EXIT\_FAILURE);

}

char input[256];

while (fgets(input, sizeof(input), stdin)) {

float sum = 0;

float number;

char \*token = strtok(input, " ");

while (token != NULL) {

if (sscanf(token, "%f", &number) == 1) {

sum += number;

}

token = strtok(NULL, " ");

}

fprintf(file, "Sum: %.2f\n", sum);

fflush(file);

printf("Sum: %.2f\n", sum);

}

fclose(file);

}

**parent.c:**

#include "parent.h"

#include <stdio.h>

#include <stdlib.h>

#include <unistd.h>

#include <string.h>

#include <sys/wait.h>

#include <fcntl.h>

void parent\_process() {

int pipe1[2], pipe2[2];

pid\_t pid;

char filename[100];

if (pipe(pipe1) == -1 || pipe(pipe2) == -1) {

perror("pipe");

exit(EXIT\_FAILURE);

}

printf("Enter name file: ");

scanf("%s", filename);

pid = fork();

if (pid == -1) {

perror("fork");

exit(EXIT\_FAILURE);

}

if (pid == 0) {

if (dup2(pipe1[0], STDIN\_FILENO) == -1 || dup2(pipe2[1], STDOUT\_FILENO) == -1) {

perror("dup2");

exit(EXIT\_FAILURE);

}

close(pipe1[1]);

close(pipe2[0]);

execl("../", "lab1", filename, NULL); // Путь до исполняемого файла

perror("execl");

exit(EXIT\_FAILURE);

} else {

close(pipe1[0]);

close(pipe2[1]);

char input[256];

while (1) {

printf("Enter numbers (or 'exit'): ");

fgets(input, sizeof(input), stdin);

if (strcmp(input, "exit\n") == 0) {

break;

}

write(pipe1[1], input, strlen(input));

fsync(pipe1[1]);

char result[256];

int bytesRead = read(pipe2[0], result, sizeof(result) - 1);

result[bytesRead] = '\0';

printf("Result: %s\n", result);

}

close(pipe1[1]);

wait(NULL);

close(pipe2[0]);

}

}

**main.c:**  
  
#include <stdio.h>

#include <string.h>

#include <stdlib.h>

#include "../include/parent.h"

#include "../include/child.h"

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

if (argc == 1) {

parent\_process();

} else if (argc == 2) {

child\_process(argv[1]);

} else {

fprintf(stderr, "Error\n");

return EXIT\_FAILURE;

}

return EXIT\_SUCCESS;

}

**Демонстрация работы программы**

n0wee@DESKTOP-8QSPN1P:~/Coding/os\_labs/build/lab1$ ./lab1

Enter name file: result.txt

Enter numbers (or 'exit'): 1 2 3

Result: Sum: 6.00

Enter numbers (or 'exit'): 4 5 6

Result: Sum: 15.00

Enter numbers (or 'exit'): exit

**Содержимое файла result.txt:**

Sum: 6.00

Sum: 15.00

**Выводы**

В ходе выполнения лабораторной работы я познакомился с механизмом взаимодействия процессов через каналы (**pipe**) в UNIX-системах. Я изучил использование системных вызовов **fork()**, **dup2()**, **execl()** и **wait()**. Новым для меня стало понимание перенаправления стандартных потоков ввода/вывода и обработки данных в дочерних процессах.