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»

Группа: М8О-212БВ-24

Студент: Мышакин М.С.

Преподаватель: Бахарев В.Д.

Оценка: _____

Дата: 04.10.25

Москва, 2025

Постановка задачи

Вариант 7.

В файле записаны команды вида: «число число число». Дочерний процесс считает их сумму и выводит результат в стандартный поток вывода. Числа имеют тип float. Количество чисел может быть произвольным.

Общий метод и алгоритм решения

Использованные системные вызовы:

- pid_t fork(void) – Создаёт дочерний процесс, клонируя текущий. Возвращает 0 в дочернем процессе и PID дочернего — в родительском.
- int pipe(int *fd) – Создаёт анонимный канал и записывает два дескриптора: fd[0]— для чтения, fd[1]— для записи.
- int open(const char *pathname, int flags) – Открывает файл на чтение (O_RDONLY) для перенаправления в stdin дочернего процесса.
- int close(int fd) – Закрывает ненужные концы канала и файловые дескрипторы для корректной работы pipe и освобождения ресурсов.
- int dup2(int oldfd, int newfd) – Дублирует файловый дескриптор: используется для перенаправления stdin на файл и stdout — в канал.
- int execl(const char *path, const char *arg, ...) – Заменяет текущий процесс (дочерний) на новую программу (./child).
- ssize_t read(int fd, void *buf, size_t count) – Читает данные: из консоли (в родителе), из канала (в родителе), посимвольно из stdin (в child).
- ssize_t write(int fd, const void *buf, size_t count) – Выводит данные: приглашение, ошибки, результаты — в stdout или stderr.
- pid_t wait(int *status) – Родительский процесс ожидает завершения дочернего, предотвращая зомби-процессы.
- void exit(int status) – Немедленно завершает процесс с указанным кодом возврата (используется при ошибках).

Описание работы программы

В рамках лабораторной работы была реализована система взаимодействия между двумя отдельными программами — родительским и дочерним процессами — с использованием только системных вызовов Unix.

Родительский процесс (parent) сначала запрашивает у пользователя имя файла, который необходимо обработать. После получения имени файла создаётся анонимный канал (pipe) с помощью системного вызова pipe(). Затем с помощью fork() порождается дочерний процесс.

Дочерний процесс открывает указанный файл на чтение через open(), после чего перенаправляет свой стандартный ввод (stdin) на содержимое этого файла с помощью dup2(). Также стандартный вывод (stdout) дочернего процесса перенаправляется в записывающий

конец канала (pipe[1]) — это позволяет передавать результаты работы обратно родителю. Далее дочерний процесс заменяется на отдельную программу child с помощью execl().

Программа child читает входные данные посимвольно из stdin (который теперь связан с файлом), разбивает их на строки, парсит каждую строку в поиске трёх чисел с плавающей точкой, вычисляет их сумму и форматирует результат в виде строки «Сумма: X.XX». Результат выводится в stdout, который перенаправлен в канал.

Родительский процесс, в свою очередь, закрывает записывающий конец канала и читает все данные из читающего конца (pipe[0]) с помощью read(), последовательно выводя их в свой стандартный вывод через write(). После завершения чтения родитель ожидает окончания дочернего процесса с помощью wait(), чтобы избежать появления зомби-процесса.

Код программы

parent.c

```
#include <unistd.h>

#include <sys/wait.h>

#include <fcntl.h>

#include <stdlib.h>

#include <string.h>

#include <errno.h>

#include <stdio.h>


const int BUFFER_SIZE = 256;


int main()

{

    int pipeFd[2];

    if (pipe(pipeFd) == -1)

    {

        write(STDERR_FILENO, "pipe error\n", 11);

        exit(1);

    }
```

```
const char prompt[] = "Введите имя файла: ";
write(STDOUT_FILENO, prompt, strlen(prompt));

char fileName[BUFFER_SIZE];
ssize_t n = read(STDIN_FILENO, fileName, BUFFER_SIZE - 1);
if (n <= 0)
{
    write(STDERR_FILENO, "read error\n", 11);
    exit(1);
}

if (fileName[n - 1] == '\n')
{
    fileName[n - 1] = '\0';
}
else
{
    fileName[n] = '\0';
}

pid_t pid = fork();
if (pid < 0)
{
    write(STDERR_FILENO, "fork error\n", 11);
    exit(1);
}

if (pid == 0)
{
    close(pipeFd[0]);
```

```
        int fileFd = open(fileName, O_RDONLY);
        if (fileFd < 0)
        {
            char msg[256];
            snprintf(msg, sizeof(msg), "Ошибка открытия файла '%s'\n",
fileName);
            write(STDERR_FILENO, msg, strlen(msg));
            exit(1);
        }

        if (dup2(fileFd, STDIN_FILENO) == -1)
        {
            write(STDERR_FILENO, "dup2 stdin error\n", 17);
            exit(1);
        }
        close(fileFd);

        if (dup2(pipeFd[1], STDOUT_FILENO) == -1)
        {
            write(STDERR_FILENO, "dup2 stdout error\n", 18);
            exit(1);
        }
        close(pipeFd[1]);

        execl("./child", "child", NULL);
        write(STDERR_FILENO, "execl failed\n", 14);
        exit(1);
    }
    else
    {
        close(pipeFd[1]);
```

```
        char buffer[BUFFER_SIZE];
        while ((n = read(pipeFd[0], buffer, sizeof(buffer))) > 0)
        {
            write(STDOUT_FILENO, buffer, n);
        }
        if (n < 0)
        {
            write(STDERR_FILENO, "read from pipe error\n", 21);
        }
        close(pipeFd[0]);
        wait(NULL);
    }

    return 0;
}
```

child.c

```
#include <unistd.h>
#include <stdlib.h>
#include <string.h>
#include <stdio.h>

const int BUFFER_SIZE = 256;

void FormatSum(float sum, char* out)
{
    int sign = 0;
    if (sum < 0.0f)
    {
        sign = 1;
        sum = -sum;
```

```
    }

    int intPart = (int)sum;
    int fracPart = (int)((sum - (float)intPart) * 100.0f + 0.5f);
    if (fracPart >= 100)
    {
        fracPart -= 100;
        intPart++;
    }

    char buf[64];
    if (sign)
    {
        snprintf(buf, sizeof(buf), "Сумма: -%d.%02d\n", intPart, fracPart);
    }
    else
    {
        snprintf(buf, sizeof(buf), "Сумма: %d.%02d\n", intPart, fracPart);
    }
    strcpy(out, buf);
}

int main()
{
    char line[BUFFER_SIZE];
    char c;
    int pos = 0;

    while (1)
    {
        ssize_t r = read(STDIN_FILENO, &c, 1);
```

```
if (r <= 0) break;

if (c == '\n')
{
    line[pos] = '\0';
    if (pos == 0)
    {
        pos = 0;
        continue;
    }

    char* start = line;
    char* end;
    int count = 0;
    float nums[3];

    while (*start && count < 3)
    {
        while (*start == ' ' || *start == '\t') start++;
        if (!*start) break;

        nums[count] = strtof(start, &end);
        if (end == start) break;
        count++;
        start = end;
    }

    if (count == 3)
    {
        float sum = nums[0] + nums[1] + nums[2];
        char output[64];
```

```
                FormatSum(sum, output);
                write(STDOUT_FILENO, output, strlen(output));
            }
            pos = 0;
        }
        else
        {
            if (pos < BUFFER_SIZE - 1)
            {
                line[pos++] = c;
            }
        }
    }

    return 0;
}
```

Протокол работы программы

```
data.txt
1.2 3.4 5.6
0.5 -1.5 2.0
10.0 20.0 30.0


misha@misha-VirtualBoxUbuntu:~/Documents/cApps/OS-LABS$ gcc -o child child.c
misha@misha-VirtualBoxUbuntu:~/Documents/cApps/OS-LABS$ gcc -o parent parent.c
misha@misha-VirtualBoxUbuntu:~/Documents/cApps/OS-LABS$ ./parent
Введите имя файла: da
Ошибка открытия файла: da
misha@misha-VirtualBoxUbuntu:~/Documents/cApps/OS-LABS$ ./parent
Введите имя файла: data.txt
Сумма: 10.20
Сумма: 1.00
Сумма: 60.00
misha@misha-VirtualBoxUbuntu:~/Documents/cApps/OS-LABS$
```

Вывод

В ходе выполнения лабораторной работы были успешно реализованы межпроцессное взаимодействие через pipe, перенаправление стандартных потоков с помощью dup2 и запуск отдельной программы через execl. Программа корректно обрабатывает входной файл, вычисляет суммы троек чисел и возвращает результат родительскому процессу.