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-210БВ-24

Студент: Резинкин Д.В.

Преподаватель: Бахарев В.Д.

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

Дата: 20.09.25

Москва, 2025

**Постановка задачи**

**Вариант 1.**

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 int. Количество чисел может быть произвольным.

**Общий метод и алгоритм решения**

В данной лабораторной работе была реализована программа, демонстрирующая взаимодействие между родительским и дочерним процессами посредством каналов (pipe).  
Алгоритм следующий:

1. Родительский процесс создаёт два канала: для передачи данных ребёнку и для обратной связи.
2. Выполняется fork().
3. В дочернем процессе выполняется перенаправление стандартного ввода/вывода в концы каналов и запуск программы child через execve().
4. Родитель принимает имя выходного файла от пользователя, передаёт его дочернему процессу как аргумент, затем отправляет строки с числами через канал.
5. Дочерний процесс вычисляет сумму чисел в строке, отправляет результат родителю и записывает его в файл.
6. Завершение происходит при вводе пустой строки.

Использованные системные вызовы:

* pid\_t fork(void); – создание дочернего процесса.
* int pipe(int fd[2]); – создание канала для межпроцессного взаимодействия.
* int dup2(int oldfd, int newfd); – переназначение файлового дескриптора (подмена stdin/stdout у дочернего процесса).
* int execve(const char \*pathname, char \*const argv[], char \*const envp[]); – замена кода текущего процесса на новый (запуск дочерней программы).
* ssize\_t read(int fd, void \*buf, size\_t count); – чтение данных из файлового дескриптора (в том числе из pipe).
* ssize\_t write(int fd, const void \*buf, size\_t count); – запись данных в файловый дескриптор (в том числе в pipe и в файл).
* int open(const char \*pathname, int flags, mode\_t mode); – открытие/создание файла для записи.
* int close(int fd); – закрытие файлового дескриптора.
* pid\_t waitpid(pid\_t pid, int \*status, int options); – ожидание завершения дочернего процесса.
* \_exit(int status); – немедленное завершение процесса.

**Код программы**

**parent.c**

#define \_GNU\_SOURCE

#include <unistd.h>

#include <fcntl.h>

#include <sys/types.h>

#include <sys/wait.h>

#include <errno.h>

#include <string.h>

#include <stdlib.h>

static void die(const char\* msg) {

    ssize\_t \_ \_\_attribute\_\_((unused)) = write(2, msg, strlen(msg));

    ssize\_t \_\_ \_\_attribute\_\_((unused)) = write(2, "\n", 1);

    \_exit(1);

}

static ssize\_t write\_all(int fd, const void\* buf, size\_t n) {

    const char\* p = (const char\*)buf;

    size\_t left = n;

    while (left) {

        ssize\_t w = write(fd, p, left);

        if (w < 0) {

            if (errno == EINTR) continue;

            return -1;

        }

        left -= (size\_t)w;

        p += w;

    }

    return (ssize\_t)n;

}

// читает одну строку (заканчивается '\n' или EOF). Возвращает длину, 0 при EOF.

static ssize\_t read\_line(int fd, char\* buf, size\_t cap) {

    size\_t i = 0;

    while (i + 1 < cap) {

        char c;

        ssize\_t r = read(fd, &c, 1);

        if (r == 0) break;         // EOF

        if (r < 0) {

            if (errno == EINTR) continue;

            return -1;

        }

        buf[i++] = c;

        if (c == '\n') break;

    }

    buf[i] = '\0';

    return (ssize\_t)i;

}

// обрезать завершающий \n, если есть

static void chomp(char\* s) {

    size\_t n = strlen(s);

    if (n && s[n-1] == '\n') s[n-1] = '\0';

}

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

    (void)argc; (void)argv;

    const char\* prompt1 = "Введите имя файла: ";

    write\_all(1, prompt1, strlen(prompt1));

    char fileName[512];

    ssize\_t fnlen = read\_line(0, fileName, sizeof(fileName));

    if (fnlen < 0) die("не удалось выполнить чтение (fileName)");

    if (fnlen == 0) die("имя файла не указано");

    chomp(fileName);

    int p1[2], p2[2];

    if (pipe(p1) < 0) die("pipe1 failed");

    if (pipe(p2) < 0) die("pipe2 failed");

    pid\_t pid = fork();

    if (pid < 0) die("fork failed");

    if (pid == 0) {

        // child (запустит отдельную программу ./child)

        // p1: parent->child (stdin ребёнка)  => читаем с p1[0]

        // p2: child->parent (stdout ребёнка) => пишем в p2[1]

        if (dup2(p1[0], 0) < 0) die("dup2 p1->stdin failed");

        if (dup2(p2[1], 1) < 0) die("dup2 p2->stdout failed");

        // закрыть лишнее

        close(p1[0]); close(p1[1]);

        close(p2[0]); close(p2[1]);

        // argv для execve: child fileName

        char\* args[3];

        args[0] = (char\*)"child";

        args[1] = fileName;

        args[2] = NULL;

        // запускаем исполняемый файл "child" из текущего каталога

        execve("./child", args, envp);

        // если вернулись — ошибка

        die("execve ./child failed");

    }

    // parent

    close(p1[0]); // не читаем из pipe1

    close(p2[1]); // не пишем в pipe2

    const char\* prompt2 =

        "Введите строку, например: \"12 -3 7\" и нажмите Ентер.\n"

        "Пустая строка для завершения.\n";

    write\_all(1, prompt2, strlen(prompt2));

    char inbuf[2048];

    char outbuf[2048];

    for (;;) {

        write\_all(1, "> ", 2);

        ssize\_t n = read\_line(0, inbuf, sizeof(inbuf));

        if (n < 0) die("read(user line) failed");

        if (n == 0) { // EOF

            // закрываем запись — ребёнок увидит EOF

            close(p1[1]);

            break;

        }

        // пустая строка — завершить

        if (inbuf[0] == '\n' || inbuf[0] == '\0') {

            close(p1[1]);

            break;

        }

        // отправляем строку ребёнку

        if (write\_all(p1[1], inbuf, (size\_t)n) < 0) die("не удалось выполнить запись в дочерний файл");

        // ждём ответ одной строкой и печатаем пользователю

        ssize\_t m = read\_line(p2[0], outbuf, sizeof(outbuf));

        if (m < 0) die("не удалось выполнить чтение из дочернего файла");

        if (m == 0) {

            write\_all(1, "(child closed pipe)\n", 20);

            break;

        }

        write\_all(1, outbuf, (size\_t)m);

    }

    // дочитать всё, что осталось у ребёнка (на случай буфера)

    for (;;) {

        ssize\_t m = read(p2[0], outbuf, sizeof(outbuf));

        if (m < 0) {

            if (errno == EINTR) continue;

            break;

        }

        if (m == 0) break;

        write\_all(1, outbuf, (size\_t)m);

    }

    close(p2[0]);

    close(p1[1]);

    int status = 0;

    waitpid(pid, &status, 0);

    return WIFEXITED(status) ? WEXITSTATUS(status) : 1;

}

**child.c**

#include <unistd.h>

#include <fcntl.h>

#include <sys/types.h>

#include <errno.h>

#include <string.h>

#include <stdlib.h>

static void die(const char\* msg) {

    ssize\_t \_ \_\_attribute\_\_((unused)) = write(2, msg, strlen(msg));

    ssize\_t \_\_ \_\_attribute\_\_((unused)) = write(2, "\n", 1);

    \_exit(1);

}

static ssize\_t write\_all(int fd, const void\* buf, size\_t n) {

    const char\* p = (const char\*)buf;

    size\_t left = n;

    while (left) {

        ssize\_t w = write(fd, p, left);

        if (w < 0) { if (errno == EINTR) continue; return -1; }

        left -= (size\_t)w; p += w;

    }

    return (ssize\_t)n;

}

static ssize\_t read\_line(int fd, char\* buf, size\_t cap) {

    size\_t i = 0;

    while (i + 1 < cap) {

        char c;

        ssize\_t r = read(fd, &c, 1);

        if (r == 0) break; // EOF

        if (r < 0) { if (errno == EINTR) continue; return -1; }

        buf[i++] = c;

        if (c == '\n') break;

    }

    buf[i] = '\0';

    return (ssize\_t)i;

}

// простой парсер long long (десятичный)

static int parse\_ll(const char\* s, size\_t\* i, long long\* out) {

    while (s[\*i] == ' ' || s[\*i] == '\t' || s[\*i] == '\r') (\*i)++;

    int sign = 1;

    if (s[\*i] == '+' || s[\*i] == '-') {

        if (s[\*i] == '-') sign = -1;

        (\*i)++;

    }

    if (s[\*i] < '0' || s[\*i] > '9') return 0;

    long long val = 0;

    int found\_digit = 0;

    while (s[\*i] >= '0' && s[\*i] <= '9') {

        int d = s[\*i] - '0';

        val = val \* 10 + d;

        (\*i)++;

        found\_digit = 1;

    }

    // проверка на . или ,

    if (s[\*i] == '.' || s[\*i] == ',') {

        \*out = 0;

        while (s[\*i] && s[\*i] != ' ' && s[\*i] != '\n') (\*i)++;

        errno = EINVAL;

        return -1;

    }

    if (!found\_digit) return 0;

    \*out = val \* sign;

    return 1;

}

// целое -> строка; пишет прямо в buf, возвращает длину

static int ll\_to\_buf(long long v, char\* buf) {

    char tmp[32];

    int neg = v < 0;

    unsigned long long x = neg ? (unsigned long long)(-(v+1)) + 1ULL : (unsigned long long)v;

    int n = 0;

    do {

        tmp[n++] = (char)('0' + (x % 10ULL));

        x /= 10ULL;

    } while (x);

    int k = 0;

    if (neg) buf[k++] = '-';

    for (int i = n - 1; i >= 0; --i) buf[k++] = tmp[i];

    return k;

}

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

    if (argc < 2) die("дочь: требуется аргумент имени файла");

    const char\* fileName = argv[1];

    int fd = open(fileName, O\_WRONLY | O\_CREAT | O\_APPEND, 0644);

    if (fd < 0) die("child: open(file) failed");

    char line[2048];

    for (;;) {

        ssize\_t n = read\_line(0, line, sizeof(line));

        if (n < 0) die("дочь: не удалось прочитать строку");

        if (n == 0) break;                 // EOF

        if (line[0] == '\n' || line[0] == '\0') break; // пустая строка — конец

        size\_t i = 0;

        int found\_any = 0;

        long long sum = 0;

        long long val;

        while (1) {

            int st = parse\_ll(line, &i, &val);

            if (st == 1) {

                found\_any = 1;

                sum += val;

            } else if (st == -1) {

                const char\* msg = "ERR: invalid number format\n";

                write\_all(1, msg, strlen(msg));

                write\_all(fd, msg, strlen(msg));

                found\_any = 0; // сбросим, чтобы не писать результат

                break;

            } else {

                break; // st == 0 → конец строки

            }

        }

        if (!found\_any) {

            const char\* msg = "Бро, ошибка, тут числа нет либо что-то чужеродное\n";

            write\_all(1, msg, strlen(msg));

            write\_all(fd, msg, strlen(msg));

            continue;

        }

        char out[128];

        int k = 0;

        memcpy(out + k, "sum=", 4);  k += 4;

        k += ll\_to\_buf(sum, out + k);

        out[k++] = '\n';

        write\_all(1, out, (size\_t)k);   // в parent

        write\_all(fd, out, (size\_t)k);  // в файл

    }

    close(fd);

    return 0;

}

**Протокол работы программы**

diwan@DESKTOP-FVGD4PE:/mnt/e/Учеба/2 курс/ос/lab1$ make

gcc -Wall -Wextra -O2 parent.c -o parent

gcc -Wall -Wextra -O2 child.c -o child

diwan@DESKTOP-FVGD4PE:/mnt/e/Учеба/2 курс/ос/lab1$ ./parent

Введите имя файла: 123

Введите строку, например: "12 -3 7" и нажмите Ентер.

Пустая строка для завершения.

> 12 32 -12 -30 2

sum=4

> 12.2 2

ERR: invalid number format

>

**Вывод**

В ходе выполнения лабораторной работы были изучены механизмы создания процессов и межпроцессного взаимодействия через каналы в операционной системе Linux. Получены практические навыки работы с системными вызовами fork, pipe, execve, read, write и другими.  
Проблемы, с которыми пришлось столкнуться: необходимость отказаться от стандартных функций ввода-вывода (stdio.h) и реализовать собственные функции для работы со строками и числами. Также возникали трудности с обработкой ошибок и корректным завершением процессов.  
В результате была создана корректно работающая программа, выполняющая требования варианта 1.