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-209БВ-24

Студент: Махова А.Б.

Преподаватель: Миронов Е.С.

Оценка: _____

Дата: 16.11.25

Москва, 2025

Постановка задачи

Вариант 6.

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.

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

Общий метод и алгоритм решения

Родительский процесс запрашивает имя файла и открывает на чтение. В дочернем процессе перенаправляет STDIN на входной файл, а STDOUT - в pipe, затем запускает программу child. Родительский процесс считывает результаты из pipe и выводит их в консоль. Дочерний процесс считывает данные из STDIN построчно: проверяет наличие только числовых символов, пробелов и знаков, при невалидных данных выводит ошибку, при валидных - вычисляет сумму чисел и выводит результат в STDOUT.

Использованные системные вызовы:

- pid_t fork(void) – создание дочернего процесса
- int pipe(int *fd) – создание неименованного канала
- ssize_t write(int fd, const void *buf, size_t count) – запись данных
- ssize_t read(int fd, void *buf, size_t count) – чтение данных
- int open(const char *pathname, int flags) – открытие файла
- int close(int fd) – закрытие файлового дескриптора
- int dup2(int oldfd, int newfd) – перенаправление потоков
- int execl(const char *path, const char *arg, ...) – запуск программы
- pid_t wait(int *status) – ожидание завершения процесса
- void exit(int status) – завершение процесса

Код программы

parent.c

```
#include <unistd.h>
#include <stdlib.h> //здесь лежит pid_t
#include <sys/wait.h>
#include <stdint.h>
#include <fcntl.h>
#include <stdio.h>

int main(int argc, char *argv[]) {
    char progpath[1024];
    {
        const char message[] = "Input file: ";
        write(STDOUT_FILENO, message, sizeof(message) - 1);

        ssize_t n = read(STDIN_FILENO, progpath, sizeof(progpath) - 1);
        if (n <= 0) {
            const char message[] = "Error: can't read the file\n";
            write(STDERR_FILENO, message, sizeof(message) - 1);
            exit(EXIT_FAILURE);
        }
        progpath[n - 1] = '\0';
    }

    int child_to_parent[2];
    if (pipe(child_to_parent) == -1) {
        const char message[] = "Error: can't make the pipe\n";
        write(STDERR_FILENO, message, sizeof(message) - 1);
        exit(EXIT_FAILURE);
    }
```

```
pid_t pid = fork();

switch(pid) {
    case -1: {
        const char message[] = "Error: can't make a new process\n";
        write(STDERR_FILENO, message, sizeof(message) - 1);
        exit(EXIT_FAILURE);
    } break;

    case 0: {
        close(child_to_parent[0]);

        int file = open(progpath, O_RDONLY);
        if (file == -1) {
            const char message[] = "Error: can't open file\n";
            write(STDERR_FILENO, message, sizeof(message) - 1);
            exit(EXIT_FAILURE);
        }

        dup2(file, STDIN_FILENO);
        close(file);

        dup2(child_to_parent[1], STDOUT_FILENO);
        close(child_to_parent[1]);

        execl("./child", "child", NULL);
        const char message[] = "Error: can't exec child\n";
        write(STDERR_FILENO, message, sizeof(message) - 1);
        exit(EXIT_FAILURE);
```

```
        } break;

        default: {
            close(child_to_parent[1]);

            char buf[4096];
            ssize_t n;
            while ((n = read(child_to_parent[0], buf, sizeof(buf))) > 0) {
                write(STDOUT_FILENO, buf, n);
            }

            close(child_to_parent[0]);
            wait(NULL);
        } break;
    }

    return 0;
}
```

child.c

```
#include <stdlib.h>
#include <unistd.h> //здесь read, write, отсюла и STDIN/ERR/OUT
#include <ctype.h>
#include <string.h>
#include <stdio.h>



void process_line(char *line) {
    int only_spaces = 1;
    for (char *p = line; *p; p++) {
```

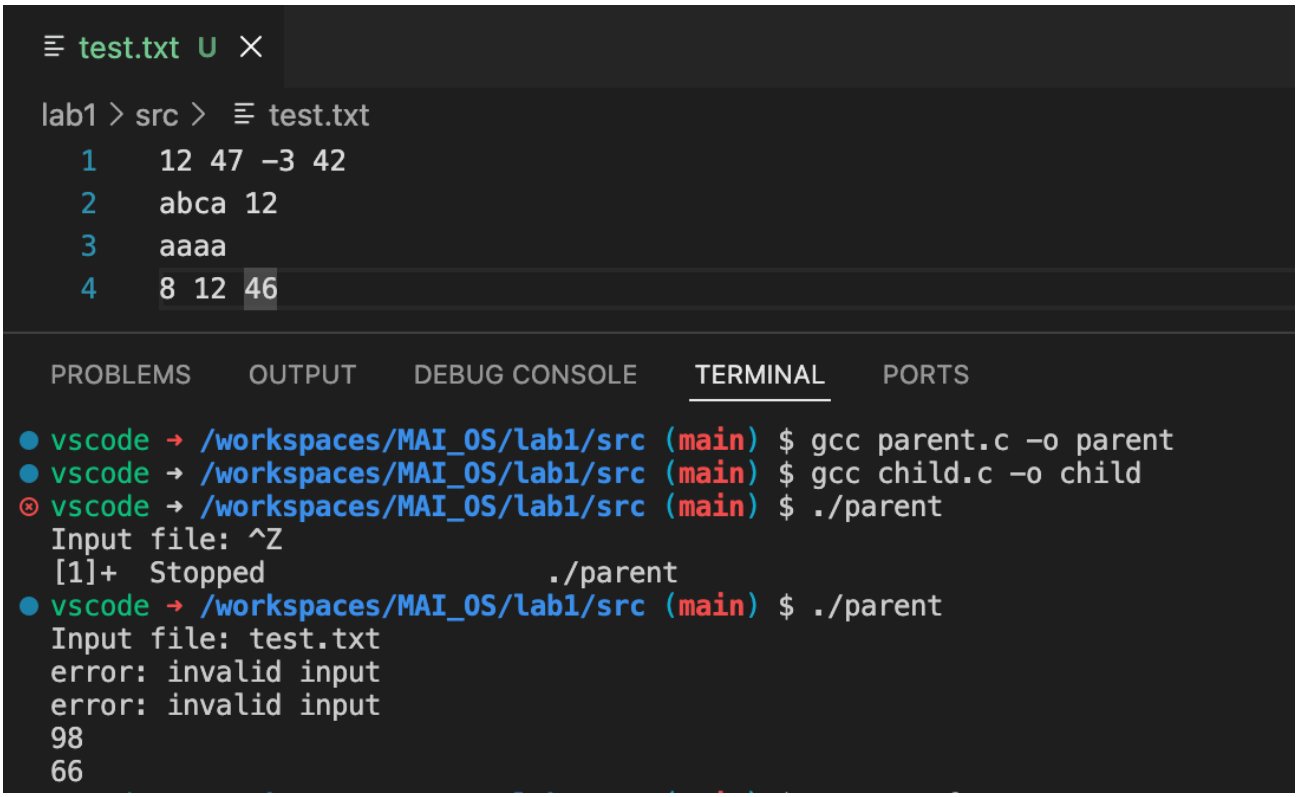
```
    if (!isspace((unsigned char)*p)) {
        only_spaces = 0;
        break;
    }
}
if (only_spaces) {
    return;
}
int sum = 0;
int invalid_input = 0;
char *p = line;

while (*p) {
    // Пропускаем пробелы
    while (*p && isspace((unsigned char)*p)) p++;
    if (!*p) break;

    // Обработка знака
    int is_neg = 1;
    if (*p == '-') {
        is_neg = -1;
        p++;
    }

    // Проверка, что после знака идёт цифра
    if (!isdigit((unsigned char)*p)) {
        invalid_input = 1;
        break;
    }
```

```
        // Чтение числа
        int num = 0;
        while (*p && isdigit((unsigned char)*p)) {
            num = num * 10 + (*p - '0');
            p++;
        }

        // После числа должны быть пробелы или конец строки
        if (*p && !isspace((unsigned char)*p)) {
            invalid_input = 1;
            break;
        }

        sum += num * is_neg;

        // Пропускаем оставшиеся непробельные символы
        while (*p && !isspace((unsigned char)*p)) p++;
    }

    // Вывод результата
    if (invalid_input) {
        const char msg[] = "error: invalid input\n";
        write(STDERR_FILENO, msg, sizeof(msg) - 1);
    } else {
        char out[64];
        int len = snprintf(out, sizeof(out), "%d\n", sum);
        write(STDOUT_FILENO, out, len);
    }
}
```

```
int main() {
    char buf[4096];
    ssize_t n;
    size_t pos = 0;

    while ((n = read(STDIN_FILENO, buf + pos, sizeof(buf) - pos - 1)) > 0) {
        pos += n;
        buf[pos] = '\0';

        char *cur_line = buf;
        char *tmp_line;

        // Обработка полных строк (закончившихся \n)
        while ((tmp_line = strchr(cur_line, '\n'))) {
            *tmp_line = '\0';
            process_line(cur_line);
            cur_line = tmp_line + 1;
        }

        // Обработка последней неполной строки (если есть)
        if (*cur_line) {
            process_line(cur_line);
        }

        pos = 0;
    }

    return 0;
}
```

Протокол работы программы

Тестирование:



Strace:

vscode ➜ /workspaces/MAI_OS/lab1/src (main) $ strace -f ./parent

execve("./parent", ["./parent"], 0xffffffb6a68c8 /* 35 vars */) = 0 < запуск родительского процесса

brk(NULL) = 0xaaaae1fc9000

mmap(NULL, 8192, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_ANONYMOUS, -1, 0) = 0xffffb49b4000

faccessat(AT_FDCWD, "/etc/ld.so.preload", R_OK) = -1 ENOENT (No such file or directory)

openat(AT_FDCWD, "/etc/ld.so.cache", O_RDONLY|O_CLOEXEC) = 3

newfstatat(3, "", {st_mode=S_IFREG|0644, st_size=15048, ...}, AT_EMPTY_PATH) = 0

mmap(NULL, 15048, PROT_READ, MAP_PRIVATE, 3, 0) = 0xffffb49b0000

close(3) = 0

openat(AT_FDCWD, "/lib/aarch64-linux-gnu/libc.so.6", O_RDONLY|O_CLOEXEC) = 3

read(3, "\177ELF\2\1\1\3\0\0\0\0\0\0\0\0\3\0\267\0\1\0\0\0\340u\2\0\0\0\0\0"..., 832) = 832

newfstatat(3, "", {st_mode=S_IFREG|0755, st_size=1637400, ...}, AT_EMPTY_PATH) = 0

mmap(NULL, 1805928, PROT_NONE, MAP_PRIVATE|MAP_ANONYMOUS, -1, 0) = 0xffffb47c6000

mmap(0xffffb47d0000, 1740392, PROT_READ|PROT_EXEC, MAP_PRIVATE|MAP_FIXED|MAP_DENYWRITE, 3, 0) = 0xffffb47d0000

munmap(0xffffb47c6000, 40960)   = 0

munmap(0xffffb4979000, 24168)   = 0

mprotect(0xffffb4958000, 61440, PROT_NONE) = 0

mmap(0xffffb4967000, 24576, PROT_READ|PROT_WRITE,
MAP_PRIVATE|MAP_FIXED|MAP_DENYWRITE, 3, 0x187000) = 0xffffb4967000

mmap(0xffffb496d000, 48744, PROT_READ|PROT_WRITE,
MAP_PRIVATE|MAP_FIXED|MAP_ANONYMOUS, -1, 0) = 0xffffb496d000

close(3)   = 0

set_tid_address(0xffffb49b4f50)   = 5974

set_robust_list(0xffffb49b4f60, 24)   = 0

rseq(0xffffb49b5620, 0x20, 0, 0xd428bc00) = 0

mprotect(0xffffb4967000, 16384, PROT_READ) = 0

mprotect(0xaaaad5661000, 4096, PROT_READ) = 0

mprotect(0xffffb49b9000, 8192, PROT_READ) = 0

prlimit64(0, RLIMIT_STACK, NULL, {rlim_cur=8192*1024, rlim_max=RLIM64_INFINITY}) = 0

munmap(0xffffb49b0000, 15048)   = 0

write(1, "Input file: ", 12Input file:)   = 12

read(0, test.txt

"test.txt\n", 1023)   = 9

pipe2([3, 4], 0)   = 0 < pipe

**clone(child_stack=NULL,
flags=CLONE_CHILD_CLEARTID|CLONE_CHILD_SETTID|SIGCHLDstrace: Process 6076 attached**

, child_tidptr=0xffffb49b4f50) = 6076

[pid 5974] close(4 <unfinished ...>

[pid 6076] set_robust_list(0xffffb49b4f60, 24 <unfinished ...>

[pid 5974] <... close resumed>)   = 0

[pid 5974] read(3, <unfinished ...>

[pid 6076] <... set_robust_list resumed>) = 0

[pid 6076] close(3)   = 0

[pid 6076] openat(AT_FDCWD, "test.txt", O_RDONLY) = 3

[pid 6076] dup3(3, 0, 0)   = 0 < dup2(stdin)

[pid 6076] close(3)   = 0

[pid 6076] dup3(4, 1, 0)   = 1 < dup2(stdout)

[pid 6076] close(4)   = 0

[pid 6076] execve("./child", ["child"], 0xffffeb613658 /* 35 vars */) = 0 < запуск дочернего процесса

[pid 6076] brk(NULL)   = 0xaaaae9cb8000

[pid 6076] mmap(NULL, 8192, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_ANONYMOUS, -1, 0) = 0xffff9d513000

[pid 6076] faccessat(AT_FDCWD, "/etc/ld.so.preload", R_OK) = -1 ENOENT (No such file or directory)

[pid 6076] openat(AT_FDCWD, "/etc/ld.so.cache", O_RDONLY|O_CLOEXEC) = 3

[pid 6076] newfstatat(3, "", {st_mode=S_IFREG|0644, st_size=15048, ...}, AT_EMPTY_PATH) = 0

[pid 6076] mmap(NULL, 15048, PROT_READ, MAP_PRIVATE, 3, 0) = 0xffff9d50f000

[pid 6076] close(3)   = 0

[pid 6076] openat(AT_FDCWD, "/lib/aarch64-linux-gnu/libc.so.6", O_RDONLY|O_CLOEXEC) = 3

[pid 6076] read(3, "\177ELF\2\1\1\3\0\0\0\0\0\0\0\0\3\0\267\0\1\0\0\0\340u\2\0\0\0\0"..., 832) = 832

[pid 6076] newfstatat(3, "", {st_mode=S_IFREG|0755, st_size=1637400, ...}, AT_EMPTY_PATH) = 0

[pid 6076] mmap(NULL, 1805928, PROT_NONE, MAP_PRIVATE|MAP_ANONYMOUS, -1, 0) = 0xffff9d325000

[pid 6076] mmap(0xffff9d330000, 1740392, PROT_READ|PROT_EXEC, MAP_PRIVATE|MAP_FIXED|MAP_DENYWRITE, 3, 0) = 0xffff9d330000

[pid 6076] munmap(0xffff9d325000, 45056) = 0

[pid 6076] munmap(0xffff9d4d9000, 20072) = 0

[pid 6076] mprotect(0xffff9d4b8000, 61440, PROT_NONE) = 0

[pid 6076] mmap(0xffff9d4c7000, 24576, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_FIXED|MAP_DENYWRITE, 3, 0x187000) = 0xffff9d4c7000

[pid 6076] mmap(0xffff9d4cd000, 48744, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_FIXED|MAP_ANONYMOUS, -1, 0) = 0xffff9d4cd000

[pid 6076] close(3)   = 0

[pid 6076] set_tid_address(0xffff9d513f50) = 6076

[pid 6076] set_robust_list(0xffff9d513f60, 24) = 0

[pid 6076] rseq(0xffff9d514620, 0x20, 0, 0xd428bc00) = 0

[pid 6076] mprotect(0xffff9d4c7000, 16384, PROT_READ) = 0

[pid 6076] mprotect(0xaaaace2b1000, 4096, PROT_READ) = 0

[pid 6076] mprotect(0xffff9d518000, 8192, PROT_READ) = 0

[pid 6076] prlimit64(0, RLIMIT_STACK, NULL, {rlim_cur=8192*1024, rlim_max=RLIM64_INFINITY}) = 0

[pid 6076] munmap(0xffff9d50f000, 15048) = 0

[pid 6076] read(0, "12 47 -3 42\n8 12 46\n90 102 -90 2"..., 4095) = 43

[pid 6076] write(1, "98\n", 3 <unfinished ...>

[pid 5974] <... read resumed>"98\n", 4096) = 3

[pid 6076] <... write resumed>) = 3

[pid 5974] write(1, "98\n", 3 <unfinished ...>

[pid 6076] write(1, "66\n", 398

<unfinished ...>

[pid 5974] <... write resumed>) = 3

[pid 6076] <... write resumed>) = 3

[pid 5974] read(3, <unfinished ...>

[pid 6076] write(1, "104\n", 4 <unfinished ...>

[pid 5974] <... read resumed>"66\n", 4096) = 3

[pid 6076] <... write resumed>) = 4

[pid 5974] write(1, "66\n", 3 <unfinished ...>

[pid 6076] write(1, "102\n", 466

<unfinished ...>

[pid 5974] <... write resumed>) = 3

[pid 6076] <... write resumed>) = 4

[pid 5974] read(3, <unfinished ...>

[pid 6076] read(0, <unfinished ...>

[pid 5974] <... read resumed>"104\n102\n", 4096) = 8

[pid 6076] <... read resumed>"", 4095) = 0

[pid 5974] write(1, "104\n102\n", 8 <unfinished ...>

[pid 6076] exit_group(0104

102

<unfinished ...>

[pid 5974] <... write resumed>) = 8

[pid 6076] <... exit_group resumed>) = ?

[pid 5974] read(3, "", 4096) = 0

[pid 5974] close(3 <unfinished ...>

[pid 6076] +++ exited with 0 +++

<... close resumed>) = 0

--- SIGCHLD {si_signo=SIGCHLD, si_code=CLD_EXITED, si_pid=6076, si_uid=1000, si_status=0, si_utime=0, si_stime=0} ---

wait4(-1, NULL, 0, NULL) = 6076

exit_group(0) = ?

+++ exited with 0 +++

Вывод

Лабораторная работа была посвящена применению системных вызовов операционной системы Linux для создания параллельных процессов и организации взаимодействия между ними с помощью pipe. В реализованной программе функционал разделён: родительский процесс осуществляет общее управление, ввод и вывод, в то время как дочерний процесс выполняет проверку входных строк на соответствие числовому формату и последующее вычисление суммы допустимых чисел.