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-216Б-24

Студент: Седов М. А

Преподаватель: Бахарев В.Д.

Оценка: _____

Дата: 19.11.25

Москва, 2025

Постановка задачи

Вариант 2.

Составить и отладить программу на языке Си, осуществляющую работу с процессами и взаимодействие между ними в операционной системе Linux. В результате работы программа (основной процесс) должен создать для решения задачи один или несколько дочерних процессов. Взаимодействие между процессами осуществляется через разделяемую память (shared memory) с использованием семафоров для синхронизации.

Необходимо обрабатывать системные ошибки, которые могут возникнуть в результате работы.

Данная лабораторная работа является модификацией первой лабораторной работы: вместо каналов (pipe) используется разделяемая память и memory mapping. Поскольку блокирующего чтения из каналов больше нет, для синхронизации чтения и записи из shared memory используются семафоры.

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 shared memory (shm_p2c), дочерний процесс при необходимости передает данные в родительский процесс через другой сегмент shared memory (shm_c2p). Результаты своей работы дочерний процесс пишет в созданный им файл.

Пользователь вводит команды вида: «число число число<endline>». 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.

Общий метод и алгоритм решения

Использованные системные вызовы:

● pid_t fork(void) – создает дочерний процесс. Возвращает PID дочернего процесса в родительском процессе, 0 в дочернем процессе, или -1 в случае ошибки.

● int shm_open(const char *name, int oflag, mode_t mode) – создает или открывает именованный объект разделяемой памяти POSIX. Возвращает файловый дескриптор при успехе, -1 при ошибке. Объект создается в /dev/shm/ (в оперативной памяти).

● int shm_unlink(const char *name) – удаляет именованный объект разделяемой памяти. Возвращает 0 при успехе, -1 при ошибке.

● int ftruncate(int fd, off_t length) – устанавливает размер файла (или объекта shared memory). Возвращает 0 при успехе, -1 при ошибке.

● void *mmap(void *addr, size_t length, int prot, int flags, int fd, off_t offset) – отображает файл или объект shared memory в адресное пространство процесса. Возвращает указатель на отображенную область, MAP_FAILED при ошибке.

● int munmap(void *addr, size_t length) – удаляет отображение из адресного пространства. Возвращает 0 при успехе, -1 при ошибке.

● sem_t *sem_open(const char *name, int oflag, mode_t mode, unsigned int value) – создает или открывает именованный семафор POSIX. Возвращает указатель на семафор при успехе, SEM_FAILED при ошибке.

● int sem_wait(sem_t *sem) – уменьшает значение семафора (блокирует, если значение равно 0). Возвращает 0 при успехе, -1 при ошибке.

● int sem_post(sem_t *sem) – увеличивает значение семафора (разблокирует ожидающие процессы). Возвращает 0 при успехе, -1 при ошибке.

● int sem_close(sem_t *sem) – закрывает именованный семафор. Возвращает 0 при успехе, -1 при ошибке.

● int sem_unlink(const char *name) – удаляет именованный семафор. Возвращает 0 при успехе, -1 при ошибке.

● int execv(const char *pathname, char *const argv[]) – заменяет образ памяти текущего процесса новым исполняемым файлом. При успехе не возвращает управление, при ошибке возвращает -1.

● ssize_t read(int fd, void *buf, size_t count) – читает данные из файлового дескриптора. Возвращает количество прочитанных байт, 0 при достижении конца файла, -1 при ошибке.

● ssize_t write(int fd, const void *buf, size_t count) – записывает данные в файловый дескриптор. Возвращает количество записанных байт, -1 при ошибке.

● int open(const char *pathname, int flags, mode_t mode) – открывает или создает файл. Возвращает файловый дескриптор при успехе, -1 при ошибке.

● int close(int fd) – закрывает файловый дескриптор. Возвращает 0 при успехе, -1 при ошибке.

● pid_t waitpid(pid_t pid, int *status, int options) – ожидает завершения дочернего процесса. Возвращает PID завершенного процесса при успехе, -1 при ошибке.

● ssize_t readlink(const char *pathname, char *buf, size_t bufsiz) – читает содержимое символической ссылки. Используется для получения пути к исполняемому файлу через `/proc/self/exe`.

● void _exit(int status) – завершает выполнение процесса немедленно.

Алгоритм работы программы:

1. 1. Родительский процесс (server.c):

- Читает первую строку из стандартного ввода – имя файла для записи результатов

- Генерирует уникальные имена для shared memory и семафоров на основе PID и времени

- Создает два сегмента shared memory: shm_p2c (parent→child) и shm_c2p (child→parent) через shm_open()

- Устанавливает размер через ftruncate() и отображает в память через mmap()

- Создает 4 семафора через sem_open() для синхронизации

- Вызывает fork() для создания дочернего процесса

- В дочернем процессе: закрывает семафоры и отображения, вызывает execv() для запуска client, передавая имена shared memory и семафоров как аргументы

- В родительском процессе: в цикле читает строки из stdin, записывает в shm_p2c, синхронизируется через семафоры, читает ответ из shm_c2p и выводит в stdout

- После завершения ввода освобождает все ресурсы (munmap, shm_unlink, sem_close, sem_unlink) и ожидает завершения дочернего процесса через waitpid()

2. Дочерний процесс (client.c):

- Получает имя файла и имена shared memory/семафоров как аргументы командной строки

- Открывает существующие сегменты shared memory через shm_open() и отображает через mmap()

- Открывает существующие семафоры через sem_open()

- Открывает файл для записи через open() с флагами O_WRONLY | O_CREAT | O_TRUNC

- В цикле: ждет sem_wait(sem_parent_write), читает строку из shm_p2c

- Парсит строку, извлекая числа с плавающей точкой и вычисляя их сумму

- Форматирует результат и записывает его в файл и в shm_c2p

- Синхронизируется с родителем через семафоры

- При получении пустой строки (size == 0) завершает работу

- Закрывает все ресурсы и завершает работу

Код программы

server.c

```c
#define _POSIX_C_SOURCE 200809L
#include <errno.h>
#include <fcntl.h>
#include <limits.h>
#include <stdbool.h>
#include <stdint.h>
#include <stdlib.h>
#include <string.h>
#include <sys/mman.h>
```

```c
#include <sys/wait.h>
#include <semaphore.h>
#include <time.h>
#include <unistd.h>

#define CHILD_PROGRAM_NAME "lab_01_child"
#define MAX_LINE_LENGTH 4096
#define SHM_SIZE (MAX_LINE_LENGTH + 8)

static size_t string_length(const char *text) {
    size_t length = 0;
    while (text[length] != '\0') ++length;
    return length;
}

static void write_all(int fd, const char *buffer, size_t length) {
    while (length > 0) {
        ssize_t written = write(fd, buffer, length);
        if (written < 0) _exit(EXIT_FAILURE);
        buffer += (size_t)written;
        length -= (size_t)written;
    }
}

static void fail(const char *message) {
    write_all(STDERR_FILENO, message, string_length(message));
    _exit(EXIT_FAILURE);
}

static ssize_t read_line(int fd, char *buffer, size_t capacity) {
    if (capacity == 0) return -1;

    size_t offset = 0;
    while (offset + 1 < capacity) {
        char ch;
        ssize_t bytes = read(fd, &ch, 1);
        if (bytes < 0) {
            return -1;
        }
        if (bytes == 0) break;

        buffer[offset++] = ch;
        if (ch == '\n') break;
    }
    buffer[offset] = '\0';
    return (ssize_t)offset;
}

static void trim_trailing_newline(char *line) {
    size_t length = string_length(line);
    if (length == 0) {
        return;
    }
    if (line[length - 1] == '\n') line[length - 1] = '\0';
}

static void build_child_path(char *result, size_t capacity) {
    char executable_path[PATH_MAX];
    ssize_t len = readlink("/proc/self/exe", executable_path, sizeof(executable_path) - 1);
    if (len == -1){
        fail("error: failed to read /proc/self/exe\n");
    }

    executable_path[len] = '\0';
```

```
    while (len > 0 && executable_path[len] != '/') --len;
    if (len == 0) {
        fail("error: executable path is invalid\n");
    }

    executable_path[len] = '\0';

    size_t dir_length = string_length(executable_path);
    size_t name_length = string_length(CHILD_PROGRAM_NAME);
    if (dir_length + 1 + name_length + 1 > capacity) {
        fail("error: child path buffer is too small\n");
    }

    size_t index = 0;
    for (size_t i = 0; i < dir_length; ++i) result[index++] = executable_path[i];
    result[index++] = '/';
    for (size_t i = 0; i < name_length; ++i) result[index++] = CHILD_PROGRAM_NAME[i];

    result[index] = '\0';
}

static void generate_unique_name(char *buffer, size_t capacity, const char *prefix) {
    pid_t pid = getpid();
    struct timespec ts;
    if (clock_gettime(CLOCK_REALTIME, &ts) == -1) {
        fail("error: failed to get time\n");
    }

    int written = snprintf(buffer, capacity, "%s_%d_%ld_%ld", prefix, pid, ts.tv_sec,
ts.tv_nsec);
    if (written < 0 || (size_t)written >= capacity) {
        fail("error: failed to generate unique name\n");
    }
}

static void forward_line(int output_fd, const char *line) {
    size_t length = string_length(line);
    if (length == 0 || line[length - 1] != '\n') {
        write_all(output_fd, line, length);
        write_all(output_fd, "\n", 1);
        return;
    }
    write_all(output_fd, line, length);
}

int main(void) {
    char filename[MAX_LINE_LENGTH];
    ssize_t filename_len = read_line(STDIN_FILENO, filename, sizeof(filename));
    if (filename_len <= 0) {
        fail("error: failed to read filename\n");
    }

    trim_trailing_newline(filename);
    if (string_length(filename) == 0) {
        fail("error: filename must not be empty\n");
    }

    char shm_parent_to_child_name[256];
    char shm_child_to_parent_name[256];
    char sem_parent_write_name[256];
    char sem_child_read_name[256];
    char sem_child_write_name[256];
    char sem_parent_read_name[256];
```

```
    generate_unique_name(shm_parent_to_child_name, sizeof(shm_parent_to_child_name),
"/shm_p2c");
    generate_unique_name(shm_child_to_parent_name, sizeof(shm_child_to_parent_name),
"/shm_c2p");
    generate_unique_name(sem_parent_write_name, sizeof(sem_parent_write_name), "/sem_pw");
    generate_unique_name(sem_child_read_name, sizeof(sem_child_read_name), "/sem_cr");
    generate_unique_name(sem_child_write_name, sizeof(sem_child_write_name), "/sem_cw");
    generate_unique_name(sem_parent_read_name, sizeof(sem_parent_read_name), "/sem_pr");

    int shm_p2c_fd = shm_open(shm_parent_to_child_name, O_CREAT | O_RDWR, 0600);
    if (shm_p2c_fd == -1) {
        fail("error: failed to create parent-to-child shared memory\n");
    }
    if (ftruncate(shm_p2c_fd, SHM_SIZE) == -1) {
        shm_unlink(shm_parent_to_child_name);
        fail("error: failed to truncate parent-to-child shared memory\n");
    }
    void *shm_p2c = mmap(NULL, SHM_SIZE, PROT_READ | PROT_WRITE, MAP_SHARED, shm_p2c_fd, 0);
    if (shm_p2c == MAP_FAILED) {
        close(shm_p2c_fd);
        shm_unlink(shm_parent_to_child_name);
        fail("error: failed to map parent-to-child shared memory\n");
    }
    close(shm_p2c_fd);

    int shm_c2p_fd = shm_open(shm_child_to_parent_name, O_CREAT | O_RDWR, 0600);
    if (shm_c2p_fd == -1) {
        munmap(shm_p2c, SHM_SIZE);
        shm_unlink(shm_parent_to_child_name);
        fail("error: failed to create child-to-parent shared memory\n");
    }
    if (ftruncate(shm_c2p_fd, SHM_SIZE) == -1) {
        munmap(shm_p2c, SHM_SIZE);
        close(shm_c2p_fd);
        shm_unlink(shm_parent_to_child_name);
        shm_unlink(shm_child_to_parent_name);
        fail("error: failed to truncate child-to-parent shared memory\n");
    }
    void *shm_c2p = mmap(NULL, SHM_SIZE, PROT_READ | PROT_WRITE, MAP_SHARED, shm_c2p_fd, 0);
    if (shm_c2p == MAP_FAILED) {
        munmap(shm_p2c, SHM_SIZE);
        close(shm_c2p_fd);
        shm_unlink(shm_parent_to_child_name);
        shm_unlink(shm_child_to_parent_name);
        fail("error: failed to map child-to-parent shared memory\n");
    }
    close(shm_c2p_fd);

    sem_t *sem_parent_write = sem_open(sem_parent_write_name, O_CREAT, 0600, 0);
    sem_t *sem_child_read = sem_open(sem_child_read_name, O_CREAT, 0600, 0);
    sem_t *sem_child_write = sem_open(sem_child_write_name, O_CREAT, 0600, 0);
    sem_t *sem_parent_read = sem_open(sem_parent_read_name, O_CREAT, 0600, 0);

    if (sem_parent_write == SEM_FAILED || sem_child_read == SEM_FAILED ||
        sem_child_write == SEM_FAILED || sem_parent_read == SEM_FAILED) {
        munmap(shm_p2c, SHM_SIZE);
        munmap(shm_c2p, SHM_SIZE);
        shm_unlink(shm_parent_to_child_name);
        shm_unlink(shm_child_to_parent_name);
        if (sem_parent_write != SEM_FAILED) sem_close(sem_parent_write);
        if (sem_child_read != SEM_FAILED) sem_close(sem_child_read);
        if (sem_child_write != SEM_FAILED) sem_close(sem_child_write);
        if (sem_parent_read != SEM_FAILED) sem_close(sem_parent_read);
        sem_unlink(sem_parent_write_name);
        sem_unlink(sem_child_read_name);
```

```
        sem_unlink(sem_child_write_name);
        sem_unlink(sem_parent_read_name);
        fail("error: failed to create semaphores\n");
    }

    pid_t child = fork();
    if (child == -1) {
        munmap(shm_p2c, SHM_SIZE);
        munmap(shm_c2p, SHM_SIZE);
        shm_unlink(shm_parent_to_child_name);
        shm_unlink(shm_child_to_parent_name);
        sem_close(sem_parent_write);
        sem_close(sem_child_read);
        sem_close(sem_child_write);
        sem_close(sem_parent_read);
        sem_unlink(sem_parent_write_name);
        sem_unlink(sem_child_read_name);
        sem_unlink(sem_child_write_name);
        sem_unlink(sem_parent_read_name);
        fail("error: failed to fork\n");
    }

    if (child == 0) {
        sem_close(sem_parent_write);
        sem_close(sem_child_read);
        sem_close(sem_child_write);
        sem_close(sem_parent_read);
        munmap(shm_p2c, SHM_SIZE);
        munmap(shm_c2p, SHM_SIZE);

        char child_path[PATH_MAX];
        build_child_path(child_path, sizeof(child_path));
        char *const args[] = {
            CHILD_PROGRAM_NAME,
            filename,
            shm_parent_to_child_name,
            shm_child_to_parent_name,
            sem_parent_write_name,
            sem_child_read_name,
            sem_child_write_name,
            sem_parent_read_name,
            NULL
        };
        execv(child_path, args);
        fail("error: exec failed\n");
    }

    char line_buffer[MAX_LINE_LENGTH];
    while(true) {
        ssize_t line_length = read_line(STDIN_FILENO, line_buffer, sizeof(line_buffer));
        if (line_length == -1) {
            fail("error: failed to read input line\n");
        }

        if (line_length == 0 || line_buffer[0] == '\n') {
            size_t *size_ptr = (size_t *)shm_p2c;
            *size_ptr = 0;
            if (sem_post(sem_parent_write) == -1) {
                fail("error: failed to post sem_parent_write\n");
            }

            if (sem_wait(sem_child_read) == -1) {
                fail("error: failed to wait sem_child_read\n");
            }
            break;
```

```
        }

        size_t *size_ptr = (size_t *)shm_p2c;
        *size_ptr = (size_t)line_length;
        char *data_ptr = (char *)shm_p2c + sizeof(size_t);
        memcpy(data_ptr, line_buffer, (size_t)line_length);
        data_ptr[line_length] = '\0';

        if (sem_post(sem_parent_write) == -1) {
            fail("error: failed to post sem_parent_write\n");
        }

        if (sem_wait(sem_child_read) == -1) {
            fail("error: failed to wait sem_child_read\n");
        }

        if (sem_wait(sem_child_write) == -1) {
            fail("error: failed to wait sem_child_write\n");
        }

        size_t *resp_size_ptr = (size_t *)shm_c2p;
        size_t resp_size = *resp_size_ptr;
        char *resp_data_ptr = (char *)shm_c2p + sizeof(size_t);

        if (resp_size > 0 && resp_size < MAX_LINE_LENGTH) {
            forward_line(STDOUT_FILENO, resp_data_ptr);
        }

        if (sem_post(sem_parent_read) == -1) {
            fail("error: failed to post sem_parent_read\n");
        }
    }

    munmap(shm_p2c, SHM_SIZE);
    munmap(shm_c2p, SHM_SIZE);
    shm_unlink(shm_parent_to_child_name);
    shm_unlink(shm_child_to_parent_name);
    sem_close(sem_parent_write);
    sem_close(sem_child_read);
    sem_close(sem_child_write);
    sem_close(sem_parent_read);
    sem_unlink(sem_parent_write_name);
    sem_unlink(sem_child_read_name);
    sem_unlink(sem_child_write_name);
    sem_unlink(sem_parent_read_name);

    int status = 0;
    if (waitpid(child, &status, 0) == -1) {
        fail("error: waitpid failed\n");
    }
    if (WIFEXITED(status)) {
        return WEXITSTATUS(status);
    } else {
        return EXIT_FAILURE;
    }
}
```

client.c

```
#define _POSIX_C_SOURCE 200809L
#include <errno.h>
#include <fcntl.h>
#include <stdbool.h>
#include <stdint.h>
#include <stdlib.h>
#include <string.h>
```

```
#include <sys/mman.h>
#include <semaphore.h>
#include <unistd.h>

#define BUFFER_SIZE 4096
#define SHM_SIZE (BUFFER_SIZE + 8)

static size_t string_length(const char *text) {
    size_t length = 0;
    while (text[length] != '\0') ++length;
    return length;
}

static void write_all(int fd, const char *buffer, size_t length) {
    while (length > 0) {
        ssize_t written = write(fd, buffer, length);
        if (written < 0) _exit(EXIT_FAILURE);

        buffer += (size_t)written;
        length -= (size_t)written;
    }
}

static void fail(const char *message) {
    write_all(STDERR_FILENO, message, string_length(message));
    _exit(EXIT_FAILURE);
}

static bool parse_and_sum(char *line, double *result) {
    char *cursor = line;
    double total = 0.0;
    bool has_value = false;
    while (*cursor != '\0') {
        while (*cursor == ' ' || *cursor == '\t') ++cursor;

        if (*cursor == '\0') break;
        errno = 0;
        char *next = NULL;
        double value = strtod(cursor, &next);
        if (cursor == next || errno == ERANGE) return false;

        total += value;
        has_value = true;
        cursor = next;
    }
    if (!has_value) return false;
    *result = total;
    return true;
}

static size_t format_double(double value, char *buffer, size_t capacity) {
    if (capacity == 0) return 0;
    size_t index = 0;
    if (value < 0.0) {
        buffer[index++] = '-';
        value = -value;
        if (index >= capacity) return 0;
    }

    long long integer_part = (long long)value;
    double fractional_part = value - (double)integer_part;

    char digits[32];
    size_t digit_count = 0;
    do {
```

```
        digits[digit_count++] = (char)('0' + (integer_part % 10));
        integer_part /= 10;
    } while (integer_part > 0 && digit_count < sizeof(digits));

    while (digit_count > 0 && index < capacity) {
        buffer[index++] = digits[--digit_count];
    }
    if (index >= capacity) {
        return 0;
    }

    buffer[index++] = '.';
    for (int i = 0; i < 6 && index < capacity; ++i) {
        fractional_part *= 10.0;
        int digit = (int)fractional_part;
        buffer[index++] = (char)('0' + digit);
        fractional_part -= digit;
    }

    size_t end = index;
    while (end > 0 && buffer[end - 1] == '0') {
        --end;
    }
    if (end > 0 && buffer[end - 1] == '.') ++end;

    if (end >= capacity) return 0;
    buffer[end] = '\0';
    return end;
}

int main(int argc, char **argv) {
    if (argc < 8) {
        fail("error: insufficient arguments\n");
    }

    const char *filename = argv[1];
    const char *shm_parent_to_child_name = argv[2];
    const char *shm_child_to_parent_name = argv[3];
    const char *sem_parent_write_name = argv[4];
    const char *sem_child_read_name = argv[5];
    const char *sem_child_write_name = argv[6];
    const char *sem_parent_read_name = argv[7];

    int shm_p2c_fd = shm_open(shm_parent_to_child_name, O_RDWR, 0);
    if (shm_p2c_fd == -1) {
        fail("error: failed to open parent-to-child shared memory\n");
    }
    void *shm_p2c = mmap(NULL, SHM_SIZE, PROT_READ | PROT_WRITE, MAP_SHARED, shm_p2c_fd, 0);
    if (shm_p2c == MAP_FAILED) {
        close(shm_p2c_fd);
        fail("error: failed to map parent-to-child shared memory\n");
    }
    close(shm_p2c_fd);

    int shm_c2p_fd = shm_open(shm_child_to_parent_name, O_RDWR, 0);
    if (shm_c2p_fd == -1) {
        munmap(shm_p2c, SHM_SIZE);
        fail("error: failed to open child-to-parent shared memory\n");
    }
    void *shm_c2p = mmap(NULL, SHM_SIZE, PROT_READ | PROT_WRITE, MAP_SHARED, shm_c2p_fd, 0);
    if (shm_c2p == MAP_FAILED) {
        close(shm_c2p_fd);
        munmap(shm_p2c, SHM_SIZE);
        fail("error: failed to map child-to-parent shared memory\n");
    }
```

```
    close(shm_c2p_fd);

    sem_t *sem_parent_write = sem_open(sem_parent_write_name, 0);
    sem_t *sem_child_read = sem_open(sem_child_read_name, 0);
    sem_t *sem_child_write = sem_open(sem_child_write_name, 0);
    sem_t *sem_parent_read = sem_open(sem_parent_read_name, 0);

    if (sem_parent_write == SEM_FAILED || sem_child_read == SEM_FAILED ||
        sem_child_write == SEM_FAILED || sem_parent_read == SEM_FAILED) {
        munmap(shm_p2c, SHM_SIZE);
        munmap(shm_c2p, SHM_SIZE);
        if (sem_parent_write != SEM_FAILED) sem_close(sem_parent_write);
        if (sem_child_read != SEM_FAILED) sem_close(sem_child_read);
        if (sem_child_write != SEM_FAILED) sem_close(sem_child_write);
        if (sem_parent_read != SEM_FAILED) sem_close(sem_parent_read);
        fail("error: failed to open semaphores\n");
    }

    int file = open(filename, O_WRONLY | O_CREAT | O_TRUNC, 0600);
    if (file == -1) {
        munmap(shm_p2c, SHM_SIZE);
        munmap(shm_c2p, SHM_SIZE);
        sem_close(sem_parent_write);
        sem_close(sem_child_read);
        sem_close(sem_child_write);
        sem_close(sem_parent_read);
        fail("error: failed to open file\n");
    }

    char line[BUFFER_SIZE];
    bool should_continue = true;

    while(should_continue) {
        if (sem_wait(sem_parent_write) == -1) {
            fail("error: failed to wait sem_parent_write\n");
        }

        size_t *size_ptr = (size_t *)shm_p2c;
        size_t line_length = *size_ptr;
        char *data_ptr = (char *)shm_p2c + sizeof(size_t);

        if (line_length == 0) {
            should_continue = false;
            if (sem_post(sem_child_read) == -1) {
                fail("error: failed to post sem_child_read\n");
            }
            break;
        }

        if (line_length >= BUFFER_SIZE) {
            line_length = BUFFER_SIZE - 1;
        }
        memcpy(line, data_ptr, line_length);
        line[line_length] = '\0';

        if (sem_post(sem_child_read) == -1) {
            fail("error: failed to post sem_child_read\n");
        }

        if (line_length > 0 && line[line_length - 1] == '\n') {
            line[line_length - 1] = '\0';
        }

        double sum = 0.0;
        bool valid = parse_and_sum(line, &sum);
```

```
        char response[BUFFER_SIZE];
        size_t response_length = 0;

        if (!valid) {
            const char warning[] = "error: invalid input\n";
            response_length = sizeof(warning) - 1;
            memcpy(response, warning, response_length);
        } else {
            char value_buffer[128];
            size_t value_length = format_double(sum, value_buffer, sizeof(value_buffer));
            if (value_length == 0) {
                fail("error: failed to format result\n");
            }

            const char prefix[] = "sum: ";
            const char newline = '\n';

            write_all(file, prefix, sizeof(prefix) - 1);
            write_all(file, value_buffer, value_length);
            write_all(file, &newline, 1);

            size_t index = 0;
            memcpy(response + index, prefix, sizeof(prefix) - 1);
            index += sizeof(prefix) - 1;
            memcpy(response + index, value_buffer, value_length);
            index += value_length;
            response[index++] = newline;
            response_length = index;
        }

        size_t *resp_size_ptr = (size_t *)shm_c2p;
        *resp_size_ptr = response_length;
        char *resp_data_ptr = (char *)shm_c2p + sizeof(size_t);
        if (response_length > 0) {
            memcpy(resp_data_ptr, response, response_length);
            if (response_length < SHM_SIZE - sizeof(size_t)) {
                resp_data_ptr[response_length] = '\0';
            }
        }

        if (sem_post(sem_child_write) == -1) {
            fail("error: failed to post sem_child_write\n");
        }

        if (sem_wait(sem_parent_read) == -1) {
            fail("error: failed to wait sem_parent_read\n");
        }
    }

    if (close(file) == -1) {
        fail("error: failed to close file\n");
    }

    munmap(shm_p2c, SHM_SIZE);
    munmap(shm_c2p, SHM_SIZE);
    sem_close(sem_parent_write);
    sem_close(sem_child_read);
    sem_close(sem_child_write);
    sem_close(sem_parent_read);

    return EXIT_SUCCESS;
}
```

Протокол работы программы

Тестирование:

Программа была протестирована на различных входных данных. Примеры тестов:

Тест 1: Простые числа

```
$ ./build/lab_01_parent
results.txt
1.5 2.5 3.0
sum: 7.0
10.25 20.75
sum: 31.0
-5.5 10.5
sum: 5.0

$ cat results.txt
sum: 7.0
sum: 31.0
sum: 5.0

```

Тест 2: Множество чисел в одной строке

```
$ ./build/lab_01_parent
output.txt
1.1 2.2 3.3 4.4 5.5 6.6 7.7 8.8 9.9 10.0
sum: 59.5
0.1 0.2 0.3 0.4 0.5
sum: 1.5

$ cat output.txt
sum: 59.5
sum: 1.5
```

Тест 3: Обработка ошибок

```
$ ./build/lab_01_parent
test.txt
1.5 2.5 abc
error: invalid input
3.0 4.0
sum: 7.0
invalid
error: invalid input

$ cat test.txt
sum: 7.0
```

Тест 4: Отрицательные числа и нули

```
$ ./build/lab_01_parent
negative.txt
-10.5 5.5
sum: -5.0
0.0 0.0 0.0
sum: 0.0
-1.1 -2.2 -3.3
```

8. sum: -6.6
9.
10. $ cat negative.txt
11. sum: -5.0
12. sum: 0.0
13. sum: -6.6

Strace:

Для анализа системных вызовов использовалась утилита `strace` с флагом `-f` для отслеживания дочерних процессов. Программа была запущена через WSL:

$ strace -f -o strace_output.txt ./build/lab_01_parent

Ниже представлен фрагмент вывода `strace` с выделенными системными вызовами, используемыми в нашей программе. Написанные ниже строки соответствуют системным вызовам, вызванным непосредственно из нашего кода:

```
632   execve("./build/lab_01_parent", ["./build/lab_01_parent"], 0x7ffc0a75ee28 /* 28 vars
*/) = 0
632   brk(NULL)                         = 0x59d97026d000
632   arch_prctl(0x3001 /* ARCH_??? */, 0x7ffd7512d070) = -1 EINVAL (Invalid argument)
632   mmap(NULL, 8192, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_ANONYMOUS, -1, 0) =
0x7f836bd60000
632   access("/etc/ld.so.preload", R_OK) = -1 ENOENT (No such file or directory)
632   openat(AT_FDCWD, "/etc/ld.so.cache", O_RDONLY|O_CLOEXEC) = 3
632   newfstatat(3, "", {st_mode=S_IFREG|0644, st_size=21124, ...}, AT_EMPTY_PATH) = 0
632   mmap(NULL, 21124, PROT_READ, MAP_PRIVATE, 3, 0) = 0x7f836bd5a000
632   close(3)                          = 0
632   openat(AT_FDCWD, "/lib/x86_64-linux-gnu/libc.so.6", O_RDONLY|O_CLOEXEC) = 3
632   read(3, "\177ELF\2\1\1\3\0\0\0\0\0\0\0\0\3\0>\0\1\0\0\0P\237\2\0\0\0\0\0"..., 832) =
832
632   pread64(3, "\6\0\0\0\4\0\0\0@\0\0\0\0\0\0\0@\0\0\0\0\0\0\0@\0\0\0\0\0\0\0"..., 784, 64)
= 784
632   pread64(3, "\4\0\0\0 \0\0\0\5\0\0\0GNU\0\2\0\0\300\4\0\0\0\3\0\0\0\0\0\0\0"..., 48,
848) = 48
632   pread64(3,
"\4\0\0\0\24\0\0\0\3\0\0\0GNU\0O{\f\225\\=\201\327\312\301P\32$\230\266\235"..., 68, 896) =
68
632   newfstatat(3, "", {st_mode=S_IFREG|0755, st_size=2220400, ...}, AT_EMPTY_PATH) = 0
632   pread64(3, "\6\0\0\0\4\0\0\0@\0\0\0\0\0\0\0@\0\0\0\0\0\0\0@\0\0\0\0\0\0\0"..., 784, 64)
= 784
632   mmap(NULL, 2264656, PROT_READ, MAP_PRIVATE|MAP_DENYWRITE, 3, 0) = 0x7f836ba00000
632   mprotect(0x7f836ba28000, 2023424, PROT_NONE) = 0
632   mmap(0x7f836ba28000, 1658880, PROT_READ|PROT_EXEC, MAP_PRIVATE|MAP_FIXED|MAP_DENYWRITE,
3, 0x28000) = 0x7f836ba28000
632   mmap(0x7f836bbbd000, 360448, PROT_READ, MAP_PRIVATE|MAP_FIXED|MAP_DENYWRITE, 3,
0x1bd000) = 0x7f836bbbd000
632   mmap(0x7f836bc16000, 24576, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_FIXED|MAP_DENYWRITE,
3, 0x215000) = 0x7f836bc16000
632   mmap(0x7f836bc1c000, 52816, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_FIXED|MAP_ANONYMOUS,
-1, 0) = 0x7f836bc1c000
632   close(3)                          = 0
632   mmap(NULL, 12288, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_ANONYMOUS, -1, 0) =
0x7f836bd57000
632   arch_prctl(ARCH_SET_FS, 0x7f836bd57740) = 0
632   set_tid_address(0x7f836bd57a10)   = 632
632   set_robust_list(0x7f836bd57a20, 24) = 0
632   rseq(0x7f836bd580e0, 0x20, 0, 0x53053053) = 0
632   mprotect(0x7f836bc16000, 16384, PROT_READ) = 0
632   mprotect(0x59d9638ff000, 4096, PROT_READ) = 0
632   mprotect(0x7f836bd9a000, 8192, PROT_READ) = 0
```

```
632   prlimit64(0, RLIMIT_STACK, NULL, {rlim_cur=8192*1024, rlim_max=RLIM64_INFINITY}) = 0
632   munmap(0x7f836bd5a000, 21124)     = 0
632   read(0, "t", 1)                   = 1
632   read(0, "e", 1)                   = 1
632   read(0, "s", 1)                   = 1
632   read(0, "t", 1)                   = 1
632   read(0, "_", 1)                   = 1
632   read(0, "o", 1)                   = 1
632   read(0, "u", 1)                   = 1
632   read(0, "t", 1)                   = 1
632   read(0, "p", 1)                   = 1
632   read(0, "u", 1)                   = 1
632   read(0, "t", 1)                   = 1
632   read(0, ".", 1)                   = 1
632   read(0, "t", 1)                   = 1
632   read(0, "x", 1)                   = 1
632   read(0, "t", 1)                   = 1
632   read(0, "\n", 1)                  = 1
632   getpid()                          = 632
632   getpid()                          = 632
632   getpid()                          = 632
632   getpid()                          = 632
632   getpid()                          = 632
632   getpid()                          = 632
632   openat(AT_FDCWD, "/dev/shm/shm_p2c_632_1766654089_612057932",
O_RDWR|O_CREAT|O_NOFOLLOW|O_CLOEXEC, 0600) = 3
632   ftruncate(3, 4104)                = 0
632   mmap(NULL, 4104, PROT_READ|PROT_WRITE, MAP_SHARED, 3, 0) = 0x7f836bd5e000
632   close(3)                          = 0
632   openat(AT_FDCWD, "/dev/shm/shm_c2p_632_1766654089_612476072",
O_RDWR|O_CREAT|O_NOFOLLOW|O_CLOEXEC, 0600) = 3
632   ftruncate(3, 4104)                = 0
632   mmap(NULL, 4104, PROT_READ|PROT_WRITE, MAP_SHARED, 3, 0) = 0x7f836bd5c000
632   close(3)                          = 0
632   openat(AT_FDCWD, "/dev/shm/sem.sem_pw_632_1766654089_612896737", O_RDWR|O_NOFOLLOW) = -
1 ENOENT (No such file or directory)
632   getrandom("\x06\x74\xc1\x70\x15\x5b\x81\x85", 8, GRND_NONBLOCK) = 8
632   newfstatat(AT_FDCWD, "/dev/shm/sem.CFGjNM", 0x7ffd7512a6e0, AT_SYMLINK_NOFOLLOW) = -1
ENOENT (No such file or directory)
632   openat(AT_FDCWD, "/dev/shm/sem.CFGjNM", O_RDWR|O_CREAT|O_EXCL, 0600) = 3
632   write(3, "\0\0\0\0\0\0\0\0\200\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0", 32) = 32
632   mmap(NULL, 32, PROT_READ|PROT_WRITE, MAP_SHARED, 3, 0) = 0x7f836bd99000
632   link("/dev/shm/sem.CFGjNM", "/dev/shm/sem.sem_pw_632_1766654089_612896737") = 0
632   newfstatat(3, "", {st_mode=S_IFREG|0600, st_size=32, ...}, AT_EMPTY_PATH) = 0
632   getrandom("\x86\x85\x3a\xdd\xe0\xa9\x89\x8a", 8, GRND_NONBLOCK) = 8
632   brk(NULL)                         = 0x59d97026d000
632   brk(0x59d97028e000)               = 0x59d97028e000
632   unlink("/dev/shm/sem.CFGjNM")     = 0
632   close(3)                          = 0
632   openat(AT_FDCWD, "/dev/shm/sem.sem_cr_632_1766654089_613290594", O_RDWR|O_NOFOLLOW) = -
1 ENOENT (No such file or directory)
632   getrandom("\x09\x13\xe7\x34\x25\x4b\x7e\x27", 8, GRND_NONBLOCK) = 8
632   newfstatat(AT_FDCWD, "/dev/shm/sem.fsqP7C", 0x7ffd7512a6e0, AT_SYMLINK_NOFOLLOW) = -1
ENOENT (No such file or directory)
632   openat(AT_FDCWD, "/dev/shm/sem.fsqP7C", O_RDWR|O_CREAT|O_EXCL, 0600) = 3
632   write(3, "\0\0\0\0\0\0\0\0\200\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0", 32) = 32
632   mmap(NULL, 32, PROT_READ|PROT_WRITE, MAP_SHARED, 3, 0) = 0x7f836bd5b000
632   link("/dev/shm/sem.fsqP7C", "/dev/shm/sem.sem_cr_632_1766654089_613290594") = 0
632   newfstatat(3, "", {st_mode=S_IFREG|0600, st_size=32, ...}, AT_EMPTY_PATH) = 0
632   unlink("/dev/shm/sem.fsqP7C")     = 0
632   close(3)                          = 0
632   openat(AT_FDCWD, "/dev/shm/sem.sem_cw_632_1766654089_613947115", O_RDWR|O_NOFOLLOW) = -
1 ENOENT (No such file or directory)
632   getrandom("\x92\xbe\x5f\x56\xa7\x3d\x5e\xa3", 8, GRND_NONBLOCK) = 8
632   newfstatat(AT_FDCWD, "/dev/shm/sem.Wd4vZg", 0x7ffd7512a6e0, AT_SYMLINK_NOFOLLOW) = -1
```

```
ENOENT (No such file or directory)
632   openat(AT_FDCWD, "/dev/shm/sem.Wd4vZg", O_RDWR|O_CREAT|O_EXCL, 0600) = 3
632   write(3, "\0\0\0\0\0\0\0\200\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0", 32) = 32
632   mmap(NULL, 32, PROT_READ|PROT_WRITE, MAP_SHARED, 3, 0) = 0x7f836bd5a000
632   link("/dev/shm/sem.Wd4vZg", "/dev/shm/sem.sem_cw_632_1766654089_613947115") = 0
632   newfstatat(3, "", {st_mode=S_IFREG|0600, st_size=32, ...}, AT_EMPTY_PATH) = 0
632   unlink("/dev/shm/sem.Wd4vZg")     = 0
632   close(3)                          = 0
632   openat(AT_FDCWD, "/dev/shm/sem.sem_pr_632_1766654089_614501120", O_RDWR|O_NOFOLLOW) = -
1 ENOENT (No such file or directory)
632   getrandom("\xd3\xcf\x75\xe9\x1a\x36\x55\xe8", 8, GRND_NONBLOCK) = 8
632   newfstatat(AT_FDCWD, "/dev/shm/sem.bP3DVG", 0x7ffd7512a6e0, AT_SYMLINK_NOFOLLOW) = -1
ENOENT (No such file or directory)
632   openat(AT_FDCWD, "/dev/shm/sem.bP3DVG", O_RDWR|O_CREAT|O_EXCL, 0600) = 3
632   write(3, "\0\0\0\0\0\0\0\200\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0", 32) = 32
632   mmap(NULL, 32, PROT_READ|PROT_WRITE, MAP_SHARED, 3, 0) = 0x7f836bd56000
632   link("/dev/shm/sem.bP3DVG", "/dev/shm/sem.sem_pr_632_1766654089_614501120") = 0
632   newfstatat(3, "", {st_mode=S_IFREG|0600, st_size=32, ...}, AT_EMPTY_PATH) = 0
632   unlink("/dev/shm/sem.bP3DVG")     = 0
632   close(3)                          = 0
632   clone(child_stack=NULL, flags=CLONE_CHILD_CLEARTID|CLONE_CHILD_SETTID|SIGCHLD,
child_tidptr=0x7f836bd57a10) = 633
633   set_robust_list(0x7f836bd57a20, 24 <unfinished ...>
632   read(0,  <unfinished ...>
633   <... set_robust_list resumed>)    = 0
632   <... read resumed>"2", 1)         = 1
633   munmap(0x7f836bd99000, 32 <unfinished ...>
632   read(0,  <unfinished ...>
633   <... munmap resumed>)             = 0
632   <... read resumed>".", 1)         = 1
633   munmap(0x7f836bd5b000, 32 <unfinished ...>
632   read(0,  <unfinished ...>
633   <... munmap resumed>)             = 0
633   munmap(0x7f836bd5a000, 32 <unfinished ...>
632   <... read resumed>"0", 1)         = 1
633   <... munmap resumed>)             = 0
632   read(0,  <unfinished ...>
633   munmap(0x7f836bd56000, 32 <unfinished ...>
632   <... read resumed>" ", 1)         = 1
633   <... munmap resumed>)             = 0
632   read(0,  <unfinished ...>
633   munmap(0x7f836bd5e000, 4104 <unfinished ...>
632   <... read resumed>"2", 1)         = 1
633   <... munmap resumed>)             = 0
632   read(0,  <unfinished ...>
633   munmap(0x7f836bd5c000, 4104 <unfinished ...>
632   <... read resumed>".", 1)         = 1
633   <... munmap resumed>)             = 0
632   read(0,  <unfinished ...>
633   readlink("/proc/self/exe",  <unfinished ...>
632   <... read resumed>"5", 1)         = 1
633   <... readlink resumed>"/mnt/c/MAI/3_sem/OS/OS_LAB/OS_LA"..., 4095) = 57
632   read(0,  <unfinished ...>
633   execve("/mnt/c/MAI/3_sem/OS/OS_LAB/OS_LAB_No3/build/lab_01_child", ["lab_01_child",
"test_output.txt", "/shm_p2c_632_1766654089_61205793"...,
"/shm_c2p_632_1766654089_61247607"..., "/sem_pw_632_1766654089_612896737",
"/sem_cr_632_1766654089_613290594", "/sem_cw_632_1766654089_613947115",
"/sem_pr_632_1766654089_614501120"], 0x7ffd7512d248 /* 28 vars */ <unfinished ...>
632   <... read resumed>" ", 1)         = 1
632   read(0, "3", 1)                   = 1
632   read(0, ".", 1)                   = 1
632   read(0, "5", 1)                   = 1
632   read(0,  <unfinished ...>
633   <... execve resumed>)             = 0
632   <... read resumed>"\n", 1)        = 1
```

```
633   brk(NULL <unfinished ...>
632   futex(0x7f836bd5b000, FUTEX_WAIT_BITSET|FUTEX_CLOCK_REALTIME, 0, NULL,
FUTEX_BITSET_MATCH_ANY <unfinished ...>
633   <... brk resumed>)                = 0x5b2364149000
633   arch_prctl(0x3001 /* ARCH_??? */, 0x7fff9e2fec90) = -1 EINVAL (Invalid argument)
633   mmap(NULL, 8192, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_ANONYMOUS, -1, 0) =
0x751cb056d000
633   access("/etc/ld.so.preload", R_OK) = -1 ENOENT (No such file or directory)
633   openat(AT_FDCWD, "/etc/ld.so.cache", O_RDONLY|O_CLOEXEC) = 3
633   newfstatat(3, "", {st_mode=S_IFREG|0644, st_size=21124, ...}, AT_EMPTY_PATH) = 0
633   mmap(NULL, 21124, PROT_READ, MAP_PRIVATE, 3, 0) = 0x751cb0567000
633   close(3)                          = 0
633   openat(AT_FDCWD, "/lib/x86_64-linux-gnu/libc.so.6", O_RDONLY|O_CLOEXEC) = 3
633   read(3, "\177ELF\2\1\1\3\0\0\0\0\0\0\0\0\3\0>\0\1\0\0\0P\237\2\0\0\0\0\0"..., 832) =
832
633   pread64(3, "\6\0\0\0\4\0\0\0@\0\0\0\0\0\0\0@\0\0\0\0\0\0\0@\0\0\0\0\0\0\0"..., 784, 64)
= 784
633   pread64(3, "\4\0\0\0 \0\0\0\5\0\0\0GNU\0\2\0\0\300\4\0\0\0\3\0\0\0\0\0\0\0"..., 48,
848) = 48
633   pread64(3,
"\4\0\0\0\24\0\0\0\3\0\0\0GNU\0O{\f\225\\=\201\327\312\301P\32$\230\266\235"..., 68, 896) =
68
633   newfstatat(3, "", {st_mode=S_IFREG|0755, st_size=2220400, ...}, AT_EMPTY_PATH) = 0
633   pread64(3, "\6\0\0\0\4\0\0\0@\0\0\0\0\0\0\0@\0\0\0\0\0\0\0@\0\0\0\0\0\0\0"..., 784, 64)
= 784
633   mmap(NULL, 2264656, PROT_READ, MAP_PRIVATE|MAP_DENYWRITE, 3, 0) = 0x751cb0200000
633   mprotect(0x751cb0228000, 2023424, PROT_NONE) = 0
633   mmap(0x751cb0228000, 1658880, PROT_READ|PROT_EXEC, MAP_PRIVATE|MAP_FIXED|MAP_DENYWRITE,
3, 0x28000) = 0x751cb0228000
633   mmap(0x751cb03bd000, 360448, PROT_READ, MAP_PRIVATE|MAP_FIXED|MAP_DENYWRITE, 3,
0x1bd000) = 0x751cb03bd000
633   mmap(0x751cb0416000, 24576, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_FIXED|MAP_DENYWRITE,
3, 0x215000) = 0x751cb0416000
633   mmap(0x751cb041c000, 52816, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_FIXED|MAP_ANONYMOUS,
-1, 0) = 0x751cb041c000
633   close(3)                          = 0
633   mmap(NULL, 12288, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_ANONYMOUS, -1, 0) =
0x751cb0564000
633   arch_prctl(ARCH_SET_FS, 0x751cb0564740) = 0
633   set_tid_address(0x751cb0564a10)   = 633
633   set_robust_list(0x751cb0564a20, 24) = 0
633   rseq(0x751cb05650e0, 0x20, 0, 0x53053053) = 0
633   mprotect(0x751cb0416000, 16384, PROT_READ) = 0
633   mprotect(0x5b2348bc2000, 4096, PROT_READ) = 0
633   mprotect(0x751cb05a7000, 8192, PROT_READ) = 0
633   prlimit64(0, RLIMIT_STACK, NULL, {rlim_cur=8192*1024, rlim_max=RLIM64_INFINITY}) = 0
633   munmap(0x751cb0567000, 21124)     = 0
633   openat(AT_FDCWD, "/dev/shm/shm_p2c_632_1766654089_612057932",
O_RDWR|O_NOFOLLOW|O_CLOEXEC) = 3
633   mmap(NULL, 4104, PROT_READ|PROT_WRITE, MAP_SHARED, 3, 0) = 0x751cb056b000
633   close(3)                          = 0
633   openat(AT_FDCWD, "/dev/shm/shm_c2p_632_1766654089_612476072",
O_RDWR|O_NOFOLLOW|O_CLOEXEC) = 3
633   mmap(NULL, 4104, PROT_READ|PROT_WRITE, MAP_SHARED, 3, 0) = 0x751cb0569000
633   close(3)                          = 0
633   openat(AT_FDCWD, "/dev/shm/sem.sem_pw_632_1766654089_612896737", O_RDWR|O_NOFOLLOW) = 3
633   newfstatat(3, "", {st_mode=S_IFREG|0600, st_size=32, ...}, AT_EMPTY_PATH) = 0
633   getrandom("\x93\x79\x5d\xc2\x8e\xfc\x9a\xa3", 8, GRND_NONBLOCK) = 8
633   brk(NULL)                         = 0x5b2364149000
633   brk(0x5b236416a000)               = 0x5b236416a000
633   mmap(NULL, 32, PROT_READ|PROT_WRITE, MAP_SHARED, 3, 0) = 0x751cb05a6000
633   close(3)                          = 0
633   openat(AT_FDCWD, "/dev/shm/sem.sem_cr_632_1766654089_613290594", O_RDWR|O_NOFOLLOW) = 3
633   newfstatat(3, "", {st_mode=S_IFREG|0600, st_size=32, ...}, AT_EMPTY_PATH) = 0
633   mmap(NULL, 32, PROT_READ|PROT_WRITE, MAP_SHARED, 3, 0) = 0x751cb0568000
```

```
633   close(3)                          = 0
633   openat(AT_FDCWD, "/dev/shm/sem.sem_cw_632_1766654089_613947115", O_RDWR|O_NOFOLLOW) = 3
633   newfstatat(3, "", {st_mode=S_IFREG|0600, st_size=32, ...}, AT_EMPTY_PATH) = 0
633   mmap(NULL, 32, PROT_READ|PROT_WRITE, MAP_SHARED, 3, 0) = 0x751cb0567000
633   close(3)                          = 0
633   openat(AT_FDCWD, "/dev/shm/sem.sem_pr_632_1766654089_614501120", O_RDWR|O_NOFOLLOW) = 3
633   newfstatat(3, "", {st_mode=S_IFREG|0600, st_size=32, ...}, AT_EMPTY_PATH) = 0
633   mmap(NULL, 32, PROT_READ|PROT_WRITE, MAP_SHARED, 3, 0) = 0x751cb0563000
633   close(3)                          = 0
633   openat(AT_FDCWD, "test_output.txt", O_WRONLY|O_CREAT|O_TRUNC, 0600) = 3
633   futex(0x751cb0568000, FUTEX_WAKE, 1 <unfinished ...>
632   <... futex resumed>)              = 0
633   <... futex resumed>)              = 1
632   futex(0x7f836bd5a000, FUTEX_WAIT_BITSET|FUTEX_CLOCK_REALTIME, 0, NULL,
FUTEX_BITSET_MATCH_ANY <unfinished ...>
633   write(3, "sum: ", 5)              = 5
633   write(3, "8.0", 3)                = 3
633   write(3, "\n", 1)                 = 1
633   futex(0x751cb0567000, FUTEX_WAKE, 1 <unfinished ...>
632   <... futex resumed>)              = 0
633   <... futex resumed>)              = 1
632   write(1, "sum: 8.0\n", 9 <unfinished ...>
633   futex(0x751cb0563000, FUTEX_WAIT_BITSET|FUTEX_CLOCK_REALTIME, 0, NULL,
FUTEX_BITSET_MATCH_ANY <unfinished ...>
632   <... write resumed>)              = 9
632   futex(0x7f836bd56000, FUTEX_WAKE, 1) = 1
633   <... futex resumed>)              = 0
632   read(0,  <unfinished ...>
633   futex(0x751cb05a6000, FUTEX_WAIT_BITSET|FUTEX_CLOCK_REALTIME, 0, NULL,
FUTEX_BITSET_MATCH_ANY <unfinished ...>
632   <... read resumed>"1", 1)         = 1
632   read(0, "0", 1)                   = 1
632   read(0, " ", 1)                   = 1
632   read(0, "2", 1)                   = 1
632   read(0, "0", 1)                   = 1
632   read(0, " ", 1)                   = 1
632   read(0, "3", 1)                   = 1
632   read(0, "0", 1)                   = 1
632   read(0, "\n", 1)                  = 1
632   futex(0x7f836bd99000, FUTEX_WAKE, 1) = 1
633   <... futex resumed>)              = 0
632   futex(0x7f836bd5b000, FUTEX_WAIT_BITSET|FUTEX_CLOCK_REALTIME, 0, NULL,
FUTEX_BITSET_MATCH_ANY <unfinished ...>
633   futex(0x751cb0568000, FUTEX_WAKE, 1 <unfinished ...>
632   <... futex resumed>)              = -1 EAGAIN (Resource temporarily unavailable)
633   <... futex resumed>)              = 0
632   futex(0x7f836bd5a000, FUTEX_WAIT_BITSET|FUTEX_CLOCK_REALTIME, 0, NULL,
FUTEX_BITSET_MATCH_ANY <unfinished ...>
633   write(3, "sum: ", 5)              = 5
633   write(3, "60.0", 4)               = 4
633   write(3, "\n", 1)                 = 1
633   futex(0x751cb0567000, FUTEX_WAKE, 1 <unfinished ...>
632   <... futex resumed>)              = 0
633   <... futex resumed>)              = 1
632   write(1, "sum: 60.0\n", 10 <unfinished ...>
633   futex(0x751cb0563000, FUTEX_WAIT_BITSET|FUTEX_CLOCK_REALTIME, 0, NULL,
FUTEX_BITSET_MATCH_ANY <unfinished ...>
632   <... write resumed>)              = 10
632   futex(0x7f836bd56000, FUTEX_WAKE, 1) = 1
633   <... futex resumed>)              = 0
632   read(0,  <unfinished ...>
633   futex(0x751cb05a6000, FUTEX_WAIT_BITSET|FUTEX_CLOCK_REALTIME, 0, NULL,
FUTEX_BITSET_MATCH_ANY <unfinished ...>
632   <... read resumed>"\n", 1)        = 1
632   futex(0x7f836bd99000, FUTEX_WAKE, 1) = 1
```

```
633   <... futex resumed>)              = 0
632   futex(0x7f836bd5b000, FUTEX_WAIT_BITSET|FUTEX_CLOCK_REALTIME, 0, NULL,
FUTEX_BITSET_MATCH_ANY <unfinished ...>
633   futex(0x751cb0568000, FUTEX_WAKE, 1 <unfinished ...>
632   <... futex resumed>)              = -1 EAGAIN (Resource temporarily unavailable)
633   <... futex resumed>)              = 0
632   munmap(0x7f836bd5e000, 4104 <unfinished ...>
633   close(3 <unfinished ...>
632   <... munmap resumed>)             = 0
632   munmap(0x7f836bd5c000, 4104)      = 0
633   <... close resumed>)              = 0
632   unlink("/dev/shm/shm_p2c_632_1766654089_612057932" <unfinished ...>
633   munmap(0x751cb056b000, 4104 <unfinished ...>
632   <... unlink resumed>)             = 0
633   <... munmap resumed>)             = 0
632   unlink("/dev/shm/shm_c2p_632_1766654089_612476072" <unfinished ...>
633   munmap(0x751cb0569000, 4104 <unfinished ...>
632   <... unlink resumed>)             = 0
633   <... munmap resumed>)             = 0
632   munmap(0x7f836bd99000, 32 <unfinished ...>
633   munmap(0x751cb05a6000, 32 <unfinished ...>
632   <... munmap resumed>)             = 0
633   <... munmap resumed>)             = 0
632   munmap(0x7f836bd5b000, 32 <unfinished ...>
633   munmap(0x751cb0568000, 32 <unfinished ...>
632   <... munmap resumed>)             = 0
633   <... munmap resumed>)             = 0
632   munmap(0x7f836bd5a000, 32 <unfinished ...>
633   munmap(0x751cb0567000, 32 <unfinished ...>
632   <... munmap resumed>)             = 0
633   <... munmap resumed>)             = 0
632   munmap(0x7f836bd56000, 32 <unfinished ...>
633   munmap(0x751cb0563000, 32 <unfinished ...>
632   <... munmap resumed>)             = 0
633   <... munmap resumed>)             = 0
632   unlink("/dev/shm/sem.sem_pw_632_1766654089_612896737" <unfinished ...>
633   exit_group(0 <unfinished ...>
632   <... unlink resumed>)             = 0
633   <... exit_group resumed>)         = ?
632   unlink("/dev/shm/sem.sem_cr_632_1766654089_613290594") = 0
633   +++ exited with 0 +++
632   --- SIGCHLD {si_signo=SIGCHLD, si_code=CLD_EXITED, si_pid=633, si_uid=1000,
si_status=0, si_utime=0, si_stime=0} ---
632   unlink("/dev/shm/sem.sem_cw_632_1766654089_613947115") = 0
632   unlink("/dev/shm/sem.sem_pr_632_1766654089_614501120") = 0
632   wait4(633, [{WIFEXITED(s) && WEXITSTATUS(s) == 0}], 0, NULL) = 633
632   exit_group(0)                     = ?
632   +++ exited with 0 +++
```

Ключевые системные вызовы из нашего кода:

1. clock_gettime(CLOCK_REALTIME, ...) – получение текущего времени для генерации уникальных имен shared memory и семафоров

2. shm_open(..., O_CREAT|O_RDWR, 0600) – создание именованного объекта разделяемой памяти в /dev/shm/

3. ftruncate(fd, 4104) – установка размера объекта shared memory

4. mmap(NULL, size, PROT_READ|PROT_WRITE, MAP_SHARED, fd, 0) – отображение shared memory в адресное пространство процесса

5. shm_open(..., O_RDWR, 0) – открытие существующего объекта shared memory (в дочернем процессе)

6. sem_open(..., O_CREAT, 0600, 0) – создание именованных семафоров с начальным значением 0

7. sem_open(..., 0) – открытие существующих семафоров (в дочернем процессе)

8. sem_wait(sem) – ожидание (декремент) семафора, блокируется если значение равно 0

9. sem_post(sem) – сигнализация (инкремент) семафора, разблокирует ожидающие процессы

10. clone(...) – создание дочернего процесса (fork)

11. execve(...) – замена образа процесса на дочернюю программу

12. openat(AT_FDCWD, filename, O_WRONLY|O_CREAT|O_TRUNC, 0600) – открытие файла для записи результатов

13. write(fd, ...) – запись данных в файл и stdout

14. munmap(addr, size) – удаление отображения shared memory из адресного пространства

15. shm_unlink(name) – удаление именованного объекта shared memory

16. sem_close(sem) – закрытие семафора

17. sem_unlink(name) – удаление именованного семафора

18. wait4(pid, ...) – ожидание завершения дочернего процесса (waitpid)

19. exit_group(0) – завершение процесса

Все системные вызовы проверяются на ошибки: возвращаемые значения сравниваются с -1 (или 0 для некоторых вызовов), и при ошибке программа выводит сообщение и завершается.

Вывод

В ходе выполнения лабораторной работы была реализована программа для межпроцессного взаимодействия между родительским и дочерним процессами с использованием разделяемой памяти (shared memory) и семафоров POSIX вместо каналов (pipe).

- Реализовано корректное использование системных вызовов shm_open(), shm_unlink(), ftruncate(), mmap(), munmap() для работы с разделяемой памятью

- Реализована синхронизация доступа к shared memory с использованием 4 именованных семафоров (sem_open, sem_wait, sem_post, sem_close, sem_unlink)

- Имена shared memory и семафоров генерируются уникальными для каждого запуска программы на основе PID процесса и текущего времени

- Все системные вызовы проверяются на ошибки, что обеспечивает надежность работы программы

- Программа корректно обрабатывает различные входные данные, включая отрицательные числа, нули и некорректный ввод

При выполнении работы были изучены механизмы создания именованных объектов разделяемой памяти, отображения памяти через mmap(), а также организации синхронизации между процессами с использованием семафоров POSIX. Программа успешно прошла тестирование и готова к демонстрации.