Московский Авиационный Институт

(Национальный Исследовательский Университет)

Институт №8 “Компьютерные науки и прикладная математика”

Кафедра №806 “Вычислительная математика и программирование”

Лабораторная работа №3 по курсу

«Операционные системы»

Группа: М8О-211Б-23

Студент:Лапин Д.А.

Преподаватель: Бахарев В.Д.

Оценка: _____

Дата: 08.01.24

Москва, 2024

Постановка задачи

Вариант 16.

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 строкой пользователь в консоль родительского процесса вводит имя файла, которое будет использовано для открытия File с таким именем на запись. Перенаправление стандартных потоков ввода-вывода показано на картинке выше. Родительский и дочерний процесс должны быть представлены разными программами. Родительский процесс принимает от пользователя строки произвольной длины и пересылает их в shm_data. Процесс child проверяет строки на валидность правилу. Если строка соответствует правилу, то она выводится в стандартный поток вывода дочернего процесса, иначе в shm_error выводится информация об ошибке. Родительский процесс полученные от child ошибки выводит в стандартный поток вывода. Правило проверки: строка должна оканчиваться на «.» или «;»

Общий метод и алгоритм решения

Использованные системные вызовы:

1. pid_t fork(void); – создает новый дочерний процесс, который является копией родительского. В родительском процессе возвращает PID дочернего процесса, а в дочернем возвращает 0.

2. int shm_open(const char *__name, int __oflag, mode_t __mode) - открывает сегмент разделяемой памяти по названию файла __name с флагами oflag и режимом __mode

3. int munmap(void *__addr, size_t __len) - удаляет все отражения из заданной области памяти по адресу __addr длины __len.

4. int shm_unlink(const char *__name) - удаляет сегмент разделяемой памяти по имени.

5. void *mmap(void *__addr, size_t __len, int __prot, int __flags, int __fd, __off_t __offset) - выполняет отображения устройства или файла в память по адресу __addr длиной len.

6. sem_t sem_open(const char *name, int oflag); – открывает семафор по указанному имени файла с заданными флагами. Возвращает идентификатор семафора.

7. ssize_t read(int fd, void *buf, size_t count); – считывает до count байт данных из дескриптора fd в буфер buf. Возвращает количество считанных байт.

8. ssize_t write(int fd, const void *buf, size_t count); – записывает до count байт из буфера buf в файловый дескриптор fd. Возвращает количество записанных байт.

9. int sem_close(sem_t *__sem); – закрывает семафор __sem. Возвращает код результата.

10. pid_t getpid(void); – возвращает идентификатор текущего процесса (PID).

11. int execv(const char *path, char *const argv[]); – используется для выполнения другой программы, загружаемой поверх программы, содержащей вызов execv. Имя файла, содержащего процесс-потомок, задано с помощью параметра path. Какие-либо аргументы, передаваемые процессу-потомку, задаются с помощью массива arg[]. Аргументы передаются процессу-потомку в массиве. Этот способ используется тогда, когда заранее не известно, сколько аргументов будет передано процессу-потомку, либо же число аргументов может изменяться во время выполнения программы. Обычно конец массива обозначается нулевым указателем.

12. ssize_t readlink(const char *path, char *buf, size_t bufsiz); – считывает символическую ссылку path и записывает содержимое в buf. Используется для получения пути к текущему исполняемому файлу.

13. int wait(int *status); – ожидает завершения дочернего процесса и возвращает его статус через указатель status.

14. void exit(int status); – завершает процесс с указанным кодом выхода status. Используется для обработки ошибок и завершения программы.

15. int snprintf(char *str, size_t size, const char *format, ...); – форматирует строку в буфер str размером size, возвращая длину записанных символов. Используется для создания сообщений перед выводом.

16. int sem_unlink(const char *__name) - удаляет семафор по имени __name. Возвращает статус удаления семафора.
17. int sem_trywait(sem_t *__sem) - неблокирующе проверяет, свободен ли семафор __sem.

18. int sem_post(sem_t *__sem) - увеличивает значение семафора __sem.

Программа делает так, чтобы родительский процесс и дочерний могли обмениваться данными через разделяемую память, а дочерний процесс запускает клиентскую программу для записи данных в файл.

1. Сначала родитель спрашивает у пользователя имя файла, чтобы передать его дочернему процессу. Также он узнаёт путь к текущей программе с помощью readlink.

2. Затем родитель создаёт два сегмента разделяемой памяти: один для передачи строк в дочерний процесс (shm_data), другой для получения ошибок от дочернего процесса (shm_errors). После этого с помощью fork создаётся дочерний процесс:
   - Родительский процесс читает строки от пользователя и отправляет их в shm_data, а ещё отслеживает ошибки из shm_errors.
   - Дочерний процесс перенаправляет свои потоки stdin и stderr на каналы и запускает клиентскую программу через execv.

3. Клиентская программа проверяет строки, которые ей присылают: они должны заканчиваться на точку (.) или точку с запятой (;). Если всё нормально, строка записывается в файл. Если нет — ошибка отправляется обратно через stderr.

4. В конце родительский процесс ждёт, пока дочерний завершится, с помощью wait. Все открытые сегменты разделяемой памяти закрываются перед завершением работы.

Таким образом, программа показывает, как процессы могут взаимодействовать между собой и как данные передаются через разделяемую память.

Код программы

posix_ipc-example-server.c

```c
#include <stdlib.h>
#include <stdint.h>
#include <fcntl.h>
#include <unistd.h>
#include <sys/mman.h>
#include <sys/stat.h>
#include <sys/types.h>
#include <sys/wait.h>
#include <string.h>
#include <semaphore.h>
#include <errno.h>
#include <limits.h>


// ------------------------------------------------
// Общие параметры для data-шм
// ------------------------------------------------
#define SHM_DATA_NAME          "/posix_ipc_example_data"
#define SEM_CAN_WRITE_DATA     "/posix_ipc_example_can_write_data"
#define SEM_CAN_READ_DATA      "/posix_ipc_example_can_read_data"


// ------------------------------------------------
// Общие параметры для err-шм
// ------------------------------------------------
#define SHM_ERR_NAME           "/posix_ipc_example_err"
```

```c
#define SEM_CAN_WRITE_ERR     "/posix_ipc_example_can_write_err"
#define SEM_CAN_READ_ERR      "/posix_ipc_example_can_read_err"


// -----------------------------------------------
#define SHM_SIZE 4096


static char CLIENT_PROGRAM_NAME[] = "client";


//---------------------------------------------------------------------------------
// Простейшая функция для вывода C-строки (null-terminated) в указанный дескриптор.
static void write_str_to_fd(int fd, const char *s)
{
   if (s) {
       write(fd, s, strlen(s));
   }
}


//---------------------------------------------------------------------------------
// Считываем *одну строку* с терминала (STDIN_FILENO) без использования stdio.h.
// Возвращаем количество реально прочитанных байт (не включая '\0' в конце).
// Если прочитан '\n', заменяем его на '\0'. Если EOF или ошибка — возвращаем 0.
// Буфер должен быть размером не меньше (buf_size).
//---------------------------------------------------------------------------------
static ssize_t read_line_from_stdin(char *buf, size_t buf_size)
{
   if (!buf || buf_size == 0) {
       return 0;
   }


   size_t pos = 0;
   char ch;
   while (1) {
       ssize_t rd = read(STDIN_FILENO, &ch, 1);
```

```
        if (rd < 1) {
            // EOF или ошибка
            break;
        }

        // Если встретили '\n' — завершаем строку
        if (ch == '\n') {
            break;
        }

        if (pos < buf_size - 1) {
            buf[pos++] = ch;
        }
        else {
            // Достигли конца буфера — больше не пишем
            // но продолжаем считывать, чтобы очистить stdin до конца строки
        }
    }

    // Добавим null-terminator
    buf[pos] = '\0';
    return (ssize_t)pos;
}

//-----------------------------------------------------------------------------
// Простейшая функция вместо perror — выводит указанное сообщение + '\n'
// Можно дополнительно вывести значение errno, если нужно.
//-----------------------------------------------------------------------------
static void simple_perror(const char *msg)
{
    // Выводим сам msg
    write_str_to_fd(STDERR_FILENO, msg);
```

```
    // Можно добавить что-то вроде ": error\n"
    // Или, если хотим вывести errno в виде числа, нужен int->string
    write_str_to_fd(STDERR_FILENO, "\n");
}

//---------------------------------------------------------------------------

int main(void)
{
    // 1) Считываем имя файла
    char file[4096];
    write_str_to_fd(STDOUT_FILENO, "Enter filename: ");
    ssize_t flen = read_line_from_stdin(file, sizeof(file));
    if (flen <= 0) {
        write_str_to_fd(STDERR_FILENO, "Enter filename failed or EOF\n");
        exit(EXIT_FAILURE);
    }

    // 2) Создаем/открываем shm (DATA)
    int shm_fd_data = shm_open(SHM_DATA_NAME, O_CREAT | O_RDWR, 0666);
    if (shm_fd_data == -1) {
        simple_perror("shm_open data");
        exit(EXIT_FAILURE);
    }
    if (ftruncate(shm_fd_data, SHM_SIZE) == -1) {
        simple_perror("ftruncate data");
        shm_unlink(SHM_DATA_NAME);
        exit(EXIT_FAILURE);
    }
    char *shm_data = mmap(NULL, SHM_SIZE, PROT_READ | PROT_WRITE, MAP_SHARED,
shm_fd_data, 0);
    if (shm_data == MAP_FAILED) {
        simple_perror("mmap data");
        shm_unlink(SHM_DATA_NAME);
```

```
        exit(EXIT_FAILURE);
    }
    memset(shm_data, 0, SHM_SIZE);


    // 3) Создаем/открываем shm (ERRORS)
    int shm_fd_err = shm_open(SHM_ERR_NAME, O_CREAT | O_RDWR, 0666);
    if (shm_fd_err == -1) {
        simple_perror("shm_open err");
        munmap(shm_data, SHM_SIZE);
        shm_unlink(SHM_DATA_NAME);
        exit(EXIT_FAILURE);
    }
    if (ftruncate(shm_fd_err, SHM_SIZE) == -1) {
        simple_perror("ftruncate err");
        shm_unlink(SHM_ERR_NAME);
        munmap(shm_data, SHM_SIZE);
        shm_unlink(SHM_DATA_NAME);
        exit(EXIT_FAILURE);
    }
    char *shm_err = mmap(NULL, SHM_SIZE, PROT_READ | PROT_WRITE, MAP_SHARED,
shm_fd_err, 0);
    if (shm_err == MAP_FAILED) {
        simple_perror("mmap err");
        munmap(shm_data, SHM_SIZE);
        shm_unlink(SHM_DATA_NAME);
        shm_unlink(SHM_ERR_NAME);
        exit(EXIT_FAILURE);
    }
    memset(shm_err, 0, SHM_SIZE);


    // 4) Создаем семафоры для data
    sem_unlink(SEM_CAN_WRITE_DATA);
    sem_t* sem_can_write_data = sem_open(SEM_CAN_WRITE_DATA, O_CREAT, 0666, 1);
    if (sem_can_write_data == SEM_FAILED) {
```

```
        simple_perror("sem_open write_data");
        munmap(shm_data, SHM_SIZE);
        munmap(shm_err, SHM_SIZE);
        shm_unlink(SHM_DATA_NAME);
        shm_unlink(SHM_ERR_NAME);
        exit(EXIT_FAILURE);
    }

    sem_unlink(SEM_CAN_READ_DATA);
    sem_t* sem_can_read_data = sem_open(SEM_CAN_READ_DATA, O_CREAT, 0666, 0);
    if (sem_can_read_data == SEM_FAILED) {
        simple_perror("sem_open read_data");
        sem_close(sem_can_write_data);
        sem_unlink(SEM_CAN_WRITE_DATA);
        munmap(shm_data, SHM_SIZE);
        munmap(shm_err, SHM_SIZE);
        shm_unlink(SHM_DATA_NAME);
        shm_unlink(SHM_ERR_NAME);
        exit(EXIT_FAILURE);
    }

    // 5) Создаем семафоры для err
    sem_unlink(SEM_CAN_WRITE_ERR);
    sem_t* sem_can_write_err = sem_open(SEM_CAN_WRITE_ERR, O_CREAT, 0666, 1);
    if (sem_can_write_err == SEM_FAILED) {
        simple_perror("sem_open write_err");
        // по аналогии освобождаем всё
        // ...
        exit(EXIT_FAILURE);
    }

    sem_unlink(SEM_CAN_READ_ERR);
    sem_t* sem_can_read_err = sem_open(SEM_CAN_READ_ERR, O_CREAT, 0666, 0);
```

```
if (sem_can_read_err == SEM_FAILED) {
    simple_perror("sem_open read_err");
    // по аналогии освобождаем всё
    // ...
    exit(EXIT_FAILURE);
}

// 6) Получаем путь к каталогу текущего исполняемого файла (readlink)
//    Ищем последний '/', чтобы обрезать до директории
char progpath[1024];
ssize_t len = readlink("/proc/self/exe", progpath, sizeof(progpath)-1);
if (len == -1) {
    simple_perror("readlink");
    // чистим всё
    exit(EXIT_FAILURE);
}
while (len > 0 && progpath[len] != '/')
    --len;
progpath[len] = '\0';

// 7) Запускаем дочерний процесс (fork)
pid_t child = fork();
if (child < 0) {
    simple_perror("fork");
    // чистим
    exit(EXIT_FAILURE);
}
else if (child == 0) {
    // child
    char path[1024];
    // Собираем что-то вроде /home/user/.../client
    // без snprintf (он из stdio.h), используем strncat и т.д.
    // Но можно сделать так:
```

```
        memset(path, 0, sizeof(path));
        strncpy(path, progpath, sizeof(path) - 1);
        strncat(path, "/", sizeof(path) - strlen(path) - 1);
        strncat(path, CLIENT_PROGRAM_NAME, sizeof(path) - strlen(path) - 1);

        // Формируем аргументы для execv
        char *const args[] = {
                CLIENT_PROGRAM_NAME,
                file,
                SHM_DATA_NAME,
                SHM_ERR_NAME,
                NULL
        };
        execv(path, args);

        // если execv вернулся — ошибка
        write_str_to_fd(STDERR_FILENO, "execv failed\n");
        _exit(EXIT_FAILURE);
    }
    else {
        // parent
        pid_t pid = getpid();

        // Аналог "printf("%d: I'm a parent...")":
        char msg[256];
        memset(msg, 0, sizeof(msg));

        // Для вывода pid и child можно написать небольшую int->str, но для краткости выведем без них:
        // Или вывести "Parent started\n"
        write_str_to_fd(STDOUT_FILENO, "Parent started. Start typing lines. Press Enter on empty line or Ctrl-D to exit\n");

        // Основной цикл ввода от пользователя
```

```
    char input_buf[4096];
    while (1) {
        // 1) Неблокирующе проверим, нет ли ошибок от ребёнка
        if (sem_trywait(sem_can_read_err) == 0) {
            // читаем ошибку из shm_err
            // затем освобождаем буфер для следующей ошибки
            write_str_to_fd(STDOUT_FILENO, "[Parent] Child error message: ");
            write_str_to_fd(STDOUT_FILENO, shm_err);
            write_str_to_fd(STDOUT_FILENO, "\n");
            sem_post(sem_can_write_err);
        }
        // если -1 с EAGAIN — нет ошибок, идём дальше

        // 2) Читаем строку
        write_str_to_fd(STDOUT_FILENO, "> "); // чтобы было видно приглашение
        ssize_t rdlen = read_line_from_stdin(input_buf, sizeof(input_buf));
        if (rdlen <= 0) {
            // EOF или ошибка
            break;
        }
        // Если пользователь ввёл пустую строку — завершаем
        if (input_buf[0] == '\0') {
            break;
        }

        // 3) Пишем в shm_data
        sem_wait(sem_can_write_data); // ожидаем, что буфер свободен
        strncpy(shm_data, input_buf, SHM_SIZE - 1);
        shm_data[SHM_SIZE - 1] = '\0';
        sem_post(sem_can_read_data);
    }

    // Сигнализируем ребёнку, что данных больше не будет (пустая строка)
```

```
        sem_wait(sem_can_write_data);
        shm_data[0] = '\0';
        sem_post(sem_can_read_data);

        // Ждём, пока ребёнок завершится
        int status = 0;
        waitpid(child, &status, 0);

        // Закрываем семафоры
        sem_close(sem_can_write_data);
        sem_close(sem_can_read_data);
        sem_unlink(SEM_CAN_WRITE_DATA);
        sem_unlink(SEM_CAN_READ_DATA);

        sem_close(sem_can_write_err);
        sem_close(sem_can_read_err);
        sem_unlink(SEM_CAN_WRITE_ERR);
        sem_unlink(SEM_CAN_READ_ERR);

        // Отключаем shm
        munmap(shm_data, SHM_SIZE);
        shm_unlink(SHM_DATA_NAME);

        munmap(shm_err, SHM_SIZE);
        shm_unlink(SHM_ERR_NAME);

        write_str_to_fd(STDOUT_FILENO, "Parent finished.\n");
    }

    return 0;
}
```

posix_ipc-example-client.c

```c
#include <stdlib.h>      // для exit()
#include <stdint.h>
#include <fcntl.h>       // для open()
#include <unistd.h>      // для write(), read(), close(),
 _exit()
#include <sys/mman.h>    // для mmap(), munmap()
#include <sys/stat.h>    // для shm_open()
#include <sys/types.h>   // для pid_t
#include <string.h>      // для strlen(), strcpy(), strcat(),
 memset() и т.д.
#include <semaphore.h>   // для sem_open(), sem_wait() и т.д.

#include <errno.h>       // для errno

// ------------------------------------------------
// Общие параметры для data-шм
// ------------------------------------------------
#define SHM_DATA_NAME         "/posix_ipc_example_data"
#define SEM_CAN_WRITE_DATA
"/posix_ipc_example_can_write_data"
#define SEM_CAN_READ_DATA
     "/posix_ipc_example_can_read_data"

// ------------------------------------------------
// Общие параметры для err-шм
// ------------------------------------------------
#define SHM_ERR_NAME          "/posix_ipc_example_err"
#define SEM_CAN_WRITE_ERR
    "/posix_ipc_example_can_write_err"
#define SEM_CAN_READ_ERR
     "/posix_ipc_example_can_read_err"

// ------------------------------------------------
#define SHM_SIZE 4096

//-----------------------------------------------------------------------------
// Функция для вывода C-строки (null-terminated) в указанный дескриптор
static void write_str_to_fd(int fd, const char *s)
{
   if (s) {
       write(fd, s, strlen(s));
   }
}

//-----------------------------------------------------------------------------
// Упрощённая функция вместо perror — выводит указанное сообщение + "\n"
static void simple_perror(const char *msg)
{
```

```
    write_str_to_fd(STDERR_FILENO, msg);
    write_str_to_fd(STDERR_FILENO, "\n");
}

//-----------------------------------------------------------------------------
// Завершение с очисткой ресурсов
static void cleanup_and_exit(
        int fd_file,
        char *shm_data,
        char *shm_err,
        sem_t *sem_can_write_data,
        sem_t *sem_can_read_data,
        sem_t *sem_can_write_err,
        sem_t *sem_can_read_err,
        int exit_code
) {
    // Закрываем файл
    if (fd_file >= 0) {
        close(fd_file);
    }
    // Отключаем shm_data
    if (shm_data && shm_data != MAP_FAILED) {
        munmap(shm_data, SHM_SIZE);
    }
    // Отключаем shm_err
    if (shm_err && shm_err != MAP_FAILED) {
        munmap(shm_err, SHM_SIZE);
    }
    // Закрываем семафоры
    if (sem_can_write_data && sem_can_write_data != SEM_FAILED) {
        sem_close(sem_can_write_data);
    }
    if (sem_can_read_data && sem_can_read_data != SEM_FAILED) {
        sem_close(sem_can_read_data);
    }
    if (sem_can_write_err && sem_can_write_err != SEM_FAILED) {
        sem_close(sem_can_write_err);
    }
    if (sem_can_read_err && sem_can_read_err != SEM_FAILED) {
        sem_close(sem_can_read_err);
    }

    // Выходим
    exit(exit_code);
}

//-----------------------------------------------------------------------------
// Примерная логика клиента:
// Ожидается, что аргументы:
```

```
//    argv[1] = имя файла (куда писать корректные строки)
//    argv[2] = имя shm для data
//    argv[3] = имя shm для err
//-----------------------------------------------------------------------------
int main(int argc, char *argv[])
{
    if (argc < 4) {
        write_str_to_fd(STDERR_FILENO, "Usage: client <filename> <shm_data>
<shm_err>\n");
        return 1;
    }

    char *filename    = argv[1]; // Выходной файл
    char *shm_data_nm = argv[2]; // Имя shm (data)
    char *shm_err_nm  = argv[3]; // Имя shm (err)

    // 1) Открываем файл на запись (перезапись + добавление)
    int fd = open(filename, O_WRONLY | O_CREAT | O_TRUNC | O_APPEND, 0600);
    if (fd == -1) {
        simple_perror("open output file failed");
        return 1;
    }

    // 2) Подключаемся к разделяемой памяти (DATA)
    int shm_fd_data = shm_open(shm_data_nm, O_RDWR, 0666);
    if (shm_fd_data == -1) {
        simple_perror("shm_open data failed");
        close(fd);
        return 1;
    }
    char *shm_data = mmap(NULL, SHM_SIZE, PROT_READ | PROT_WRITE, MAP_SHARED,
shm_fd_data, 0);
    if (shm_data == MAP_FAILED) {
        simple_perror("mmap data failed");
        close(fd);
        close(shm_fd_data);
        return 1;
    }
    close(shm_fd_data);  // дескриптор можно закрыть после mmap

    // 3) Подключаемся к разделяемой памяти (ERR)
    int shm_fd_err = shm_open(shm_err_nm, O_RDWR, 0666);
    if (shm_fd_err == -1) {
        simple_perror("shm_open err failed");
        munmap(shm_data, SHM_SIZE);
        close(fd);
        return 1;
    }
    char *shm_err = mmap(NULL, SHM_SIZE, PROT_READ | PROT_WRITE, MAP_SHARED,
shm_fd_err, 0);
    if (shm_err == MAP_FAILED) {
```

```
        simple_perror("mmap err failed");
        munmap(shm_data, SHM_SIZE);
        close(fd);
        close(shm_fd_err);
        return 1;
    }
    close(shm_fd_err);

    // 4) Открываем семафоры (уже созданные родителем)
    sem_t *sem_can_write_data = sem_open(SEM_CAN_WRITE_DATA, 0);
    if (sem_can_write_data == SEM_FAILED) {
        simple_perror("sem_open(SEM_CAN_WRITE_DATA) failed");
        cleanup_and_exit(fd, shm_data, shm_err, NULL, NULL, NULL, NULL, 1);
    }
    sem_t *sem_can_read_data  = sem_open(SEM_CAN_READ_DATA, 0);
    if (sem_can_read_data == SEM_FAILED) {
        simple_perror("sem_open(SEM_CAN_READ_DATA) failed");
        cleanup_and_exit(fd, shm_data, shm_err, sem_can_write_data, NULL, NULL,
NULL, 1);
    }
    sem_t *sem_can_write_err = sem_open(SEM_CAN_WRITE_ERR, 0);
    if (sem_can_write_err == SEM_FAILED) {
        simple_perror("sem_open(SEM_CAN_WRITE_ERR) failed");
        cleanup_and_exit(fd, shm_data, shm_err, sem_can_write_data,
sem_can_read_data, NULL, NULL, 1);
    }
    sem_t *sem_can_read_err  = sem_open(SEM_CAN_READ_ERR, 0);
    if (sem_can_read_err == SEM_FAILED) {
        simple_perror("sem_open(SEM_CAN_READ_ERR) failed");
        cleanup_and_exit(fd, shm_data, shm_err, sem_can_write_data,
sem_can_read_data, sem_can_write_err, NULL, 1);
    }

    // Сообщаем, что клиент запустился
    write_str_to_fd(STDOUT_FILENO, "[Child] started, reading from shm_data...\n");

    while (1) {
        // Ждём разрешения на чтение строки
        if (sem_wait(sem_can_read_data) < 0) {
            // Ошибка или сигнал
            simple_perror("sem_wait(sem_can_read_data) failed");
            break;
        }

        // Проверяем, не пустая ли строка => признак окончания
        if (shm_data[0] == '\0') {
            // Родитель сообщил об окончании ввода
            break;
        }

        // Копируем строку локально
```

```
        char buf[4096];
        memset(buf, 0, sizeof(buf));
        strncpy(buf, shm_data, sizeof(buf)-1);

        // Проверяем, заканчивается ли '.' или ';'
        size_t len = strlen(buf);
        if (len > 0) {
            char last_char = buf[len - 1];
            if (last_char != '.' && last_char != ';') {
                // Ошибка. Пишем в shm_err
                if (sem_wait(sem_can_write_err) == 0) {
                    // Собираем сообщение в shm_err
                    shm_err[0] = '\0';
                    strcat(shm_err, "child error: string does not end with '.' or
';'. The string was: ");
                    strcat(shm_err, buf);
                    sem_post(sem_can_read_err);
                } else {
                    // sem_wait не сработал
                    simple_perror("sem_wait(sem_can_write_err) failed");
                }
            } else {
                // Строка корректна, пишем в файл + \n
                strcat(buf, "\n");
                ssize_t written = write(fd, buf, strlen(buf));
                if (written < 0) {
                    // Ошибка записи
                    if (sem_wait(sem_can_write_err) == 0) {
                        shm_err[0] = '\0';
                        strcat(shm_err, "child error: write to file failed,
errno=");
                        // Если нужно вывести число errno, нужно int->str.
Упростим:
                        // Выведем просто "... , errno\n"
                        strcat(shm_err, "?\n");
                        sem_post(sem_can_read_err);
                    }
                }
            }
        }

        // Освобождаем буфер data
        sem_post(sem_can_write_data);
    }

    // Сообщаем, что завершаемся
    write_str_to_fd(STDOUT_FILENO, "[Child] finishing.\n");

    cleanup_and_exit(fd, shm_data, shm_err,
                     sem_can_write_data, sem_can_read_data,
                     sem_can_write_err, sem_can_read_err,
```

```
                        0);
    return 0;
}
```

Протокол работы программы

Тестирование:

```
qbzy@QBZstation:/mnt/c/Users/mrbor/CLionProjects/osi/lab3$ strace -o strace_output.txt
-f ./server
Enter filename: f.txt
Parent started. Start typing lines. Press Enter on empty line or Ctrl-D to exit
> [Child] started, reading from shm_data...
fkjsdlfjsdf;
> l.;
> l;l;
> fjdlsaf.
> fjdlskajf;
>
[Child] finishing.
Parent finished.
```

f.txt:

fkjsdlfjsdf;

l.;

l;l;

fjdlsaf.

fjdlskajf;

Strace:

```
3868  execve("./server", ["./server"], 0x7ffc94c4f7a8 /* 26 vars */) = 0

3868  brk(NULL)                           = 0x5625c7e0c000

3868  mmap(NULL, 8192, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_ANONYMOUS, -1, 0) =
0x7f0771beb000

3868  access("/etc/ld.so.preload", R_OK) = -1 ENOENT (No such file or directory)

3868  openat(AT_FDCWD, "/etc/ld.so.cache", O_RDONLY|O_CLOEXEC) = 3

3868  fstat(3, {st_mode=S_IFREG|0644, st_size=21363, ...}) = 0

3868  mmap(NULL, 21363, PROT_READ, MAP_PRIVATE, 3, 0) = 0x7f0771be5000

3868  close(3)                            = 0

3868  openat(AT_FDCWD, "/lib/x86_64-linux-gnu/libc.so.6", O_RDONLY|O_CLOEXEC) = 3

3868  read(3, "\177ELF\2\1\1\3\0\0\0\0\0\0\0\0\3\0>\0\1\0\0\0\220\243\2\0\0\0\0\0"...,
832) = 832

3868  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

3868  fstat(3, {st_mode=S_IFREG|0755, st_size=2125328, ...}) = 0

3868  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

3868  mmap(NULL, 2170256, PROT_READ, MAP_PRIVATE|MAP_DENYWRITE, 3, 0) = 0x7f07719d3000

3868  mmap(0x7f07719fb000, 1605632, PROT_READ|PROT_EXEC,
MAP_PRIVATE|MAP_FIXED|MAP_DENYWRITE, 3, 0x28000) = 0x7f07719fb000

3868  mmap(0x7f0771b83000, 323584, PROT_READ, MAP_PRIVATE|MAP_FIXED|MAP_DENYWRITE, 3,
0x1b0000) = 0x7f0771b83000

3868  mmap(0x7f0771bd2000, 24576, PROT_READ|PROT_WRITE,
MAP_PRIVATE|MAP_FIXED|MAP_DENYWRITE, 3, 0x1fe000) = 0x7f0771bd2000

3868  mmap(0x7f0771bd8000, 52624, PROT_READ|PROT_WRITE,
MAP_PRIVATE|MAP_FIXED|MAP_ANONYMOUS, -1, 0) = 0x7f0771bd8000

3868  close(3)                            = 0

3868  mmap(NULL, 12288, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_ANONYMOUS, -1, 0) =
0x7f07719d0000

3868  arch_prctl(ARCH_SET_FS, 0x7f07719d0740) = 0

3868  set_tid_address(0x7f07719d0a10)   = 3868

3868  set_robust_list(0x7f07719d0a20, 24) = 0

3868  rseq(0x7f07719d1060, 0x20, 0, 0x53053053) = 0

3868  mprotect(0x7f0771bd2000, 16384, PROT_READ) = 0

3868  mprotect(0x56258d401000, 4096, PROT_READ) = 0

3868  mprotect(0x7f0771c23000, 8192, PROT_READ) = 0
```

```
3868  prlimit64(0, RLIMIT_STACK, NULL, {rlim_cur=8192*1024, rlim_max=RLIM64_INFINITY})
= 0

3868  munmap(0x7f0771be5000, 21363)       = 0

3868  write(1, "Enter filename: ", 16)  = 16

3868  read(0, "f", 1)                     = 1

3868  read(0, ".", 1)                     = 1

3868  read(0, "t", 1)                     = 1

3868  read(0, "x", 1)                     = 1

3868  read(0, "t", 1)                     = 1

3868  read(0, "\n", 1)                    = 1

3868  openat(AT_FDCWD, "/dev/shm/posix_ipc_example_data",
O_RDWR|O_CREAT|O_NOFOLLOW|O_CLOEXEC, 0666) = 3

3868  ftruncate(3, 4096)                  = 0

3868  mmap(NULL, 4096, PROT_READ|PROT_WRITE, MAP_SHARED, 3, 0) = 0x7f0771bea000

3868  openat(AT_FDCWD, "/dev/shm/posix_ipc_example_err",
O_RDWR|O_CREAT|O_NOFOLLOW|O_CLOEXEC, 0666) = 4

3868  ftruncate(4, 4096)                  = 0

3868  mmap(NULL, 4096, PROT_READ|PROT_WRITE, MAP_SHARED, 4, 0) = 0x7f0771be9000

3868  unlink("/dev/shm/sem.posix_ipc_example_can_write_data") = -1 ENOENT (No such
file or directory)

3868  openat(AT_FDCWD, "/dev/shm/sem.posix_ipc_example_can_write_data",
O_RDWR|O_NOFOLLOW|O_CLOEXEC) = -1 ENOENT (No such file or directory)

3868  getrandom("\xd4\x5b\x7d\x3b\xef\xd9\x56\x90", 8, GRND_NONBLOCK) = 8

3868  newfstatat(AT_FDCWD, "/dev/shm/sem.K8QgrB", 0x7ffe9890c260, AT_SYMLINK_NOFOLLOW)
= -1 ENOENT (No such file or directory)

3868  openat(AT_FDCWD, "/dev/shm/sem.K8QgrB",
O_RDWR|O_CREAT|O_EXCL|O_NOFOLLOW|O_CLOEXEC, 0666) = 5

3868  write(5, "\1\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

3868  mmap(NULL, 32, PROT_READ|PROT_WRITE, MAP_SHARED, 5, 0) = 0x7f0771be8000

3868  link("/dev/shm/sem.K8QgrB", "/dev/shm/sem.posix_ipc_example_can_write_data") = 0

3868  fstat(5, {st_mode=S_IFREG|0644, st_size=32, ...}) = 0

3868  getrandom("\xd8\x2f\xdc\x14\x8a\xb0\xa5\x9e", 8, GRND_NONBLOCK) = 8

3868  brk(NULL)                           = 0x5625c7e0c000

3868  brk(0x5625c7e2d000)                 = 0x5625c7e2d000

3868  unlink("/dev/shm/sem.K8QgrB")       = 0

3868  close(5)                            = 0
```

```
3868  unlink("/dev/shm/sem.posix_ipc_example_can_read_data") = -1 ENOENT (No such file
or directory)

3868  openat(AT_FDCWD, "/dev/shm/sem.posix_ipc_example_can_read_data",
O_RDWR|O_NOFOLLOW|O_CLOEXEC) = -1 ENOENT (No such file or directory)

3868  getrandom("\x14\x74\x7d\x41\xab\xba\x5f\xb1", 8, GRND_NONBLOCK) = 8

3868  newfstatat(AT_FDCWD, "/dev/shm/sem.gvc2SH", 0x7ffe9890c260, AT_SYMLINK_NOFOLLOW)
= -1 ENOENT (No such file or directory)

3868  openat(AT_FDCWD, "/dev/shm/sem.gvc2SH",
O_RDWR|O_CREAT|O_EXCL|O_NOFOLLOW|O_CLOEXEC, 0666) = 5

3868  write(5, "\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

3868  mmap(NULL, 32, PROT_READ|PROT_WRITE, MAP_SHARED, 5, 0) = 0x7f0771be7000

3868  link("/dev/shm/sem.gvc2SH", "/dev/shm/sem.posix_ipc_example_can_read_data") = 0

3868  fstat(5, {st_mode=S_IFREG|0644, st_size=32, ...}) = 0

3868  unlink("/dev/shm/sem.gvc2SH")       = 0

3868  close(5)                            = 0

3868  unlink("/dev/shm/sem.posix_ipc_example_can_write_err") = -1 ENOENT (No such file
or directory)

3868  openat(AT_FDCWD, "/dev/shm/sem.posix_ipc_example_can_write_err",
O_RDWR|O_NOFOLLOW|O_CLOEXEC) = -1 ENOENT (No such file or directory)

3868  getrandom("\x5c\x6d\xd0\x67\xb4\x6e\xe5\xc4", 8, GRND_NONBLOCK) = 8

3868  newfstatat(AT_FDCWD, "/dev/shm/sem.gbcqve", 0x7ffe9890c260, AT_SYMLINK_NOFOLLOW)
= -1 ENOENT (No such file or directory)

3868  openat(AT_FDCWD, "/dev/shm/sem.gbcqve",
O_RDWR|O_CREAT|O_EXCL|O_NOFOLLOW|O_CLOEXEC, 0666) = 5

3868  write(5, "\1\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

3868  mmap(NULL, 32, PROT_READ|PROT_WRITE, MAP_SHARED, 5, 0) = 0x7f0771be6000

3868  link("/dev/shm/sem.gbcqve", "/dev/shm/sem.posix_ipc_example_can_write_err") = 0

3868  fstat(5, {st_mode=S_IFREG|0644, st_size=32, ...}) = 0

3868  unlink("/dev/shm/sem.gbcqve")       = 0

3868  close(5)                            = 0

3868  unlink("/dev/shm/sem.posix_ipc_example_can_read_err") = -1 ENOENT (No such file
or directory)

3868  openat(AT_FDCWD, "/dev/shm/sem.posix_ipc_example_can_read_err",
O_RDWR|O_NOFOLLOW|O_CLOEXEC) = -1 ENOENT (No such file or directory)

3868  getrandom("\x64\xa8\xf0\x1d\x85\x28\x1d\x9b", 8, GRND_NONBLOCK) = 8

3868  newfstatat(AT_FDCWD, "/dev/shm/sem.E9QNFf", 0x7ffe9890c260, AT_SYMLINK_NOFOLLOW)
= -1 ENOENT (No such file or directory)
```

```
3868  openat(AT_FDCWD, "/dev/shm/sem.E9QNFf",
O_RDWR|O_CREAT|O_EXCL|O_NOFOLLOW|O_CLOEXEC, 0666) = 5

3868  write(5, "\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

3868  mmap(NULL, 32, PROT_READ|PROT_WRITE, MAP_SHARED, 5, 0) = 0x7f0771be5000

3868  link("/dev/shm/sem.E9QNFf", "/dev/shm/sem.posix_ipc_example_can_read_err") = 0

3868  fstat(5, {st_mode=S_IFREG|0644, st_size=32, ...}) = 0

3868  unlink("/dev/shm/sem.E9QNFf")       = 0

3868  close(5)                            = 0

3868  readlink("/proc/self/exe", "/mnt/c/Users/mrbor/CLionProjects"..., 1023) = 48

3868  clone(child_stack=NULL, flags=CLONE_CHILD_CLEARTID|CLONE_CHILD_SETTID|SIGCHLD,
child_tidptr=0x7f07719d0a10) = 3869

3869  set_robust_list(0x7f07719d0a20, 24 <unfinished ...>

3868  getpid( <unfinished ...>

3869  <... set_robust_list resumed>)      = 0

3868  <... getpid resumed>)               = 3868

3869  execve("/mnt/c/Users/mrbor/CLionProjects/osi/lab3/client", ["client", "f.txt",
"/posix_ipc_example_data", "/posix_ipc_example_err"], 0x7ffe9890ecb8 /* 26 vars */
<unfinished ...>

3868  write(1, "Parent started. Start typing lin"..., 80) = 80

3868  write(1, "> ", 2)                   = 2

3868  read(0,  <unfinished ...>

3869  <... execve resumed>)               = 0

3869  brk(NULL)                           = 0x55ae8d923000

3869  mmap(NULL, 8192, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_ANONYMOUS, -1, 0) =
0x7f66bb420000

3869  access("/etc/ld.so.preload", R_OK) = -1 ENOENT (No such file or directory)

3869  openat(AT_FDCWD, "/etc/ld.so.cache", O_RDONLY|O_CLOEXEC) = 3

3869  fstat(3, {st_mode=S_IFREG|0644, st_size=21363, ...}) = 0

3869  mmap(NULL, 21363, PROT_READ, MAP_PRIVATE, 3, 0) = 0x7f66bb41a000

3869  close(3)                            = 0

3869  openat(AT_FDCWD, "/lib/x86_64-linux-gnu/libc.so.6", O_RDONLY|O_CLOEXEC) = 3

3869  read(3, "\177ELF\2\1\1\3\0\0\0\0\0\0\0\3\0>\0\1\0\0\0\220\243\2\0\0\0\0\0"...,
832) = 832

3869  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

3869  fstat(3, {st_mode=S_IFREG|0755, st_size=2125328, ...}) = 0
```

```
3869  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

3869  mmap(NULL, 2170256, PROT_READ, MAP_PRIVATE|MAP_DENYWRITE, 3, 0) = 0x7f66bb208000

3869  mmap(0x7f66bb230000, 1605632, PROT_READ|PROT_EXEC, MAP_PRIVATE|MAP_FIXED|MAP_DENYWRITE, 3, 0x28000) = 0x7f66bb230000

3869  mmap(0x7f66bb3b8000, 323584, PROT_READ, MAP_PRIVATE|MAP_FIXED|MAP_DENYWRITE, 3, 0x1b0000) = 0x7f66bb3b8000

3869  mmap(0x7f66bb407000, 24576, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_FIXED|MAP_DENYWRITE, 3, 0x1fe000) = 0x7f66bb407000

3869  mmap(0x7f66bb40d000, 52624, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_FIXED|MAP_ANONYMOUS, -1, 0) = 0x7f66bb40d000

3869  close(3)                          = 0

3869  mmap(NULL, 12288, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_ANONYMOUS, -1, 0) = 0x7f66bb205000

3869  arch_prctl(ARCH_SET_FS, 0x7f66bb205740) = 0

3869  set_tid_address(0x7f66bb205a10)   = 3869

3869  set_robust_list(0x7f66bb205a20, 24) = 0

3869  rseq(0x7f66bb206060, 0x20, 0, 0x53053053) = 0

3869  mprotect(0x7f66bb407000, 16384, PROT_READ) = 0

3869  mprotect(0x55ae5b16e000, 4096, PROT_READ) = 0

3869  mprotect(0x7f66bb458000, 8192, PROT_READ) = 0

3869  prlimit64(0, RLIMIT_STACK, NULL, {rlim_cur=8192*1024, rlim_max=RLIM64_INFINITY}) = 0

3869  munmap(0x7f66bb41a000, 21363)      = 0

3869  openat(AT_FDCWD, "f.txt", O_WRONLY|O_CREAT|O_TRUNC|O_APPEND, 0600) = 3

3869  openat(AT_FDCWD, "/dev/shm/posix_ipc_example_data", O_RDWR|O_NOFOLLOW|O_CLOEXEC) = 4

3869  mmap(NULL, 4096, PROT_READ|PROT_WRITE, MAP_SHARED, 4, 0) = 0x7f66bb41f000

3869  close(4)                          = 0

3869  openat(AT_FDCWD, "/dev/shm/posix_ipc_example_err", O_RDWR|O_NOFOLLOW|O_CLOEXEC) = 4

3869  mmap(NULL, 4096, PROT_READ|PROT_WRITE, MAP_SHARED, 4, 0) = 0x7f66bb41e000

3869  close(4)                          = 0

3869  openat(AT_FDCWD, "/dev/shm/sem.posix_ipc_example_can_write_data", O_RDWR|O_NOFOLLOW|O_CLOEXEC) = 4

3869  fstat(4, {st_mode=S_IFREG|0644, st_size=32, ...}) = 0

3869  getrandom("\x76\xcf\x22\xb3\xfc\xac\xa7\x8f", 8, GRND_NONBLOCK) = 8

3869  brk(NULL)                         = 0x55ae8d923000
```

```
3869  brk(0x55ae8d944000)                = 0x55ae8d944000

3869  mmap(NULL, 32, PROT_READ|PROT_WRITE, MAP_SHARED, 4, 0) = 0x7f66bb41d000

3869  close(4)                           = 0

3869  openat(AT_FDCWD, "/dev/shm/sem.posix_ipc_example_can_read_data",
O_RDWR|O_NOFOLLOW|O_CLOEXEC) = 4

3869  fstat(4, {st_mode=S_IFREG|0644, st_size=32, ...}) = 0

3869  mmap(NULL, 32, PROT_READ|PROT_WRITE, MAP_SHARED, 4, 0) = 0x7f66bb41c000

3869  close(4)                           = 0

3869  openat(AT_FDCWD, "/dev/shm/sem.posix_ipc_example_can_write_err",
O_RDWR|O_NOFOLLOW|O_CLOEXEC) = 4

3869  fstat(4, {st_mode=S_IFREG|0644, st_size=32, ...}) = 0

3869  mmap(NULL, 32, PROT_READ|PROT_WRITE, MAP_SHARED, 4, 0) = 0x7f66bb41b000

3869  close(4)                           = 0

3869  openat(AT_FDCWD, "/dev/shm/sem.posix_ipc_example_can_read_err",
O_RDWR|O_NOFOLLOW|O_CLOEXEC) = 4

3869  fstat(4, {st_mode=S_IFREG|0644, st_size=32, ...}) = 0

3869  mmap(NULL, 32, PROT_READ|PROT_WRITE, MAP_SHARED, 4, 0) = 0x7f66bb41a000

3869  close(4)                           = 0

3869  write(1, "[Child] started, reading from sh"..., 42) = 42

3869  futex(0x7f66bb41c000, FUTEX_WAIT_BITSET|FUTEX_CLOCK_REALTIME, 0, NULL,
FUTEX_BITSET_MATCH_ANY <unfinished ...>

3868  <... read resumed>"f", 1)          = 1

3868  read(0, "k", 1)                    = 1

3868  read(0, "j", 1)                    = 1

3868  read(0, "s", 1)                    = 1

3868  read(0, "d", 1)                    = 1

3868  read(0, "l", 1)                    = 1

3868  read(0, "f", 1)                    = 1

3868  read(0, "j", 1)                    = 1

3868  read(0, "s", 1)                    = 1

3868  read(0, "d", 1)                    = 1

3868  read(0, "f", 1)                    = 1

3868  read(0, ";", 1)                    = 1

3868  read(0, "\n", 1)                   = 1

3868  futex(0x7f0771be7000, FUTEX_WAKE, 1) = 1
```

```
3869  <... futex resumed>)              = 0

3868  write(1, "> ", 2 <unfinished ...>

3869  write(3, "fkjsdlfjsdf;\n", 13 <unfinished ...>

3868  <... write resumed>)              = 2

3869  <... write resumed>)              = 13

3868  read(0,  <unfinished ...>

3869  futex(0x7f66bb41c000, FUTEX_WAIT_BITSET|FUTEX_CLOCK_REALTIME, 0, NULL,
FUTEX_BITSET_MATCH_ANY <unfinished ...>

3868  <... read resumed>"l", 1)         = 1

3868  read(0, ".", 1)                   = 1

3868  read(0, ";", 1)                   = 1

3868  read(0, "\n", 1)                  = 1

3868  futex(0x7f0771be7000, FUTEX_WAKE, 1) = 1

3869  <... futex resumed>)              = 0

3868  write(1, "> ", 2 <unfinished ...>

3869  write(3, "l.;\n", 4 <unfinished ...>

3868  <... write resumed>)              = 2

3868  read(0,  <unfinished ...>

3869  <... write resumed>)              = 4

3869  futex(0x7f66bb41c000, FUTEX_WAIT_BITSET|FUTEX_CLOCK_REALTIME, 0, NULL,
FUTEX_BITSET_MATCH_ANY <unfinished ...>

3868  <... read resumed>"l", 1)         = 1

3868  read(0, ";", 1)                   = 1

3868  read(0, "l", 1)                   = 1

3868  read(0, ";", 1)                   = 1

3868  read(0, "\n", 1)                  = 1

3868  futex(0x7f0771be7000, FUTEX_WAKE, 1) = 1

3869  <... futex resumed>)              = 0

3868  write(1, "> ", 2 <unfinished ...>

3869  write(3, "l;l;\n", 5 <unfinished ...>

3868  <... write resumed>)              = 2

3868  read(0,  <unfinished ...>

3869  <... write resumed>)              = 5

3869  futex(0x7f66bb41c000, FUTEX_WAIT_BITSET|FUTEX_CLOCK_REALTIME, 0, NULL,
FUTEX_BITSET_MATCH_ANY <unfinished ...>
```

```
3868  <... read resumed>"f", 1)           = 1
3868  read(0, "j", 1)                     = 1
3868  read(0, "d", 1)                     = 1
3868  read(0, "l", 1)                     = 1
3868  read(0, "s", 1)                     = 1
3868  read(0, "a", 1)                     = 1
3868  read(0, "f", 1)                     = 1
3868  read(0, ".", 1)                     = 1
3868  read(0, "\n", 1)                    = 1
3868  futex(0x7f0771be7000, FUTEX_WAKE, 1) = 1
3869  <... futex resumed>)                = 0
3868  write(1, "> ", 2 <unfinished ...>
3869  write(3, "fjdlsaf.\n", 9 <unfinished ...>
3868  <... write resumed>)                = 2
3868  read(0,  <unfinished ...>
3869  <... write resumed>)                = 9
3869  futex(0x7f66bb41c000, FUTEX_WAIT_BITSET|FUTEX_CLOCK_REALTIME, 0, NULL,
FUTEX_BITSET_MATCH_ANY <unfinished ...>
3868  <... read resumed>"f", 1)           = 1
3868  read(0, "j", 1)                     = 1
3868  read(0, "d", 1)                     = 1
3868  read(0, "l", 1)                     = 1
3868  read(0, "s", 1)                     = 1
3868  read(0, "k", 1)                     = 1
3868  read(0, "a", 1)                     = 1
3868  read(0, "j", 1)                     = 1
3868  read(0, "f", 1)                     = 1
3868  read(0, ";", 1)                     = 1
3868  read(0, "\n", 1)                    = 1
3868  futex(0x7f0771be7000, FUTEX_WAKE, 1) = 1
3869  <... futex resumed>)                = 0
3868  write(1, "> ", 2 <unfinished ...>
3869  write(3, "fjdlskajf;\n", 11 <unfinished ...>
3868  <... write resumed>)                = 2
```

```
3868  read(0,  <unfinished ...>
3869  <... write resumed>)              = 11
3869  futex(0x7f66bb41c000, FUTEX_WAIT_BITSET|FUTEX_CLOCK_REALTIME, 0, NULL,
FUTEX_BITSET_MATCH_ANY <unfinished ...>
3868  <... read resumed>"\n", 1)        = 1
3868  futex(0x7f0771be7000, FUTEX_WAKE, 1) = 1
3869  <... futex resumed>)              = 0
3868  wait4(3869,  <unfinished ...>
3869  write(1, "[Child] finishing.\n", 19) = 19
3869  close(3)                          = 0
3869  munmap(0x7f66bb41f000, 4096)      = 0
3869  munmap(0x7f66bb41e000, 4096)      = 0
3869  munmap(0x7f66bb41d000, 32)        = 0
3869  munmap(0x7f66bb41c000, 32)        = 0
3869  munmap(0x7f66bb41b000, 32)        = 0
3869  munmap(0x7f66bb41a000, 32)        = 0
3869  exit_group(0)                     = ?
3869  +++ exited with 0 +++
3868  <... wait4 resumed>[{WIFEXITED(s) && WEXITSTATUS(s) == 0}], 0, NULL) = 3869
3868  --- SIGCHLD {si_signo=SIGCHLD, si_code=CLD_EXITED, si_pid=3869, si_uid=1000,
si_status=0, si_utime=0, si_stime=1 /* 0.01 s */} ---
3868  munmap(0x7f0771be8000, 32)        = 0
3868  munmap(0x7f0771be7000, 32)        = 0
3868  unlink("/dev/shm/sem.posix_ipc_example_can_write_data") = 0
3868  unlink("/dev/shm/sem.posix_ipc_example_can_read_data") = 0
3868  munmap(0x7f0771be6000, 32)        = 0
3868  munmap(0x7f0771be5000, 32)        = 0
3868  unlink("/dev/shm/sem.posix_ipc_example_can_write_err") = 0
3868  unlink("/dev/shm/sem.posix_ipc_example_can_read_err") = 0
3868  munmap(0x7f0771bea000, 4096)      = 0
3868  unlink("/dev/shm/posix_ipc_example_data") = 0
3868  munmap(0x7f0771be9000, 4096)      = 0
3868  unlink("/dev/shm/posix_ipc_example_err") = 0
3868  write(1, "Parent finished.\n", 17) = 17
```

```
3868  exit_group(0)                     = ?

3868  +++ exited with 0 +++
```

Вывод

Программа успешно реализована и протестирована. Все требования задания выполнены:

- **Родительский и дочерний процессы реализованы в виде отдельных программ.**
- **Взаимодействие между процессами осуществлено через разделяемую память.**
- **Проверка строк на валидность выполнена в дочернем процессе.**
- **Системные ошибки обработаны.**
- **Результаты обработки корректно выводятся в стандартные потоки вывода родительского и дочернего процессов.**