

**Министерство науки и высшего образования Российской Федерации  
ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ АВТОНОМНОЕ ОБРАЗОВАТЕЛЬНОЕ  
УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ  
НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ УНИВЕРСИТЕТ ИТМО**

**Факультет безопасности информационных технологий**

**Дисциплина:**

«Основы системного программирования»

**ОТЧЕТ ПО ЛАБОРАТОРНОЙ РАБОТЕ №2**

«Процессы, потоки и средства межпроцессного взаимодействия»

*Вариант 2*

**Выполнил:**

Суханкулиев Мухаммет,  
студент группы N3246



(подпись)

**Проверил:**

Грозов В. А.,  
преподаватель практики

(отметка о выполнении)

(подпись)

Санкт-Петербург

2025 г.

## СОДЕРЖАНИЕ

Введение .....	3
1    Описание .....	5
1.1    Сервер .....	5
1.2    Клиент .....	5
1.3    Makefile .....	5
2    Тестирование .....	6
2.1    Скриншоты тестирования .....	6
2.2    Тестирование утечек памяти (valgrind) .....	9
Заключение .....	10
Список использованных источников .....	11

## ВВЕДЕНИЕ

**Цель работы** – написать на языке С две программы для ОС Linux:

- сервер, поддерживающий заданный вариантом тип многозадачности (Табл. 2), транспортный протокол (Табл. 3) и прикладной протокол (Табл. 4);
- клиент, поддерживающий заданный вариантом протокол и предназначенный для тестирования сервера.

**Таблица 1 – Переменные среды и опции командной строки, поддерживаемые программой-сервером и программой-клиентом**

Опция	Переменные среды	Назначение	Значение по умолчанию	Поддерживается
-w N	LAB2WAIT	Имитация работы путем приостановки обслуживающего запрос процесса на N секунд. Если опция не задана, обслуживать запрос без задержки.	0	Сервером
-d		Работа в режиме демона.		Сервером
-l /path/to/log	LAB2LOGFILE	Путь к лог-файлу.	/tmp/lab2.log	Сервером
-a ip	LAB2ADDR	Адрес, на котором слушает сервер и к которому подключается клиент.		Сервером и клиентом
-p port	LAB2PORT	Порт, на котором слушает сервер и к которому подключается клиент.		Сервером и клиентом
-v		Вывод версии программы.		Сервером и клиентом
-h		Вывод справки по опциям.		Сервером и клиентом
	LAB2DEBUG	Включение вывода отладочной информации.		Сервером и клиентом

**Таблица 2 – Тип многозадачности программы-сервера**

Вариант (условие)	Тип многозадачности
Количество букв в фамилии студента, выполняющего работу, нечетное.	Многопроцессность (создание рабочих задач с помощью вызова fork).

**Таблица 3 – Транспортный протокол, поддерживаемый программой-сервером и программой-клиентом**

Вариант (условие)	Транспортный протокол
Количество букв в имени студента, выполняющего работу, четное.	TCP

**Таблица 4 – Прикладной протокол, поддерживаемый программой-сервером и программой-клиентом**

Вариант (номер)	Тип многозадачности
2	<p>Запрос: 2 строки, каждая из которых завершается символом LF. Первая строка содержит список вещественных чисел в обычном или научном формате, разделенных пробелами или символами табуляции (возможно, несколькими подряд). Вторая строка содержит способ округления (в сторону минус бесконечности, в сторону плюс бесконечности, в сторону нуля).</p> <p>Ответ, если ошибок не было: строка, содержащая округленные по заданному способу вещественные числа (с сохранением порядка следования в запросе, разделитель — пробел), завершающаяся символом LF.</p> <p>Ответ, если были ошибки: строка «ERROR N», завершающаяся символом LF, где N — код ошибки.</p>

## 1 ОПИСАНИЕ

Проект представляет собой клиент-серверное приложение на языке C, реализующее TCP-протокол с функционалом округления чисел по заданному режиму.

### 1.1 Сервер

- Запускается на заданном IP и порту (по умолчанию 127.0.0.1:7777), поддерживает многопроцессную обработку подключений (fork).
- Принимает от клиента две текстовые строки: список чисел и режим округления.
- Выполняет соответствующее округление и отправляет результат обратно.
- Ведёт логирование запросов и ошибок, поддерживает работу в фоновом режиме (демон).
- Обрабатывает системные сигналы для корректного завершения, сбора статистики и управления дочерними процессами.

### 1.2 Клиент

- Подключается к серверу по заданному IP и порту.
- Считывает две строки с числами и режимом округления, отправляет их серверу.
- Получает и выводит ответ от сервера.
- Поддерживает конфигурацию через переменные окружения, аргументы командной строки и режим отладки.

### 1.3 Makefile

```
CC = gcc
CFLAGS = -Wall -Wextra -Werror -O3
LDFLAGS = -lm
TARGETS = lab2msN3246_client lab2msN3246_server

.PHONY: all clean

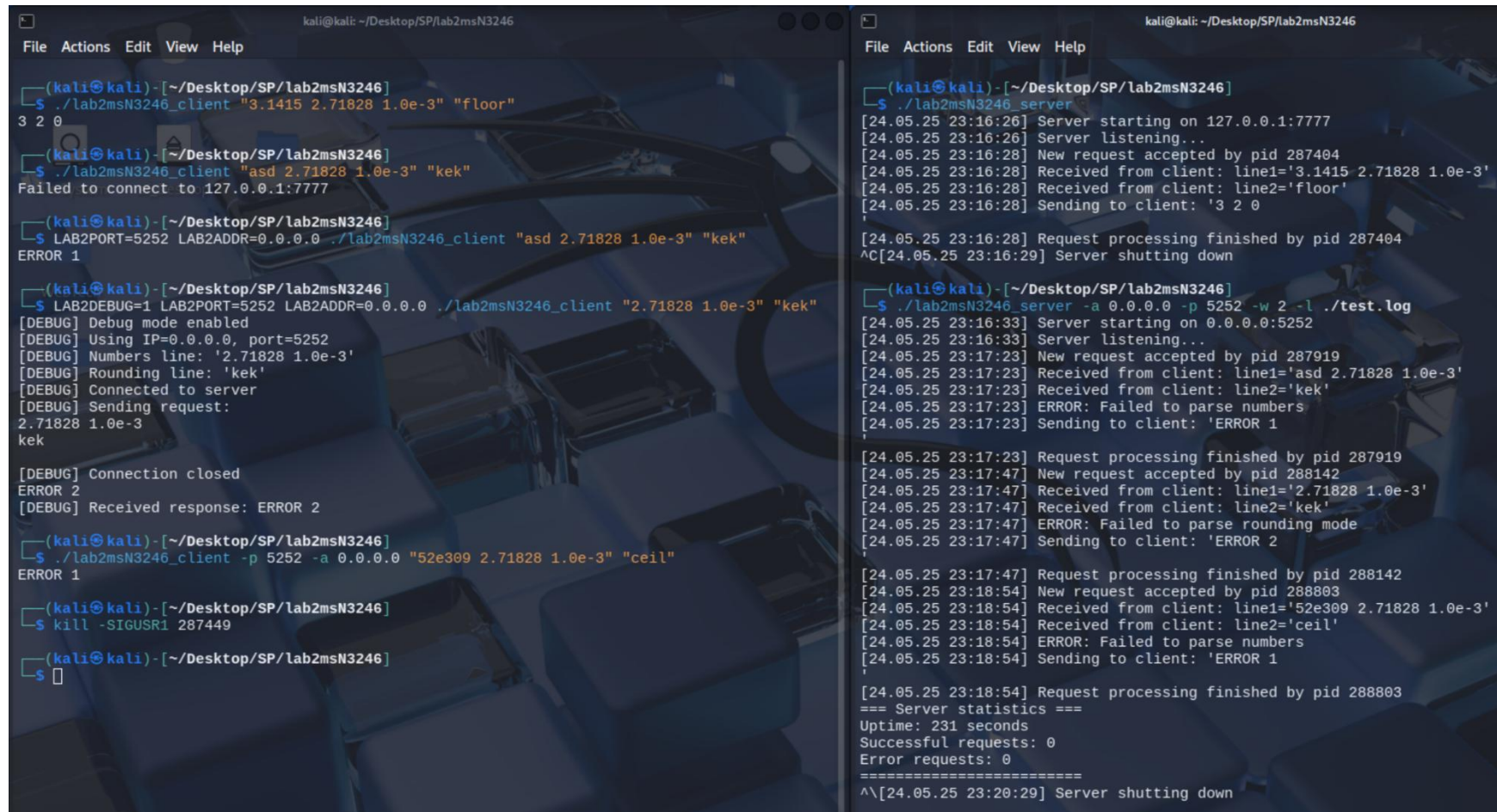
all: $(TARGETS)

%: %.c
    $(CC) $(CFLAGS) -o $@ $^ $(LDFLAGS)

clean:
    rm -f $(TARGETS)
```

## 2 ТЕСТИРОВАНИЕ

### 2.1 Скриншоты тестирования



```
(kali@kali) - [~/Desktop/SP/lab2msN3246]
$ ./lab2msN3246_client "3.1415 2.71828 1.0e-3" "floor"
3 2 0

(kali@kali) - [~/Desktop/SP/lab2msN3246]
$ ./lab2msN3246_client "asd 2.71828 1.0e-3" "kek"
Failed to connect to 127.0.0.1:7777

(kali@kali) - [~/Desktop/SP/lab2msN3246]
$ LAB2PORT=5252 LAB2ADDR=0.0.0.0 ./lab2msN3246_client "asd 2.71828 1.0e-3" "kek"
ERROR 1

(kali@kali) - [~/Desktop/SP/lab2msN3246]
$ LAB2DEBUG=1 LAB2PORT=5252 LAB2ADDR=0.0.0.0 ./lab2msN3246_client "2.71828 1.0e-3" "kek"
[DEBUG] Debug mode enabled
[DEBUG] Using IP=0.0.0.0, port=5252
[DEBUG] Numbers line: '2.71828 1.0e-3'
[DEBUG] Rounding line: 'kek'
[DEBUG] Connected to server
[DEBUG] Sending request:
2.71828 1.0e-3
kek
[DEBUG] Connection closed
ERROR 2
[DEBUG] Received response: ERROR 2

(kali@kali) - [~/Desktop/SP/lab2msN3246]
$ ./lab2msN3246_client -p 5252 -a 0.0.0.0 "52e309 2.71828 1.0e-3" "ceil"
ERROR 1

(kali@kali) - [~/Desktop/SP/lab2msN3246]
$ kill -SIGUSR1 287449

(kali@kali) - [~/Desktop/SP/lab2msN3246]
$

(kali@kali) - [~/Desktop/SP/lab2msN3246]
$ ./lab2msN3246_server
[24.05.25 23:16:26] Server starting on 127.0.0.1:7777
[24.05.25 23:16:26] Server listening...
[24.05.25 23:16:28] New request accepted by pid 287404
[24.05.25 23:16:28] Received from client: line1='3.1415 2.71828 1.0e-3'
[24.05.25 23:16:28] Received from client: line2='floor'
[24.05.25 23:16:28] Sending to client: '3 2 0'

[24.05.25 23:16:28] Request processing finished by pid 287404
^C[24.05.25 23:16:29] Server shutting down

(kali@kali) - [~/Desktop/SP/lab2msN3246]
$ ./lab2msN3246_server -a 0.0.0.0 -p 5252 -w 2 -l ./test.log
[24.05.25 23:16:33] Server starting on 0.0.0.0:5252
[24.05.25 23:16:33] Server listening...
[24.05.25 23:17:23] New request accepted by pid 287919
[24.05.25 23:17:23] Received from client: line1='asd 2.71828 1.0e-3'
[24.05.25 23:17:23] Received from client: line2='kek'
[24.05.25 23:17:23] ERROR: Failed to parse numbers
[24.05.25 23:17:23] Sending to client: 'ERROR 1'

[24.05.25 23:17:23] Request processing finished by pid 287919
[24.05.25 23:17:47] New request accepted by pid 288142
[24.05.25 23:17:47] Received from client: line1='2.71828 1.0e-3'
[24.05.25 23:17:47] Received from client: line2='kek'
[24.05.25 23:17:47] ERROR: Failed to parse rounding mode
[24.05.25 23:17:47] Sending to client: 'ERROR 2'

[24.05.25 23:17:47] Request processing finished by pid 288142
[24.05.25 23:18:54] New request accepted by pid 288803
[24.05.25 23:18:54] Received from client: line1='52e309 2.71828 1.0e-3'
[24.05.25 23:18:54] Received from client: line2='ceil'
[24.05.25 23:18:54] ERROR: Failed to parse numbers
[24.05.25 23:18:54] Sending to client: 'ERROR 1'

[24.05.25 23:18:54] Request processing finished by pid 288803
=== Server statistics ===
Uptime: 231 seconds
Successful requests: 0
Error requests: 0
=====
^C[24.05.25 23:20:29] Server shutting down
```

Рисунок 1 – Базовое тестирование

```
(kali@kali)-[~/Desktop/SP/lab2msN3246]
$ ./lab2msN3246_client "" "ceil"
ERROR 1

(kali@kali)-[~/Desktop/SP/lab2msN3246]
$ ./lab2msN3246_client "3.14" "ceil" -q
Warning: too many arguments provided, only first two will be used.
4

(kali@kali)-[~/Desktop/SP/lab2msN3246]
$ ./lab2msN3246_client "3.14" "ceil" "asdads"
Warning: too many arguments provided, only first two will be used.
4

(kali@kali)-[~/Desktop/SP/lab2msN3246]
$ kill 290678

(kali@kali)-[~/Desktop/SP/lab2msN3246]
$ ./lab2msN3246_server -d

(kali@kali)-[~/Desktop/SP/lab2msN3246]
$ ps aux | grep lab2msN3246
kali 3238 1.8 1.1 491776 213112 ? Sl 15:09 8:52 /opt/sublime_text/sublim
e_text --detached /home/kali/Desktop/SP/lab2msN3246/Makefile
kali 290678 0.0 0.0 3528 1492 ? Ss 23:22 0:00 ./lab2msN3246_server -d
kali 290790 0.0 0.0 6528 2172 pts/0 S+ 23:22 0:00 grep --color=auto lab2ms
N3246

(kali@kali)-[~/Desktop/SP/lab2msN3246]
$ ps aux | grep lab2msN3246
kali 3238 1.7 1.1 491776 214008 ? Sl 15:09 8:52 /opt/sublime_text/sublim
e_text --detached /home/kali/Desktop/SP/lab2msN3246/Makefile
kali 291901 0.0 0.0 6528 2232 pts/0 S+ 23:24 0:00 grep --color=auto lab2ms
N3246
```

Рисунок 2 – Демон

```
Rounding options: floor, ceil, trunc

(kali@kali)-[~/Desktop/SP/lab2msN3246]
$ ./lab2msN3246_client -v
Version lab2msN3246_client 1.0
Author: Суханкулиев Мухаммет
Group: N3246
Поток: ОСП N23 1.2
Variant: 2

(kali@kali)-[~/Desktop/SP/lab2msN3246]
$ ./lab2msN3246_client -q
./lab2msN3246_client: invalid option -- 'q'
Unknown option
Usage: ./lab2msN3246_client [options] ["numbers"] [rounding]
Options:
-a ip      IP address of server (env LAB2ADDR), default 127.0.0.1
-p port    Port of server (env LAB2PORT), default 7777
-v         Print version and student information
-h         Print this help

If numbers and rounding are not given as arguments, input is read from stdin.
Rounding options: floor, ceil, trunc

(kali@kali)-[~/Desktop/SP/lab2msN3246]
$ LAB2DEBUG=1 ./lab2msN3246_server -v
lab2msN3246_server 1.0
Author: Суханкулиев Мухаммет
Group: N3246
Поток: ОСП N23 1.2
Variant: 2

(kali@kali)-[~/Desktop/SP/lab2msN3246]
$ LAB2DEBUG=1 ./lab2msN3246_server -q
./lab2msN3246_server: invalid option -- 'q'
Usage: ./lab2msN3246_server [options]
Options:
-w N       Wait N seconds before processing request (env LAB2WAIT), default 0
-d         Run as daemon
-l logfile Log file path (env LAB2LOGFILE), default /tmp/lab2.log
-a ip      Server listen IP (env LAB2ADDR), default 127.0.0.1
-p port    Server listen port (env LAB2PORT), default 7777
-v         Print version and student information
-h         Print this help

Environment variables:
LAB2ADDR, LAB2PORT, LAB2LOGFILE, LAB2WAIT, LAB2DEBUG

(kali@kali)-[~/Desktop/SP/lab2msN3246]
```

Рисунок 3 – Проверка -v и -h



```
kali@kali: ~/Desktop/SP/lab2msN3246
File Actions Edit View Help

(kali@kali)~-[~/Desktop/SP/lab2msN3246]
$ ./lab2msN3246_client "2.72 5.52 6.85 -5.25 -8.25" "ceil"
3 6 7 -5 -8

(kali@kali)~-[~/Desktop/SP/lab2msN3246]
$

kali@kali: ~/Desktop/SP/lab2msN3246
File Actions Edit View Help

(kali@kali)~-[~/Desktop/SP/lab2msN3246]
$ LAB2DEBUG=1 ./lab2msN3246_server
[DEBUG] Debug mode enabled
[24.05.25 23:28:58] Server starting on 127.0.0.1:7777
[24.05.25 23:28:58] Server listening...
[24.05.25 23:30:12] New request accepted by pid 294616
[24.05.25 23:30:12] Received from client: line1='2.72 5.52 6.85 -5.25 -8.25'
[24.05.25 23:30:12] Received from client: line2='ceil'
[24.05.25 23:30:12] Sending to client: '3 6 7 -5 -8'
,
[24.05.25 23:30:12] Request processing finished by pid 294616
[24.05.25 23:30:12] [DEBUG] Reaped child process pid=294616
[24.05.25 23:30:34] New request accepted by pid 294868
[24.05.25 23:30:34] Received from client: line1='5.52'
[24.05.25 23:30:34] Received from client: line2='trunc'
[24.05.25 23:30:34] Sending to client: '5'
,
[24.05.25 23:30:34] Request processing finished by pid 294868
[24.05.25 23:30:34] [DEBUG] Reaped child process pid=294868
[24.05.25 23:31:25] New request accepted by pid 295323
[24.05.25 23:31:25] Received from client: line1='8.85 -5e-25 -20e+6'
[24.05.25 23:31:25] Received from client: line2='floor'
[24.05.25 23:31:25] Sending to client: '8 -1 -20000000'
,
[24.05.25 23:31:25] Request processing finished by pid 295323
[24.05.25 23:31:25] [DEBUG] Reaped child process pid=295323
[]

kali@kali: ~/Desktop/SP/lab2msN3246
File Actions Edit View Help

(kali@kali)~-[~/Desktop/SP/lab2msN3246]
$ LAB2DEBUG=1 ./lab2msN3246_client
[DEBUG] Debug mode enabled
[DEBUG] Using IP=127.0.0.1, port=7777
[DEBUG] Enter numbers line:

8.85 -5e-25 -20e+6
[DEBUG] Enter rounding line
floor (в сторону минус бесконечности),
ceil (в сторону плюс бесконечности),
trunc (в сторону нуля):

floor
[DEBUG] Numbers line: '8.85 -5e-25 -20e+6'
[DEBUG] Rounding line: 'floor'
[DEBUG] Connected to server
[DEBUG] Sending request:
8.85 -5e-25 -20e+6
floor

[DEBUG] Connection closed
8 -1 -20000000
[DEBUG] Received response: 8 -1 -20000000

(kali@kali)~-[~/Desktop/SP/lab2msN3246]
$

5.52
trunc
5

(kali@kali)~-[~/Desktop/SP/lab2msN3246]
$
```

Рисунок 4 – Многопроцессность



## 2.2 Тестирование утечек памяти (valgrind)

```
(kali@kali)-[~/Desktop/SP/lab2msN3246]
$ valgrind ./lab2msN3246_server
==299340== Memcheck, a memory error detector
==299340== Copyright (C) 2002-2024, and GNU GPL'd, by Julian Seward et al.
==299340== Using Valgrind-3.24.0 and LibVEX; rerun with -h for copyright info
==299340== Command: ./lab2msN3246_server
==299340==
[24.05.25 23:39:13] Server starting on 127.0.0.1:7777
[24.05.25 23:39:13] Server listening...
[24.05.25 23:39:27] New request accepted by pid 299465
[24.05.25 23:39:27] Received from client: line1='5.52 10.62'
[24.05.25 23:39:27] Received from client: line2='floor'
[24.05.25 23:39:27] Sending to client: '5 10'

[24.05.25 23:39:27] Request processing finished by pid 299465
==299465==
==299465== HEAP SUMMARY:
==299465==   in use at exit: 472 bytes in 1 blocks
==299465==   total heap usage: 20 allocs, 19 frees, 11,625 bytes allocated
==299465==
==299465== LEAK SUMMARY:
==299465==   definitely lost: 0 bytes in 0 blocks
==299465==   indirectly lost: 0 bytes in 0 blocks
==299465==   possibly lost: 0 bytes in 0 blocks
==299465==   still reachable: 472 bytes in 1 blocks
==299465==   suppressed: 0 bytes in 0 blocks
==299465== Rerun with --leak-check=full to see details of leaked memory
==299465==
==299465== For lists of detected and suppressed errors, rerun with: -s
==299465== ERROR SUMMARY: 0 errors from 0 contexts (suppressed: 0 from 0)
^Z^[24.05.25 23:39:32] Server shutting down
==299340==
==299340== HEAP SUMMARY:
==299340==   in use at exit: 0 bytes in 0 blocks
==299340==   total heap usage: 16 allocs, 16 frees, 11,239 bytes allocated
==299340==
==299340== All heap blocks were freed -- no leaks are possible
==299340==
==299340== For lists of detected and suppressed errors, rerun with: -s
==299340== ERROR SUMMARY: 0 errors from 0 contexts (suppressed: 0 from 0)
```

Рисунок 5 – Сервер

```
(kali@kali)-[~/Desktop/SP/lab2msN3246]
$ valgrind ./lab2msN3246_client "5.52 10.62" "floor"
==299464== Memcheck, a memory error detector
==299464== Copyright (C) 2002-2024, and GNU GPL'd, by Julian Seward et al.
==299464== Using Valgrind-3.24.0 and LibVEX; rerun with -h for copyright info
==299464== Command: ./lab2msN3246_client 5.52\ 10.62 floor
==299464==
5 10
==299464==
==299464== HEAP SUMMARY:
==299464==   in use at exit: 0 bytes in 0 blocks
==299464==   total heap usage: 6 allocs, 6 frees, 1,379 bytes allocated
==299464==
==299464== All heap blocks were freed -- no leaks are possible
==299464==
==299464== For lists of detected and suppressed errors, rerun with: -s
==299464== ERROR SUMMARY: 0 errors from 0 contexts (suppressed: 0 from 0)
```

Рисунок 6 – Клиент

## **ЗАКЛЮЧЕНИЕ**

В ходе лабораторной работы были разработаны и протестированы две программы на языке C для ОС Linux — многопроцессный TCP-сервер и клиент, реализующие протокол округления чисел. Реализация обеспечила поддержку командной строки, переменных среды, логирования, демонизации, а также корректную обработку системных сигналов.

Программа успешно прошла тестирование, включая проверку на утечки памяти с использованием Valgrind.

## СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

1. Керниган, Брайан У., Ритчи, Деннис М.: Язык программирования С, 2-е издание.: Пер. с англ. – М.: Издательский дом «Вильямс» – 2009. – 304 с.: ил. – Парал. тит. англ. – URL: [Керниган, Ритчи. Язык программирования С.pdf - Google Диск](#)
2. Гирик А. В.: Основы системного программирования. Многозадачность и процессы. Трассировка программ – Университет ИТМО – 2025. – URL: [04. ОСП. Многозадачность и процессы. Трассировка программ.pdf - Google Диск](#)
3. Гирик А. В.: Основы системного программирования. Потоки и синхронизация – Университет ИТМО – 2025. – URL: [05. ОСП. Потоки и синхронизация.pdf - Google Диск](#)
4. Гирик А. В.: Основы системного программирования. Сигналы и средства межпроцессного взаимодействия – Университет ИТМО – 2025. – URL: [06. ОСП. Сигналы и средства межпроцессного взаимодействия.pdf - Google Диск](#)
5. Гирик А. В.: Основы системного программирования. Сетевое программирование и сокеты – Университет ИТМО – 2025. – URL: [07. ОСП. Сетевое программирование и сокеты.pdf - Google Диск](#)

## ПРИЛОЖЕНИЕ А

### (обязательное)

#### Исходные коды с комментариями

##### Листинг А.1 – lab2msN3246\_server.c

```
#define _POSIX_C_SOURCE 200809L
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <strings.h>
#include <ctype.h>
#include <errno.h>
#include <time.h>
#include <unistd.h>
#include <signal.h>
#include <sys/types.h>
#include <sys/socket.h>
#include <netinet/in.h>
#include <arpa/inet.h>
#include <sys/wait.h>
#include <math.h>
#include <stdarg.h>

#define DEFAULT_ADDR "127.0.0.1"
#define DEFAULT_PORT 7777
#define DEFAULT_LOGFILE "/tmp/lab2.log"
#define VERSION "lab2msN3246_server 1.0\nAuthor: Суханкулиев Мухаммет\nGroup: N3246\nПоток: ОСП N23 1.2\nVariant: 2\n"

// Для обработки сигналов
static volatile sig_atomic_t stop_flag = 0;
static volatile sig_atomic_t print_stats_flag = 0;

// Логгирование
static FILE *log_file = NULL;
static int debug_enabled = 0;
static unsigned long successful_requests = 0;
static unsigned long error_requests = 0;
static time_t server_start_time;

// Опции сервера
static unsigned int wait_seconds = 0; // DEFAULTWAIT
static int daemon_mode = 0;

static void print_help(const char *prog) {
    fprintf(stderr,
        "Usage: %s [options]\n"
        "Options:\n"
        "  -w N      Wait N seconds before processing request (env LAB2WAIT),\n"
        "  -d        Run as daemon\n"
        "  -l logfile Log file path (env LAB2LOGFILE), default %s\n"
        "  -a ip      Server listen IP (env LAB2ADDR), default %s\n"
        "  -p port    Server listen port (env LAB2PORT), default %d\n"
        "  -v        Print version and student information\n"
        "  -h        Print this help\n"
        "Environment variables:\n"
        "  LAB2ADDR, LAB2PORT, LAB2LOGFILE, LAB2WAIT, LAB2DEBUG\n",
        prog, DEFAULT_LOGFILE, DEFAULT_ADDR, DEFAULT_PORT);
}

static void debug_printf(const char *fmt, ...) {
    if (!debug_enabled) return;
    va_list ap;
```

```

    va_start(ap, fmt);
    fprintf(stderr, "[DEBUG] ");
    vfprintf(stderr, fmt, ap);
    fprintf(stderr, "\n");
    va_end(ap);
}

//
// Записать текущее время в формате "дд.мм.гг ЧЧ:ММ:СС"
//
static void timestamp(char *buf, size_t size) {
    time_t now = time(NULL);
    struct tm tm;
    localtime_r(&now, &tm);
    strftime(buf, size, "%d.%m.%y %H:%M:%S", &tm);
}

//
// Запись лога в консоль и в файл
//
static void log_msg(const char *fmt, ...) {
    if (!log_file) return;
    char timebuf[32];
    timestamp(timebuf, sizeof timebuf);

    va_list args;

    fprintf(log_file, "[%s] ", timebuf);
    va_start(args, fmt);
    vfprintf(log_file, fmt, args);
    va_end(args);
    fprintf(log_file, "\n");
    fflush(log_file);

    printf("[%s] ", timebuf);
    va_start(args, fmt);
    vprintf(fmt, args);
    va_end(args);
    printf("\n");
    fflush(stdout);
}

//
// Логгирование ошибки
//
static void log_error(const char *fmt, ...) {
    if (!log_file) return;
    char timebuf[32];
    timestamp(timebuf, sizeof timebuf);

    va_list args;

    fprintf(log_file, "[%s] ERROR: ", timebuf);
    va_start(args, fmt);
    vfprintf(log_file, fmt, args);
    va_end(args);
    fprintf(log_file, "\n");
    fflush(log_file);

    fprintf(stderr, "[%s] ERROR: ", timebuf);
    va_start(args, fmt);
    vfprintf(stderr, fmt, args);
    va_end(args);
    fprintf(stderr, "\n");
    fflush(stderr);
}

//

```

```

// Обработчик сигнала для убийства детей (child processes)
//
void reap_children(int signo) {
    (void)signo;
    int status;
    pid_t pid;
    while ((pid = waitpid(-1, &status, WNOHANG)) > 0) {
        if (debug_enabled) {
            log_msg("[DEBUG] Reaped child process pid=%d", pid);
        }
    }
}

//
// Обработчик SIGINT, SIGTERM, SIGQUIT
//
void termination_handler(int signo) {
    (void)signo;
    stop_flag = 1;
}

//
// Обработчик SIGUSR1
//
void sigusr1_handler(int signo) {
    (void)signo;
    print_stats_flag = 1;
}

//
// Донастройка обработчиков
//
void setup_signal_handlers() {
    struct sigaction sa;

    // SIGINT, SIGTERM, SIGQUIT для выключения
    sa.sa_handler = termination_handler;
    sigemptyset(&sa.sa_mask);
    sa.sa_flags = 0;
    sigaction(SIGINT, &sa, NULL);
    sigaction(SIGTERM, &sa, NULL);
    sigaction(SIGQUIT, &sa, NULL);

    // SIGUSR1 чтобы вывести статистику
    sa.sa_handler = sigusr1_handler;
    sigemptyset(&sa.sa_mask);
    sa.sa_flags = 0;
    sigaction(SIGUSR1, &sa, NULL);

    // SIGCHLD РИПнуть детей
    sa.sa_handler = reap_children;
    sigemptyset(&sa.sa_mask);
    sa.sa_flags = SA_RESTART | SA_NOCLDSTOP;
    sigaction(SIGCHLD, &sa, NULL);
}

void print_stats() {
    time_t now = time(NULL);
    unsigned long uptime = now - server_start_time;

    fprintf(stderr, "=== Server statistics ===\n");
    fprintf(stderr, "Uptime: %lu seconds\n", uptime);
    fprintf(stderr, "Successful requests: %lu\n", successful_requests);
    fprintf(stderr, "Error requests: %lu\n", error_requests);
    fprintf(stderr, "=====\n");
    fflush(stderr);
}

```



```

//
// Демонизация с помощью fork и перенаправления stdio в /dev/null
//
static void daemonize(void) {
    pid_t pid = fork();
    if (pid < 0) {
        perror("fork");
        exit(EXIT_FAILURE);
    }
    if (pid > 0) {
        exit(EXIT_SUCCESS);
    }
    if (setsid() < 0) {
        perror("setsid");
        exit(EXIT_FAILURE);
    }
    freopen("/dev/null", "r", stdin);
    freopen("/dev/null", "w", stdout);
    freopen("/dev/null", "w", stderr);
}

static void open_log_file(const char *path) {
    if (log_file) fclose(log_file);
    log_file = fopen(path, "a");
    if (!log_file) {
        fprintf(stderr, "ERROR: cannot open log file %s: %s\n", path,
strerror(errno));
        exit(EXIT_FAILURE);
    }
}

//
// Проверка на пробельные символы
//
static int is_lab2_space(char c) {
    return c == 0x20 || c == 0x09 || c == 0x0B || c == 0x0C || c == 0x0D;
}

//
// Обработка строки чисел
//
static double* parse_numbers(const char *line, size_t *count) {
    if (!line || !count) return NULL;

    const char *p = line;
    while (*p && is_lab2_space(*p)) p++;
    if (*p == '\0' || *p == '\n') {
        // Пустая строка
        return NULL;
    }

    size_t capacity = 8;
    double *numbers = malloc(capacity * sizeof(double));
    if (!numbers) return NULL;

    *count = 0;

    while (*p) {
        while (*p && is_lab2_space(*p)) p++;
        if (!*p || *p == '\n') break;

        char *endptr;
        errno = 0;
        double val = strtod(p, &endptr);
        if (p == endptr) {
            free(numbers);
            return NULL;
        }
    }
}

```

```

        if (errno == ERANGE) {
            free(numbers);
            return NULL;
        }
        if (*endptr && !is_lab2_space(*endptr) && *endptr != '\n') {
            free(numbers);
            return NULL; // Инвалидный характер
        }

        if (*count >= capacity) {
            capacity *= 2;
            double *tmp = realloc(numbers, capacity * sizeof(double));
            if (!tmp) {
                free(numbers);
                return NULL;
            }
            numbers = tmp;
        }
        numbers[*count] = val;
        (*count)++;

        p = endptr;
    }
    return numbers;
}

//
// Способы округления
//
enum RoundMode { ROUND_FLOOR, ROUND_CEIL, ROUND_TRUNC };

//
// Обработка второй строки
//
static int parse_round_mode(const char *line, enum RoundMode *mode) {
    if (!line || !mode) return -1;

    char buf[64];
    size_t len = strlen(line);
    if (len >= sizeof(buf)) return -1;
    size_t j = 0;
    for (size_t i = 0; i < len; i++) {
        if (!is_lab2_space(line[i]) && line[i] != '\n') {
            buf[j++] = (char)tolower((unsigned char)line[i]);
        }
    }
    buf[j] = '\0';

    if (strcmp(buf, "floor") == 0) {
        *mode = ROUND_FLOOR;
        return 0;
    }
    if (strcmp(buf, "ceil") == 0) {
        *mode = ROUND_CEIL;
        return 0;
    }
    if (strcmp(buf, "trunc") == 0) {
        *mode = ROUND_TRUNC;
        return 0;
    }
    return -1;
}

static void apply_rounding(double *nums, size_t count, enum RoundMode mode) {
    for (size_t i = 0; i < count; i++) {
        switch (mode) {
            case ROUND_FLOOR: nums[i] = floor(nums[i]); break;
            case ROUND_CEIL:  nums[i] = ceil(nums[i]); break;
        }
    }
}

```

```

        case ROUND_TRUNC: nums[i] = trunc(nums[i]); break;
    }
}

static char* format_response(const double *nums, size_t count) {
    if (!nums) return NULL;

    // Приблизительный размер (max 30 chars per number + spaces + LF)
    size_t bufsize = count * 32 + 2;
    char *resp = malloc(bufsize);
    if (!resp) return NULL;

    size_t offset = 0;
    for (size_t i = 0; i < count; i++) {
        int written = snprintf(resp + offset, bufsize - offset, "%.0f", nums[i]);
        if (written < 0 || (size_t)written >= bufsize - offset) {
            free(resp);
            return NULL;
        }
        offset += (size_t)written;
        if (i + 1 < count) {
            if (offset < bufsize - 1) {
                resp[offset] = ' ';
                offset++;
                resp[offset] = '\\0';
            }
        }
    }
    // Добавить завершающий LF (в выводе некрасиво получается, но зато все работает)
    if (offset < bufsize - 1) {
        resp[offset] = '\\n';
        offset++;
        resp[offset] = '\\0';
    } else {
        free(resp);
        return NULL;
    }
    return resp;
}

//
// ERROR N
//
static char* format_error(int code) {
    char *resp = malloc(32);
    if (!resp) return NULL;
    snprintf(resp, 32, "ERROR %d\\n", code);
    return resp;
}

static char* read_line(int sockfd) {
    size_t bufsize = 128;
    char *buffer = malloc(bufsize);
    if (!buffer) return NULL;

    size_t pos = 0;
    while (1) {
        char c;
        ssize_t r = recv(sockfd, &c, 1, 0);
        if (r < 0) {
            free(buffer);
            return NULL;
        }
        if (r == 0) {
            // EOF before LF, treat as end
            break;
        }
    }
}

```

```

        if (c == '\n') {
            buffer[pos] = '\0';
            return buffer;
        }
        buffer[pos++] = c;
        if (pos >= bufsize) {
            bufsize *= 2;
            char *tmp = realloc(buffer, bufsize);
            if (!tmp) {
                free(buffer);
                return NULL;
            }
            buffer = tmp;
        }
    }
    buffer[pos] = '\0';
    return buffer;
}

//
// Обработка клиента
//
static void handle_client(int client_fd) {
    if (wait_seconds > 0) {
        sleep(wait_seconds);
    }

    log_msg("New request accepted by pid %d", getpid());

    // Две строки от клиента
    char *line1 = read_line(client_fd);
    if (!line1) {
        log_error("Failed to read first line");
        debug_printf("read_line(client_fd) returned NULL at first line");
        goto error;
    }
    char *line2 = read_line(client_fd);
    if (!line2) {
        log_error("Failed to read second line");
        free(line1);
        goto error;
    }

    log_msg("Received from client: line1='%s'", line1);
    log_msg("Received from client: line2='%s'", line2);

    // Числа (первая строка)
    size_t count;
    double *numbers = parse_numbers(line1, &count);
    if (!numbers) {
        log_error("Failed to parse numbers");
        free(line1);
        free(line2);
        char *resp = format_error(1);
        if (resp) {
            log_msg("Sending to client: '%s'", resp);
            send(client_fd, resp, strlen(resp), 0);
        }
        free(resp);
        error_requests++;
        goto done;
    }

    // Способ округления (вторая строка)
    enum RoundMode mode;
    if (parse_round_mode(line2, &mode) != 0) {
        log_error("Failed to parse rounding mode");
        free(line1);
    }

```

```

        free(line2);
        free(numbers);
        char *resp = format_error(2);
        if (resp) {
            log_msg("Sending to client: '%s'", resp);
            send(client_fd, resp, strlen(resp), 0);
        }
        free(resp);
        error_requests++;
        goto done;
    }

    apply_rounding(numbers, count, mode);

    char *resp = format_response(numbers, count);
    if (!resp) {
        log_error("Failed to format response");
        free(line1);
        free(line2);
        free(numbers);
        char *resp_err = format_error(3);
        if (resp_err) {
            log_msg("Sending to client: '%s'", resp_err);
            send(client_fd, resp_err, strlen(resp_err), 0);
        }
        free(resp_err);
        error_requests++;
        goto done;
    }

    log_msg("Sending to client: '%s'", resp);

    // Отправка пакета
    ssize_t sent = send(client_fd, resp, strlen(resp), 0);
    if (sent < 0) {
        log_error("Failed to send response");
        error_requests++;
    } else {
        successful_requests++;
    }
    free(resp);
    free(numbers);
    free(line1);
    free(line2);

done:
    log_msg("Request processing finished by pid %d", getpid());
    close(client_fd);
    exit(EXIT_SUCCESS);

error:
    close(client_fd);
    exit(EXIT_FAILURE);
}

//
// Это мейн
//
int main(int argc, char *argv[]) {
    const char *addr = NULL;
    unsigned short port = 0;
    const char *logpath = NULL;

    char *env_addr = getenv("LAB2ADDR");
    char *env_port = getenv("LAB2PORT");
    char *env_logfile = getenv("LAB2LOGFILE");
    char *env_wait = getenv("LAB2WAIT");
    char *env_debug = getenv("LAB2DEBUG");

```

```

if (env_addr && *env_addr) addr = env_addr;
if (env_port) port = (unsigned short)atoi(env_port);
if (env_logfile && *env_logfile) logpath = env_logfile;
if (env_wait) wait_seconds = (unsigned int)atoi(env_wait);
if (env_debug && (strcmp(env_debug, "1") == 0 || strcasecmp(env_debug, "true") ==
0))
    debug_enabled = 1;

if (!addr) addr = DEFAULT_ADDR;
if (port == 0) port = DEFAULT_PORT;
if (!logpath) logpath = DEFAULT_LOGFILE;

int opt;
while ((opt = getopt(argc, argv, "a:p:l:w:dvh")) != -1) {
    switch (opt) {
        case 'a': addr = optarg; break;
        case 'p': port = (unsigned short)atoi(optarg); break;
        case 'l': logpath = optarg; break;
        case 'w': wait_seconds = (unsigned int)atoi(optarg); break;
        case 'd': daemon_mode = 1; break;
        case 'v':
            printf("%s\n", VERSION);
            exit(EXIT_SUCCESS);
        case 'h':
        default:
            print_help(argv[0]);
            exit(EXIT_SUCCESS);
    }
}

open_log_file(logpath);

if (daemon_mode) {
    daemonize();
}

debug_printf("Debug mode enabled");
log_msg("Server starting on %s:%u", addr, port);
server_start_time = time(NULL);

setup_signal_handlers();

// Создание сокета
int server_fd = socket(AF_INET, SOCK_STREAM, 0);
if (server_fd < 0) {
    log_error("socket() failed: %s", strerror(errno));
    exit(EXIT_FAILURE);
}

// Разрешить REUSEADDR
int optval = 1;
if (setsockopt(server_fd, SOL_SOCKET, SO_REUSEADDR, &optval, sizeof(optval)) < 0)
{
    log_error("setsockopt() failed: %s", strerror(errno));
    close(server_fd);
    exit(EXIT_FAILURE);
}

struct sockaddr_in serv_addr = {0};
serv_addr.sin_family = AF_INET;
serv_addr.sin_port = htons(port);
if (inet_pton(AF_INET, addr, &serv_addr.sin_addr) != 1) {
    log_error("Invalid IP address: %s", addr);
    close(server_fd);
    exit(EXIT_FAILURE);
}

```



```

if (bind(server_fd, (struct sockaddr *)&serv_addr, sizeof(serv_addr)) < 0) {
    log_error("bind() failed: %s", strerror(errno));
    close(server_fd);
    exit(EXIT_FAILURE);
}

// Слушать
if (listen(server_fd, 5) < 0) {
    log_error("listen() failed: %s", strerror(errno));
    close(server_fd);
    exit(EXIT_FAILURE);
}

log_msg("Server listening...");

// Мэйн цикл
while (!stop_flag) {
    if (print_stats_flag) {
        print_stats();
        print_stats_flag = 0;
    }

    struct sockaddr_in client_addr;
    socklen_t client_len = sizeof(client_addr);
    int client_fd = accept(server_fd, (struct sockaddr *)&client_addr,
&client_len);
    if (client_fd < 0) {
        if (errno == EINTR) continue; // Interrupted by signal, retry
        log_error("accept() failed: %s", strerror(errno));
        break;
    }

    pid_t pid = fork();
    if (pid < 0) {
        log_error("fork() failed: %s", strerror(errno));
        close(client_fd);
        continue;
    } else if (pid == 0) {
        // Обработка клиента дочерним процессом (ребенком)
        close(server_fd);
        handle_client(client_fd);
        close(client_fd);
        exit(EXIT_SUCCESS);
    } else {
        // Родитель закрывает ребенка
        close(client_fd);
    }
}

log_msg("Server shutting down");
close(server_fd);
if (log_file) fclose(log_file);

return 0;
}

```

## Листинг А.2 – lab2msN3246\_client.c

```

#define _POSIX_C_SOURCE 200809L
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <errno.h>
#include <unistd.h>
#include <arpa/inet.h>
#include <netdb.h>
#include <sys/socket.h>
#include <sys/types.h>
#include <getopt.h>

```

```

#include <stdarg.h>

#define DEFAULT_ADDR "127.0.0.1"
#define DEFAULT_PORT "7777"
#define VERSION "lab2msN3246_client 1.0\nAuthor: Суханкулиев Мухаммет\nGroup: N3246\nПоток: ОСП N23 1.2\nVariant: 2\n"

static int debug_enabled = 0;

static void print_help(const char *prog) {
    fprintf(stdout,
        "Usage: %s [options] [\"numbers\"] [rounding]\n"
        "Options:\n"
        "  -a ip      IP address of server (env LAB2ADDR), default %s\n"
        "  -p port    Port of server (env LAB2PORT), default %s\n"
        "  -v         Print version and student information\n"
        "  -h         Print this help\n\n"
        "If numbers and rounding are not given as arguments, input is read from"
        stdin.\n"
        "Rounding options: floor, ceil, trunc\n"
        , prog, DEFAULT_ADDR, DEFAULT_PORT);
}

static void debug_printf(const char *fmt, ...) {
    if (!debug_enabled) return;
    va_list ap;
    va_start(ap, fmt);
    fprintf(stderr, "[DEBUG] ");
    vfprintf(stderr, fmt, ap);
    fprintf(stderr, "\n");
    va_end(ap);
}

//
// Валидность IPv4
//
static int validate_ip(const char *ip) {
    struct sockaddr_in sa;
    return inet_pton(AF_INET, ip, &(sa.sin_addr)) == 1;
}

static int validate_port(const char *port_str) {
    char *endptr;
    long port = strtol(port_str, &endptr, 10);
    if (*endptr != '\0' || port < 1 || port > 65535) return 0;
    return 1;
}

//
// Возвращает преобразованную строку без пробельных символов
// p.s. на сервере тоже обрабатывается
//
static char *read_line(FILE *f) {
    size_t size = 256;
    size_t len = 0;
    char *buf = malloc(size);
    if (!buf) return NULL;

    while (1) {
        if (fgets(buf + len, (int)(size - len), f) == NULL) {
            if (len == 0) {
                free(buf);
                return NULL;
            }
            break;
        }
        len += strlen(buf + len);
        if (len > 0 && buf[len - 1] == '\n') {

```

```

        buf[len - 1] = '\\0'; // Удалить завершающую новую строку
        break;
    }
    // Увеличить буффер
    if (len == size - 1) {
        size *= 2;
        char *tmp = realloc(buf, size);
        if (!tmp) {
            free(buf);
            return NULL;
        }
        buf = tmp;
    }
    return buf;
}

int main(int argc, char *argv[]) {
    // Получить переменные окружения, если заданы
    const char *ip = getenv("LAB2ADDR");
    if (!ip) ip = DEFAULT_ADDR;
    const char *port = getenv("LAB2PORT");
    if (!port) port = DEFAULT_PORT;
    const char *debug_env = getenv("LAB2DEBUG");
    debug_enabled = (debug_env != NULL && strlen(debug_env) > 0);

    int opt;
    while ((opt = getopt(argc, argv, "a:p:vh")) != -1) {
        switch (opt) {
            case 'a':
                ip = optarg;
                break;
            case 'p':
                port = optarg;
                break;
            case 'v':
                printf("Version %s\\n", VERSION);
                return 0;
            case 'h':
                print_help(argv[0]);
                return 0;
            default:
                fprintf(stderr, "Unknown option\\n");
                print_help(argv[0]);
                return 1;
        }
    }

    debug_printf("Debug mode enabled");
    debug_printf("Using IP=%s, port=%s", ip, port);

    if (!validate_ip(ip)) {
        fprintf(stderr, "Invalid IP address: %s\\n", ip);
        return 1;
    }
    if (!validate_port(port)) {
        fprintf(stderr, "Invalid port: %s\\n", port);
        return 1;
    }

    // Прочитать входные строки
    char *numbers_line = NULL;
    char *rounding_line = NULL;

    int remaining_args = argc - optind;

    if (remaining_args > 2) {

```

```

    fprintf(stderr, "Warning: too many arguments provided, only first two will be
used.\n");
}
if (remaining_args >= 2) {
    numbers_line = strdup(argv[optind]);
    rounding_line = strdup(argv[optind + 1]);
    if (!numbers_line || !rounding_line) {
        fprintf(stderr, "Memory allocation failed\n");
        free(numbers_line);
        free(rounding_line);
        return 1;
    }
} else if (remaining_args == 1) {
    // Числа из первого аргумента, округление из stdin
    numbers_line = strdup(argv[optind]);
    if (!numbers_line) {
        fprintf(stderr, "Memory allocation failed\n");
        return 1;
    }
    debug_printf("Enter rounding line\n          floor (в сторону минус
бесконечности),\n          ceil (в сторону плюс бесконечности),\n          trunc (в
сторону нуля):\n");
    rounding_line = read_line(stdin);
    if (!rounding_line) {
        fprintf(stderr, "Failed to read rounding line\n");
        free(numbers_line);
        return 1;
    }
} else {
    debug_printf("Enter numbers line:\n");
    numbers_line = read_line(stdin);
    if (!numbers_line) {
        fprintf(stderr, "Failed to read numbers line\n");
        return 1;
    }
    debug_printf("Enter rounding line\n          floor (в сторону минус
бесконечности),\n          ceil (в сторону плюс бесконечности),\n          trunc (в
сторону нуля):\n");
    rounding_line = read_line(stdin);
    if (!rounding_line) {
        fprintf(stderr, "Failed to read rounding line\n");
        free(numbers_line);
        return 1;
    }
}

debug_printf("Numbers line: '%s'", numbers_line);
debug_printf("Rounding line: '%s'", rounding_line);

// Подготовка к подключению
struct addrinfo hints = {0}, *res = NULL;
hints.ai_family = AF_INET; // IPv4
hints.ai_socktype = SOCK_STREAM; // TCP

int gai_err = getaddrinfo(ip, port, &hints, &res);
if (gai_err != 0) {
    fprintf(stderr, "getaddrinfo: %s\n", gai_strerror(gai_err));
    free(numbers_line);
    free(rounding_line);
    return 1;
}

// Подключение
int sock = -1;
for (struct addrinfo *p = res; p != NULL; p = p->ai_next) {
    sock = socket(p->ai_family, p->ai_socktype, p->ai_protocol);
    if (sock == -1) continue;
}

```

```

        if (connect(sock, p->ai_addr, p->ai_addrlen) == 0) {
            debug_printf("Connected to server");
            break;
        }
        close(sock);
        sock = -1;
    }
    freeaddrinfo(res);

    if (sock == -1) {
        fprintf(stderr, "Failed to connect to %s:%s\n", ip, port);
        free(numbers_line);
        free(rounding_line);
        return 1;
    }

    // Подготовка в отправке запроса
    size_t req_len = strlen(numbers_line) + strlen(rounding_line) + 2;
    char *request = malloc(req_len + 1);
    if (!request) {
        fprintf(stderr, "Memory allocation failed\n");
        free(numbers_line);
        free(rounding_line);
        close(sock);
        return 1;
    }
    snprintf(request, req_len + 1, "%s\n%s\n", numbers_line, rounding_line);

    debug_printf("Sending request:\n%s", request);

    // Отправка
    ssize_t sent = 0;
    size_t to_send = req_len;
    while (to_send > 0) {
        ssize_t n = send(sock, request + sent, to_send, 0);
        if (n <= 0) {
            fprintf(stderr, "Send error: %s\n", strerror(errno));
            free(numbers_line);
            free(rounding_line);
            free(request);
            close(sock);
            return 1;
        }
        sent += n;
        to_send -= n;
    }

    free(numbers_line);
    free(rounding_line);
    free(request);

    // Получить от сервера
    size_t bufsize = 256;
    size_t pos = 0;
    char *response = malloc(bufsize);
    if (!response) {
        fprintf(stderr, "Memory allocation failed\n");
        close(sock);
        return 1;
    }

    while (1) {
        ssize_t n = recv(sock, response + pos, 1, 0);
        if (n <= 0) {
            if (n == 0)
                fprintf(stderr, "Connection closed by server unexpectedly\n");
            else
                fprintf(stderr, "Recv error: %s\n", strerror(errno));
        }
    }

```

```

        free(response);
        close(sock);
        return 1;
    }
    if (response[pos] == '\n') {
        response[pos] = '\0';
        break;
    }
    pos++;
    if (pos + 1 >= bufsize) {
        bufsize *= 2;
        char *tmp = realloc(response, bufsize);
        if (!tmp) {
            fprintf(stderr, "Memory allocation failed\n");
            free(response);
            close(sock);
            return 1;
        }
        response = tmp;
    }
}

close(sock);
debug_printf("Connection closed");

// Вывести ответ сервера в stdout
printf("%s\n", response);

if (debug_enabled) {
    fprintf(stderr, "[DEBUG] Received response: %s\n", response);
}

free(response);
return 0;
}

```