

Министерство науки и высшего образования Российской Федерации
Федеральное государственное автономное образовательное учреждение
высшего образования

**«Пермский национальный исследовательский
политехнический университет»**

Электротехнический факультет

Кафедра «Информационные технологии и автоматизированные системы»
направление подготовки: 09.03.04 Программная инженерия

Системное программирование

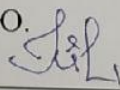
Курсовая работа

Тема: «Взаимодействие процессов в многозадачной среде»

Вариант 18

Выполнил студент группы РИС-19-16

Миннахметов Э.Ю.



(подпись)

Проверил:

ст. преподаватель Кузнецов Д. Б.



(оценка)

(подпись)

28.12.2021
(дата)

Пермь 2021

СОДЕРЖАНИЕ

ВВЕДЕНИЕ.....	3
1 Анализ поставленной задачи	4
1.1 Постановка задачи.....	4
1.2 Анализ задачи	4
1.3 Анализ средств для решения задачи	5
2 Технология реализации	7
2.1 Алгоритм работы.....	8
2.2 Данные.....	11
2.3 Функции	12
2.4 Сценарий компиляции	13
2.5 Демонстрация работы	14
ЗАКЛЮЧЕНИЕ	15
СПИСОК ИСПОЛЬЗУЕМЫХ ИСТОЧНИКОВ	16
ПРИЛОЖЕНИЕ А	17
ПРИЛОЖЕНИЕ Б.....	18
ПРИЛОЖЕНИЕ В	19

ВВЕДЕНИЕ

Одной из проблем разработки программного обеспечения является проблема многопоточности, то есть распределение ресурсов системы для более эффективного и корректного выполнения программы в более, чем одной нити или, говоря иначе, ядре процессора.^[1]

Ярким примером частой встречи такой проблемы являются серверные приложения. Сервер должен отвечать по мере поступления запросов, исключая нарастание очереди. Процесс формирования ответа, таким образом, отводится отдельной нитью, а та, в свою очередь должна иметь доступ к некоторым общим ресурсам.

Разрешение доступа к общим ресурсам является сутью проблемы многопоточных программ, а суть курсовой работы – провести контроль знаний решения таковой проблемы, владения сетевыми протоколами и операционными системами.

Перед исполнителем курсовой работы ставится **цель** – разработать приложение для подсчета суммы чисел, получаемых сервером от клиентов, и отправкой её текущего состояния каждому клиенту в ответ.

Поставленная цель достигается за счет последовательного решения ряда **задач**:

- провести анализ предметной области и средств необходимых для достижения цели;
- разработать технология реализации приложения;
- сконструировать алгоритм работы программы;
- определить типы данных и функции, реализующие логику работы;
- провести подробный разбор работы приложения с заданными параметрами.

1 Анализ предметной области

1.1 Постановка задачи

Необходимо разработать программу на Си. Обмен между клиентом и сервером должен происходить с использованием ТСР-подключения. При этом каждое обращение клиента сервер должен обрабатывать в отдельной нити. Каждый клиент должен посылать на сервер число. В ответе сервер должен передать сумму чисел клиентов на текущий момент. Программа должна выполняться на операционной системе из семейства Linux.

1.2 Анализ задачи

Создание клиент-серверного приложения подразумевает разработку отдельно серверной и клиентской части так, чтобы можно было запустить один сервер, к которому подключается и отключаются клиенты, запуск которых производится из терминала (командного интерпретатора).^[2]

Для того, чтобы на каждый запрос клиента сервер возвращал текущую сумму чисел, необходимо объявить переменную, в которой будет она храниться. Изначально ей будет присвоен ноль. Также будет иметься счетчик клиентов для информативности вывода в терминале сервера.

Доступ к счетчику и сумме чисел будут иметь все нити, поэтому они является критической секцией программы, работа с которой в многопоточном приложении может сопровождаться коллизиями, ошибками доступа, чтения и записи.

Поэтому счетчик и сумма будут защищены семафором, который контролирует доступ нитей к области памяти, в которой хранится счетчик.

Таким образом, каждая нить сможет безопасно изменить или получить текущее значение счетчика и суммы.

По условию задачи на каждого клиента необходимо выделить одну нить. Для этого после запуска приложения будет создаваться столько нитей, сколько будет клиентов.

Каждая нить будет ждать запрос от своего клиента. Как только запрос будет получен, сервер обработает его, увеличит счетчик запросов, выполнит суммирование текущей суммы с переданным числом, и вернет текущую сумму клиенту. Общение между клиентом и сервером происходит на основе TCP-подключения.

1.3 Анализ средств для решения задачи

В условиях задачи требуется написать программу на C для Linux, задействовав сетевой протокол TCP, и использовать нити, исключая формирование очереди клиентов. Но Linux – не операционная система, а семейство операционных систем, поэтому необходимо произвести выбор конкретной системы, а затем выбрать среду разработки, систему сборки и средство работы с защищенной памятью.

Операционная система

Существует несколько крупных семейств Linux: Debian, Red Hat, Arch. Новичкам обычно советуют начинать изучать Linux с Debian или его потомков, поэтому далее будет выполняться выбор из этого подмножества.

Весьма отличный потомок Debian – KDE Neon – дистрибутив, который содержит все необходимые библиотеки для разработки на языке C, имеет красивую графическую оболочку KDE Plasma и минимум предустановленных программ.

Интегрированная среда разработки

Для Linux существует большое множество хороших сред разработки на C – например, Visual Studio Code, KDevelop, QtCreator, CodeBlocks и т.д. Однако самая богатая на функционал и, в то же время, простая и готовая к работе сразу после установке – CLion.

CLion – интегрированная среда разработки для языков C и C++ от компании JetBrains. Подходит для разработки на Windows, Linux и macOS. CLion работает с такими системами сборки, как Make, CMake и даже QMake для фреймворка Qt.

Система сборки

Компилировать проект можно двумя способами: с помощью компилятора с заданием всех опций компиляции в команде, с помощью системы сборки. Первый – очень громоздок и рассматриваться не будет. Второй – призван упростить процесс компиляции с помощью файлов, описывающих процесс компиляции – примером таковой является система сборки Make. Далее будут рассматриваться второй способ.

Однако, Make – не панацея. На текущий момент, передовым средством сборки проектов на языке C является CMake. Она обладает лучшим синтаксисом.

CMake – кроссплатформенная утилита для автоматической сборки программы из исходного кода. При этом сама CMake непосредственно сборкой не занимается, а представляет из себя front-end. В качестве back-end`а могут выступать различные версии make и Ninja.

Доступ к памяти в многопоточном режиме

Существует два примитива для доступа к памяти в многопоточном режиме: мьютекс и семафор. Мьютекс имеет два состояния – он либо открыт, либо закрыт. Семафор имеет счётчик, который позволяет расставлять нити в очередь для доступа к памяти. В серверных приложениях семафор считает более эффективным средством доступа к памяти, поэтому в курсовой работе будет использоваться именно он.

Семафор – примитив синхронизации работы процессов и потоков, в основе которого лежит счётчик, над которым можно производить две атомарные операции: увеличение и уменьшение значения на единицу, при этом операция уменьшения для нулевого значения счётчика является блокирующей.

2 Технология реализации

Разработка программы на языке С состоит из нескольких этапов.

На первом этапе разрабатывается алгоритм решения поставленной задачи в форме блок-схем и текстового описания.

На втором этапе определяется, какие данные нужны для работы программы, как они будут храниться и передаваться.

На третьем этапе шаги алгоритма разбираются на функции на языке С с описание их сигнатур в форме таблицы.

На заключительном этапе разрабатывается сценарий компиляции программы, а после демонстрируется работа программы.

2.1 Алгоритм работы

Клиентская часть

Работы программы начинается с точки входа.



Рисунок 2.1 - Блок-схема клиентской программы

Программа начинается с открытия сокета и дальнейшего подключения к серверу по указанному порту и IP-адресу, который в данной программе всегда

localhost. Далее пользователь должен ввести число, после чего оно отправляется на сервер. После чего, клиент получает ответ от сервера в виде суммы всех чисел, отправленных серверу с момента его запуска.

Серверная часть

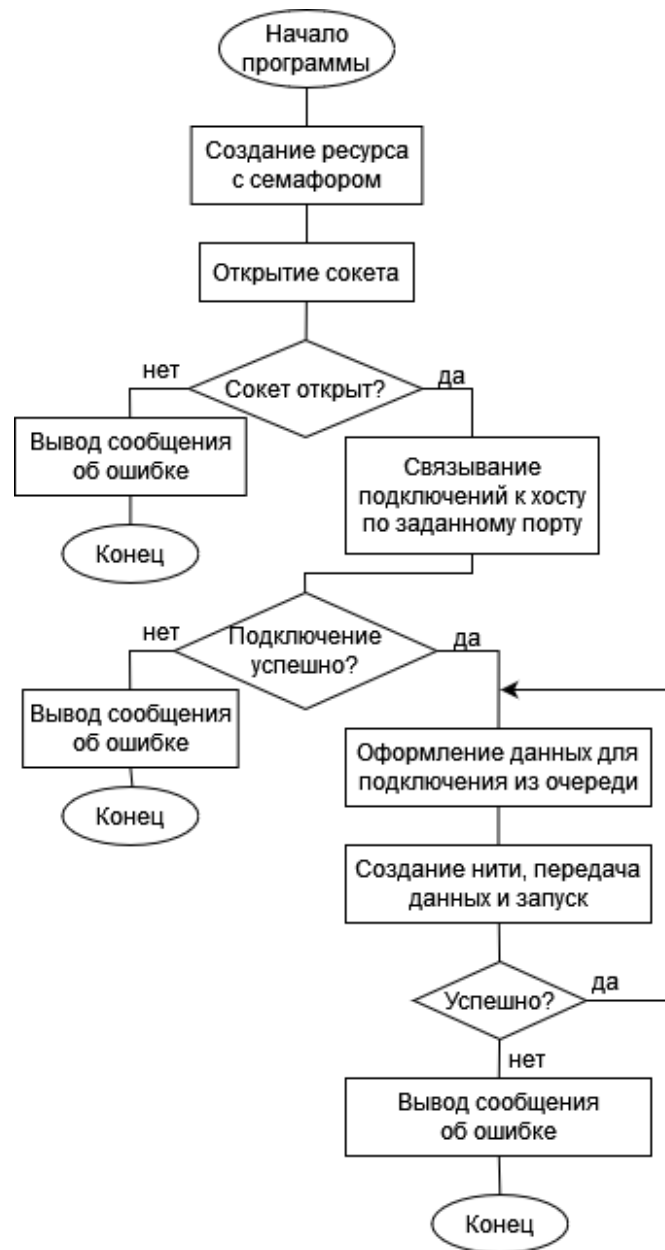


Рисунок 2.2 – Блок-схема сервера

Работа сервера начинается с создания ресурса – объекта структуры, включающей счётчик обращений к серверу, текущую сумму чисел и семафор, регулирующий работу с нитями.

Затем производится открытие сокета, а после он связывается с заданным портом и начинается его «прослушка».

Запускается цикл обработки подключений. Каждое подключение встает в очередь обработки запросов. Суть обработки запросов заключается в создании нити, чтобы переназначить задачу обработки запроса ей. Для работы каждой нити необходимы данные, этими данными являются объекты структуры, которая включает дескриптор сокета и указатель на ресурс – объект структуры со счётчиком, текущей суммы и семафором.^[3]

Нить на серверной части

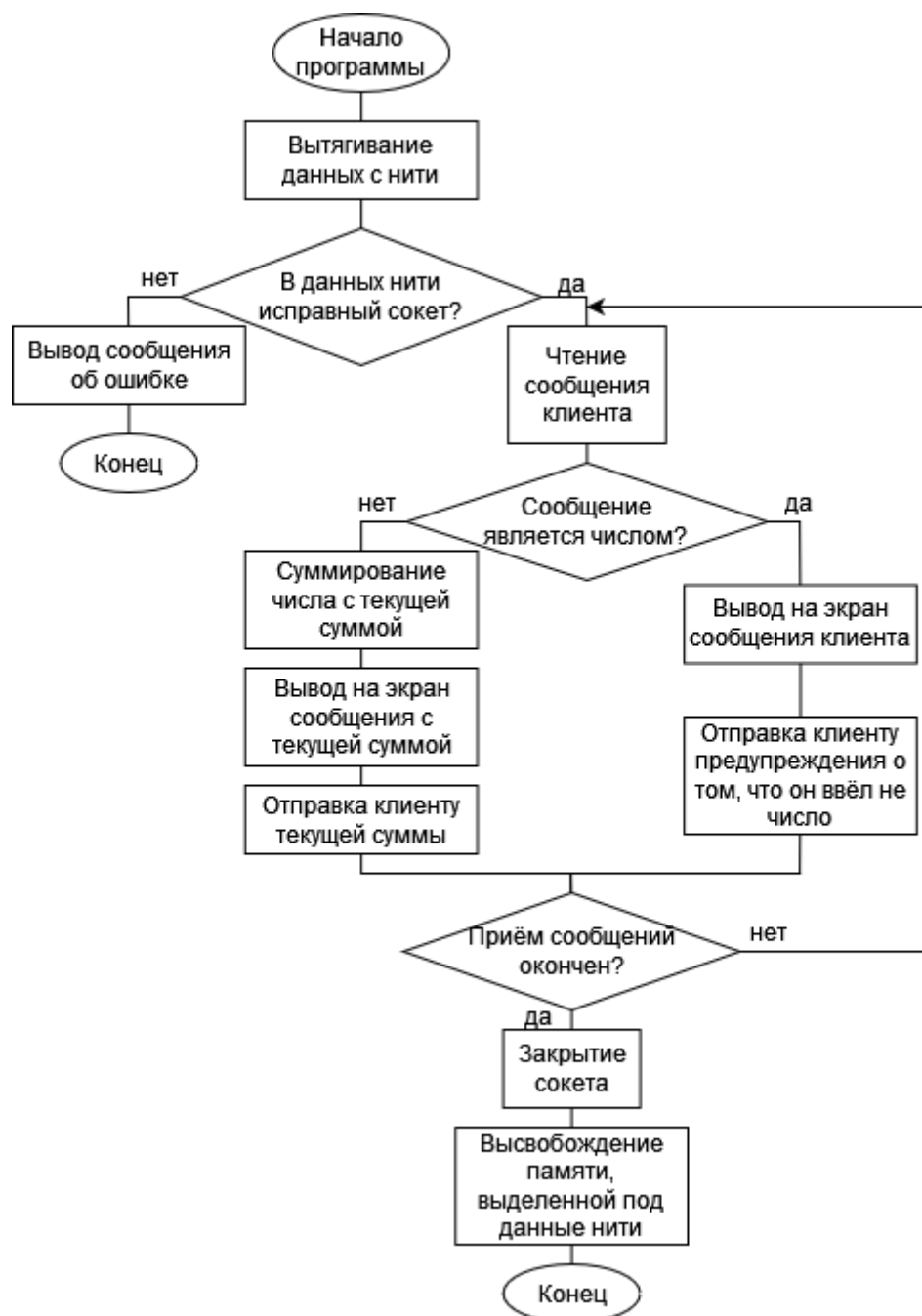


Рисунок 2.3 – Блок-схема нити сервера

Работа нити обработки подключения начинается с проверки дескриптора сокета на исправность – если он неисправен, программа завершает своё выполнение. Далее читается сообщение до тех пор, пока оно не будет прочитано полностью.

Прочитанное сообщение проверяется на то, является ли оно числом. Начало работы с защищённым ресурсом. Если переданное сообщение не является числом, то оно просто выводится на экран. Если же оно является

числом, то оно суммируется к текущей сумме. Клиенту отправляется текущая сумма переданных чисел.

Сокет закрывается и высвобождается память под данные нити. Работа нити завершена.

2.2 Данные

В ходе анализа работы было определено, что для решения задачи необходимо определить ряд структур. Каждая инкапсулирует в себе данные, которые нужны для решения одной подзадачи. Использование структур повышает удобство работы с данными и облегчает понимание исходного текста программы.

resources_t

Структура, которая хранит в себе:

- семафор *semaphore* для защищённого доступа к ресурсу;
- счётчик *current_index* для подсчёта подключений;
- текущая сумма чисел *sum_of_numbers* для хранения суммы переданных чисел.

В названиях пользовательских структур будет добавляться постфикс *_t*, поскольку они будут объявлены с помощью оператора *typedef*.^[4]

thread_data_t

Структура инкапсулирует информацию, необходимую для работы нити. К полям структур относятся:

- *socket* – дескриптор подключения;
- *resources* – указатель на защищённый ресурс.

Перечисление структур закончено.

2.3 Функции

Для решения задачи было определено 6 функций, каждая из которых инкапсулирует часть работы программы работы.

Таблица 1 – Сигнатуры функции, реализованных в программе

Программа	Название	Аргументы	Возвращаемое значение
Клиент	main		Целое число – код завершения работы функции
	client_run		Нет возвращаемого значения
Сервер	main		Целое число – код завершения работы функции
	server_run		Нет возвращаемого значения
	socket_thread	arg – данные нити типа thread_data_t	Предусмотрен указатель на void, но нет возвращаемого значения
	isNumber	input – строка output – указатель на int для записи туда числа, если строка таковой является	Целое число - код завершения работы функции
	numberToString	Number – число	Строка – строковое представление числа

Подробнее рассматривается каждая из приведенных функций.

client_run

Запуск работы клиента. Используется в клиентской программе. Работа метода описана в клиентской части предыдущего раздела.

server_run

Запуск работы сервера. Используется в серверной программе. Работа метода описана в серверной части предыдущего раздела.

socket_thread

Запуск работы нити сервераа. Используется в серверной программе. Работа метода описана в нити серверной части предыдущего раздела.

isNumber

Функция проверяет строку на то, что она является числом. Если это так, то из строки читается число и записывается в разыменованный указатель на число,

переданный вторым параметром. Функция возвращает 0, если строка не число, и 1 – если число.

numberToString

Функция возвращает строковое представление числа в виде символьного указателя.

2.4 Сценарий компиляции

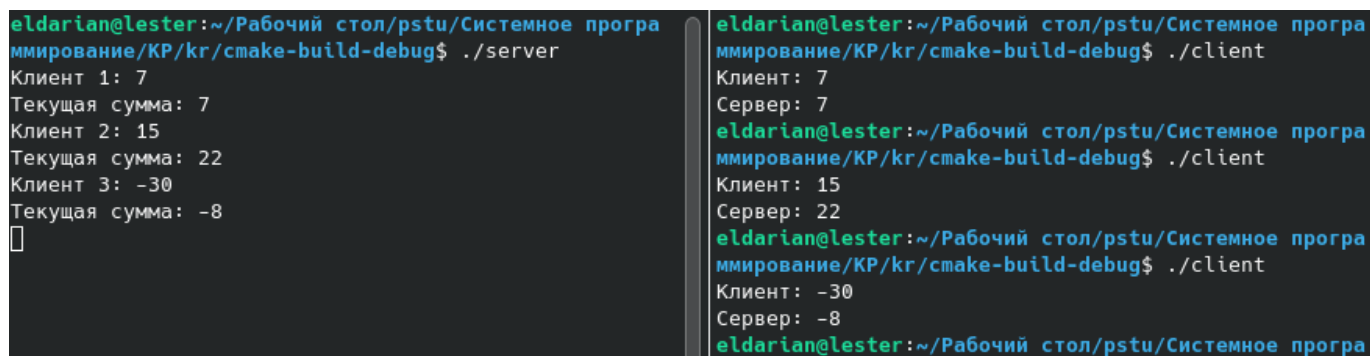
Проект курсовой работы состоит из 2 программ: сервера и клиента. Сборка выполняется с помощью системы сборки CMake. В приложении А представлен текст CMakeLists.txt. Сборка проекта выполняется командой:

```
cmake .
```

– из каталога проекта.

2.5 Демонстрация работы

Демонстрируется пример работы проекта в двух окнах терминала, в одном – сервер, во втором – клиент.



```
eldarian@lester:~/Рабочий стол/pstu/Системное програ  
мирование/КР/kr/cmake-build-debug$ ./server  
Клиент 1: 7  
Текущая сумма: 7  
Клиент 2: 15  
Текущая сумма: 22  
Клиент 3: -30  
Текущая сумма: -8  
█  
eldarian@lester:~/Рабочий стол/pstu/Системное програ  
мирование/КР/kr/cmake-build-debug$ ./client  
Клиент: 7  
Сервер: 7  
eldarian@lester:~/Рабочий стол/pstu/Системное програ  
мирование/КР/kr/cmake-build-debug$ ./client  
Клиент: 15  
Сервер: 22  
eldarian@lester:~/Рабочий стол/pstu/Системное програ  
мирование/КР/kr/cmake-build-debug$ ./client  
Клиент: -30  
Сервер: -8  
eldarian@lester:~/Рабочий стол/pstu/Системное програ
```

Рисунок 2.4 – Демонстрация работы сервера и клиента

Клиент и сервер запускаются без каких-либо параметров – они просто не требуются.

Сервер запускает цикл обработки запросов. Остановить его можно только нажатием комбинации клавиш «Ctrl + C».

Клиент с момента запуска начинает требовать ввести число. После ввода числа, клиент отправляет его на сервера, а от него получает текущую сумму всех чисел, которые сервер получал от начала своей работы.

Вывод, в разделе были рассмотрены алгоритмы клиента, сервера и нити сервера. Далее были рассмотрены данные, необходимые для работы сервера. А после, написаны функции и описано их предназначение. На завершительном этапе был рассмотрен сценарий компилирования и продемонстрирована работа серверной и клиентской программ.

ЗАКЛЮЧЕНИЕ

За время выполнения работы были изучены средства работы с протоколом TCP, нитями и семафорами в операционной системе Linux, отточены навыки взаимодействия с операционной системой, вынесены ценные уроки о возникающих ошибках и способах их избежать, получен незаменимый опыт поиска необходимой информации в печатных изданиях и в сети Интернет.

Для выполнения курсовой работы были использованы операционная система KDE Neon, среда разработки CLion, система сборки CMake, язык программирования C и его возможности в области межпроцессорного взаимодействия.

В ходе выполнения курсовой работы были выполнены все поставленные задачи: разработаны алгоритмы работы клиентской и серверной части, а также алгоритм формирования ответа на сервере, с использованием блок-схем.

Цель курсовой работы достигнута – в операционной системе Linux на языке C разработано приложение, реализующее общение между клиентом и сервером на основе TCP-подключений с выделением отдельной нити на каждого клиента. Пользователь клиента вводит число, а сервер возвращает ему сумму всех чисел, переданных ему от начала его работы.

СПИСОК ИСПОЛЬЗУЕМЫХ ИСТОЧНИКОВ

- 1) Лав Р. Linux. Системное программирование. СПб.: Питер, 2014. 448 с;
- 2) Технология "клиент-сервер" и мониторы транзакций [Электронный ресурс] URL: <https://www.osp.ru/os/1994/03/178494> (дата обращения: 10.12.21);
- 3) TCP и UDP сокет в CODESYS V3 [Электронный ресурс] URL: <http://support.fastwel.ru/AppNotes/AN/AN-0001.html#intro> (дата обращения: 20.12.21);
- 4) Ритчи Деннис М., Керниган Брайан У. Язык программирования С. Вильямс, 2017, 228 с.

ПРИЛОЖЕНИЕ А

Листинг файла CMakeLists.txt:

```
cmake_minimum_required(VERSION 3.20)
project(sysprog C)

set(CMAKE_C_STANDARD 11)
set(THREADS_PREFER_PTHREAD_FLAG ON)
find_package(Threads REQUIRED)

include_directories(.)

add_executable(server main_server.c server.h server.c consts.h)
add_executable(client main_client.c client.h client.c consts.h)

target_link_libraries(server PRIVATE Threads::Threads m)
target_link_libraries(client)
```

Листинг файла consts.h:

```
#pragma once

#define PORT 3456
```

ПРИЛОЖЕНИЕ Б

Листинг файла main_client.c:

```
#include <client.h>
```

```
int main() {  
    client_run();  
    return 0;  
}
```

Листинг файла client.h:

```
#pragma once
```

```
void client_run();
```

Листинг файла client.c:

```
#include "client.h"  
#include "consts.h"
```

```
#include <stdio.h>  
#include <unistd.h>  
#include <string.h>  
#include <sys/socket.h>  
#include <netinet/in.h>
```

```
void client_run() {  
    size_t buf_size = 1024;  
    char buf[1024];  
    char *buff = buf;  
    int sock;  
    struct sockaddr_in addr;  
  
    do {  
        sock = socket(AF_INET, SOCK_STREAM, 0);  
        if (sock < 0) usleep(100);  
    } while (sock < 0);  
  
    addr.sin_family = AF_INET;  
    addr.sin_port = htons(PORT);  
    addr.sin_addr.s_addr = htonl(INADDR_LOOPBACK);  
    int isConnect;  
    do {  
        isConnect = connect(sock, (struct sockaddr *) &addr, sizeof(addr));  
        if (isConnect < 0) usleep(100);  
    } while (isConnect < 0);  
  
    printf("Клиент: ");  
    scanf("%s", buff);  
    getchar();  
    send(sock, buf, strlen(buf), 0);  
    memset(buf, '\0', buf_size);  
    recv(sock, buf, sizeof(buf), 0);  
    printf("Сервер: %s\n", buf);  
    close(sock);  
}
```

ПРИЛОЖЕНИЕ В

Листинг файла main_server.c:

```
#include <server.h>
```

```
int main() {  
    server_run();  
    return 0;  
}
```

Листинг файла server.h:

```
#pragma once
```

```
#include <semaphore.h>
```

```
typedef struct {  
    sem_t semaphore;  
    int current_index, sum_of_numbers;  
} resources_t;
```

```
typedef struct {  
    int socket;  
    resources_t *resources;  
} thread_data_t;
```

```
void server_run();  
void *socket_thread(void *arg);
```

```
int isNumber(const char *input, int *output);  
char *numberToString(int number);
```

Листинг файла server.c:

```
#include "server.h"  
#include "consts.h"
```

```
#include <stdio.h>  
#include <stdlib.h>  
#include <string.h>  
#include <pthread.h>  
#include <sys/ipc.h>  
#include <sys/sem.h>  
#include <sys/types.h>  
#include <sys/socket.h>  
#include <netinet/in.h>
```

```
void server_run() {  
    int listener;  
    struct sockaddr_in addr;  
    resources_t resources = {{}, 0, 0};  
    sem_init(&resources.semaphore, 0, 1);  
  
    listener = socket(AF_INET, SOCK_STREAM, 0);  
    if (listener < 0) {  
        perror("socket");  
        exit(1);  
    }  
}
```

```

addr.sin_family = AF_INET;
addr.sin_port = htons(PORT);
addr.sin_addr.s_addr = htonl(INADDR_ANY);
if (bind(listener, (struct sockaddr *)&addr, sizeof(addr)) < 0) {
    perror("bind");
    exit(2);
}

listen(listener, 1);

while (1) {
    thread_data_t *data = malloc(sizeof(thread_data_t));
    data->socket = accept(listener, 0, 0);
    data->resources = &resources;
    pthread_t thread;
    if(pthread_create(&thread, NULL, &socket_thread, data)) {
        exit(-1);
    }
}

void *socket_thread(void *arg) {
    char buf[1024];
    int bytes_read;
    thread_data_t *data = (thread_data_t *)arg;

    if (data->socket < 0) {
        perror("accept");
        exit(3);
    }

    while (1) {
        memset(buf, '\0', 1024);
        bytes_read = recv(data->socket, buf, 1024, 0);
        if(bytes_read <= 0) break;
        else {
            sem_wait(&data->resources->semaphore);
            ++data->resources->current_index;
            int client_number;
            if (isNumber(buf, &client_number)) {
                char *number = numberToString(data->resources->sum_of_numbers += client_number);
                printf("Клиент %d: %s\nТекущая сумма: %d\n",
                    data->resources->current_index, buf, data->resources->sum_of_numbers);
                send(data->socket, number, strlen(number), 0);
                free(number);
            } else {
                const char *warn = "Некорректный ввод!";
                printf("Клиент %d ввёл строку: %s\n", data->resources->current_index, buf);
                send(data->socket, warn, strlen(warn), 0);
            }
            sem_post(&data->resources->semaphore);
        }
    }

    close(data->socket);
    free(arg);
    return 0;
}

int isNumber(const char *input, int *output) {

```

```

*output = 0;
int n = strlen(input), first = 0;
if(n && input[0] == '-') {
    first = 1;
}
if (n == first + 1 && input[first] == '0') {
    return 1;
} else if (n <= first || input[first] < '1' || input[first] > '9') {
    return 0;
} else {
    *output = input[first] - '0';
}
for (int i = first + 1; i < n; ++i) {
    if (input[i] < '0' || input[i] > '9') {
        return 0;
    } else {
        *output *= 10;
        *output += input[i] - '0';
    }
}
if (first) {
    *output = -*output;
}
return 1;
}

```

```

char *numberToString(int number) {
    char *numberArray;
    if (number == 0) {
        numberArray = malloc(2 * sizeof(char));
        numberArray[0] = '0';
        numberArray[1] = '\0';
        return numberArray;
    }
    int i, first;
    if (number < 0) {
        first = 1;
        number = -number;
    } else {
        first = 0;
    }
    int n = (int)log10((double)number) + first + 1;
    numberArray = calloc(n + first + 1, sizeof(char));
    for (i = n - 1; i >= first; --i, number /= 10) {
        numberArray[i] = (char)((number % 10) + '0');
    }
    if (first) {
        numberArray[0] = '-';
    }
    numberArray[n] = '\0';
    return numberArray;
}

```