Федеральное государственное автономное образовательное учреждение высшего образования «Санкт-Петербургский политехнический университет Петра Великого»

[Институт компьютерных наук и кибербезопасности](https://www.spbstu.ru/structure/institut_kompyuternykh_nauk_i_kiberbezopasnosti/)

Высшая школа технологий искусственного интеллекта

Направление 02.03.03 Математическое обеспечение и администрирование информационных систем

Отчет по предмету: «Технологии разработки программного обеспечения». Практическая работа:

«Разработка клиент-серверного приложения Калькулятор»

Выполнила: Мураев А.А. группа: 5130203/30002

Преподаватель: Силиненко А.В.

Санкт-Петербург, 2024

**Оглавление**

[Введение 2](#_heading=h.gjdgxs)

[Часть 1 4](#_heading=h.30j0zll)

[Часть 2 6](#_heading=h.1fob9te)

[Часть 3 8](#_heading=h.3znysh7)

[Часть 4 9](#_heading=h.2et92p0)

[Вывод 11](#_heading=h.tyjcwt)

[Список использованных источников 12](#_heading=h.3dy6vkm)

[Приложение №1 13](#_heading=h.1t3h5sf)

[Приложение №2 22](#_heading=h.4d34og8)

# Введение

**Актуальность работы.** В процессе выполнения данной работы, исполнителю будет необходимо изучить относительно большой объем информации, в результате чего студент научиться следующим навыкам:

* Работа с сетевыми протоколами TCP и UDP.
* Работа с командной строкой OC Linux (Ubuntu).
* Углубление навыков программирования на языке C/C++.

Работа посвящена созданию клиент-серверного сетевого приложения.

**Цель работ**ы. Разработка клиент-серверного сетевого приложения на языке С++ в OC Linux.

**Задачи:**

* Получение базовых знаний по разработке клиент-серверных

приложений в ОС Linux;

* Разработка клиент-серверного приложения-калькулятора.

Для ведения проекта был заведен репозиторий [GitHub [1]](https://github.com/SashaNeist/Net_Calculator):

# Часть 1

1. Общее описание задачи.

Реализовать клиент-серверное сетевое приложение - Калькулятор. Пользователь может выбрать протокол взаимодействия (TCP либо UDP). Сервер поддерживает оба протокола и может быть запущен отдаленно (в одной локальной сети), но пользователю необходимо знать IP-адрес сервера для подключения.

2. Описание интерфейса

2.1 Входные данные. Программа от клиента принимает строку, проверяет формат и передает ее серверу для дальнейшей обработки.

2.2 Выходные данные. Сервер передает клиенту строку, в которой содержится результат вычислений. Клиент выводит ее на экран.

2.3 Хранимые данные. В данном приложении хранение данных не предусмотрено.

3. Функциональные требования.

3.1 Ввод данных. Клиент вводит данные в формате без пробелов в одну строку, либо стоп-слово: “exit”.

3.2 Проверка корректности данных. Проверка корректности формата введенных данных осуществляется с помощью трех регулярных выражений (для 2 чисел, для 3 чисел без скобок, для 3 чисел со скобками), в случае неверного формата введенных данных выводится сообщение о несоответствии формата. На сервере осуществляется проверка деления на 0. Так же в программе присутствуют различные уровни обработки ошибок и исключений.

3.3 Содержательная часть. Сервер получает от клиента строковое сообщение, сравнивает формат с тремя регулярными выражениями и вызывает соответствующий блок обработки. Строка разбивается на числа и операторы. Если в выражении 2 числа - происходит прямое вычисление через конструкцию switch - case (от оператора). Если в выражении 3 числа со скобками, так же используется конструкция switch - case (порядок учитывается в регулярном выражении). В случае 3 чисел без скобок используется конструкция if - else с использованием тернарного оператора ?:. Результат записывается в строку и отправляется клиенту.

3.4 Вывод результатов и сообщения об ошибке. Сервер отправляет клиенту результат в виде строки и он выводится клиенту на консоль. На каждом этапе у сервера и клиента осуществляется обработка ошибок. Например: при создании сокета, при подключении клиента, при отправке данных и т.д.. Каждой ошибке соответствует вывод информации о ней.

4. Требования к тестированию.

4.1 Проверка приветственного сообщения.

4.2 Проверка, допускает ли программа к вводу целые и дробные числа и числовые выражения со скобками.

4.3 Проверка на корректность вывода ошибки некорректного формата ввода.

4.4 Проверка вывода корректного результата числового выражения.

4.5 Проверка на корректность завершения работы клиента и сервера.

# Часть 2

Рассмотрим более детально осуществление протокола взаимодействия UDP [1]. При запуске клиентской части пользователь указывает через какой протокол хочет осуществлять передачу информации. tcp протокол TCP, udp протокол UDP. Запуск программы может выглядеть так:  
Запуск сервера: *./server*Запуск клиента: *./client 127.0.0.1 udp.*

Рис. 2.1 соответствует фрагменту программы серверной клиентской части (Приложение №1, Приложение №2). В данном соединении осуществляется отправка сообщения с клиента на сервер. В случае успешного получения сервером сообщения, он должен отправить подтверждающее сообщение клиенту, который в свою очередь отправляет подтверждение о получении результата позже. Если же подтверждающее сообщение клиентом не получено, то клиент отправляет серверу сообщение еще два раза. Если не было получено ответа, клиент завершает свою работу.

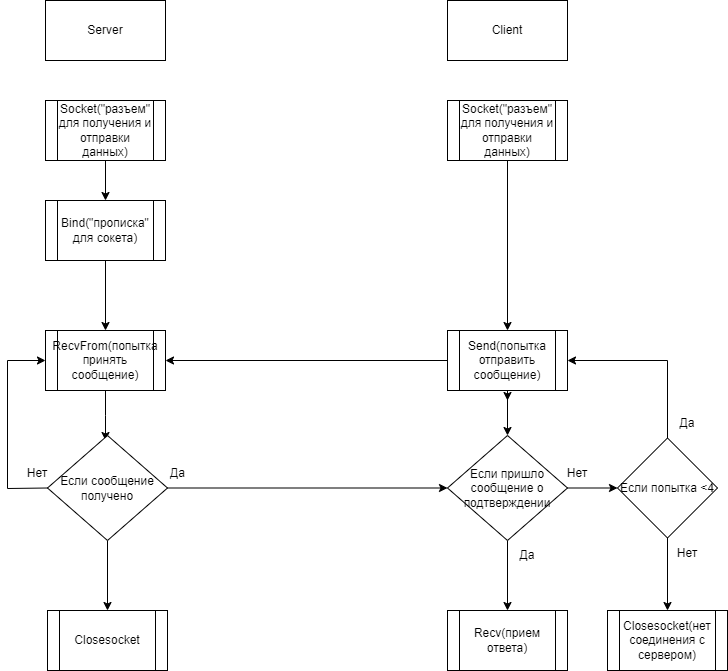


Рис. 2.1. Схема взаимодействия протокола UDP

# Часть 3

Была написан expect-сценарий для тестирования написанного консольного приложения. Код тестирующей программы находится в Приложении №3, Приложение №4 и №5 содержат вывод тестового сценария. В массив записываем все тестовые примеры и ожидаемый ответ. Сценарий запускает клиент и имитирует ввод ряда примеров. Тестируется как TCP, так и UDP соединение.

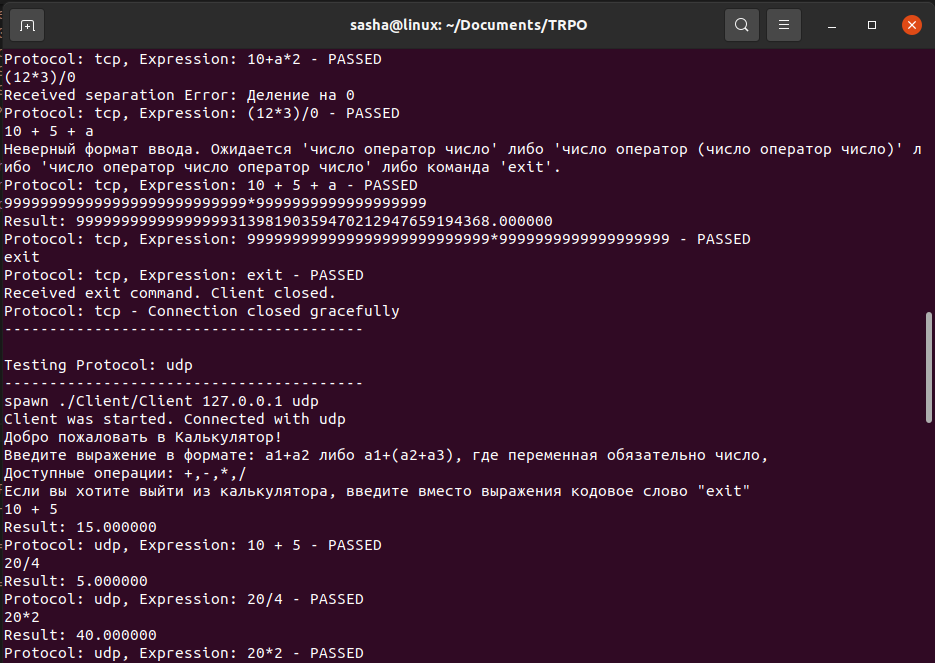


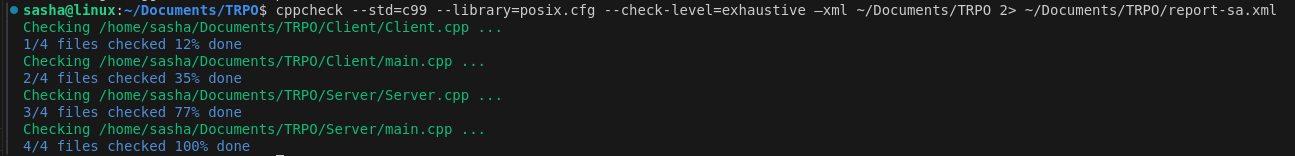
Рис. 3.1. Фрагмент результатов тестирования

На рисунке 3.1 приведены результаты тестирования калькулятора с помощью expect-сценария. Результаты тестирования положительные, что говорит о корректности программы.

# Часть 4

Следующий этап тестирования приложения – статический анализатор [2]. Для этого следовало установить анализатор и проверить версию. Далее с помощью команды 4.1 выполнен статический анализ программы.

(4.1)



Поясним обозначения в команде 4.1:

* *~/Documents/TRPO* – корневой каталог проекта.
* *~/Documents/TRPO/report-sa.xml* - файл, который записывает результаты анализа.

<?xml version="1.0" encoding="UTF-8"?>

<results version="2">

<cppcheck version="2.16.0"/>

<errors>

</errors>

</results>

Рис. 4.1. Результаты статического анализа программы.

Статистический анализатор не обнаружил ошибок в программе, результаты представлены на рисунке 4.1.

Далее нужно перевести результаты в формат HTML. Для этого использовалась предполагалось использовать утилиту *cppcheck-htmlreport*. Однако, возникли проблемы с корректной установкой cppcheck версии >=2.11, в связи с чем пришлось написать файл html самостоятельно. В результате имеем файл с расширением .html. Рисунок 4.2.



Рис. 4.2. Результаты тестирования в формате HTML.

# Заключение

В ходе выполнения работы были получены следующие навыки:

* Работа с командной строкой OC Linux.
* Работа с GitHub
* Построение алгоритмов на языке C/C++.
* Получение общего представления о работе протоколов сетевых взаимодействий TCP и UDP.
* Получение представления об клиент-серверной архитектуре.
* Ознакомление с разработкой bash-скриптов и expect-сценариев для тестирования.
* Ознакомление с утилитой cppcheck, для статистического анализа.

Были выполнены следующие задачи:

1. Получение базовых знаний по разработке клиент-серверных приложений в ОС Linux
2. Разработка клиент-серверного приложения - калькулятор.

В итоге мы получили полностью рабочее приложение-калькулятор, состоящее из 2 частей: сервер и клиент. Сервер и клиент могут взаимодействовать в двух режимах, по протоколу TCP или UDP. Сервер работает сразу в двух режимах. Клиент задает режим при запуске. Клиент выключается с помощью стоп-слова *“exit”.* Сервер завершает работу только принудительно.

# Список использованных источников

1. Собственный репозиторий GitHub [Электронный ресурс] / SashaNeist  — URL:  <https://github.com/SashaNeist/Net_Calculator> (дата обращения: 10.01.2025).
2. Реализация UDP-сервера и клиента на C [Электронный ресурс] // GeeksforGeeks. — URL: <https://www.geeksforgeeks.org/udp-server-client-implementation-c/> (дата обращения: 07.01.2025).
3. *Силиненко, А. В. Технология разработки программного обеспечения* [Электронный ресурс] : Лаб. 2 / А.В. Силиненко ; Министерство науки и высшего образования Российской Федерации. - [б. м.], [б. г.] - URL: <https://dl.spbstu.ru/course/view.php?id=6217> (дата обращения: 09.01.2025).
4. *Королев, А.* Протокол UDP: основы, преимущества и недостатки [Электронный ресурс] / А. Королев // Хабр. – 2017. – 17 сент. – URL: <https://habr.com/ru/articles/357566/> (дата обращения: 02.10.2024).

# Приложение №1

Серверная часть(main.cpp), с полным кодом можно ознакомиться на [GitHub[1]](https://github.com/SashaNeist/Net_Calculator):

#include "Server.h"

**int** **main**()

{

setlocale(LC\_ALL, "rus");

std::cout << "Server was started**\n**";

**int** tcpSock, udpSock, port = **5175**; // Объявляем переменные для сокета и порта сервера.

**socklen\_t** cliLength; // Объявляем переменную для хранения размера структуры адреса клиента.

**struct** sockaddr\_in servAddres, cliAddres; // Объявляем структуры для адресов сервера и клиента.

**pthread\_t** thread\_id; // Объявляем переменную для идентификатора потока.

// Создаем сокет

tcpSock = socket(AF\_INET, SOCK\_STREAM, **0**); // Создаем сокет для TCP-соединения.

**if** (tcpSock < **0**) // Проверяем, что сокет был создан успешно

{

std::cout << " Error: opening socket **\n**"; // Выводим сообщение об ошибке создания сокета.

exit(**1**);

}

udpSock = socket(AF\_INET, SOCK\_DGRAM, **0**);

**if** (udpSock < **0**)

{

std::cerr << "Error: opening UDP socket**\n**";

close(tcpSock);

**return** **1**;

}

// Устанавливаем опцию SO\_REUSEADDR, которая позволяет переиспользовать адрес

**int** reuse = **1**; // Значение опции (1 - включить)

**if** (setsockopt(tcpSock, SOL\_SOCKET, SO\_REUSEADDR, &reuse, **sizeof**(reuse)) < **0**)

{

std::cerr << "Error setting SO\_REUSEADDR option**\n**"; // Выводим сообщение об ошибке установки опции.

close(tcpSock); // Закрываем сокет.

close(udpSock);

**return** **1**;

}

**if** (setsockopt(udpSock, SOL\_SOCKET, SO\_REUSEADDR, &reuse, **sizeof**(reuse)) < **0**)

{

std::cerr << "Error setting SO\_REUSEADDR option**\n**"; // Выводим сообщение об ошибке установки опции.

close(tcpSock); // Закрываем сокет.

close(udpSock);

**return** **1**;

}

memset(&servAddres, **0**, **sizeof**(servAddres)); // Заполняем структуру адреса сервера нулями.

servAddres.sin\_family = AF\_INET; // Указываем семейство адресов (IPv4).

servAddres.sin\_addr.s\_addr = INADDR\_ANY; // Указываем, что сервер принимает соединения на всех интерфейсах.

servAddres.sin\_port = htons(port); // Устанавливаем порт сервера, преобразуя его из host byte order в network byte order.

// Связываем TCP сокет с адресом

**if** (bind(tcpSock, (**struct** sockaddr \*)&servAddres, **sizeof**(servAddres)) < **0**)

{ // Проверяем, удалось ли связать сокет с адресом.

std::cout << " Error: building **\n**"; // Выводим сообщение об ошибке при связывании сокета.

exit(**1**);

}

// Связываем UDP сокет с адресом

**if** (bind(udpSock, (**struct** sockaddr \*)&servAddres, **sizeof**(servAddres)) < **0**)

{

std::cerr << "Error: binding UDP socket**\n**";

close(tcpSock);

close(udpSock);

**return** **1**;

}

// Переводим сокет в режим прослушивания, разрешаем до 5 ожидающих соединений.

**if** (listen(tcpSock, **5**) < **0**)

{

std::cerr << "Error: listening on TCP socket**\n**";

close(tcpSock);

close(udpSock);

**return** **1**;

}

// Переводим UDP сокет в неблокирующий режим

**int** flagsUDP = fcntl(udpSock, F\_GETFL, **0**);

**if** (flagsUDP == -**1**)

{

perror("fcntl F\_GETFL failed");

close(tcpSock);

close(udpSock);

**return** **1**;

}

**if** (fcntl(udpSock, F\_SETFL, flagsUDP | O\_NONBLOCK) == -**1**)

{

perror("fcntl F\_SETFL failed");

close(tcpSock);

close(udpSock);

**return** **1**;

}

cliLength = **sizeof**(cliAddres); // Получаем размер структуры адреса клиента.

fd\_set read\_fds; // Создаем множество дескрипторов для чтения

// Главный цикл приема соединений

**while** (true) // Бесконечный цикл для приема новых соединений.

{

FD\_ZERO(&read\_fds); // Очищаем множество дескрипторов

FD\_SET(tcpSock, &read\_fds); // Добавляем TCP сокет в множество для чтения

FD\_SET(udpSock, &read\_fds); // Добавляем UDP сокет в множество для чтения

// Определяем максимальный файловый дескриптор, который будет использован в select

// Прибавляем 1, т.к. select использует значения меньше чем max\_fd

**int** max\_fd = std::max(tcpSock, udpSock) + **1**;

// Вызываем select, чтобы дождаться готовности хотя бы одного из сокетов к чтению

// Первый аргумент - максимальный дескриптор + 1

// Второй аргумент - указатель на множество дескрипторов для чтения

// Третий и четвертый аргументы - для записи и исключений (здесь nullptr, не используются)

// Пятый аргумент - тайм-аут (здесь nullptr, т.е. select будет ждать бесконечно)

**if** (select(max\_fd, &read\_fds, nullptr, nullptr, nullptr) < **0**)

{

// Если select возвращает -1, значит произошла ошибка

perror("select failed"); // Выводим сообщение об ошибке

**continue**; // Переходим к следующей итерации цикла

}

// TCP

**if** (FD\_ISSET(tcpSock, &read\_fds))

{

**int** \*newSock\_ptr = **new** **int**; // Выделяем динамически память для хранения сокета клиента.

\*newSock\_ptr = accept(tcpSock, (**struct** sockaddr \*)&cliAddres, &cliLength); // Принимаем новое соединение от клиента.

// Проверяем, удалось ли принять соединение

**if** (\*newSock\_ptr < **0**)

{

std::cout << "Error: accept **\n**"; // Выводим сообщение об ошибке при принятии соединения.

**delete** newSock\_ptr; // Освобождаем память в случае ошибки.

**continue**; // Продолжаем слушать новые соединения.

}

// Создаем поток для обработки клиента

**if** (pthread\_create(&thread\_id, NULL, handle\_client, (**void** \*)newSock\_ptr) != **0**)

{ // Проверяем, удалось ли создать поток.

std::cerr << "Error creating thread" << std::endl; // Выводим сообщение об ошибке при создании потока.

close(\*newSock\_ptr); // Закрываем сокет клиента.

**delete** newSock\_ptr; // Освобождаем память выделенную под сокет.

**continue**; // Продолжаем слушать новые соединения.

}

pthread\_detach(thread\_id); // Освобождаем ресурсы потока после завершения его работы.

}

// UDP

**else** **if** (FD\_ISSET(udpSock, &read\_fds))

{

// Если в наборе дескрипторов read\_fds есть udp сокет, то есть пришли данные по udp

**struct** sockaddr\_in clientAddr;

**socklen\_t** clientAddrLen = **sizeof**(clientAddr);

// Пытаемся получить UDP сообщение и его тип от клиента

std::pair<MessageType, std::string> received = receiveUdpMessage(udpSock, clientAddr, clientAddrLen);

// Если тип сообщения REQUEST

**if** (received.first == MessageType::REQUEST)

{

// Отправляем клиенту подтверждение (ACK) о получении запроса

**if** (sendUdpMessage(udpSock, "", MessageType::ACK, clientAddr, clientAddrLen) < **0**)

{

std::cerr << "Error send UDP ACK**\n**";

**continue**; // Если ошибка отправки ACK, переходим к следующей итерации ожидания сообщений

}

// Обрабатываем полученные данные и получаем результат

std::string result = function(received.second);

// Отправляем клиенту результат обработки

**if** (sendUdpMessage(udpSock, result, MessageType::RESULT, clientAddr, clientAddrLen) < **0**)

{

std::cerr << "Error send UDP RESULT**\n**";

**continue**; // Если ошибка отправки RESULT, переходим к следующей итерации ожидания сообщений

}

// Ожидание ACK от клиента на отправленный результат

**bool** askReceived = false;

// Фиксируем время начала ожидания

**auto** startTime = std::chrono::steady\_clock::now();

**while** (true)

{

// Получаем текущее время

**auto** currentTime = std::chrono::steady\_clock::now();

// Вычисляем длительность ожидания в секундах

**auto** duration = std::chrono::duration\_cast<std::chrono::seconds>(currentTime - startTime).count();

// Проверяем, истек ли таймаут ожидания ACK (3 секунды)

**if** (duration >= **3**)

{

std::cout << "Timeout waiting for Result UDP ACK**\n**";

**break**; // Если таймаут истек, выходим из цикла ожидания ACK

}

// Пытаемся получить UDP сообщение, ожидая ACK от клиента на результат

std::pair<MessageType, std::string> receivedAsk = receiveUdpMessage(udpSock, clientAddr, clientAddrLen);

// Если полученное сообщение типа ACK

**if** (receivedAsk.first == MessageType::ACK)

{

askReceived = true;

**break**; // Выходим из цикла ожидания ACK, так как он получен

}

// Пауза на 10 миллисекунд

std::this\_thread::sleep\_for(std::chrono::milliseconds(**10**));

}

// Если ACK не был получен в течение таймаута

**if** (!askReceived)

std::cout << "Потеряна связь с UDP клиентом**\n**";

// Вывод сообщения об ошибке, если получен неизвестный тип сообщения

**else** **if** (received.first != MessageType::ACK && !received.second.empty())

std::cerr << "Unknow message type received: " << **static\_cast**<**int**>(received.first) << std::endl;

}

}

}

close(tcpSock); // Закрываем TCP сокет

close(udpSock); // Закрываем UDP сокет

std::cout << "Server was killed**\n**";

**return** **0**;

}

# Приложение №2

Клиентская часть(main.cpp), с полным кодом можно ознакомиться на [GitHub[1]](https://github.com/SashaNeist/Net_Calculator):

#include "Client.h"

**int** **main**(**int** argc, **char** \*argv[])

{

setlocale(LC\_ALL, "rus");

// Проверяем, передали ли IP-адрес в качестве аргумента командной строки

**if** (argc != **3**)

{

std::cerr << "Usage: client <IP\_address> <tcp/udp>**\n**";

**return** **1**; // Возвращаем код ошибки, если аргумент не передан

}

**const** **char** \*ipAddr = argv[**1**]; // Получаем IP-адрес из аргумента командной строки

**const** **char** \*netProt = argv[**2**]; // Получаем выбранный пользователем сетевой протокол

**int** sock, port = **5175**; // Определяем переменные для сокета и порта сервера

**struct** sockaddr\_in servAddres; // Определяем структуру для адреса сервера

std::string buffer; // Буфер для хранения сообщений

// Проверяем корректность формата IP-адреса

**if** (inet\_pton(AF\_INET, ipAddr, &servAddres.sin\_addr) <= **0**)

{

// Обработка ошибок преобразования IP-адреса

**if** (errno == EAFNOSUPPORT)

{

std::cerr << "Error: Invalid IP address format**\n**";

}

**else**

{

std::cerr << "Error: inet\_pton failed **\n**";

}

**return** **1**;

}

std::cout << "Client was started. Connected with " << netProt << std::endl;

**if** (!std::strcmp(netProt, "tcp")) // Проверяем выбранный протокол передачи данных

{ // Соединение с помощью TCP

// Создаем сокет

sock = socket(AF\_INET, SOCK\_STREAM, **0**);

**if** (sock < **0**)

{

std::cerr << "Error: opening socket **\n**";

**return** **1**;

}

// Заполняем структуру адреса сервера нулями

memset(&servAddres, **0**, **sizeof**(servAddres));

servAddres.sin\_family = AF\_INET; // Используем IPv4

// Преобразуем IP-адрес из текстового формата в бинарный и сохраняем в структуру

**if** (inet\_pton(AF\_INET, ipAddr, &servAddres.sin\_addr) <= **0**)

{

std::cerr << "Error: pton **\n**";

close(sock); // Закрываем сокет при ошибке

**return** **1**;

}

servAddres.sin\_port = htons(port); // Устанавливаем порт сервера, преобразуя его из host byte order в network byte order

// Пытаемся установить соединение с сервером

**if** (connect(sock, (**struct** sockaddr \*)&servAddres, **sizeof**(servAddres)) < **0**)

{

std::cerr << "Error: connecting **\n**";

close(sock); // Закрываем сокет при ошибке

**return** **1**;

}

// Выводим приветственное сообщение и инструкцию по использованию калькулятора

hello();

// Основной цикл работы клиента

**while** (true)

{

// Вызываем функцию function для получения выражения от пользователя

**if** (function(buffer))

{

// Если пользователь ввел "exit", выходим из цикла

**if** (buffer == "exit")

**break**;

// Отправляем выражение на сервер

send(sock, buffer.c\_str(), buffer.size(), **0**);

// Получаем ответ от сервера

**char** temp\_buffer[**256**];

**ssize\_t** bytes\_received = recv(sock, temp\_buffer, **255**, **0**);

**if** (bytes\_received > **0**)

{

buffer.assign(temp\_buffer, bytes\_received);

**if** (buffer == "Division by 0")

std::cerr << "Received separation Error: Деление на 0**\n**";

**else**

std::cout << "Result: " << buffer << std::endl; // Выводим результат вычисления

}

**else** **if** (bytes\_received == **0**)

{

std::cout << "Server closed the connection." << std::endl; // Сообщение об отключении сервера

**break**;

}

**else**

{

std::cerr << "Error receiving data." << std::endl; // Выводим ошибку, если не удалось получить ответ от сервера

**break**;

}

}

**else**

**continue**;

}

close(sock); // Закрываем сокет при завершении работы клиента

}

**else** // Соединение с помощью UDP

{

// Создаем UDP сокет

sock = socket(AF\_INET, SOCK\_DGRAM, **0**);

**if** (sock < **0**)

{

std::cerr << "Error: opening UDP socket" << std::endl;

**return** **1**;

}

// Заполняем структуру адреса сервера нулями

memset(&servAddres, **0**, **sizeof**(servAddres));

servAddres.sin\_family = AF\_INET; // Используем IPv4

// Преобразуем IP-адрес из текстового формата в бинарный и сохраняем в структуру

**if** (inet\_pton(AF\_INET, ipAddr, &servAddres.sin\_addr) <= **0**)

{

std::cerr << "Error: pton **\n**";

close(sock); // Закрываем сокет при ошибке

**return** **1**;

}

servAddres.sin\_port = htons(port); // Устанавливаем порт сервера, преобразуя его из host byte order в network byte order

// Переводим сокет в неблокирующий режим

**int** flags = fcntl(sock, F\_GETFL, **0**);

**if** (flags == -**1**)

{

perror("fcntl F\_GETFL failed");

close(sock);

**return** **1**;

}

**if** (fcntl(sock, F\_SETFL, flags | O\_NONBLOCK) == -**1**)

{

perror("fcntl F\_SETFL failed");

close(sock);

**return** **1**;

}

hello(); // Выводим приветственное сообщение и инструкцию по использованию калькулятора

**socklen\_t** servAddresLen = **sizeof**(servAddres);

// Основной цикл работы клиента

**while** (true)

{

// Вызываем функцию function для получения выражения от пользователя

**if** (function(buffer))

{

// Проверяем, если пользователь ввел команду "exit"

**if** (buffer == "exit")

**break**; // Завершаем цикл, если введена команда "exit"

**int** retryCount = **0**; // Счетчик повторных отправок сообщения

**bool** ackReceived = false; // Флаг, показывающий, был ли получен ACK (подтверждение)

// Цикл повторных отправок, пока не будет получено подтверждение (ACK)

// или не будет достигнуто максимальное количество попыток

**while** (retryCount < **3** && !ackReceived)

{

// Отправляем выражение на сервер

**ssize\_t** bytesSend = sendMessage(sock, buffer, MessageType::REQUEST, servAddres);

// Проверяем, успешно ли отправлено сообщение

**if** (bytesSend < **0**)

{

std::cerr << "Error: sendto failed**\n**";

close(sock);

**return** **1**; // Возвращаем код ошибки, если отправка не удалась

}

// Ждем ACK (подтверждение) от сервера

**auto** startTime = std::chrono::steady\_clock::now(); // Запоминаем время начала ожидания

// Цикл ожидания ACK, пока не истечет таймаут или не будет получен ACK

**while** (true)

{

// Получаем текущее время

**auto** currentTime = std::chrono::steady\_clock::now();

// Вычисляем длительность ожидания

**auto** duration = std::chrono::duration\_cast<std::chrono::seconds>(currentTime - startTime).count();

// Проверяем, истек ли таймаут (3 секунды)

**if** (duration >= **3**)

{

std::cout << "Timeout waiting for ACK**\n**";

**break**; // Выход из цикла ожидания ACK (по таймауту)

}

// Пытаемся получить сообщение (ACK или другой ответ) от сервера

std::pair<MessageType, std::string> received = receiveMessage(sock, servAddres, servAddresLen);

// Проверяем, получили ли мы ACK

**if** (received.first == MessageType::ACK)

{

ackReceived = true; // Устанавливаем флаг, что ACK получен

**break**; // Выход из цикла ожидания ACK (ACK получен)

}

// Пауза на 10 миллисекунд перед следующей попыткой чтения

std::this\_thread::sleep\_for(std::chrono::milliseconds(**10**));

}

// Если ACK не был получен

**if** (!ackReceived)

retryCount++; // Увеличиваем счетчик повторных отправок

}

// Проверяем, был ли получен ACK после всех попыток

**if** (!ackReceived)

{

std::cout << "Connecting lost**\n**";

**break**; // Выход из основного цикла (потеря связи)

}

// Если ACK получен, то продолжаем работу

**else**

{

// Получаем ответ с сервера

std::pair<MessageType, std::string> receivedResult;

// Цикл ожидания сообщения с результатом

**while** (true)

{

// Получаем сообщение от сервера

receivedResult = receiveMessage(sock, servAddres, servAddresLen);

// Проверяем, является ли сообщение результатом вычисления

**if** (receivedResult.first == MessageType::RESULT)

**break**; // Выход из цикла ожидания (результат получен)

// Пауза перед следующей попыткой чтения

std::this\_thread::sleep\_for(std::chrono::milliseconds(**10**));

}

// Обработка ошибки: Деление на 0

**if** (receivedResult.second == "Division by 0")

std::cerr << "Received separation Error: Деление на 0**\n**";

// Выводим результат

**else**

std::cout << "Result: " << receivedResult.second << std::endl;

// Отправляем ACK на результат

**if** (sendMessage(sock, "", MessageType::ACK, servAddres) < **0**)

std::cerr << "Error sending result ask**\n**";

}

}

}

close(sock); // Закрываем сокет при завершении работы клиента

}

**return** **0**;

}

# Приложение №3

#!/usr/bin/expect

**set** timeout **10**

**set** protocols **{**tcp udp**}**

puts "========================================"

puts " STARTING TESTS "

puts "========================================"

**foreach** protocol $protocols **{**

puts "\nTesting Protocol: $protocol"

puts "----------------------------------------"

spawn ./Client/Client **127.0**.0.1 $protocol

match\_max **100000**

expect -exact "Client was started. Connected with $protocol\r

Добро пожаловать в Калькулятор!\r

Введите выражение в формате: a1+a2 либо a1+(a2+a3), где переменная обязательно число,\r

Доступные операции: +,-,\*,/ \r

Если вы хотите выйти из калькулятора, введите вместо выражения кодовое слово \"exit\" \r

"

**set** expressions **{**

"10 + 5" "10 + 5\r

Result: 15.000000\r"

"20/4" "20/4\r

Result: 5.000000\r"

"20\*2" "20\*2\r

Result: 40.000000\r"

"2 + (3+4)" "2 + (3+4)\r

Result: 9.000000\r"

"10 - (5-2)" "10 - (5-2)\r

Result: 7.000000\r"

"12\*(3/2)" "12\*(3/2)\r

Result: 18.000000\r"

"(2+3)+4" "(2+3)+4\r

Result: 9.000000\r"

"(10-5.2)-2" "(10-5.2)-2\r

Result: 2.800000\r"

"10+5\*2" "10+5\*2\r

Result: 20.000000\r"

"20/4-1" "20/4-1\r

Result: 4.000000\r"

"2+3+4" "2+3+4\r

Result: 9.000000\r"

"10-5-2" "10-5-2\r

Result: 3.000000\r"

"12\*3/2" "12\*3/2\r

Result: 18.000000\r"

"100/5\*2" "100/5\*2\r

Result: 40.000000\r"

"1.5+2.5\*2" "1.5+2.5\*2\r

Result: 6.500000\r"

"10/3\*3" "10/3\*3\r

Result: 10.000000\r"

"1.5+2.5\*2" "1.5+2.5\*2\r

Result: 6.500000\r"

"9999999999+1\*2" "9999999999+1\*2\r

Result: 10000000001.000000\r"

"10+a\*2" "10+a\*2\r

Неверный формат ввода. Ожидается 'число оператор число' либо 'число оператор (число оператор число)' либо 'число оператор число оператор число' либо команда 'exit'.\r"

"(12\*3)/0" "(12\*3)/0\r

Received separation Error: Деление на 0\r"

"10 + 5 + a" "10 + 5 + a\r

Неверный формат ввода. Ожидается 'число оператор число' либо 'число оператор (число оператор число)' либо 'число оператор число оператор число' либо команда 'exit'.\r"

"999999999999999999999999999\*9999999999999999999" "999999999999999999999999999\*9999999999999999999\r

Result: 9999999999999999931398190359470212947659194368.000000\r"

"exit" "exit\r"

**}**

**foreach** **{expr** expected\_output**}** $expressions **{**

send -- "$expr\r"

expect **{**

-exact "$expected\_output" **{**

puts "Protocol: $protocol, Expression: $expr - PASSED"

**}**

timeout **{**

puts "Protocol: $protocol, Expression: $expr - TIMEOUT"

**}**

eof **{**

puts "Protocol: $protocol, Connection closed unexpectedly"

exit

**}**

**}**

**}**

expect eof

puts "Protocol: $protocol - Connection closed gracefully"

puts "----------------------------------------"

**}**

puts "========================================"

puts " TESTS FINISHED "

puts "========================================"

# Приложение №4

========================================

STARTING TESTS

========================================

Testing Protocol: tcp

----------------------------------------

spawn ./Client/Client 127.0.0.1 tcp

Client was started. Connected with tcp

Добро пожаловать в Калькулятор!

Введите выражение в формате: a1+a2 либо a1+(a2+a3), где переменная обязательно число,

Доступные операции: +,-,\*,/

Если вы хотите выйти из калькулятора, введите вместо выражения кодовое слово "exit"

10 + 5

Result: 15.000000

Protocol: tcp, Expression: 10 + 5 - PASSED

20/4

Result: 5.000000

Protocol: tcp, Expression: 20/4 - PASSED

20\*2

Result: 40.000000

Protocol: tcp, Expression: 20\*2 - PASSED

2 + (3+4)

Result: 9.000000

Protocol: tcp, Expression: 2 + (3+4) - PASSED

10 - (5-2)

Result: 7.000000

Protocol: tcp, Expression: 10 - (5-2) - PASSED

12\*(3/2)

Result: 18.000000

Protocol: tcp, Expression: 12\*(3/2) - PASSED

(2+3)+4

Result: 9.000000

Protocol: tcp, Expression: (2+3)+4 - PASSED

(10-5.2)-2

Result: 2.800000

Protocol: tcp, Expression: (10-5.2)-2 - PASSED

10+5\*2

Result: 20.000000

Protocol: tcp, Expression: 10+5\*2 - PASSED

20/4-1

Result: 4.000000

Protocol: tcp, Expression: 20/4-1 - PASSED

2+3+4

Result: 9.000000

Protocol: tcp, Expression: 2+3+4 - PASSED

10-5-2

Result: 3.000000

Protocol: tcp, Expression: 10-5-2 - PASSED

12\*3/2

Result: 18.000000

Protocol: tcp, Expression: 12\*3/2 - PASSED

100/5\*2

Result: 40.000000

Protocol: tcp, Expression: 100/5\*2 - PASSED

1.5+2.5\*2

Result: 6.500000

Protocol: tcp, Expression: 1.5+2.5\*2 - PASSED

10/3\*3

Result: 10.000000

Protocol: tcp, Expression: 10/3\*3 - PASSED

1.5+2.5\*2

Result: 6.500000

Protocol: tcp, Expression: 1.5+2.5\*2 - PASSED

9999999999+1\*2

Result: 10000000001.000000

Protocol: tcp, Expression: 9999999999+1\*2 - PASSED

10+a\*2

Неверный формат ввода. Ожидается 'число оператор число' либо 'число оператор (число оператор число)' либо 'число оператор число оператор число' либо команда 'exit'.

Protocol: tcp, Expression: 10+a\*2 - PASSED

(12\*3)/0

Received separation Error: Деление на 0

Protocol: tcp, Expression: (12\*3)/0 - PASSED

10 + 5 + a

Неверный формат ввода. Ожидается 'число оператор число' либо 'число оператор (число оператор число)' либо 'число оператор число оператор число' либо команда 'exit'.

Protocol: tcp, Expression: 10 + 5 + a - PASSED

999999999999999999999999999\*9999999999999999999

Result: 9999999999999999931398190359470212947659194368.000000

Protocol: tcp, Expression: 999999999999999999999999999\*9999999999999999999 - PASSED

exit

Protocol: tcp, Expression: exit - PASSED

Received exit command. Client closed.

Protocol: tcp - Connection closed gracefully

# Приложение №5

----------------------------------------

Testing Protocol: udp

----------------------------------------

spawn ./Client/Client 127.0.0.1 udp

Client was started. Connected with udp

Добро пожаловать в Калькулятор!

Введите выражение в формате: a1+a2 либо a1+(a2+a3), где переменная обязательно число,

Доступные операции: +,-,\*,/

Если вы хотите выйти из калькулятора, введите вместо выражения кодовое слово "exit"

10 + 5

Result: 15.000000

Protocol: udp, Expression: 10 + 5 - PASSED

20/4

Result: 5.000000

Protocol: udp, Expression: 20/4 - PASSED

20\*2

Result: 40.000000

Protocol: udp, Expression: 20\*2 - PASSED

2 + (3+4)

Result: 9.000000

Protocol: udp, Expression: 2 + (3+4) - PASSED

10 - (5-2)

Result: 7.000000

Protocol: udp, Expression: 10 - (5-2) - PASSED

12\*(3/2)

Result: 18.000000

Protocol: udp, Expression: 12\*(3/2) - PASSED

(2+3)+4

Result: 9.000000

Protocol: udp, Expression: (2+3)+4 - PASSED

(10-5.2)-2

Result: 2.800000

Protocol: udp, Expression: (10-5.2)-2 - PASSED

10+5\*2

Result: 20.000000

Protocol: udp, Expression: 10+5\*2 - PASSED

20/4-1

Result: 4.000000

Protocol: udp, Expression: 20/4-1 - PASSED

2+3+4

Result: 9.000000

Protocol: udp, Expression: 2+3+4 - PASSED

10-5-2

Result: 3.000000

Protocol: udp, Expression: 10-5-2 - PASSED

12\*3/2

Result: 18.000000

Protocol: udp, Expression: 12\*3/2 - PASSED

100/5\*2

Result: 40.000000

Protocol: udp, Expression: 100/5\*2 - PASSED

1.5+2.5\*2

Result: 6.500000

Protocol: udp, Expression: 1.5+2.5\*2 - PASSED

10/3\*3

Result: 10.000000

Protocol: udp, Expression: 10/3\*3 - PASSED

1.5+2.5\*2

Result: 6.500000

Protocol: udp, Expression: 1.5+2.5\*2 - PASSED

9999999999+1\*2

Result: 10000000001.000000

Protocol: udp, Expression: 9999999999+1\*2 - PASSED

10+a\*2

Неверный формат ввода. Ожидается 'число оператор число' либо 'число оператор (число оператор число)' либо 'число оператор число оператор число' либо команда 'exit'.

Protocol: udp, Expression: 10+a\*2 - PASSED

(12\*3)/0

Received separation Error: Деление на 0

Protocol: udp, Expression: (12\*3)/0 - PASSED

10 + 5 + a

Неверный формат ввода. Ожидается 'число оператор число' либо 'число оператор (число оператор число)' либо 'число оператор число оператор число' либо команда 'exit'.

Protocol: udp, Expression: 10 + 5 + a - PASSED

999999999999999999999999999\*9999999999999999999

Result: 9999999999999999931398190359470212947659194368.000000

Protocol: udp, Expression: 999999999999999999999999999\*9999999999999999999 - PASSED

exit

Protocol: udp, Expression: exit - PASSED

Received exit command. Client closed.

Protocol: udp - Connection closed gracefully

========================================

TESTS FINISHED