

Министерство образования Российской Федерации
Пензенский государственный университет
Кафедра «Вычислительная техника»

ПОЯСНИТЕЛЬНАЯ ЗАПИСКА
к курсовой работе
по курсу «Архитектура операционных систем»
на тему «Система программ обмена текстовыми
сообщениями»

Выполнил:

студент группы 21BBB1

Гришин С.В.

Принял:

к.т.н., доцент Егоров В.Ю.

удов

Вз

29.02.2024

ПЕНЗЕНСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ
Факультет Вычислительной техники
Кафедра "Вычислительная техника"

"УТВЕРЖДАЮ"

Зав. кафедрой ВТ
профессор М. А. Митрохин

« 18 20 18 г

ЗАДАНИЕ

на курсовое проектирование по курсу

Архитектура операционных систем
Студенту Тришкину Сергею Вадимовичу Группа 21ВВВ1
Тема проекта Система программ обмена текстовыми сообщениями

Исходные данные (технические требования) на проектирование

Разработать систему программ клиент-серверной архитектуры, позволяющее пользователям обмениваться текстовыми сообщениями в рамках локальной вычислительной сети. Клиент взаимодействует с сервером на основе вновь создаваемой в рамках курсовой работы протоколов обмена данными. Необходимо предусмотреть графический интерфейс пользовательского приложения, консольного интерфейса сервера. Сообщения могут быть индивидуальными и широковещательными.

Опционально*

- идентификация пользователя

Язык прог. С, С++, С#, комбинация
ОС любые.

Объем работы по курсу

1. Расчетная часть

1. Разработка протокола взаимодействия программ
2. Покомпонентная отладка приложений
3. Комплексная отладка системы программ.

2. Графическая часть

Не менее 5 ИМЛ-диаграмм, контролирующих работу системы программ.

3. Экспериментальная часть

Срок выполнения проекта по разделам

1. Согласно графике
2. работы на ф
3. проекте.
- 4.
- 5.
- 6.
- 7.
- 8.

Дата выдачи задания " ____ " ____

Дата защиты проекта " ____ " ____

Руководитель 

Задание получил " ____ " ____

Студент Гришин С. В.  20 ____ г.

Содержание

Введение	3
Основная часть	4
1 Постановка задачи.....	4
2 Теоритические сведения	5
3 Выбор решения.....	6
4 Описание протокола	7
5 Описание программы	8
6 Руководство пользователя	12
Заключение	15
Список используемых источников.....	16
Приложение А.1	17
Приложение А.2	26
Приложение В.1.....	34
Приложение В.2.....	35
Приложение В.3.....	36

Введение

Современный мир не обойтись без средств обмена информацией и коммуникации. В данной курсовой работе рассматривается процесс разработки и реализации мессенджера – программы, предназначенной для обмена сообщениями и файлами между пользователями в сети.

Целью данной курсовой работы является создание мессенджера, предназначенного для обмена текстовыми сообщениями и файлами между пользователями в локальной сети. Приложение ориентировано на обеспечение удобного общения, а также обмена файлами в реальном времени.

В ходе разработки приложения будут рассмотрены основные этапы создания мессенджера: многопользовательский доступ, управление файлами, авторизация и аутентификация. Процесс создания будет описан, начиная от проектирования структуры приложения до реализации ключевых функциональных возможностей.

Результатом работы должно стать приложение, способное обеспечивать удобное взаимодействие пользователей и эффективный обмен файлами в сети. Полученный опыт и результаты исследования могут быть полезными для понимания процесса создания подобных приложений и их дальнейшего усовершенствования.

Основная часть

1 Постановка задачи

Цель работы заключается в разработке клиент-серверной архитектуры, предназначенной для обмена текстовыми сообщениями в локальной вычислительной сети. Необходимо создать систему, в которой клиенты могут взаимодействовать с сервером на основе вновь создаваемого протокола обмена данными. Необходимой частью является наличие графического интерфейса пользовательского приложения и консольного интерфейса сервера.

Требуется разработать протокол обмена данными, который обеспечит передачу информации между клиентами и сервером. Графический интерфейс пользовательского приложения должен поддерживать отправку как индивидуальных, так и широковещательных сообщений. Консольный интерфейс сервера предоставит возможность мониторинга обмена сообщениями.

Опционально внедрить механизмы сигнализации о наличии пользователей в сети. Благодаря этой функции каждый пользователь может получать уведомления о входе и выходе своих контактов, что обеспечивает более полное представление о текущем статусе онлайн-активности.

Так же дополнительно система может предоставлять возможности идентификации и авторизации пользователей, что повышает уровень безопасности и обеспечивает защищенный доступ к обмену данными.

Язык программирования – C#. Операционная система – Windows.

2 Теоретические сведения

Программы в сети Интернет взаимодействуют друг с другом с использованием специальных правил и соглашений, называемых протоколами. Протокол представляет собой набор инструкций, определяющих, как передавать данные.

Для того чтобы программа могла функционировать в сети, ей необходимо соблюдать определенные правила при передаче и приеме данных. Одним из основных протоколов, используемых в сети Интернет, является протокол TCP/IP. TCP (Transmission Control Protocol) и IP (Internet Protocol) – это два взаимосвязанных протокола. TCP управляет передачей данных, определяя, как данные разбиваются на пакеты и отправляются по каналам связи. Он также обеспечивает правильное распределение пакетов и проверку их на ошибки передачи.

Каждый информационный пакет включает в себя IP-адреса отправителя и получателя. Маршрутизаторы, специальные устройства, используют эти IP-адреса для направления пакетов в нужном направлении, к указанному получателю.

Для эффективной работы прикладных программ, таких как почтовые клиенты, необходимы дополнительные протоколы. Эти протоколы определяют правила упаковки и передачи данных, а также процедуры обмена информацией. Например, для получения электронного письма может потребоваться предъявление пароля к почтовому ящику, что включает в себя целую последовательность действий. Таким образом, различные протоколы обеспечивают эффективное взаимодействие между программами в сети Интернет.

3 Выбор решения

Для реализации взаимодействия между сервером и клиентами был выбран протокол TCP/IP, который обеспечивает связь, используя подтверждения о получении данных и механизм повторной передачи при неудачной передаче. Широко применяется в прикладных протоколах, таких как HTTP, FTP, SMTP и Telnet. TCP требует установки соединения перед отправкой сообщения, при этом сервер ожидает входящие запросы, а клиент отправляет синхронизирующий порядковый номер при активном открытии соединения. В данной системе взаимодействия будет применена топология "звезда". Каждый клиент подключается к серверу, который является центральным узлом. Для наглядности топология представлена на рисунке 1.

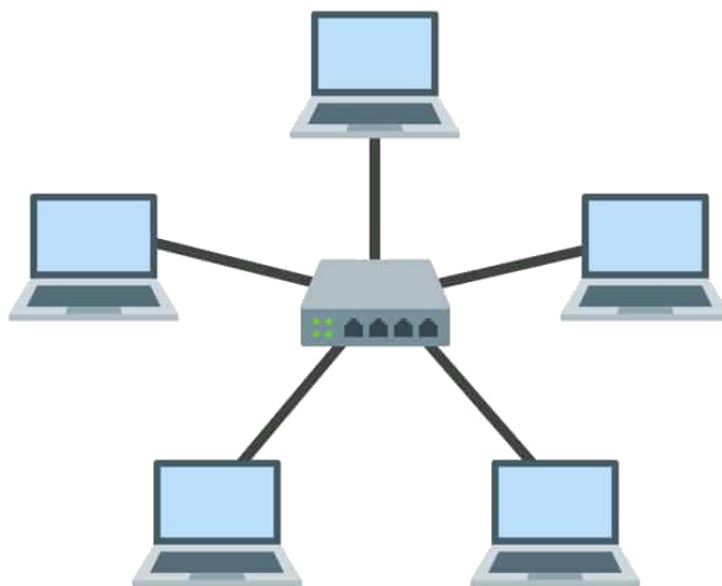


Рисунок 1 – Топология “звезда”

Для обеспечения безопасного обмена данными между клиентом и сервером, в решение был внедрен простейший механизм шифрования передаваемого сообщения.

4 Описание протокола

Протокол представляет собой договоренность, применяемую как клиентом, так и сервером. С использованием данного протокола клиент и сервер взаимодействуют, осуществляя согласованный обмен информацией.

Протокол реализован как в клиентской части (файл STMclient.cs), так и в серверной части (файл STMserver.cs). Отправляемые сообщения состоят из типа (первый байт сообщения), который указывает на содержимое сообщения и как его обработать, и самого сообщения.

Тип 'm' это обычное сообщение, которое сервер принимает и отправляет получателю, клиент выводит данные сообщения на экран.

Тип 't' это команды (такие как \list или \exit), каждая из которых обрабатывается уникальным образом, в зависимости от самой команды.

Тип 'n' используется при авторизации для отправки имени.

Тип 'a' сообщение-ответ на предыдущее сообщение.

Тип 'e' сообщения, используемые для протокола "Ping/Pong"

Для наглядной демонстрации протокола в приложении В.1 представлена UML диаграмма деятельности, которая отражает взаимодействие сервера и клиентов через протокол.

5 Описание программы

Ниже представлена работа клиент-серверного приложения.
Для начала необходимо запустить сервер (рисунок 2).

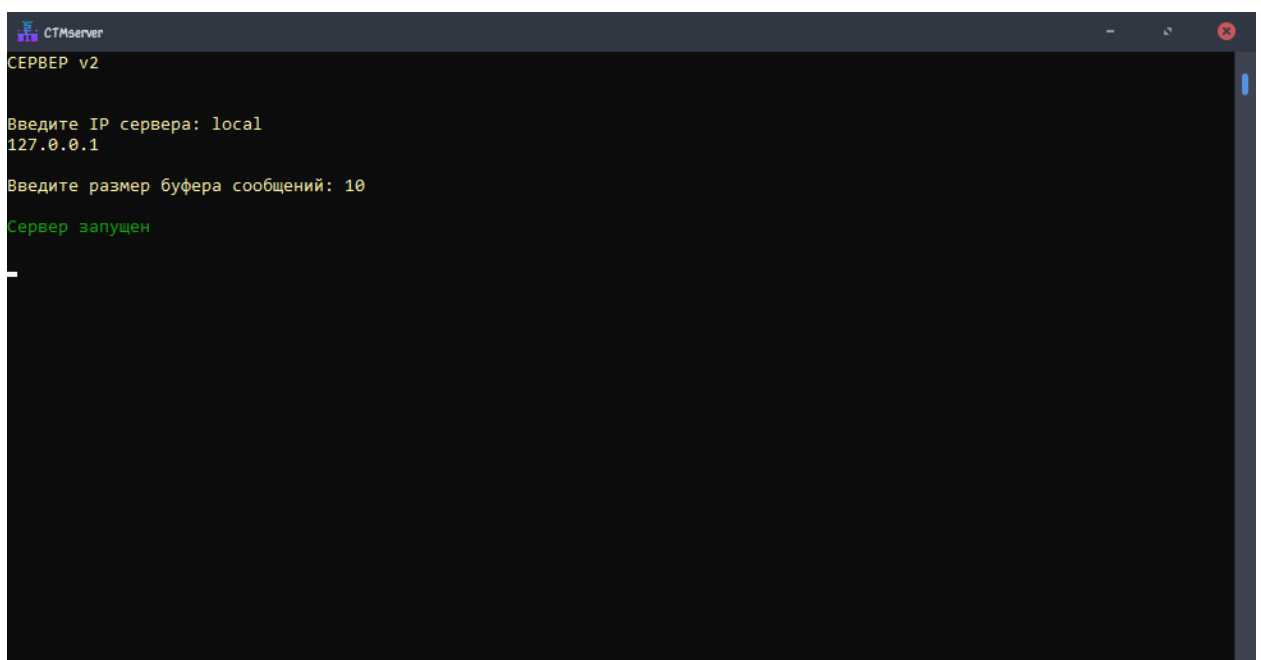


Рисунок 2 – Запуск сервера

После можно запускать клиент, который попросит ввода ip адреса сервера, после успешного подключения пользователю будут предлагать ввести имя до тех пор, пока введённое имя не будет свободно и не будет содержать запрещённых символов (рисунок 3).

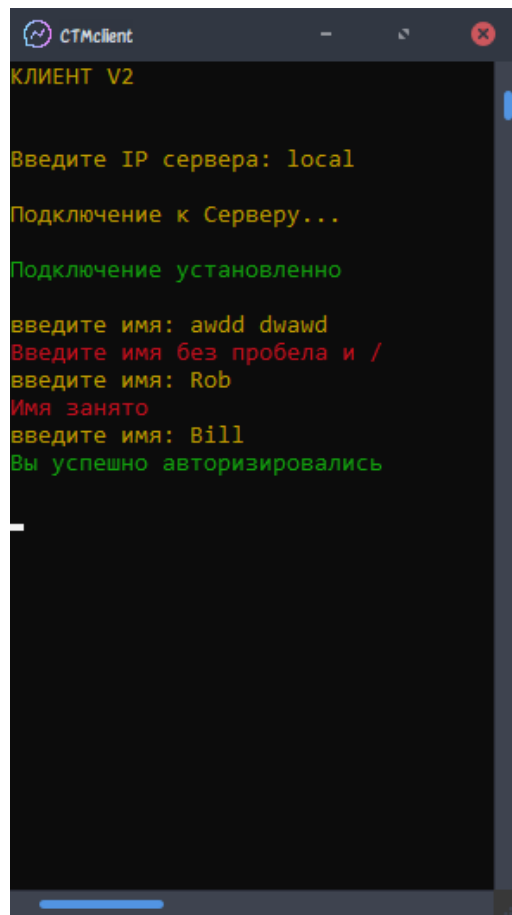


Рисунок 3 – Подключение клиента к серверу и авторизация

После успешного подключения, сервер и клиент начинают непрерывно обмениваться сообщениями (рисунок 4).

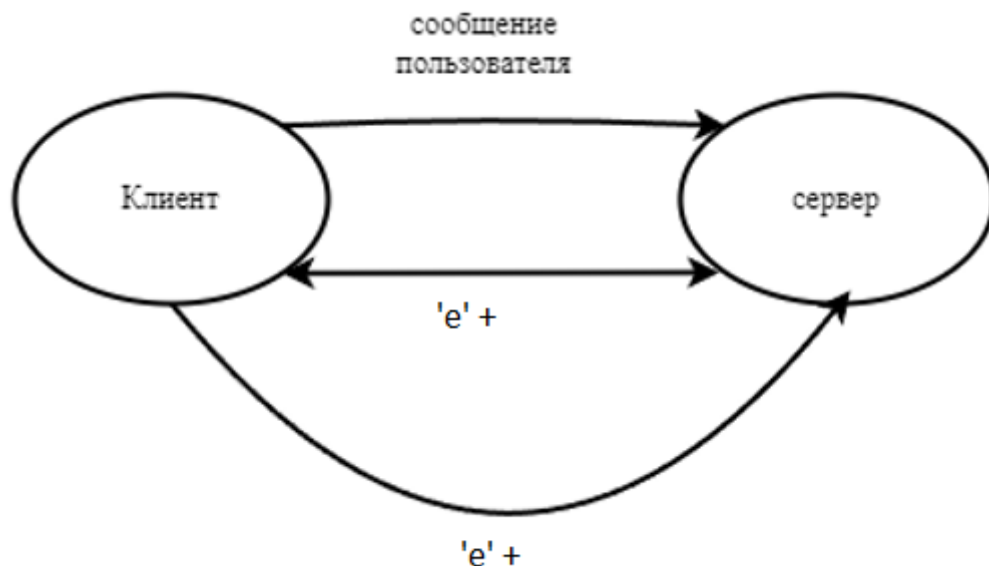


Рисунок 4 – UML-диаграмма вариантов сообщений

Каждые 2 секунды клиент и сервер отправляют друг другу сообщение “e+”. Если кто-то из них не получает такого сообщения в течении 5 секунд, то соединение разрывается. При успешном подключении клиент предложит пользователю ввести сообщение. Пользователь может послать сообщение

всех пользователям в сети, определённому пользователю и запросить список активных пользователей. После ввода сообщение шифруется путём увеличения каждого байта сообщения, которое представляет из себя массив байтов, на значение переменной *ChipKey*. Уже зашифрованное сообщение отправляет на сервер, где оно расшифровывается и обрабатывается.

Сервер и клиент ведут файл “C:\CTMlog\Log.txt”, в котором описаны все события и ошибки при работе программы, такие как отправка и получение сообщений, установка и разрыв соединений, ошибка при отправке и получении, полученные команды и тд. Для выделения основных взаимодействий между клиентской и серверной частями приложения, была разработана UML-диаграмма развертывания. Она служит для визуализации физической структуры системы и взаимодействия её компонентов. На рисунке 5 представлена данная диаграмма, где показана клиентская и серверная часть приложений, а также способы их взаимодействия.

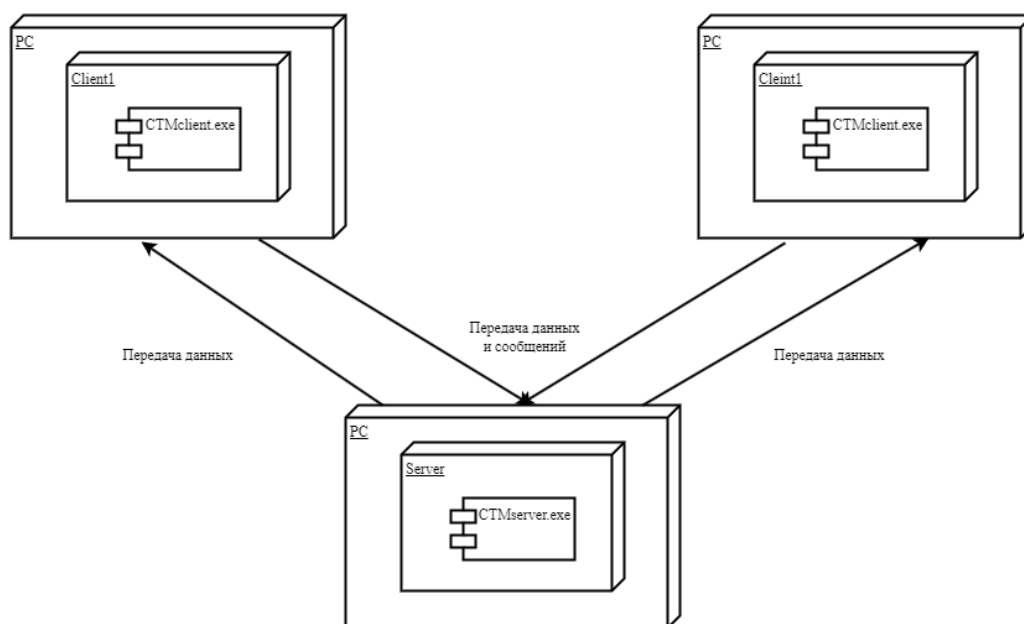


Рисунок 5 – UML - диаграмма развёртывания

В приложении В.2 представлена UML – диаграмма вариантов использования для сервера. Она описывает последовательность событий, которые происходят на сервере во время обработки команд от клиентов.

В приложении В.3 представлена UML – диаграмма вариантов использования для клиента. Она описывает последовательность событий, которые происходят клиенте при обработке команд от пользователя и ответов от сервера.

Руководство пользователя

Для начала использования необходимо запустить сервер, который при попросит ввести ip на котором он будет работать. В этот момент можно ввести сам ip, “host”, который в качестве ip запишет адрес машины в локальной сети и “local”, который в качестве ip запишет 127.0.0.1 . После ввода ip нужно ввести размер буфера сообщений, который отвечает за количество сообщений, которое сможет одновременно храниться в буфере, пока не будет обработано. Если сервер успешно запустился, то он выведет сообщение “Сервер запущен” (рисунок 6).

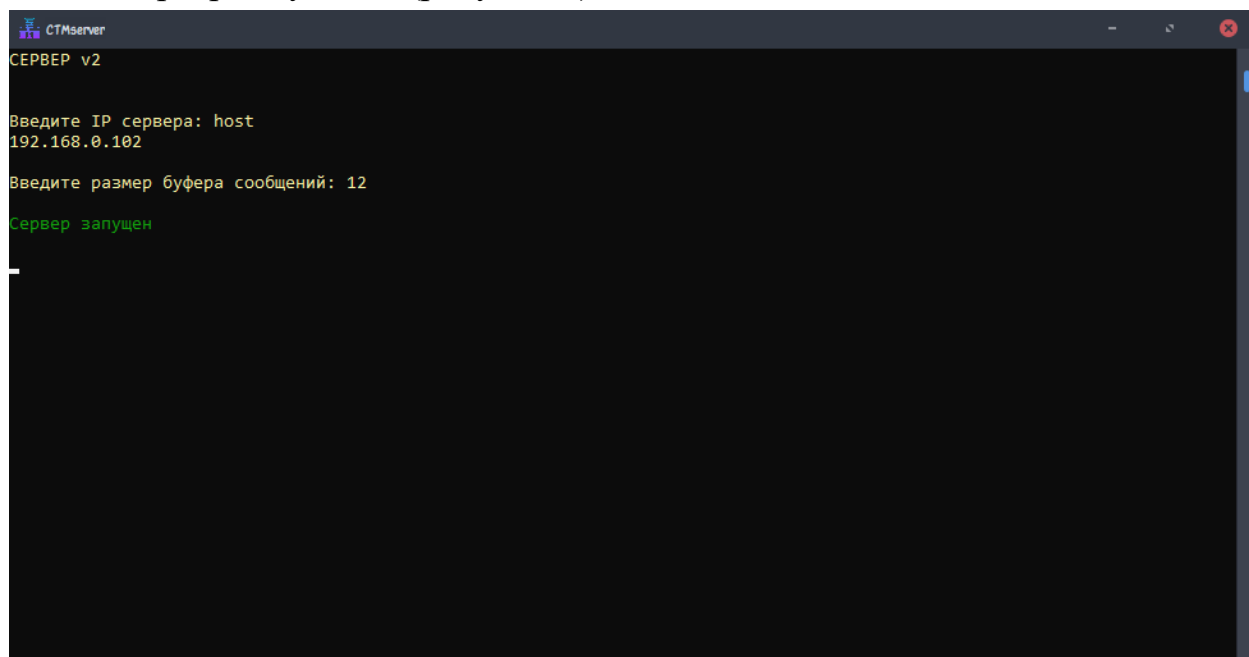


Рисунок 6 – запуск сервера

После можно запускать клиент, который при успешном подключении выведет сообщение “Подключение установлено. Если клиент не смог подключиться к серверу, то он выведет сообщение “Не удалось подключиться к серверу” и закроется. При успешном подключении клиент предложит пользователю ввести имя, которое не должно содержать знак ‘/’ и пробел, а также не должно быть занято другим пользователем(рисунок 7)

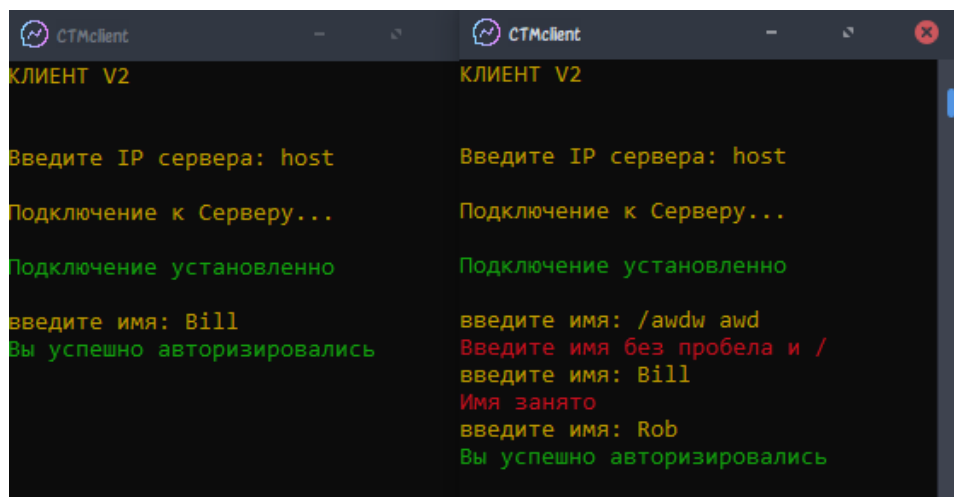


Рисунок 7 – авторизация пользователя

Так же пользователь может запросить список активных пользователей командой \list (рисунок 8)

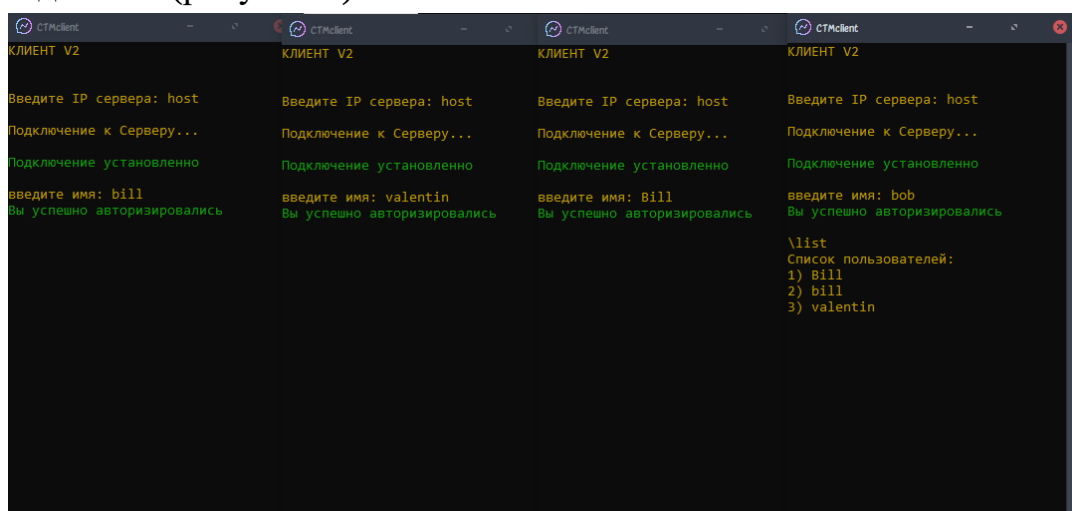


Рисунок 8 – вызов \list

Чтобы отправить сообщение всем пользователям необходимо просто ввести сообщение в консоль, чтобы оно не начиналось со знаков ‘\’ или ‘/’ (рисунок 9).

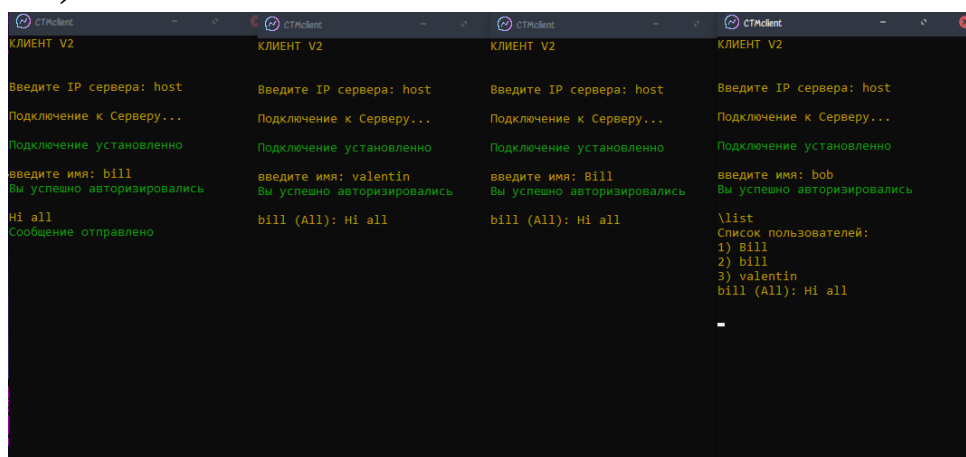


Рисунок 9 – отправка сообщения всем пользователям

Чтобы отправить сообщение определённому пользователю необходимо ввести сообщение с следующей конструкцией (`/<имя пользователя> <сообщение>`) (рисунок 10).

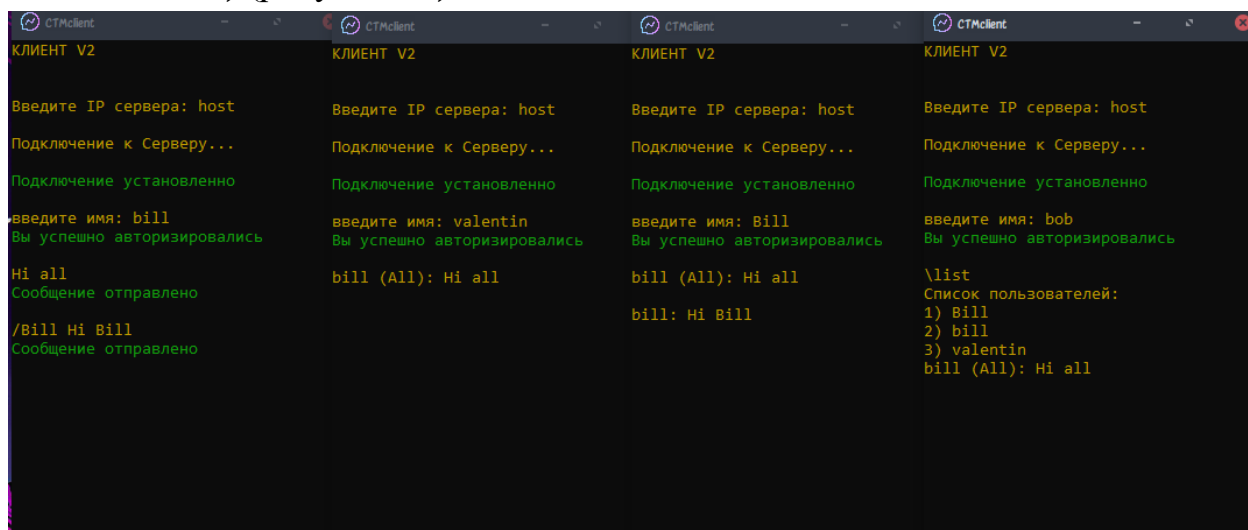


Рисунок 10 – отправка сообщения определённому пользователю

Чтобы закрыть сервер или клиент введите в консоль `/exit` (после закрытия сервера через 5 секунд закроются все клиенты).

Заключение

В ходе выполнения курсовой работы по разработке программы клиент-серверной архитектуры были получены практические навыки, программирования на языке C# с использованием библиотек System.Net и System.Serilog. Этот проект позволил углубить знания в области работы с TCP/IP и клиент-серверными приложениями.

Список используемых источников

1. Осипов Н.А. Разработка Windows приложений на С# : учебное пособие / Осипов Н.А. – Санкт-Петербург. Изд-во: НИУ ИТМО. 2012.
2. Cyberforum. [Электронный ресурс] // <https://www.cyberforum.ru/> – Режим доступа – URL.
3. Документация по С# Начало работы, руководства, справочные материалы Microsoft <https://learn.microsoft.com/ru-ru/dotnet/csharp//> – Режим доступа – URL.

Приложение А.1 Листинг STMserver.cs

```
using Serilog;
using System.Net;
using System.Net.Sockets;
using System.Text;

//сервер V2

Encoding.RegisterProvider(CodePagesEncodingProvider.Instance);
var enc1251 = Encoding.GetEncoding(1251);
System.Console.OutputEncoding = System.Text.Encoding.UTF8;
System.Console.InputEncoding = enc1251;

Console.BackgroundColor = ConsoleColor.Black;
Console.ForegroundColor = ConsoleColor.Yellow;
Console.Clear();
Console.WriteLine("СЕРВЕР v2\n\n");

string ServerIP;
Console.Write("Введите IP сервера: ");
ServerIP = Console.ReadLine();
if(ServerIP == "local") { ServerIP = "127.0.0.1"; }
if(ServerIP == "host")
{
    string Host = System.Net.Dns.GetHostName();
    foreach(var U in System.Net.Dns.GetHostByName(Host).AddressList)
    {
        if(U.ToString().Contains("192.168.0"))
        {
            ServerIP = U.ToString();
            break;
        }
    }
}

//Логер
Log.Logger = new LoggerConfiguration().WriteTo.File("C:\\STMlogs\\ServerLog.txt", rollingInterval:
RollingInterval.Day, rollOnFileSizeLimit: true).CreateLogger();
Log.Information("/////////////////НОВАЯ СЕССИЯ/////////////////");

Console.WriteLine(ServerIP + "\n");
TcpListener Server = new TcpListener(IPAddress.Parse(ServerIP), 21567); //Прослушка запросов
CancellationTokenSource cancellationTokenSource = new CancellationTokenSource(); //Токен отмены

int BufferSize;
do
Console.Write("Введите размер буфера сообщений: ");
while(!int.TryParse(Console.ReadLine(), out BufferSize));
Log.Information("Выбран буфер: " + BufferSize);

byte ChipKey = 5; //Ключ
List<User> UsersList = new List<User>();
//Ожидание команд
Task.Run(async () =>
{
    while (true)
    {
        string Command;
        Command = Console.ReadLine();
        switch (Command)
        {
            case "exit":
            {
                Environment.Exit(0);
            }
        }
    }
}
```

```

        break;
    }
    default:
    {
        ColorPrint('r', "Неизвестная команда\n\n");
        break;
    }
}
}
});

```

```
Task.Run(async () => await ProccesClientConnections(Server, cancellationTokenSource.Token));
```

```
AutoResetEvent autoResetEvent = new AutoResetEvent(false);
autoResetEvent.WaitOne();
```

```
//Ожидание клиента
```

```
async Task ProccesClientConnections(TcpListener server, CancellationToken cancellationToken)
```

```

{
    Server.Start();
    ColorPrint('g', "\nСервер запущен\n\n");
    while (!cancellationToken.IsCancellationRequested)
    {
        try
        {
            var client = await Server.AcceptTcpClientAsync();
            Task.Run(async () => await ProcessNewClientConnectionAsync(client));
        }
        catch (Exception ex)
        {
            Log.Error(ex.Message + "(Ошибка при прослушивании)");
        }
    }
    Server.Stop();
}

```

```
//Работа с клиентом-----
```

```
async Task ProcessNewClientConnectionAsync(TcpClient ClientConnection)
```

```

{
    //Подключение
    IPEndPoint ClientSocket = (IPEndPoint)ClientConnection.Client.RemoteEndPoint; //Информация о IP
    Mess[] MessageBuffer = new Mess[BufferSize]; //буфер сообщений
    var networkStream = ClientConnection.GetStream(); //поток сообщений
    User UserInfo = new User(); //информация о пользователе
    bool Connected = true;
    UserInfo.Name = string.Empty;
    UserInfo.Client = ClientConnection;
    byte[] Buffer = new byte[1024];
    //string Message = string.Empty;
    var currentTime = DateTime.Now;

```

```
//Ожидание сообщений
```

```
Task.Run(async () =>
```

```

{
    while (Connected == true)
    {
        int l = 0;
        string WaitMessage = string.Empty;
        byte[] Buffer = new byte[1024];
        try
        {
            l++;

            await networkStream.ReadAsync(Buffer, 0, Buffer.Length);
            l++;
            WaitMessage = System.Text.Encoding.UTF8.GetString(Buffer);

```

```

l++;
WaitMessage = WaitMessage.Trim('\0');
l++;
if (WaitMessage.Length > 0)
{
    l++;
    for (int i = 0; i < WaitMessage.Length; i++)
    {
        Buffer[i] -= ChipKey;
    }
    l++;
    WaitMessage = System.Text.Encoding.UTF8.GetString(Buffer);
    l++;
    WaitMessage = WaitMessage.Trim('\0');
    l++;
    Mess Cur = new Mess();
    l++;
    Cur.Type = WaitMessage[0];
    l++;
    WaitMessage = WaitMessage.Substring(1);
    l++;
    Cur.Message = WaitMessage;
    l++;
    Cur.Message = Cur.Message + "\0";
    l++;
    if (Cur.Type == 'e')
    {
        currentTime = DateTime.Now;
    }
    else
    {
        Task.Run(async () =>
        {
            MessageBuffer = await AddQueue(Cur, MessageBuffer);
        });
    }
}
}
catch (Exception ex)
{
    Log.Error(ex.Message + " (Не получено сообщение " + WaitMessage + " от " + ClientSocket + ") " + l);
}
//break;
}
});

//Эхо отправка
Task.Run(async () =>
{
    while (Connected == true)
    {
        try
        {
            Send('e', "+", ClientConnection);
            await Task.Delay(2000);
        }
        catch (Exception ex)
        {
            Log.Error(ex.Message + "(Ошибка при отправке эхо" + " от " + ClientSocket + ")");
        }
    }
});

//Проверка отклика
Task.Run(async () =>

```

```

{
    try
    {
        while (Connected == true)
        {
            var time = DateTime.Now - currentTime;
            if (time.TotalSeconds > 5)
            {
                Connected = false;
                ColorPrint('r', "Отклик от " + ClientSocket + " не получен. Соединение разорвано\n");
                Log.Information("Отклик от " + ClientSocket + " не получен. Соединение разорвано");
                if (UserInfo.Name != string.Empty)
                {
                    UsersList.Remove(UserInfo);
                }
                ClientConnection.Close();
                break;
            }
            await Task.Delay(1000);
        }
    }
    catch(Exception ex)
    {
        Log.Error(ex.Message + " (Ошибка при обработке эхо" + " от " + ClientSocket + ")");
    }
});

//Авторизация
try
{
    do
    {
        string Name;
        bool res = true;
        Name = await SearchQueue('n', MessageBuffer);
        foreach (User U in UsersList)
        {
            if (U.Name == Name)
            {
                await Send('a', "-", ClientConnection);
                res = false;
                break;
            }
        }
        if (res == true) { UserInfo.Name = Name; await Send('a', "+", ClientConnection); }
    } while (UserInfo.Name == string.Empty);
    UsersList.Add(UserInfo);
    ColorPrint('g', "Пользователь " + ClientSocket + " подключился как " + UserInfo.Name + "\n");
    Log.Information("Успешная авторизация " + ClientSocket + " под именем " + UserInfo.Name);
}
catch(Exception ex)
{
    Log.Error(ex.Message + " (Ошибка авторизации " + ClientSocket + ")");
}

//Обработка сообщений
while (Connected == true)
{
    try
    {
        Mess CurMessage = new Mess();
        CurMessage = await SearchQueueAll(MessageBuffer);
        switch (CurMessage.Type)
        {

```

```

//Обработка сообщений
case 'm':
{
    if (CurMessage.Message[0] == '/')
    {
        CurMessage.Message = CurMessage.Message.Substring(1);
        string Recipient = CurMessage.Message.Substring(0, CurMessage.Message.IndexOf(' '));
        CurMessage.Message = CurMessage.Message.Substring(CurMessage.Message.IndexOf(' ') + 1);
        byte Res = 0;
        foreach (User U in UsersList)
        {
            if (U.Name == Recipient)
            {
                {
                    if (await Send('m', UserInfo.Name + ": " + CurMessage.Message, U.Client))
                    {
                        Res = 1;
                    }
                    else
                    {
                        Res = 2;
                    }
                }
                break;
            }
        }
        switch (Res)
        {
            case 0: await Send('a', "-", ClientConnection); break;
            case 1: await Send('a', "+", ClientConnection); break;
            case 2: await Send('a', "--", ClientConnection); break;
        }
        Console.WriteLine("сообщение \'" + CurMessage.Message + "\" от " + UserInfo.Name + "(" +
ClientSocket + ") для " + Recipient);
        Log.Information("сообщение \'" + CurMessage.Message + "\" от " + UserInfo.Name + "(" +
ClientSocket + ") для " + Recipient);
    }
    else
    {
        foreach (User U in UsersList)
        {
            if (U.Name != UserInfo.Name)
            {
                {
                    await Send('m', UserInfo.Name + " (All): " + CurMessage.Message, U.Client);
                }
            }
        }
        Console.WriteLine("сообщение \'" + CurMessage.Message + "\" от " + UserInfo.Name + "(" +
ClientSocket + ") для всех");
        Log.Information("сообщение \'" + CurMessage.Message + "\" от " + UserInfo.Name + "(" +
ClientSocket + ") для всех");
        await Send('a', "+", ClientConnection);
    }
    break;
}
//Обработка команд
case 't':
{
    switch (CurMessage.Message)
    {
        case "list":
        {
            Console.WriteLine("Запрос list от " + ClientSocket + " (" + UserInfo.Name + ")");
            Send('a', (UsersList.Count() - 1).ToString(), ClientConnection);
            if ((UsersList.Count() - 1 != 0))
            {
                foreach (User U in UsersList)
                {

```

```

        if (U.Name != UserInfo.Name)
        {
            await Send('t', U.Name, ClientConnection);
            await SearchQueue('a', MessageBuffer);
        }
    }
    Log.Information("Успешная обработка List от " + ClientSocket + " (" + UserInfo.Name + ")");
    break;
}
case "exit":
{
    Connected = false;
    Log.Information("пользователь " + ClientSocket + " вышел");
    if (UserInfo.Name != string.Empty)
    {
        ColorPrint('r', "пользователь " + UserInfo.Name + " (" + ClientSocket + ") вышел\n");
        UsersList.Remove(UserInfo);
    }
    else
    {
        ColorPrint('r', "пользователь " + ClientSocket + " вышел\n");
    }
    ClientConnection.Close();
    break;
}
}
break;
}
}
}
catch (Exception ex)
{
    Log.Error(ex.Message + " (Ошибка при получении" + " от " + ClientSocket + ")");
}
}
}

//VOIDS-----
void ColorPrint(char Color, string Message)
{
    if (Color == 'G' || Color == 'g')
    {
        Console.ForegroundColor = ConsoleColor.DarkGreen;
        Console.Write(Message);
        Console.ForegroundColor = ConsoleColor.Cyan;
    }
    if (Color == 'R' || Color == 'r')
    {
        Console.ForegroundColor = ConsoleColor.DarkRed;
        Console.Write(Message);
        Console.ForegroundColor = ConsoleColor.Cyan;
    }
}

async Task<bool> Send(char Type, string Message, TcpClient ClientConnection , int Col = 1)
{
    for (int j = 0; j < Col; j++)
    {
        try
        {
            byte[] ChipMessage = Encoding.UTF8.GetBytes(Type + Message + '\0');
            for (int i = 0; i < ChipMessage.Length; i++) { ChipMessage[i] += ChipKey; }
            ClientConnection.GetStream().Write(ChipMessage);
            await ClientConnection.GetStream().FlushAsync();
        }
    }
}

```

```

    }
    catch(Exception ex)
    {
        Log.Information(ex.Message + " (Не отправлено сообщение '\" + Message + "\" типа: " + Type + " для:" +
(IPEndPoint)ClientConnection.Client.RemoteEndPoint+ "));
        return false;
    }
}
return true;
}
}
async Task<Mess[]> AddQueue(Mess Cur, Mess[] MessageBuffer)
{
    try
    {
        int i = 0;
        while (true)
        {
            if (MessageBuffer[i].Type == '\0')
            {
                MessageBuffer[i] = Cur;
                break;
            }
            i++;
            if (i >= MessageBuffer.Length) { i = 0; }
        }
        return MessageBuffer;
    }
    catch (Exception ex)
    {
        Log.Error(ex.Message + " (Ошибка при работе с очередью)");
        return MessageBuffer;
    }
}
}
async Task<string> SearchQueue(char Type, Mess[] MessageBuffer)
{
    try
    {
        int i = 0;
        if (MessageBuffer != null)
        {
            if (Type == 0)
            {
                if (MessageBuffer[i].Type != '\0')
                {
                    string Message = MessageBuffer[i].Message.Substring(0, MessageBuffer[i].Message.IndexOf('\0'));
                    MessageBuffer[i].Message = MessageBuffer[i].Message.Remove(0, MessageBuffer[i].Message.Length);
                    MessageBuffer[i].Type = '\0';
                    MessageBuffer[i].Message = null;
                    return Message;
                }
                i++;
                if (i >= MessageBuffer.Length) { i = 0; }
            }
            else
            {
                while (true)
                {
                    if (MessageBuffer[i].Type == Type)
                    {
                        string Message = MessageBuffer[i].Message.Substring(0, MessageBuffer[i].Message.IndexOf('\0'));
                        MessageBuffer[i].Message = MessageBuffer[i].Message.Remove(0, MessageBuffer[i].Message.Length);
                        MessageBuffer[i].Type = '\0';
                        MessageBuffer[i].Message = null;
                        return Message;
                    }
                }
            }
        }
    }
}

```

```

        i++;
        if (i >= MessageBuffer.Length) { i = 0; }
    }
}
return "\0";
}
catch (Exception ex)
{
    Log.Error(ex.Message + " (Ошибка при поиске в очереди)");
    return "0";
}
}

async Task<Mess> SearchQueueAll(Mess[] MessageBuffer)
{
    try
    {
        int i = 0;
        while (true)
        {
            if (MessageBuffer[i].Type != '\0')
            {
                Mess Message = new Mess();
                Message.Message = MessageBuffer[i].Message.Substring(0, MessageBuffer[i].Message.IndexOf('\0'));
                Message.Type = MessageBuffer[i].Type;
                MessageBuffer[i].Message = MessageBuffer[i].Message.Remove(0, MessageBuffer[i].Message.Length);
                MessageBuffer[i].Type = '\0';
                MessageBuffer[i].Message = null;
                return Message;
            }
            i++;
            if (i >= MessageBuffer.Length) { i = 0; }
        }
    }
    catch (Exception ex)
    {
        Log.Error(ex.Message + " (Ошибка при поиске в очереди)");
        return new Mess();
    }
}

struct User
{
    public string Name { get; set; }
    public TcpClient Client { get; set; }
};
struct Mess
{
    public string Message { get; set; }
    public char Type { get; set; }
}

```

Приложение А.2 Листинг STMclient.cs

```
using Serilog;
using System.Net;
using System.Net.Sockets;
using System.Text;
//клиент v2

Encoding.RegisterProvider(CodePagesEncodingProvider.Instance);
var enc1251 = Encoding.GetEncoding(1251);
System.Console.OutputEncoding = System.Text.Encoding.UTF8;
System.Console.InputEncoding = enc1251;

Console.SetWindowSize(35, 30);
Console.BackgroundColor = ConsoleColor.Black;
Console.ForegroundColor = ConsoleColor.DarkYellow;
Console.Clear();
Console.WriteLine("КЛИЕНТ V2\n\n");

//Логер
Log.Logger = new LoggerConfiguration().WriteTo.File("C:\\CTMlogs\\ClientLog.txt", rollingInterval:
RollingInterval.Day, rollOnFileSizeLimit: true).CreateLogger();
Log.Information("/////////////////НОВАЯ СЕССИЯ/////////////////");

//Переменные
TcpClient Connection = new TcpClient();//Тср клиент
NetworkStream networkStream = null;
Mess[] MessageBuffer = new Mess[10];
bool Connected = false;//активность подключения
byte ChipKey = 5;//Ключ
var currentTime = DateTime.Now;
string ServerIP;

//Ввод IP
Console.Write("Введите IP сервера: ");
ServerIP = Console.ReadLine();
if (ServerIP == "local") { ServerIP = "127.0.0.1"; }
```

```

if (ServerIP == "host")
{
    string Host = System.Net.Dns.GetHostName();
    foreach (var U in System.Net.Dns.GetHostByName(Host).AddressList)
    {
        if (U.ToString().Contains("192.168.0"))
        {
            ServerIP = U.ToString();
            break;
        }
    }
}

```

//Подключение

```

Console.WriteLine("\nПодключение к Серверу...");
try
{
    await Connection.ConnectAsync(IPAddress.Parse(ServerIP), 21567);
    networkStream = Connection.GetStream();
    Connected = true;
    ColorPrint('g', "\nПодключение установлено\n\n");
    Log.Information("Подключение установлено");
}
catch (Exception ex)
{
    //Ошибка подключения
    ColorPrint('r', "Не удалось подключиться к серверу");
    Log.Error(ex.Message + " (ошибка при подключении)");
    Environment.Exit(0);
}

```

//Ожидание сообщений

```

Task.Run(async () =>
{
    try
    {
        while (Connected == true)
        {
            string WaitMessage = string.Empty;
            byte[] Buffer = new byte[1024];
            await networkStream.ReadAsync(Buffer, 0, Buffer.Length);
            WaitMessage = System.Text.Encoding.UTF8.GetString(Buffer);
            WaitMessage = WaitMessage.Trim('\0');
            for (int i = 0; i < WaitMessage.Length; i++)
            {
                Buffer[i] -= ChipKey;
            }
            WaitMessage = System.Text.Encoding.UTF8.GetString(Buffer);
            WaitMessage = WaitMessage.Trim('\0');

```

```

        Mess Cur = new Mess();
        Cur.Type = WaitMessage[0];
        WaitMessage = WaitMessage.Substring(1);
        Cur.Message = WaitMessage;
        Cur.Message = Cur.Message + "\0";
        if (Cur.Type == 'e')
        {
            Task.Run(async () => {currentTime = DateTime.Now; });
        }
        else
        {
            Task.Run(async () => AddQueue(Cur));
        }
    }
}
catch (Exception ex)
{
    Log.Information(ex.Message + " (Не получено сообщение");
}
});

Thread.Sleep(15);
//Эхо отправка
Task.Run(async () =>
{
    try
    {
        while (Connected == true)
        {
            Send('e', "+");
            await Task.Delay(2000);
        }
    }
    catch (Exception ex)
    {
        Log.Error(ex.Message + " (Ошибка при отправке эхо)");
    }
});

//Проверка отклика
Task.Run(async () =>
{
    try
    {
        while (Connected == true)
        {
            var time = DateTime.Now - currentTime;
            if (time.TotalSeconds > 5)
            {
                Connected = false;
                Connection.Close();
            }
        }
    }
    catch (Exception ex)
    {
        Log.Error(ex.Message + " (Ошибка при проверке отклика)");
    }
});

```

```

        ColorPrint('r', "Отклик от сервера не получен. Соединение разорвано");
        Log.Information("Отклик от сервера не получен. Соединение разорвано");
        Environment.Exit(0);
        break;
    }
    await Task.Delay(1000);
}
}
catch (Exception ex)
{
    Log.Error(ex.Message + " (Ошибка при обработке эхо)");
}
});

//Авторизация
try
{
    string Name;
    do
    {
        Console.Write("введите имя: ");
        string bf;
        Name = Console.ReadLine();
        if (Name.Contains(' ') || Name.Contains('/'))
        {
            ColorPrint('r', "Введите имя без пробела и /\n");
            continue;
        }
        await Send('n', Name);
        bf = await SearchQueue('a');
        if (bf == "+") { ColorPrint('g', "Вы успешно авторизировались\n\n"); break; }
        if (bf == "-") { ColorPrint('r', "Имя занято\n"); }
    } while (true);
    Log.Information("Успешная авторизация под именем " + Name);
}
catch (Exception ex)
{
    Log.Error(ex.Message + " (Ошибка при авторизации)");
}

//Вывод сообщений
Task.Run(async () =>
{
    try
    {
        while (Connected == true)
        {
            string Message = await SearchQueue('m');
            Console.WriteLine(Message + "\n");
        }
    }
}

```

```

catch (Exception ex)
{
    Log.Error(ex.Message + " (Ошибка при выводе сообщения)");
}
});

//Ожидание ввода
while (Connected == true)
{
    try
    {
        string Message1 = Console.ReadLine();
        if (Message1[0] == '\\')
        {
            Message1 = Message1.Substring(1);
            switch(Message1)
            {
                case "exit":
                {
                    Log.Information("Выход");
                    await Send('t', "exit");
                    Connected = false;
                    Connection.Close();
                    Environment.Exit(0);
                    break;
                }
                case "list":
                {
                    Log.Information("Вызов списка");
                    await Send('t', "list");
                    int Col = int.Parse(await SearchQueue('a'));
                    if(Col > 0)
                    {
                        Console.WriteLine("Список пользователей:");
                        for (int i = 0; i < Col; i++)
                        {
                            Console.WriteLine(i+1 + " " + await SearchQueue('t'));
                            Send('a', " ");
                        }
                    }
                    else
                    {
                        Console.WriteLine("\nНет других пользователей\n");
                    }
                    break;
                }
                default:
                {
                    ColorPrint('r', "Неизвестная команда\n\n");
                    break;
                }
            }
        }
    }
}

```

```

    }
}
else
{
    if (Message1[0] == '/')
    {
        if (Message1.IndexOf(' ') == -1) { ColorPrint('r', "Неверный формат сообщения\n\n"); }
        else
        {
            if (Message1.IndexOf(' ') >= Message1.Length-1)
            {
                ColorPrint('r', "Неверный формат сообщения\n\n");
            }
            else
            {
                await Send('m', Message1);
                Message1 = await SearchQueue('a');
                switch (Message1)
                {
                    case "+": ColorPrint('g', "Сообщение отправлено\n\n"); break;
                    case "-": ColorPrint('r', "Пользователь не найден\n\n"); break;
                    case "--": ColorPrint('r', "Сообщение не доставленно\n\n"); break;
                }
            }
        }
    }
}
else
{
    await Send('m', Message1);
    Message1 = await SearchQueue('a');
    switch (Message1)
    {
        case "+": ColorPrint('g', "Сообщение отправлено\n\n"); break;
        case "-": ColorPrint('r', "Пользователь не найден\n\n"); break;
        case "--": ColorPrint('r', "Сообщение не доставленно\n\n"); break;
    }
}
}
}
catch (Exception ex)
{
    Log.Error(ex.Message + " (Ошибка при отправке)");
}
}

//VOIDS-----
void ColorPrint(char Color, string Message)
{
    if (Color == 'G' || Color == 'g')
    {
        Console.ForegroundColor = ConsoleColor.DarkGreen;
    }
}

```

```

        Console.Write(Message);
        Console.ForegroundColor = ConsoleColor.DarkYellow;
    }
    if (Color == 'R' || Color == 'r')
    {
        Console.ForegroundColor = ConsoleColor.DarkRed;
        Console.Write(Message);
        Console.ForegroundColor = ConsoleColor.DarkYellow;
    }
}

async Task<bool> Send(char Type , string Message)
{
    try
    {
        byte[] ChipMessage = Encoding.UTF8.GetBytes(Type + Message + '\0');
        for (int i = 0; i < ChipMessage.Length; i++) { ChipMessage[i] += ChipKey; }
        Connection.GetStream().Write(ChipMessage);
        await Connection.GetStream().FlushAsync();
        return true;
    }
    catch (Exception ex)
    {
        Log.Error(ex.Message + " (Не удалось отправить сообщение \"" + Message + "\" типа: " + Type + ")");
        return false;
    }
}

async Task AddQueue(Mess Cur)
{
    int i = 0;
    while (true)
    {
        if (MessageBuffer[i].Type == '\0')
        {
            MessageBuffer[i] = Cur;
            break;
        }
        i++;
        if(i >= MessageBuffer.Length) { i = 0; }
    }
}

async Task<string> SearchQueue(char Type)
{
    try
    {
        int i = 0;
        while (true)
        {
            if (MessageBuffer[i].Type == Type)

```

```

        {
            string Message = MessageBuffer[i].Message.Substring(0, MessageBuffer[i].Message.IndexOf('\0'));
            MessageBuffer[i].Message = MessageBuffer[i].Message.Remove(0,
MessageBuffer[i].Message.Length);
            MessageBuffer[i].Type = '\0';
            MessageBuffer[i].Message = null;
            return Message;
        }
        i++;
        if (i >= MessageBuffer.Length) { i = 0; }
    }
}
catch (Exception ex)
{
    Log.Error(ex.Message + " (Ошибка поиска в очереди)");
    return " ";
}
}

struct Mess
{
    public string Message { get; set; }
    public char Type { get; set; }

    public Mess()
    {
        Message = string.Empty;
        Type = ' ';
    }
}

```

Приложение B.1 UML – диаграмма деятельности

На рисунке 9 представлена UML – диаграмма деятельности.

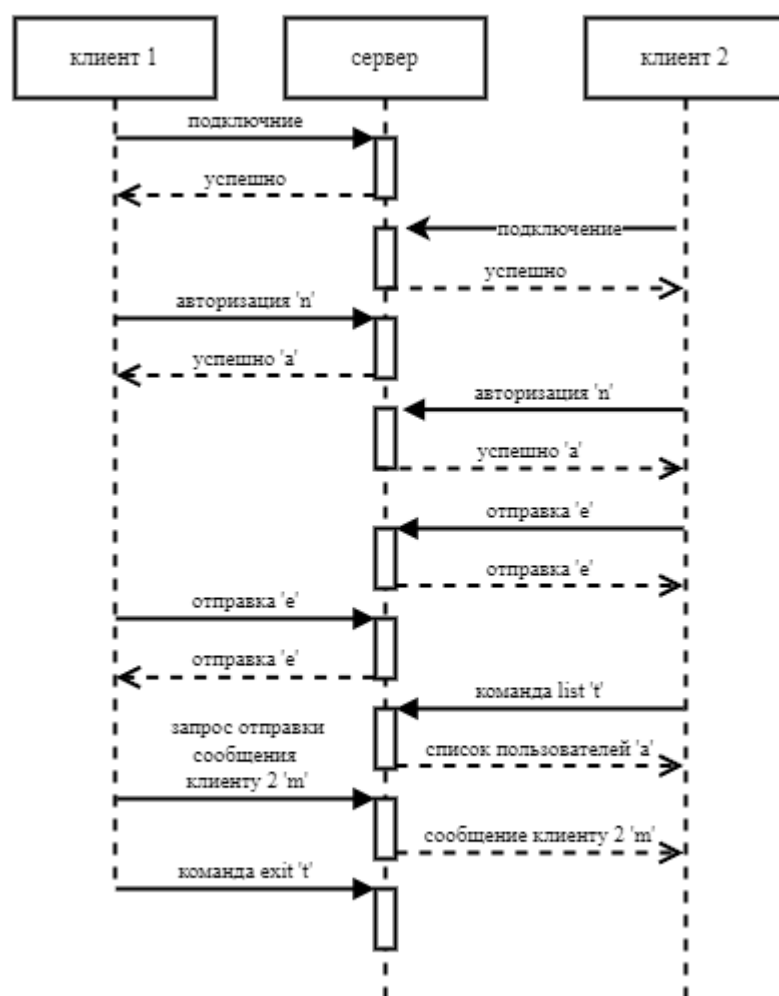


Рисунок 9 – UML - диаграмма деятельности

Приложение В.2 UML – диаграмма состояний сервера

На рисунке 10 представлена UML – диаграмма состояний сервера.

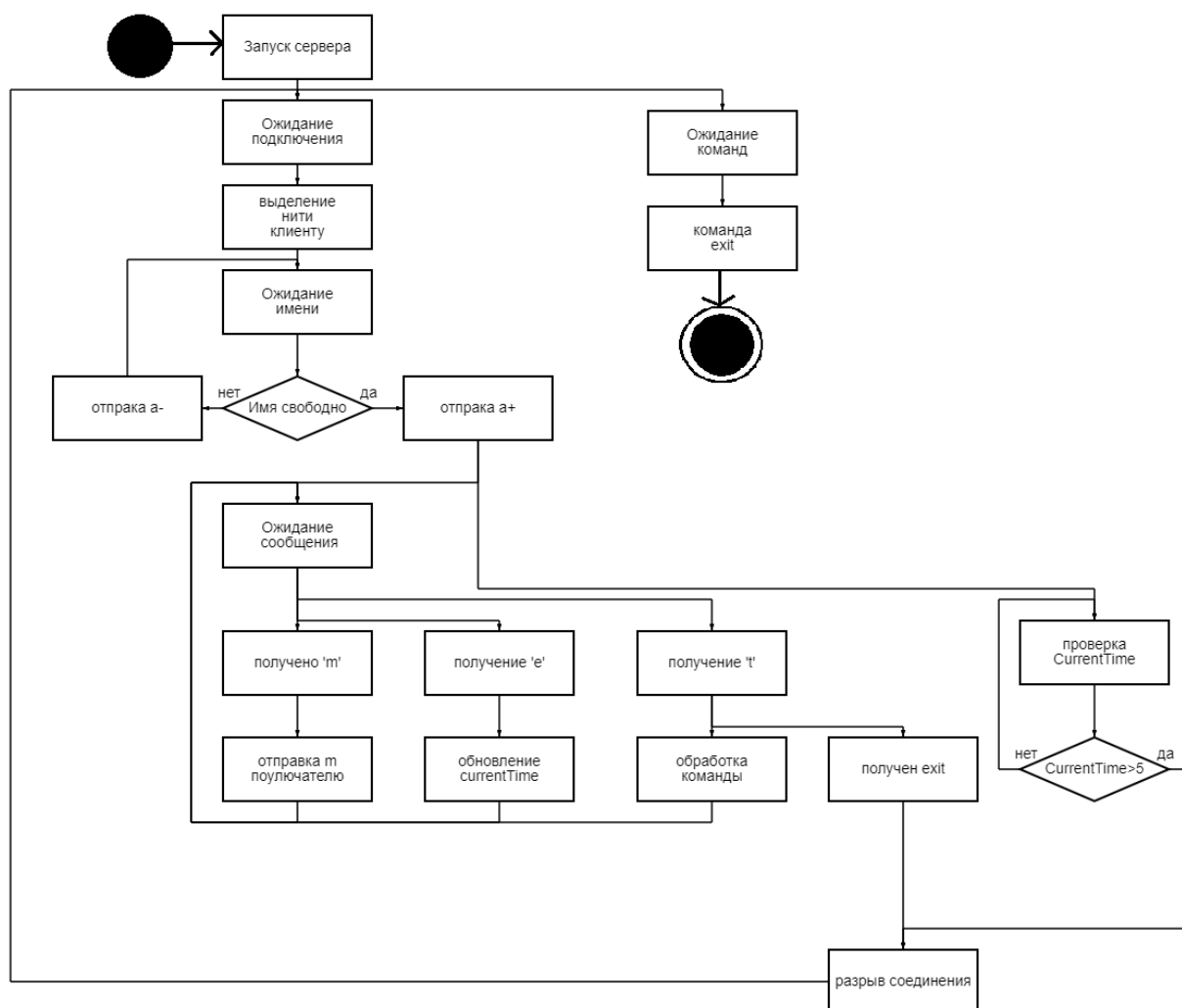


Рисунок 10 – UML - диаграмма состояний сервера

Приложение В.3 UML – диаграмма состояний клиента

На рисунке 11 представлена UML – диаграмма состояний клиента.

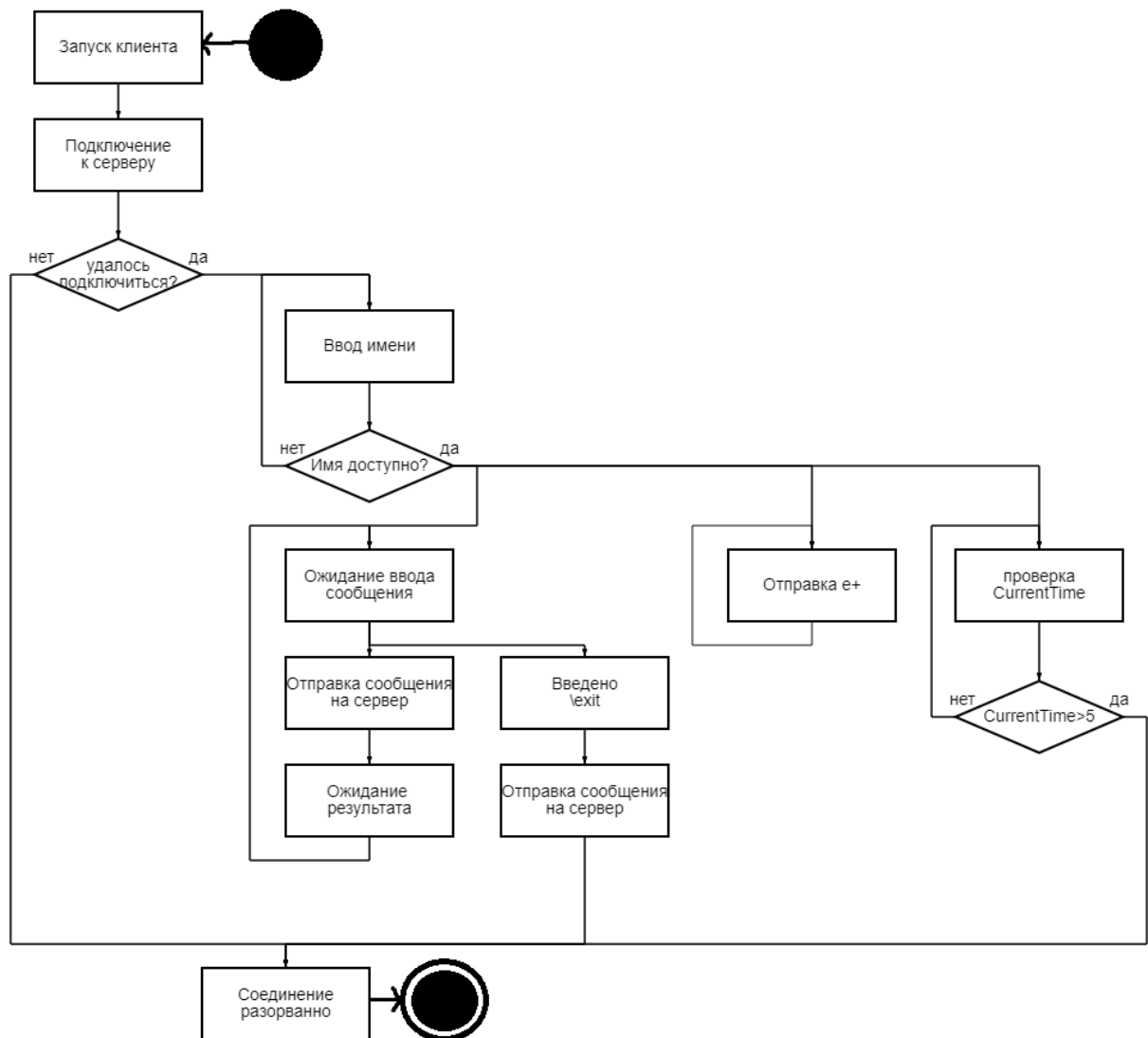


Рисунок 11 – UML - диаграмма состояний клиента