Министерство цифрового развития, связи и

массовых коммуникаций Российской Федерации

Ордена Трудового Красного Знамени федеральное государственное

бюджетное образовательное учреждение высшего образования

«Московский Технический Университет Связи и Информатики»

Кафедра «Системное программирование»

Курсовая работа

«Разработка клиент – серверного приложения»

По дисциплине: «Операционные системы»

Выполнил: Студент группы БФИ1901

Чернышов Д. В.,

Проверила: Алексанян Д. А.

Москва 2021

# 1 Задание на курсовую работу

Разработать многозадачное приложение, выполняющее получение, сбор и вывод системной информации в соответствии с вариантом задания (таблица 1).

1. Изучить методы разработки клиент-серверных приложений.
2. Изучить механизмы обмена данными в Windows NT и особенности их применения на практике.
3. Изучить средства синхронизации процессов в Windows NT и особенности их применения на практике.
4. Изучить функции Win32 API для получения системной информации, информации о процессах и потоках, состоянии памяти.
5. Разработать серверные приложения и приложение-клиент, обменивающиеся данными через заданный интерфейс.
6. Оформить результаты выполнения курсовой работы в виде пояснительной записки в соответствии с предъявляемыми требованиями.

Таблица 1 – Вариант задания

|  |  |  |
| --- | --- | --- |
| Вариант | Сервер 1 | Сервер 2 |
| Получает/выполняет и возвращает в качестве результата | |
| 33 | * Текущее положение курсора * Дескриптор раскладки клавиатуры по умолчанию | * Идентификатор серверного процесса * Перечень собственных потоков с указанием их приоритета |

## Дополнительные задания

1. При обновлении информации на клиенте по таймеру данные передается только в том случае, если что-либо изменилось с момента прошлого обновления.
2. Обновление информации на клиенте происходит по инициативе сервера как реакция на изменения на сервере.

Содержание

[1 Задание на курсовую работу 2](#_Toc90763258)

[1 Задание 5](#_Toc90763259)

# Введение

В связи с быстрым развитием операционных систем и необходимостью использовать их функций, непосредственно в разработке программного обеспечения, возникает необходимость изучения данной темы.

Помимо использования внутренних функций операционных систем, важной частью является использование клиент-серверных функций, которые помогает реализовать многие задачи.

Поэтому целью данной курсовой работы является закрепление и углубление теоретических знаний в области современных операционных систем, приобретение практических навыков разработки клиент-серверных приложений, использующих стандартные механизмы межпроцессного взаимодействия.

# 2 Основная часть

## 2.1 Постановка задачи

В соответствии с заданием необходимо разработать 3 отдельных приложения, одним из которых является клиент, а остальные – сервер.

Клиент должен устанавливать соединение с сервером по средствам отображения файлов и отправлять запросы на получение данных: количество клавиш мыши и наличие колеса прокрутки для первого сервера и размер файла подкачки в байтах и количество свободных байтов файла подкачки для второго сервера. При получении ответа от сервера – вывести данные.

Первый сервер должен ожидать запрос от клиента и при получении данного запроса определить количество клавиш мыши и наличие колеса прокрутки и отправлять обратно данные клиенту.

Второй сервер должен ожидать запрос от клиента и при получении данного запроса получить информацию о памяти, а конкретно размер файла подкачки в байтах и количество свободных байтов файла подкачки и отправлять обратно данные клиенту.

Приложения должны бать разработаны по технологии «клиент-сервер». Метод коммуникации между процессами – отображение файлов.

## 2.2 Краткие теоретические основы разрабатываемой темы

Модели клиент-сервер - это технология взаимодействия в информационной сети. Сервер обладает правом управления тем или иным ресурсом, а клиент - пользования им. Понятия "клиент" и "сервер" описывают лишь распределение ролей при выполнении конкретной задачи, но не какие-либо отдельные виды компьютеров или программ. На одном и том же компьютере могут одновременно работать программы, выполняющие как клиентские, так и серверные функции.

Каждый конкретный сервер определяется видом того ресурса, которым он владеет. Этот принцип распространяется и на взаимодействие программ. Программа, выполняющая предоставление соответствующего набора услуг, рассматривается в качестве сервера, а программы, пользующиеся этими услугами, принято называть клиентами. Программы имеют распределенный характер, т.е. одна часть функций прикладной программы реализуется в программе-клиенте, а другая - в программе-сервере, а для их взаимодействия определяется некоторый протокол. Программа клиент взаимодействует с сервером, используя определенный протокол, устанавливающий правила обмена данными. Клиент может запрашивать с сервера какие-либо документы, манипулировать данными непосредственно на сервере, запускать на нем новые программы и т. п. Полученные от сервера данные клиент может предоставлять пользователю или формировать на их основе новый запрос. Клиент и сервер могут работать как на одном и том же компьютере, так и на разных. Во втором случае для обмена информацией между ними используется сетевое соединение.

Стандартные клиент-серверные приложения по своей архитектуре являются двухуровневыми. Компоненты, отвечающие за презентационный сервис и прикладную логику, размещаются на клиентской машине и обращаются к общему источнику данных по сети. В таких приложениях пользовательский интерфейс и прикладная логика образуют один уровень на клиентском компьютере, а сервисы данных предоставляются отдельными приложениями. Двухуровневые приложения хорошо работают в масштабах подразделения при наличии умеренного количества пользователей (до 100) и быстродействующей защищенной сети. Отладку приложений клиент-сервер возможно производить на локальном компьютере.

Клиент-серверная архитектура имеет следующие преимущества и недостатки.

Преимущества

* Отсутствие дублирования кода программы-сервера программами-клиентами.
* Так как все вычисления выполняются на сервере, то требования к компьютерам, на которых установлен клиент, снижаются.
* Все данные хранятся на сервере, который, как правило, защищён гораздо лучше большинства клиентов. На сервере проще организовать контроль полномочий, чтобы разрешать доступ к данным только клиентам с соответствующими правами доступа.

Недостатки

* Неработоспособность сервера может сделать неработоспособной всю вычислительную сеть. Неработоспособным сервером следует считать сервер, производительности которого не хватает на обслуживание всех клиентов, а также сервер, находящийся на ремонте, профилактике и т. п.
* Поддержка работы данной системы требует отдельного специалиста — системного администратора.
* Высокая стоимость оборудования.

Синхронизация может быть выполнена средствами прикладной программы и средствами ОС Более универсальными и эффективными являются средства ОС. В ОС Windows предусмотрены, в частности, cледующие основные объекты синхронизации:

* события;
* мьютексы;
* семафоры.

Объектами синхронизации называют специальные системные объекты, которые видны всем потокам, в том числе и тем, которые принадлежат разным процессам. Объекты могут находиться в двух состояниях: сигнальном и занятом. Смысл «сигнальное состояние» зависит от контекста. Например, поток переходит в сигнальное состояние, когда он заканчивается. Файл переходит в сигнальное состояние, когда операция ввода-вывода для этого файла завершается. Семафор находится в сигнальном состоянии, если его счетчик больше нуля, мьютекс - если он освобожден. Событие находится в сигнальном состоянии, если его флаг установлен (событие произошло).

Объекты синхронизации используются сходным образом. Для создания объекта вызывается функция создания объекта (например. CreateMutex для создания мьютекса). Если объект синхронизации будет использоваться только текущим процессом и его потомками, то при создании можно не указывать имя объекта. Открытие существующего объекта для использования другими процессами выполняется функцией открытия (например, ОpenMutex). Освобождение объекта осуществляется функцией типа ReleaseMutex. Для других объектов синхронизации используются аналогичные функции.

Следить за состоянием объектов синхронизации можно с помощью специальных системных вызовов. Чаще всего для этого используют функцию WaitForSingleObject (). Функция проверяет, находится та соответствующий объект в сигнальном состоянии. Когда функция при вызове устанавливает, что объект имеет сигнальное состояние, то она тут же возвращает управление вызвавшей программе и сообщает ей об изменении состояния объекта

Разработанное ПО предполагает использование в учебных целях для осваивания и закрепления навыков разработки клиент-серверных приложений, использующих стандартные механизмы межпроцессного взаимодействия.

## Разработка и описание алгоритма работы приложении

Для реализации данного задания необходимо использовать локальные функции, которые предоставляют множество возможностей взаимодействия с системной информацией необходимых для реализации серверной части.

В клиент-приложении нужно обязательно указать IP адрес устройства, на котором запущены серверы, иначе не будет установлено соединение между устройствами и не будет осуществлена передача данных. Далее выбирается сервер, который будет передавать данные. Клиент отправляет запрос с помощью сокетов и возвращает на экран данные от выбранного сервера. Также реализован запуск серверов в единственном экземпляре с помощью мьютексов, т.е. сервер может быть только один, а клиентов, обращающихся к серверу – много.

Для того чтобы наглядно рассмотреть процесс взаимодействия клиента и серверов, составим диаграмму компонентов и определим их роли, а также укажем связь между ними.

На рисунке 1 изображена диаграмма компонентов:

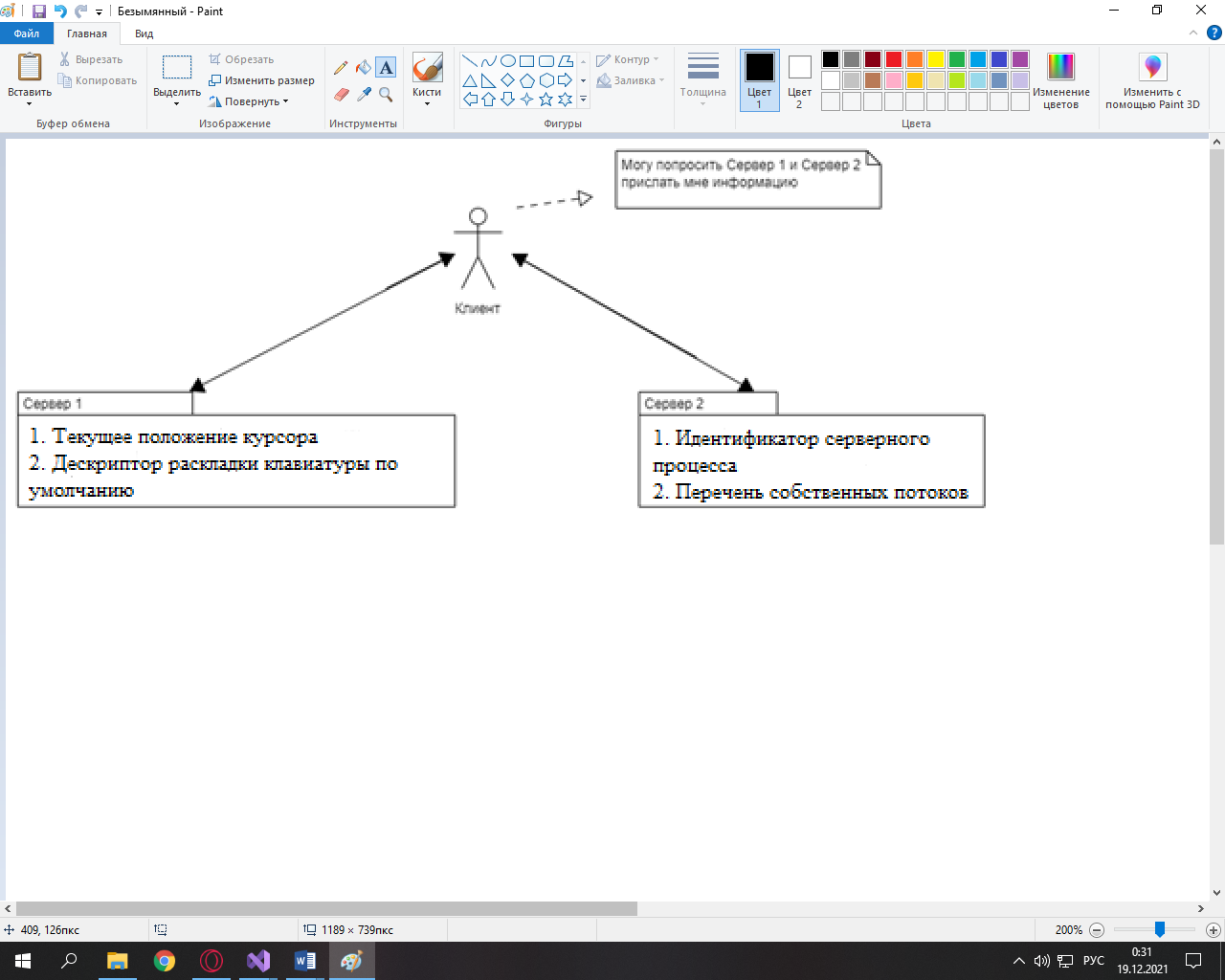


Рисунок 1 – Диаграмма компонентов

На полученной диаграмме наглядно видно, как взаимодействуют между собой клиент и сервер (в обе стороны, в форме «запрос-ответ»), сервер и сервер (не взаимодействуют), а также с помощью заметок можно понять задачи, которые ставятся перед компонентами.

## 2.4 Описание структуры программы

Функция Cursor.Position() позволяет получить положение курсора. С помощью переменных X и Y мы считываем положение курсора и выводим их в наш list\_box. Функция InputLanguage.LayoutName() позволяет нам получить текущую раскладку клавиатуры.

Функция GetProcessesByName() позволяет получить идентификатор указанного процесса. Функция GetCurrentProcess().Id позволяет нам получить процесс и связывает его с активным в данный момент процессом. Функция Process.Thread позволяет получить множество потоков, выполняющихся в связанном процессе. Функция Thread.Id() позволяет получить id потока, а функция PriorityLevel() позволяет получить уровень приоритета потока.

Первый сервер выполняет создание области отображения файлов и событий, необходимых для синхронизации работы программ. Затем он переходит к ожиданию подключения клиента, который отправляет серверу запрос на выполнение. Получив данный запрос сервер определяет расположение курсора мыши и раскладку клавиатуры, затем передаем данные в лист, а после чего в сокет и отправляет клиенту событие на получение данных.

Второй сервер схож принципом работы с первым сервером. После получения запроса от клиента, второй сервер отправляет клиенту событие для получения данных.

## 2.5 Результаты применения программы

На рисунке 2 будет представлен результат работы первого сервера.

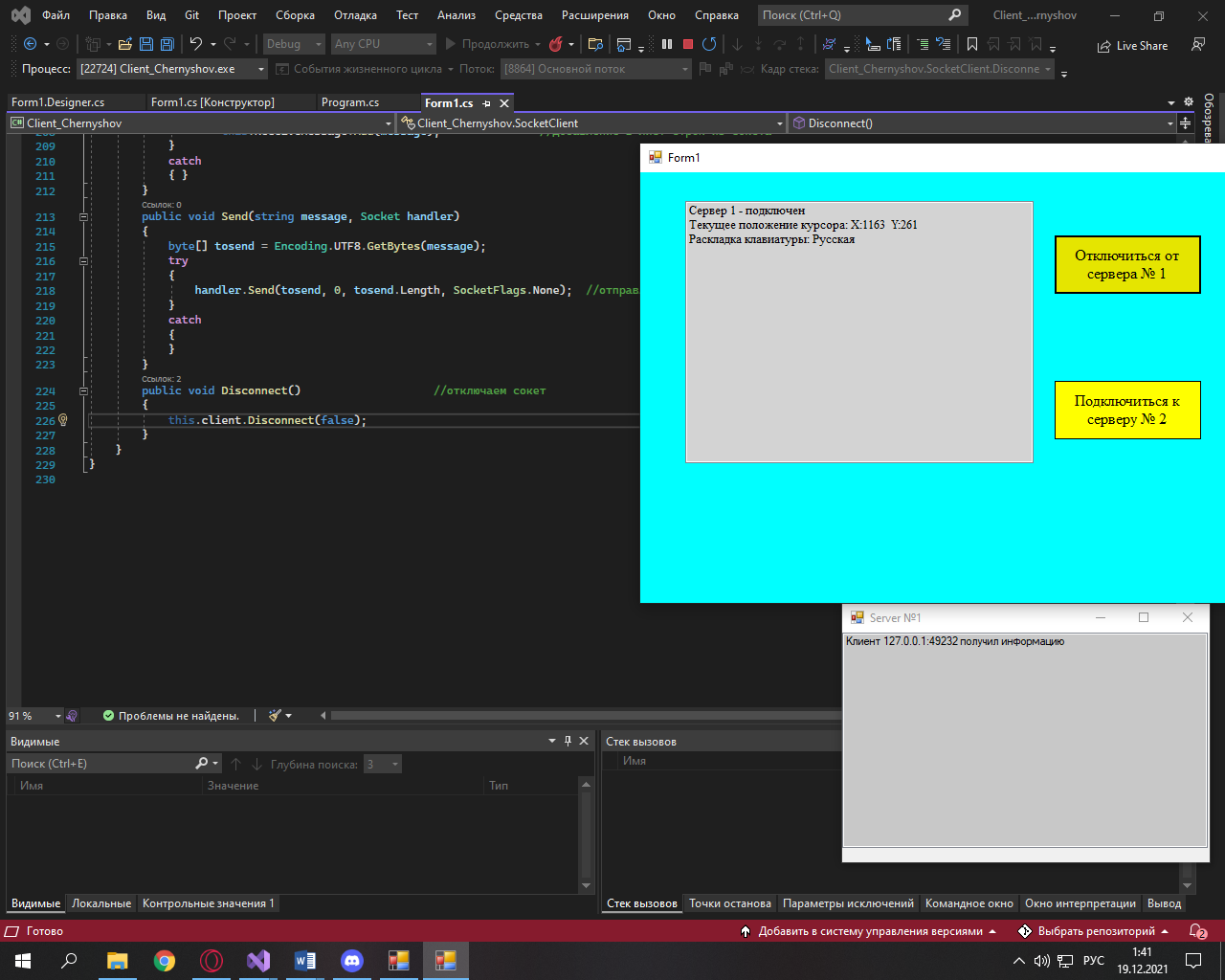
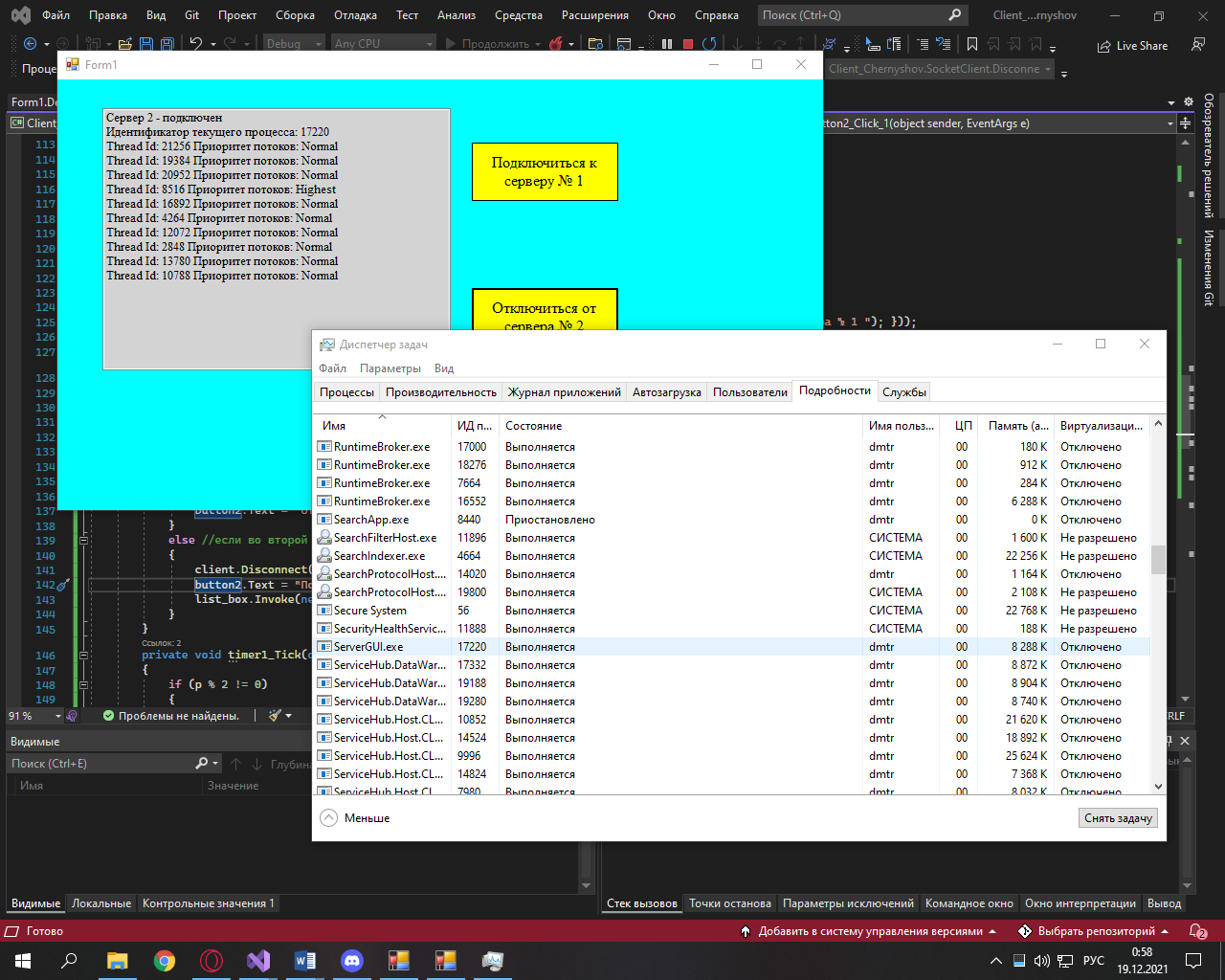


Рисунок 2 – Результат работы первого сервера

На рисунке 2 видно, что текущая раскладка клавиатуры «русская», соответственно результат работы сервера будет таким же.

На рисунке 3 будет представлен результат работы первого сервера.



На рисунке 3 – Результат работы второго сервера

На рисунке 3 можно заметить, что идентификатор процесса в диспетчере задач равен 17220, соответственно сервер выводит id этого процесса.

# ЗАКЛЮЧЕНИЕ

В ходе курсовой работы было разработано клиент-серверное приложение, состоящее из двух серверных частей и одной клиентской. Так же были закреплены теоретические знания в области современных операционных систем, а также приобретены навыки разработки клиент-серверных приложений, использующих стандартные механизмы межпроцессного взаимодействия.

# Список используемых источников

1. Википедия [Электронный ресурс]. – Режим доступа: https://ru.wikipedia.org/wiki/Клиент\_—\_сервер. – Дата доступа: 13.12.2020.
2. Документация C# [Электронный ресурс] - https://docs.microsoft.com/ru-ru/dotnet/csharp/.
3. С.В. Назаров, А.И. Широков. «Современные операционные системы». — М., Бином, 2013.
4. В.Г. Олифер, Н.А. Олифер. «Сетевые операционные системы». — СПб., Питер, 2009.
5. Э. Таненбаум, Х. Бос. «Современные операционные системы. 4-изд». — СПб., Питер, 2015.
6. Вдовикина Н.В., Машечкин И.В., Терехин А.Н., Томилин А.Н. «Операционные системы: взаимодействие процессов: учебно-методическое пособие». — Издательский отдел факультета ВМиК МГУ, 2008.

# Приложение 1

## Программный код клиента

using System;

using System.Collections.Generic;

using System.Text;

using System.Windows.Forms;

using System.Net.Sockets;

using System.Threading;

using System.Net;

namespace Client\_Chernyshov

{

public partial class Form1 : Form

{

string ReceiveMessageA;

string ReceiveMessageB;

int k, p;

int port;

static SocketClient client; //содержит ip и port клиента

public Form1()

{

InitializeComponent();

}

private void connection() //подключение к серверу

{

client = new SocketClient("127.0.0.1", this.port); //создаем сокет под нашего клиента

bool smt = client.Connect(); //подключаемся к серверу

if (smt) //если подключились

{

if (port == 8000) //порт первого сервера

list\_box.Invoke(new MethodInvoker(delegate { list\_box.Items.Add("Сервер 1 - подключен"); }));

else

list\_box.Invoke(new MethodInvoker(delegate { list\_box.Items.Add("Сервер 2 - подключен"); }));

Status();

}

else

{

if (port == 8000)

list\_box.Invoke(new MethodInvoker(delegate { list\_box.Items.Add("Ошибка подключения - Сервер 1"); }));

else

list\_box.Invoke(new MethodInvoker(delegate { list\_box.Items.Add("Ошибка подключения - Сервер 2"); }));

}

}

private void Status()

{

for (; ; ) //бесконечный цикл

{

client.Receive(); //выводим информацию из сокета

try

{

if (client.ReceiveMessage[client.ReceiveMessage.Count - 1] == "[end]") break; //закончился ли поток данных

}

catch

{ }

}

ReceiveMessageA = client.ReceiveMessage[0];

if (port == 8000)

{

for (int i = 0; i < client.ReceiveMessage.Count - 1;)

{

if (client.ReceiveMessage[i] != "[end]")

list\_box.Invoke(new MethodInvoker(delegate { list\_box.Items.Add(client.ReceiveMessage[i]); })); //выводит в лист бокс

client.ReceiveMessage.RemoveAt(i); //удаляем сообщение которое вывели

}

}

else

{

for (int i = 0; i < client.ReceiveMessage.Count - 1;)

{

if (client.ReceiveMessage[i] != "[end]")

list\_box.Invoke(new MethodInvoker(delegate { list\_box.Items.Add(client.ReceiveMessage[i]); })); //выводит в лист бокс

client.ReceiveMessage.RemoveAt(i); //удаляем сообщение которое вывели

}

}

ReceiveMessageB = ReceiveMessageA;

}

private void Status2()

{

for (; ; ) //бесконечный цикл

{

client.Receive(); //выводим информацию из сокета

try

{

if (client.ReceiveMessage[client.ReceiveMessage.Count - 1] == "[end]") break; //закончился ли поток данных

}

catch

{ }

}

ReceiveMessageA = client.ReceiveMessage[0];//запись сообщения в переменную

if (ReceiveMessageA != ReceiveMessageB)//сравнение с промежуточной переменной

{

for (int i = 0; i < client.ReceiveMessage.Count - 2;)

{

if (client.ReceiveMessage[i] != "[end]")

list\_box.Invoke(new MethodInvoker(delegate { list\_box.Items.Add(client.ReceiveMessage[i]); })); //выводит в лист бокс

client.ReceiveMessage.RemoveAt(i); //удаляем сообщение которое вывели

}

}

else

{

list\_box.Invoke(new MethodInvoker(delegate { list\_box.Items.Add("Информация не обновилась"); })); //выводит в лист бокс

}

ReceiveMessageB = ReceiveMessageA;//переопределение промежуточной переменной

ReceiveMessageA = "";

}

private void button1\_Click\_1(object sender, EventArgs e)

{

p++; //счетчик нажатия на ктопку

port = 8000; //определяем порт для подключения

if (p % 2 != 0)

{

Thread connect = new Thread(connection); //создаем поток для void connection

connect.Start(); //запуск потока

connect.IsBackground = true; //переводим поток в фоновый режим

timer1.Start();

button1.Text = "Отключиться от сервера № 1";

}

else //если во второй отключение

{

client.Disconnect();

button1.Text = "Подключиться к серверу № 1";

list\_box.Invoke(new MethodInvoker(delegate { list\_box.Items.Add("Произошло отключение от сервера № 1 "); }));

}

}

private void button2\_Click\_1(object sender, EventArgs e)

{

k++; //счетчик нажатия на ктопку

port = 9000; //определяем порт для подключения

if ( k % 2 != 0)

{

Thread connect = new Thread(connection); //создаем поток для void connection

connect.Start(); //запуск потока

connect.IsBackground = true; //переводим поток в фоновый режим

button2.Text = "Отключиться от сервера № 2";

}

else //если во второй отключение

{

client.Disconnect();

button2.Text = "Подключиться к серверу № 2";

list\_box.Invoke(new MethodInvoker(delegate { list\_box.Items.Add("Произошло отключение от сервера № 2 "); }));

}

}

private void timer1\_Tick(object sender, EventArgs e)

{

if (p % 2 != 0) //если кнопка подключения нажата 1 раз, каждый тик (=1000) запрашиваем информацию от сервера №2

{

Thread connect = new Thread(reconnection2);

connect.Start();

connect.IsBackground = true;

}

else

timer1.Stop(); //если нажали на кнопку подключения в 2 раз остановка связи с сервером

}

private void Form1\_Load(object sender, EventArgs e)

{

timer1 = new System.Windows.Forms.Timer();

timer1.Interval = 5000;

timer1.Tick += new EventHandler(timer1\_Tick);

}

private void reconnection2()//переподключение к 2 серверу для таймера (1 доп задание)

{

client = new SocketClient("127.0.0.1", this.port); //новый клиент

if (client.Connect())

{

//если законнектились то ок

Status2();

}

}

}

class SocketClient

{

public Socket client;

private IPEndPoint ip;

public List<string> SendMessage = new List<string>();

public List<string> ReceiveMessage = new List<string>();

public SocketClient(string ip, Int32 port)

{

this.ip = new IPEndPoint(IPAddress.Parse(ip), port); //используется для связывания сокета и локального адреса

this.client = new Socket(this.ip.AddressFamily, SocketType.Stream, ProtocolType.Tcp);

}

public bool Connect()

{

try

{

this.client.Connect(this.ip); //проверяем соединение

return true;

}

catch

{

return false;

}

}

public void Receive()

{

string message = String.Empty;

byte[] GetBytes = new byte[1024]; //создаем буфер

try

{

int b = client.Receive(GetBytes); //записывает данные из сокета в буфер

message = Encoding.UTF8.GetString(GetBytes, 0, b); //форматируем в стринг

if (message != "")

this.ReceiveMessage.Add(message); //добавление в лист строк из сокета

}

catch

{ }

}

public void Send(string message, Socket handler)

{

byte[] tosend = Encoding.UTF8.GetBytes(message);

try

{

handler.Send(tosend, 0, tosend.Length, SocketFlags.None); //отправляем сообщение на сервер

}

catch

{

}

}

public void Disconnect() //отключаем сокет

{

this.client.Disconnect(false);

}

}

}

## Программный код первого сервера

using System;

using System.Collections.Generic;

using System.Drawing;

using System.Text;

using System.Windows.Forms;

using System.Net.Sockets;

using System.Threading;

using System.Net;

using System.Globalization;

namespace server1

{

public partial class Form1 : Form

{

SocketServer server; //сокет нашего сервера

public Form1()

{

InitializeComponent();

server = new SocketServer("127.0.0.1", 8000);

server.Start();

Thread list = new Thread(List);

list.IsBackground = true;

list.Start();

}

private void List ()

{

for (; ;Thread.Sleep(200) )

if (server.ClientList.Count != List\_box.Items.Count)

{

List\_box.Invoke(new MethodInvoker(delegate { List\_box.Items.Clear();}));

for (int i = 0; i < server.ClientList.Count; i++)

List\_box.Invoke(new MethodInvoker(delegate { List\_box.Items.Add(server.ClientList[i]); }));

}

}

private void Form1\_Closed(object sender, System.EventArgs e)

{

server.Dispose();

this.Dispose();

Application.Exit();

}

}

class SocketServer

{

bool open = true;

public Socket server;

private IPEndPoint ip;

private List<Thread> thread\_list;

private int max\_conn;

public List<string> messageSend = new List<string>();

public List<string> ClientList = new List<string>();

public SocketServer(string ip, Int32 port)

{

this.max\_conn = 2;

this.thread\_list = new List<Thread>();

this.ip = new IPEndPoint(IPAddress.Parse(ip), port);

this.server = new Socket(this.ip.AddressFamily, SocketType.Stream, ProtocolType.Tcp);

this.server.Bind(this.ip);

this.server.Listen(this.max\_conn);

}

public void Start()

{

for (int i = 0; i < this.max\_conn; i++) //создает кол-во потоков для указанного кол-ва пользователей

{

Thread th = new Thread(Listening); //создается поток для метода listening

th.Start();

thread\_list.Add(th); //добавляет потоки в list

}

}

public void Dispose() //нужен для завершения потоков из list

{

foreach (Thread th in thread\_list)

{

th.Interrupt();

}

open = false;

server.Close();

}

private void Listening()

{

while (open) //пока сокет открыт

{

try

{

using (Socket client = this.server.Accept()) //новый сокет для клиента

{

client.Blocking = true; //блокирует сокет

if (client.Connected) //подключились ли мы к сокету?

{

this.ClientList.Add("Клиент " + client.RemoteEndPoint.ToString() + " получил информацию"); //записываем пользователя для которого создался сокет

List<string> message = new List<string>();

int CursorX = Cursor.Position.X;

int CursorY = Cursor.Position.Y;

InputLanguage myCurrentLanguage = InputLanguage.CurrentInputLanguage;

message.Add("Текущее положение курсора: " + "X:" + CursorX.ToString() + " " + " Y:" + CursorY.ToString());

message.Add("Раскладка клавиатуры: " + myCurrentLanguage.LayoutName);

message.Add("[end]");

SendAll(client, message); //выгружаем его в сокет

}

}

}

catch (Exception)

{

}

}

}

public void SendAll(Socket handler, List<string> message) //метод для отправки данных в сокет из листа

{

for (int i = 0; i < message.Count;)

{

Send(message[i], handler); //применяем send, который отправляет наши данные в сокет

message.RemoveAt(i);

Thread.Sleep(5);

}

}

public void Send(string message, Socket handler)

{

byte[] tosend = Encoding.UTF8.GetBytes(message);

try

{

handler.Send(tosend, 0, tosend.Length, SocketFlags.None);

}

catch

{

}

}

}

}

## Программный код второго сервера

using System;

using System.Collections.Generic;

using System.ComponentModel;

using System.Data;

using System.Drawing;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

using System.Windows.Forms;

using System.Net.Sockets;

using System.Threading;

using System.Net;

using System.Diagnostics;

namespace server2

{

public partial class Form2 : Form

{

SocketServer server; //сокет нашего сервера

public Form2()

{

InitializeComponent();

server = new SocketServer("127.0.0.1", 9000);

server.Start();

Thread list = new Thread(List);

list.IsBackground = true;

list.Start();

}

private void List()

{

for (; ; Thread.Sleep(200))

if (server.ClientList.Count != List\_box.Items.Count)

{

List\_box.Invoke(new MethodInvoker(delegate { List\_box.Items.Clear(); }));

for (int i = 0; i < server.ClientList.Count; i++)

List\_box.Invoke(new MethodInvoker(delegate { List\_box.Items.Add(server.ClientList[i]); }));

}

}

private void Form2\_Closed(object sender, System.EventArgs e)

{

server.Dispose();

this.Dispose();

Application.Exit();

}

}

class SocketServer

{

bool open = true;

public Socket server;

private IPEndPoint ip;

private List<Thread> thread\_list;

private int max\_conn;

public List<string> messageSend = new List<string>();

public List<string> ClientList = new List<string>();

public SocketServer(string ip, Int32 port)

{

this.max\_conn = 2;

this.thread\_list = new List<Thread>();

this.ip = new IPEndPoint(IPAddress.Parse(ip), port);

this.server = new Socket(this.ip.AddressFamily, SocketType.Stream, ProtocolType.Tcp);

this.server.Bind(this.ip);

this.server.Listen(this.max\_conn);

}

public void Start()

{

for (int i = 0; i < this.max\_conn; i++) //создает кол-во потоков для указанного кол-ва пользователей

{

Thread th = new Thread(Listening); //создается поток для метода listening

th.Start();

thread\_list.Add(th); //добавляет потоки в list

}

}

public void Dispose() //нужен для завершения потоков из list

{

foreach (Thread th in thread\_list)

{

th.Interrupt();

}

open = false;

server.Close();

}

private void Listening()

{

while (open) //пока сокет открыт

{

try

{

using (Socket client = this.server.Accept()) //новый сокет для клиента

{

client.Blocking = true; //блокирует сокет

if (client.Connected) //подключились ли мы к сокету?

{

this.ClientList.Add("Клиент " + client.RemoteEndPoint.ToString() + " получил информацию"); //записываем пользователя для которого создался сокет

List<string> message = new List<string>();

Process proc = Process.GetProcessesByName("ServerGUI")[0];

int ident = Process.GetCurrentProcess().Id;

ProcessThreadCollection processThreads = proc.Threads;

message.Add("Идентификатор текущего процесса: " + ident);

foreach (ProcessThread thread in processThreads)

{

message.Add($"Thread Id: {thread.Id} Приоритет потоков: {thread.PriorityLevel}");

}

message.Add("[end]");

SendAll(client, message); //выгружаем его в сокет

}

}

}

catch (Exception)

{

}

}

}

public void SendAll(Socket handler, List<string> message) //метод для отправки данных в сокет из листа

{

for (int i = 0; i < message.Count;)

{

Send(message[i], handler); //применяем send, который отправляет наши данные в сокет

message.RemoveAt(i);

Thread.Sleep(5);

}

}

public void Send(string message, Socket handler)

{

byte[] tosend = Encoding.UTF8.GetBytes(message);

try

{

handler.Send(tosend, 0, tosend.Length, SocketFlags.None);

}

catch

{

}

}

}

}