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»**

**(МТУСИ)**

Кафедра

Системное программирование

Курсовая работа

Научный руководитель:

Алексанян Д.А.

Выполнил студент БФИ1901

Леонов Н.Н.

Москва 2021

**ЗАДАНИЕ НА КУРСОВУЮ РАБОТУ**

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

|  |  |  |
| --- | --- | --- |
| 3, 21 | * код последней ошибки | * процент используемой физической памяти |
| * текущее положение курсора | * процент используемой виртуальной памяти |

Оглавление

**ВВЕДЕНИЕ4**

**ПОСТАНОВКА ЗАДАЧИ6**

**1.** **Краткая** **теория7**

1.1 Архитектура "клиент-сервер"7

1.2. Модели взаимодействия клиент-сервер7

1.3. Преимущества архитектуры клиент-сервер9

**2.** **Разработка ПО11**

2.1 Структура серверов11

2.2 Структура клиента12

**ЗАКЛЮЧЕНИЕ13**

**СПИСОК ЛИТЕРАТУРЫ14**

**ПРИЛОЖЕНИЕ А15**

**ВВЕДЕНИЕ**

В наше время практически во всех сферах жизни человеческого общества применяются те или иные системы обработки данных. Это могут быть банковские системы, промышленные программные комплексы, средства автоматизации торговли и многие другие приложения. Постоянное расширение областей применения вычислительной техники и всевозрастающая сложность решаемых задач требуют совершенствования методов создания современных информационных систем.  
С начала 90-х годов XX века известна такая технология реализации программных приложений как архитектура "клиент-сервер". Ее основная особенность состоит в том, что приложение делится на два уровня - представление данных (клиент) и хранение данных (сервер БД). Обработка информации происходит на клиенте, на сервер посылаются запросы и обрабатываются полученные в ответ на них данные.  
С появлением мульти-платформенных технологий программирования и телекоммуникационных систем возникла возможность внедрения архитектуры клиент-сервер в портативные устройства.  
В рамках данной работы будет рассмотрены преимущества этой архитектуры с использованием технологии .Net на языке C#.

Архитектура клиент-сервер предполагает сервер баз данных, представляющий собой приложение, осуществляющее комплекс действий по управлению данными - выполнение запросов, хранение и резервное копирование данных, отслеживание ссылочной целостности, проверку прав и привилегий пользователей, ведение журнала транзакций.

Одним из важнейших преимуществ клиент-серверных информационных систем по сравнению с их аналогами является снижение сетевого трафика при выполнении запросов.

Вторым преимуществом архитектуры клиент-сервер является возможность хранения бизнес-правил на сервере, что позволяет избежать дублирования кода в различных приложениях, использующих общую базу данных. Кроме того, в этом случае любое редактирование данных, в том числе и редактирование нештатными средствами, может быть произведено только в рамках этих правил.

Помимо перечисленных возможностей, современные серверные СУБД обладают широкими возможностями управления пользовательскими привилегиями и правами доступа к различным объектам базы данных, резервного копирования и архивации данных, а нередко и оптимизации выполнения запросов. Они также, как правило, предоставляют возможность параллельной обработки данных, особенно в случае использования многопроцессорных компьютеров в качестве сервера баз данных.

Итак, клиент-серверная информационная система состоит в простейшем случае из двух основных компонентов:

• сервер баз данных, управляющий хранением данных, доступом и защитой, резервным копированием, отслеживающий целостность данных в соответствии с бизнес-правилами и, самое главное, выполняющий запросы клиента;

• клиент, предоставляющий интерфейс пользователя, выполняющий логику приложения, проверяющий допустимость данных, посылающий запросы к серверу и получающий ответы от него;

**ПОСТАНОВКА ЗАДАЧИ**

**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.

**Задачи** курсовой работы:

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.

**1. Краткая теория**

1.1 Архитектура "клиент-сервер"

Как правило, компьютеры и программы, входящие в состав информационной системы, не являются равноправными. Некоторые из них владеют ресурсами (файловая система, процессор, принтер, база данных и т.д.), другие имеют возможность обращаться к этим ресурсам. Компьютер (или программу), управляющий ресурсом, называют сервером этого ресурса (файл-сервер, сервер базы данных, вычислительный сервер...). Клиент и сервер какого-либо ресурса могут находиться как в рамках одной вычислительной системы, так и на различных компьютерах, связанных сетью.

Основной принцип технологии "клиент-сервер" заключается в разделении функций приложения на три группы:

• ввод и отображение данных (взаимодействие с пользователем);

• прикладные функции, характерные для данной предметной области;

• функции управления ресурсами (файловой системой, базой данных и т.д.);

Поэтому, в любом приложении выделяются следующие компоненты:

• компонент представления данных;

• прикладной компонент;

• компонент управления ресурсом;

Связь между компонентами осуществляется по определенным правилам, которые называют "протокол взаимодействия"

1.2. Модели взаимодействия клиент-сервер

Исторически первой появилась модель распределенного представления данных, которая реализовывалась на универсальной ЭВМ с подключенными к ней неинтеллектуальными терминалами. Управление данными и взаимодействие с пользователем при этом объединялись в одной программе, на терминал передавалась только "картинка", сформированная на центральном компьютере.

Затем, с появлением персональных компьютеров (ПК) и локальных сетей, были реализованы модели доступа к удаленной базе данных. Некоторое время базовой для сетей ПК была архитектура файлового сервера. При этом один из компьютеров является файловым сервером, на клиентах выполняются приложения, в которых совмещены компонент представления и прикладной компонент (СУБД и прикладная программа). Протокол обмена при этом представляет набор низкоуровневых вызовов операций файловой системы. Такая архитектура, реализуемая, как правило, имеет очевидные недостатки - высокий сетевой трафик и отсутствие унифицированного доступа к ресурсам.

С появлением первых специализированных серверов баз данных появилась возможность другой реализации модели доступа к удаленной базе данных. В этом случае ядро СУБД функционирует на сервере, протокол обмена обеспечивается с помощью языка SQL. Такой подход по сравнению с файловым сервером ведет к уменьшению загрузки сети и унификации интерфейса "клиент-сервер". Однако, сетевой трафик остается достаточно высоким, кроме того, по- прежнему невозможно удовлетворительное администрирование приложений, поскольку в одной программе совмещаются различные функции.

Позже была разработана концепция активного сервера, который использовал механизм хранимых процедур. Это позволило часть прикладного компонента перенести на сервер (модель распределенного приложения). Процедуры хранятся в словаре базы данных, разделяются между несколькими клиентами и выполняются на том же компьютере, что и SQL-сервер. Преимущества такого подхода: возможно централизованное администрирование прикладных функций, значительно снижается сетевой трафик (т.к. передаются не SQL-запросы, а вызовы хранимых процедур). Недостаток - ограниченность средств разработки хранимых процедур по сравнению с языками общего назначения.

На практике сейчас обычно используется смешанный подход: простейшие прикладные функции выполняются хранимыми процедурами на сервере; более сложные реализуются на клиенте непосредственно в прикладной программе.

Сейчас ряд поставщиков коммерческих СУБД объявило о планах реализации механизмов выполнения хранимых процедур с использованием языка Java. Это соответствует концепции "тонкого клиента", функцией которого остается только отображение данных (модель удаленного представления данных).

В последнее время также наблюдается тенденция ко все большему использованию модели распределенного приложения. Характерной чертой таких приложений является логическое разделение приложения на две и более частей, каждая из которых может выполняться на отдельном компьютере. Выделенные части приложения взаимодействуют друг с другом, обмениваясь сообщениями в заранее согласованном формате. В этом случае двухзвеньевая архитектура клиент-сервер становится трехзвеньевой, а в некоторых случаях, она может включать и больше звеньев (рис 1).

1.3. Преимущества архитектуры клиент-сервер

Силой, способствующей развитию систем с архитектурой клиент-сервер, служит то, что подключение к недорогим серверам недорогих ПК позволит получить оптимальное соотношение цены и производительности. Чаще всего это предположение верно.

Используя множество небольших компьютеров, разработчики систем клиент-сервер могут эмулировать вычислительную мощность больших ЭВМ, распределяя прикладную задачу по различным микрокомпьютерам и серверам.

Каждый из них берет на себя свою часть вычислительной нагрузки, используя информацию совместно с другими процессорами сети. Суть идеи в том, что чтобы повысить мощность системы, не наращивая производительность одного компьютера, а суммируя средства многих.

Быстродействие - основной фактор целесообразности разработки систем для архитектуры клиент-сервер. Технологи серверов баз данных становится проще в использовании и сочетаются в одних системах со средствами RAD. Таким образом, ожидается, что с помощью этой скоростной и практически защищенной платформы сократится время, необходимое для подготовки и передачи прикладной программы пользователю.

Привлекательность ГИП (графического интерфейса пользователя) - еще один фактор, склоняющий пользователей к применению архитектуры клиент-сервер . К преимуществам прикладных программ клиент-сервер относятся также простота использования и удобство для пользователей, работающих с такими ГИП, как Windows 3.1, Windows 95, Windows NT и OS/2 Warp. Пользователи чувствуют себя более комфортно и работают продуктивнее с такими прикладными программами, такими как текстовые процессоры, средства просмотра Internet (браузеры) и электронные таблицы, и подобными им .

Наконец, клиент-сервер - это технология, предоставляющая больше самостоятельности пользователям. Прикладные программы и данные находятся на небольших, более доступных системах, а не в недрах мини-ЭВМ и мощных вычислительных центров.

Наиболее простая форма архитектуры клиент-сервер - это разделение вычислительной нагрузки между двумя отдельными процессами: клиентом и сервером. Хотя и клиент, и сервер могут находиться на одном и том же компьютере, большинство систем этой архитектуры запускают клиентский процесс на одном компьютере, а процесс-сервер на другом, используя для обмена информацией сетевые связи. В этой модели один процесс может работать независимо от другого, выполнять определенные задания и разделять вычислительную нагрузку.

Обычно клиентом служит настольный ПК, выполняющий программное обеспечение конечного пользователя. Программное обеспечение (ПО) конечного пользователя (front-end software) - это любая прикладная программа или пакет, способные направлять запросы по сети серверу и обрабатывать получаемую в ответ информацию. Сервер, в свою очередь, получает запросы и предпринимает действия от имени клиента.

Промежуточное обеспечение (middleware) предоставляет общий интерфейс для ПО конечного пользователя и сервера, проникающий сквозь слои ГИП, ОС, вычислительной сети и собственных драйверов базы данных с помощью общих вызовов. Для завершения операции сервер базы данных выполняет запрос и передает клиенту затребованные данные для обработки их программой клиента.

**2. Разработка ПО**

Многозадачное приложение реализуется на основе технологии «клиент-сервер» средствами интегрированной среды разработки Microsoft Visual Studio на языке C#. Процессы 1 и 2 выступают как серверы приложений, третий процесс является их клиентом. Обмен данными между клиентами и серверами осуществляется при помощи сокетов.

И клиентский и серверные процессы оформляются как отдельные приложения, запускаются и завершают работу независимо друг от друга. Серверы реализованы многопоточными посредством возможности одновременного подключения нескольких клиентов. Серверные процессы по запросу получают системную информацию в соответствии с вариантом задания и передают ее клиенту. На стороне клиента необходимо реализуется интерфейс, позволяющий пользователю подключиться по выбору к первому или второму серверу, а также возможность раздельного получения и отображения данных по каждому из пунктов задания.

2.1 Структура серверов.

Приложения обоих серверов состоят из трех классов: SystemInfo (реализуется индивидуальное задание), ClientObject (получение и обработка запроса, создание и отправление ответа), Program (обработка клиентов по локальной сети).

В SystemInfo первого сервера получаем позицию курсора с использованием Windows.Forms через класс Cursor. Последнюю ошибку получаем с помощью System.Management через класс Marshal.

В SystemInfo второго сервера для получения данных о физ. и вирт памяти мы должны обратиться к WMI с запросом на языке WQL с использованием System.Management через класс ManagementObjectSearcher. Далее мы достаем из объекта необходимые данные и вычисляем процент использующейся вирт. и физ. памяти.

В ClientObject мы создаем TCP клиента (класс TcpClient, использующий сокет и работающий с ним по протоколу TCP). Далее мы по каналу получаем байты информации, и переводим их в строчное значение, после чего формируем ответ в обратной последовательности и отправляем по каналу связи.

В Program мы создаем TcpListener, куда принимаем несколько клиентов, обратившихся одновременно. Далее в цикле мы вытаскиваем одного клиента и выделяем для его обработки поток с помощью ThreadStart

2.2 Структура клиента.

Приложение клиента состоит из четырех классов: RequestHandler (обработка запроса с сервера), ConnectToServer(отправка запроса, получение ответа), Form1 (графический интерфейс для нашего пользователя), Program(запуск формы).

В RequestHandler создаётся конструктор, принимающий ответ с сервера. Он имеет вид «volOfRequest<Количество поступающих данных>$<данные>$<данные>». В зависимости от количества данных мы извлекаем необходимую подстроку и расформировываем данные по членам класса, далее на уменьшение количества запросов применяется рекурсия данной функции.

ConnectToServer является аналогом серверного ClientObject, разве что в обратном порядке.

В Form1 реализуется подключение к соответствующим серверам по нажатию соответствующих кнопок. Получается ответ с сервера, передается обработчику запросов и расформировывается по текстбоксам.

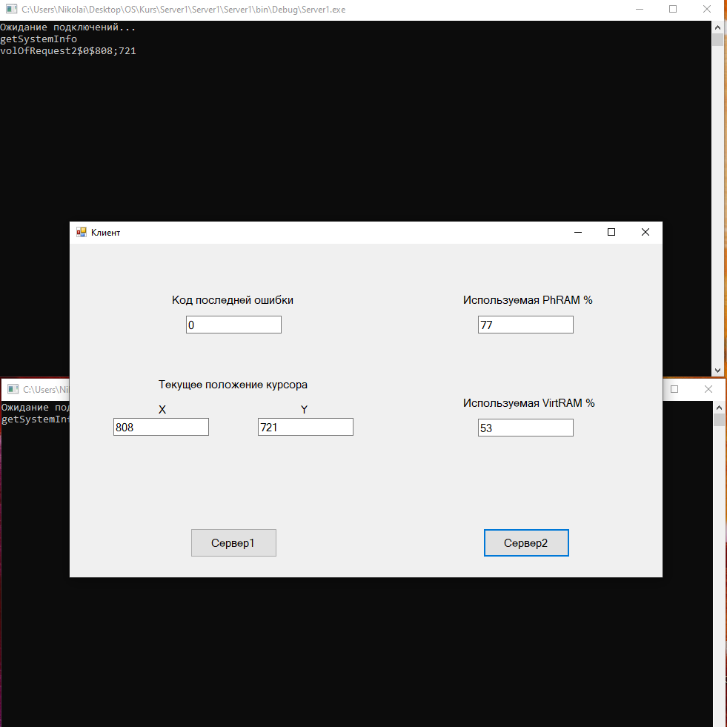


Рисунок 2.1 – Результат работа программы

**ЗАКЛЮЧЕНИЕ**

В течение работы над данной курсовой была изучена клиент-серверная архитектура, приобретены навыки написания программ межпроцессного взаимодействия с использованием технологии многопоточности. Реализовано индивидуальное задание, и результаты работа программы показывают, что приложения функционируют без ошибок.

**СПИСОК ЛИТЕРАТУРЫ**

1. <https://habr.com/ru/post/495698/>
2. Принципы **архитектуры**: • **Клиент**-**серверная** **архитектура** • Каждый слой (данные, представление и логика) не. зависит от остальных и не. ... Тузовский, А. Ф. Проектирование и разработка web-приложений : **учебное** **пособие** для академического бакалавриата / А. Ф. Тузовский.М. : Издательство Юрайт, 2018. — 218 с. — (Серия : Университеты России). — ISBN 978-5-534-00515-8

**ПРИЛОЖЕНИЕ А**

**Server1:**

SystemInfo:

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

using System.Windows.Forms;

using System.Drawing;

using System.Runtime.InteropServices;

using System.Management;

namespace Server1

{

internal class SystemInfo

{

private int cursorX;

private int cursorY;

private int lastErr;

public SystemInfo()

{

this.cursorX = Cursor.Position.X;

this.cursorY = Cursor.Position.Y;

this.lastErr = Marshal.GetLastWin32Error();

}

public string getCursorPosition()

{

return Convert.ToString(this.cursorX) + ";" + Convert.ToString(this.cursorY);

}

public string getlastError() { return Convert.ToString(this.lastErr);}

}

}

ClientObject:

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

using System.Net.Sockets;

namespace Server1

{

public class ClientObject

{

public TcpClient client;

public ClientObject(TcpClient tcpClient)

{

client = tcpClient;

}

public void Process()

{

NetworkStream stream = null;

try

{

stream = client.GetStream();

byte[] data = new byte[64]; // буфер для получаемых данных

if (true)

{

// получаем сообщение

StringBuilder builder = new StringBuilder();

int bytes = 0;

do

{

bytes = stream.Read(data, 0, data.Length);

builder.Append(Encoding.Unicode.GetString(data, 0, bytes));

}

while (stream.DataAvailable);

SystemInfo systemInfo = new SystemInfo();

string message = builder.ToString();

Console.WriteLine(message);

// отправляем обратно сообщение

message = "volOfRequest2"+"$"+systemInfo.getlastError() + "$"+systemInfo.getCursorPosition(); //+systemInfo.getlastError()+"$"

Console.WriteLine(message);

data = Encoding.Unicode.GetBytes(message);

stream.Write(data, 0, data.Length);

}

}

catch (Exception ex)

{

Console.WriteLine(ex.ToString());

}

finally

{

if (stream != null)

stream.Close();

if (client != null)

client.Close();

}

}

}

}

Program:

using System;

using System.Net;

using System.Net.Sockets;

using System.Threading;

namespace Server1

{

class Program

{

const int port = 8888;

static TcpListener listener;

static void Main(string[] args)

{

try

{

Exception exception = null;

listener = new TcpListener(IPAddress.Parse("127.0.0.1"), port);

listener.Start();

Console.WriteLine("Ожидание подключений...");

while (true)

{

TcpClient client = listener.AcceptTcpClient();

ClientObject clientObject = new ClientObject(client);

// создаем новый поток для обслуживания нового клиента

Thread clientThread = new Thread(new ThreadStart(clientObject.Process));

clientThread.Start();

}

}

catch (Exception ex)

{

Console.WriteLine(ex.Message);

}

finally

{

if (listener != null)

listener.Stop();

}

}

}

}

Server2:

SystemInfo:

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

using System.Management;

using System.Windows.Forms;

namespace Server2

{

internal class SystemInfo

{

ulong totalVisMemory;

ulong totalVirtMemory;

ulong freePhysMemory;

ulong freeVirtMemory;

ulong usingPhysRam;

ulong usingVirtRam;

public SystemInfo()

{

ManagementObjectSearcher memMonitor = //запрос к WMI для получения памяти ПК

new ManagementObjectSearcher("SELECT TotalVisibleMemorySize, FreePhysicalMemory, FreeVirtualMemory, TotalVirtualMemorySize FROM Win32\_OperatingSystem");

foreach (ManagementObject management in memMonitor.Get())

{

totalVisMemory = Convert.ToUInt64(management["TotalVisibleMemorySize"]);

totalVirtMemory = Convert.ToUInt64(management["TotalVirtualMemorySize"]);

freePhysMemory = Convert.ToUInt64(management["FreePhysicalMemory"]);

freeVirtMemory = Convert.ToUInt64(management["FreeVirtualMemory"]);

}

usingPhysRam = (ulong)((totalVisMemory - freePhysMemory)\*100 / totalVisMemory);

usingVirtRam = (ulong)((totalVirtMemory - freeVirtMemory)\*100 / totalVirtMemory);

}

public string getUsingPhysRam() { return Convert.ToString(usingPhysRam); }

public string getUsingVirtRam() { return Convert.ToString(usingVirtRam); }

public string getfreePhysMemory() { return Convert.ToString(freePhysMemory); }

public string gettotalVisMemory() { return Convert.ToString(totalVisMemory); }

}

}

ClientObject:

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

using System.Net.Sockets;

using System.Windows.Forms;

namespace Server2

{

public class ClientObject

{

public TcpClient client;

public ClientObject(TcpClient tcpClient)

{

client = tcpClient;

}

public void Process()

{

NetworkStream stream = null;

try

{

stream = client.GetStream();

byte[] data = new byte[64]; // буфер для получаемых данных

if (true)

{

// получаем сообщение

StringBuilder builder = new StringBuilder();

int bytes = 0;

do

{

bytes = stream.Read(data, 0, data.Length);

builder.Append(Encoding.Unicode.GetString(data, 0, bytes));

}

while (stream.DataAvailable);

SystemInfo systemInfo = new SystemInfo();

string message = builder.ToString();

Console.WriteLine(message);

// отправляем обратно сообщение

message = "volOfRequest2" + "$" + systemInfo.getUsingPhysRam() + ";" + systemInfo.getUsingVirtRam();

data = Encoding.Unicode.GetBytes(message);

stream.Write(data, 0, data.Length);

}

}

catch (Exception ex)

{

}

finally

{

if (stream != null)

stream.Close();

if (client != null)

client.Close();

}

}

}

}

Program:

using System;

using System.Net;

using System.Net.Sockets;

using System.Threading;

namespace Server2

{

class Program

{

const int port = 8887;

static TcpListener listener;

static void Main(string[] args)

{

try

{

listener = new TcpListener(IPAddress.Parse("127.0.0.1"), port);

listener.Start();

Console.WriteLine("Ожидание подключений...");

while (true)

{

TcpClient client = listener.AcceptTcpClient();

ClientObject clientObject = new ClientObject(client);

// создаем новый поток для обслуживания нового клиента

Thread clientThread = new Thread(new ThreadStart(clientObject.Process));

clientThread.Start();

}

}

catch (Exception ex)

{

Console.WriteLine(ex.Message);

}

finally

{

if (listener != null)

listener.Stop();

}

}

}

}

Client:

RequestHandler:

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

using System.Windows.Forms;

namespace Client

{

internal class RequestHandler

{

string message="";

string X = "";

string Y = "";

string er = "";

public RequestHandler(string message) { this.message = message; }

public string getEr() { return this.er; }

public string getX() { return this.X; }

public string getY() { return this.Y; }

public int volOfRequest() //На сколько запросов ответил сервер

{

return Convert.ToInt32(this.message[12]);

}

public void getOutInfo(int vol, int index) //Распределяем данные из ответа

{

string msg = this.message.Substring(index);

if (vol==1)

{

int a = 0;

while (msg[a] !=';')

{

this.X+=msg[a];

a += 1;

}

this.Y=msg.Substring(a+1);

}

if (vol == 2)

{

int a = 0;

while (msg[a]!='$')

{

this.er+=msg[a];

a += 1;

}

getOutInfo(1, index + a + 1);

}

}

}

}

ConnectToServer:

using System;

using System.Windows.Forms;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

using System.Net.Sockets;

namespace Client

{

internal class ConnectToServer

{

const string address = "127.0.0.1";

int port;

//string message;

public ConnectToServer(int port)

{

this.port = port;

}

public string getSystemInfo()

{

string message = "getSystemInfo";

TcpClient tcpClient = null;

tcpClient = new TcpClient(address,port);

NetworkStream stream = tcpClient.GetStream();

while (true)

{

byte[] data = Encoding.Unicode.GetBytes(message);

stream.Write(data, 0, data.Length);

// получаем ответ

data = new byte[64]; // буфер для получаемых данных

StringBuilder builder = new StringBuilder();

int bytes = 0;

do

{

bytes = stream.Read(data, 0, data.Length);

builder.Append(Encoding.Unicode.GetString(data, 0, bytes));

}

while (stream.DataAvailable);

message = builder.ToString();

tcpClient.Close();

return message;

}

}

//public string getMessage() { return message; }

}

}

Form1:

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;

namespace Client

{

public partial class Form1 : Form

{

public Form1()

{

InitializeComponent();

}

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

{

ConnectToServer connectToServer1 = new ConnectToServer(8888);

//if (true)

//{

RequestHandler handler = new RequestHandler(connectToServer1.getSystemInfo());

handler.getOutInfo(2, 14);

lastErrTB.Text = handler.getEr();

cursorXTB.Text = handler.getX();

cursorYTB.Text = handler.getY();

//lastErrTB.Text = connectToServer1.getMessage();

//}

}

private void server2BT\_Click(object sender, EventArgs e)

{

ConnectToServer connectToServer1 = new ConnectToServer(8887);

RequestHandler handler = new RequestHandler(connectToServer1.getSystemInfo());

handler.getOutInfo(1, 14);

phRamTB.Text = handler.getY();

virtRamTB.Text = handler.getX();

}

}

}

Program:

using System;

using System.Collections.Generic;

using System.Linq;

using System.Threading.Tasks;

using System.Windows.Forms;

using System.Net.Sockets;

using System.Text;

namespace Client

{

internal static class Program

{

/// <summary>

/// Главная точка входа для приложения.

/// </summary>

[STAThread]

static void Main()

{

Application.EnableVisualStyles();

Application.SetCompatibleTextRenderingDefault(false);

Application.Run(new Form1());

}

}

}