Министерство цифрового развития, связи и массовых коммуникаций Российской Федерации

Ордена Трудового Красного Знамени

Федеральное государственное образовательное бюджетное учреждение высшего профессионального образования

Московский технический университет связи и информатики

Курсовая работа

по предмету: «Операционные системы»

Выполнила: Фролова А.Д.

Группа: БСТ2003

Вариант: 21

Проверила: Алексанян Д.А.

Москва, 2022

# Задание на курсовую работу

**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.

Также необходимо реализовать дополнительные функции, а именно:

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

[Задание на курсовую работу 2](#_Toc122658002)

[Введение 4](#_Toc122658003)

[Постановка задачи 5](#_Toc122658004)

[Краткие теоретические основы разрабатываемой темы 6](#_Toc122658005)

[Разработка и описание алгоритма работы приложения 11](#_Toc122658006)

[Описание структуры программы 12](#_Toc122658007)

[Результаты применения программы 13](#_Toc122658008)

[Заключение 15](#_Toc122658009)

[Список использованных источников 16](#_Toc122658010)

[Приложение 17](#_Toc122658011)

# Введение

В наше время важнейшим ресурсом является информацию. С ней можно выполнять различные действия: хранить, удалять, обрабатывать, передавать и тд. В данной работе конкретнее будет рассмотрена обработка и передача данных.

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 для работы с информацией 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.

Клиент и сервер — это программы, расположенное на разных компьютерах, в разных контроллерах и других подобных устройствах. Между собой они взаимодействуют через вычислительную сеть с помощью сетевых протоколов.

Программы-серверы являются поставщиками услуг. Они постоянно ожидают запросы от программ-клиентов и предоставляют им свои услуги (передают данные, решают вычислительные задачи, управляют чем-либо и т.п.). Сервер должен быть постоянно включен и “прослушивать” сеть. Каждая программа-сервер, как правило, может выполнять запросы от нескольких программ-клиентов.

Программа-клиент является инициатором запроса, который может произвести в любой момент. В отличие от сервера клиент не должен быть постоянно включен. Достаточно подключиться в момент запроса.

# Постановка задачи

1. Реализовать многозадачное приложение на основе технологии «клиент-сервер» средствами интегрированной среды разработки Microsoft Visual Studio/Visual C++. Обмен данными между клиентами и серверами осуществляется при помощи сокетов.
2. Реализовать выполнение и передачу положения курсора и кода последней ошибки от сервера 1 клиенту.
3. Реализовать выполнение и передачу процента используемой физической и виртуальной памяти от сервера 2 клиенту.
4. Реализовать работу, чтобы при обновлении информации на клиенте по таймеру данные передавались только в том случае, если что-либо изменилось с момента прошлого обновления.
5. Реализовать работу, чтобы обновление информации на клиенте происходило по инициативе сервера как реакция на изменения на сервере.
6. Оформить пояснительную записку.

# Краткие теоретические основы разрабатываемой темы

Суть вычислительной архитектуры «клиент-сервер»: это достаточно большая сеть, которая может состоять из неограниченного числа клиентов и одного или нескольких серверов, при этом неважно, по какой сети происходит коммуникация: по глобальной или по локальной.

Клиент-серверное взаимодействие происходит даже тогда, когда пользователи отправляют друг другу электронные письма или общаются через мессенджер. Даже в этих случаях сообщение или письмо вначале отправляется на сервер, где оно проходит небольшую обработку, а потом — получателю. При этом сервер может сохранить отправленный файл, чтобы отправитель и получатель всегда имели к нему доступ: скачали, отредактировали или удалили.

В глобальном смысле весь интернет — это один большой сервер, а в роли клиента выступает каждое устройство, выходящее в сеть.

Благодаря архитектуре клиент-сервер обслуживается любой запрос пользователей в поисковой системе. О каждом своем клиенте серверы хранят определенную информацию, создавая обезличенный облик своего клиента.

**Преимущества клиент-серверной архитектуры:**

1. делает возможным, в большинстве случаев, распределение функций вычислительной системы между несколькими независимыми компьютерами в сети;
2. все данные хранятся на сервере, который, как правило, защищён гораздо лучше большинства клиентов. На сервере проще обеспечить контроль полномочий, чтобы разрешать доступ к данным только клиентам с соответствующими правами доступа;
3. позволяет объединить различные клиенты. Использовать ресурсы одного сервера часто могут клиенты с разными аппаратными платформами, операционными системами и т.п.

**Недостатки клиент-серверной архитектуры:**

1. неработоспособность сервера может сделать неработоспособной всю вычислительную сеть;
2. поддержка работы данной системы требует отдельного специалиста -- системного администратора;
3. высокая стоимость оборудования.

Обзор системных средств коммуникации и синхронизации процессов:

**Межпроцессное взаимодействие** (inter-process communication, IPC) — обмен данными между потоками одного или разных процессов. Реализуется посредством механизмов, предоставляемых ядром ОС или процессом, использующим механизмы ОС и реализующим новые возможности IPC. Может осуществляться как на одном компьютере, так и между несколькими компьютерами сети.

Один из наиболее популярных интерфейсов межпроцессного взаимодействия называется сокеты.

Интерфейс сокетов — это API для сетей TCP/IP.

Интерфейс прикладной программы (API) представляет собой просто набор функций (интерфейс), использующийся программистами для разработки прикладных программ в определенных компьютерных сре­дах.

**Сокет** (socket) — название программного интерфейса для обеспечения обмена данными между процессами. Процессы при таком обмене могут исполняться как на одной ЭВМ, так и на различных ЭВМ, связанных между собой сетью. Сокет — абстрактный объект, представляющий конечную точку соединения.

Интерфейс сокетов впервые появился в BSD Unix. Программный интерфейс сокетов описан в стандарте POSIX.1 и в той или иной мере поддерживается всеми современными операционными системами. Так как изначально разработчиками интерфейса сокетов были иссле­дователи университета в г. Беркли, этот интерфейс часто называется сокетами Беркли (Berkeley sockets) или сокетами в стиле Беркли.

Сокеты определяют конечные точки взаимодействия (обычно через сеть). Они поддерживают целый ряд протоколов, самыми популярными из которых сегодня являются User Datagram Protocol (UDP) и Transmission Control Protocol (TCP).

UDP-сокеты не требуют установления логических соединений и обычно применяются для широковещательной и многоадресной (multicast) связи. В UDP нет средств надежной доставки сообщений и контроля правильного порядка пакетов, поэтому за обнаружение потери пакетов, устранение таких проблем и упорядочение пакетов отвечает приложение-получатель.

TCP-сокеты ориентированы на логические соединения, предоставляя надежный коммуникационный путь двумя конечными точками. Важное преимущество TCP в том, что он гарантирует доставку сообщений и правильный порядок пакетов. В этой статье основное внимание уделяется TCP-сокетам.

TCP-сокеты могут быть либо клиентскими, либо серверными. Серверный сокет ожидает запросы на установление соединений, а клиентский — инициирует соединение. Как только соединение между сокетами установлено, клиент и сервер могут передавать и принимать данные или закрыть это соединение.

Процессам часто нужно взаимодействовать друг с другом, например, один процесс может передавать данные другому процессу, или несколько процессов могут обрабатывать данные из общего файла. Во всех этих случаях возникает проблема синхронизации процессов, которая может решаться приостановкой и активизацией процессов, организацией очередей, блокированием и освобождением ресурсов.

**Синхронизация потоков** (thread synchronization) — это обобщенный термин, относящийся к процессу взаимодействия и взаимосвязи потоков. Нужно учесть, что синхронизация потоков требует привлечения в качестве посредника самой операционной системы. Потоки не могут взаимодействовать друг с другом без ее участия.

В Win32 существует несколько методов синхронизации потоков. Бывает, что в конкретной ситуации один метод более предпочтителен, чем другой.

Один из методов синхронизации потоков состоит в использовании **критических секций**(critical sections). Это единственный метод синхронизации потоков, который не требует привлечения ядра Windows. (Критическая секция не является объектом ядра). Однако этот метод может использоваться только для синхронизации потоков одного процесса.

Критическая секция — это некоторый участок кода, который в каждый момент времени может выполняться только одним из потоков. Если код, используемый для инициализации массива, поместить в критическую секцию, то другие потоки не смогут войти в этот участок кода до тех пор, пока первый поток не завершит его выполнение.

**Мьютекс** (MUTual Exclusions - взаимоисключения) — это объект ядра, который можно использовать для синхронизации потоков из разных процессов. Он может принадлежать или не принадлежать некоторому потоку. Если мьютекс принадлежит потоку, то он находится в состоянии «занято». Если данный объект не относится ни к одному потоку, то он находится в состоянии «свободно». Другими словами, принадлежать для него означает быть в состоянии «занято». Мьютекс часто используют для проверки на повторный запуск приложения.

**События** используются в качестве сигналов о завершении какой-либо операции. Однако в отличие от мьютексов, они не принадлежат ни одному потоку.

Существует еще один метод синхронизации потоков, в котором используются семафорные объекты API. В **семафорах** применен принцип действия мьютексов, но с добавлением одной существенной детали. В них заложена возможность подсчета ресурсов, что позволяет заранее определенному числу потоков одновременно войти в синхронизуемый участок кода.

**Ждущий таймер** (waitable timer) представляет собой новый тип объектов синхронизации, поддерживаемый в Windows NT версии 4.0 и выше. Это полноценный объект синхронизации, который может использоваться для организации задержки в одном или нескольких приложениях.

Ждущий таймер работает в трех режимах. В режиме «ручного сброса» таймер переходит в установленное состояние при истечении заданной задержки и остается установленным до тех пор, пока функция SetWaitableTimer не задаст новую задержку. В режиме «автоматического сброса» таймер переходит в установленное состояние при истечении заданной задержки и остается установленным до первого успешного вызова функции ожидания. В этом режиме он напоминает объект Event в режиме автоматического сброса, поскольку каждый раз при истечении времени задержки разрешается выполнение лишь одной нити. Наконец, ждущий таймер может выполнять функции интервального таймера, который перезапускается с заданной задержкой после каждого срабатывания объекта.

Главная особенность, отличающая ждущие таймеры от системных, — то, что ждущие таймеры могут совместно использоваться несколькими приложениями.

**Почтовые ящики** –обеспечивают только однонаправленные соединения. Каждый процесс, который создает почтовый ящик, является «сервером почтовых ящиков» (mailslot server). Другие процессы, называемые «клиентами почтовых ящиков» (mailslot clients), посылают сообщения серверу, записывая их в почтовый ящик. Входящие сообщения всегда дописываются в почтовый ящик и сохраняются до тех пор, пока сервер их не прочтет. Каждый процесс может одновременно быть и сервером, и клиентом почтовых ящиков, создавая, таким образом, двунаправленные коммуникации между процессами.

**Каналы** (безымянные и именованные каналы) – способы организовать двунаправленное соединение, обычно, безымянные каналы используются для перенаправления стандартного ввода/вывода дочернего процесса так, чтобы он мог обмениваться данными с родительским процессом. Именованные каналы используются для передачи данных между независимыми процессами или между процессами, работающими на разных компьютерах.

**Файловое отображение** – с помощью физической памяти совместно используется (shared), если она отображается на виртуальное адресное пространство нескольких процессов, хотя виртуальные адреса в каждом процессе могут отличаться.

# Разработка и описание алгоритма работы приложения

Общее описание: приложение клиента отправляет запросы на получение информации двум серверам. Первый сервер передает информацию о коде последней ошибки и текущем положении курсора. Второй сервер передает информацию о проценте использования физической и виртуальной памяти. Приложения серверов работают независимо от клиента.

Каждый сервер может быть запущен только один раз, при попытке повторного запуска при запущенном приложении, появляется сообщение о том, что приложение уже запущено. Проверка на повторный запуск происходит с помощью мьютекса. Он присваивается потоку и при первом запуске переходит в состояние «занято». Пока мьютекс занят, повторный запуск невозможен. На рисунке 1 представлена реализация проверки для 1 сервера. Для второго проверка выполняется аналогичным образом.

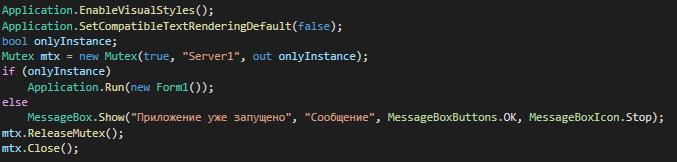


Рисунок 1 – проверка на повторный запуск

На клиенте: пользователь может получить данные с серверов, очистить выводы сообщений и выйти из программы.

Обновление информации с 1 сервера происходит по таймеру. Обновление информации с 2 сервера происходит по инициативе сервера.

На сервере 1: вычисляется информация о коде последней ошибки и текущем положении курсора. С клиента по таймеру приходит запрос на обновление информации. Если с последнего запроса информация не изменилась, то сообщение не передается.

На сервере 2 вычисляется информация о проценте используемой физической и виртуальной памяти. Информация передается клиенту по инициативе сервера, как реакция на изменения на сервере.

# Описание структуры программы

Для создания серверов используем конструктор public SocketServer(string ip, Int32 port, Form form) в котором задаем количество возможных клиентов, список потоков подключения, ip адрес сервера, сокет сервера.

На обоих серверах используются функции создания и разрыва подключения. При создании подключения создается новый поток, чтобы «Слушать» запросы клиента. При разрыве подключения этот поток прерывается.

Оба сервера так же имеют одинаковые функции отправки сообщений клиенту public void SendAll(Socket handler, List<string> message) public void Send(string message, Socket handler)

. Сначала сообщение разбивается на байты(Send), далее(SendAll) передается клиенту и лист сообщений очищается для записи нового

На первом сервере используются системные метрики GetLastWin32Error() и Cursor.Position

На втором сервере используется структура MEMORYSTATUS и параметры public uint dwTotalPhys public uint dwAvailPhys public uint dwTotalVirtual public uint dwAvailVirtual

Для реализации задания «обновление информации на клиенте происходило по инициативе сервера как реакция на изменения на сервере» создаем проверку обновлений на сервере, запуская бесконечный цикл, в котором если полученные данные не равны предыдущим, отправляем эти данные клиенту. Если же данные не изменились, то сервер ничего не отправляет клиенту.

Для реализации задания «обновление информации на клиенте по таймеру, данные передается только в том случае, если что-либо изменилось с момента прошлого обновления» Запрос осуществляется по таймеру. Каждый тик с клиента на сервер отправляется запрос. Если сообщение не равно предыдущему, клиент получает его и выводит. Если нет, то пропускается и не выводится.

Для инициализации клиента используется конструктор public SocketClient(string ipAddress, Int32 port) в котором инициализируется клиент и ip адрес

На клиенте реализованы функции подключения по определенному ip, получения сообщений по байтам и функция разрыва подключения. Так же реализованы функции работы с 1 и 2 сервером: подключение, обработка сообщений, вывод.

# Результаты применения программы

На рисунках 2-4 представлены скриншоты работы программы в различных условиях.

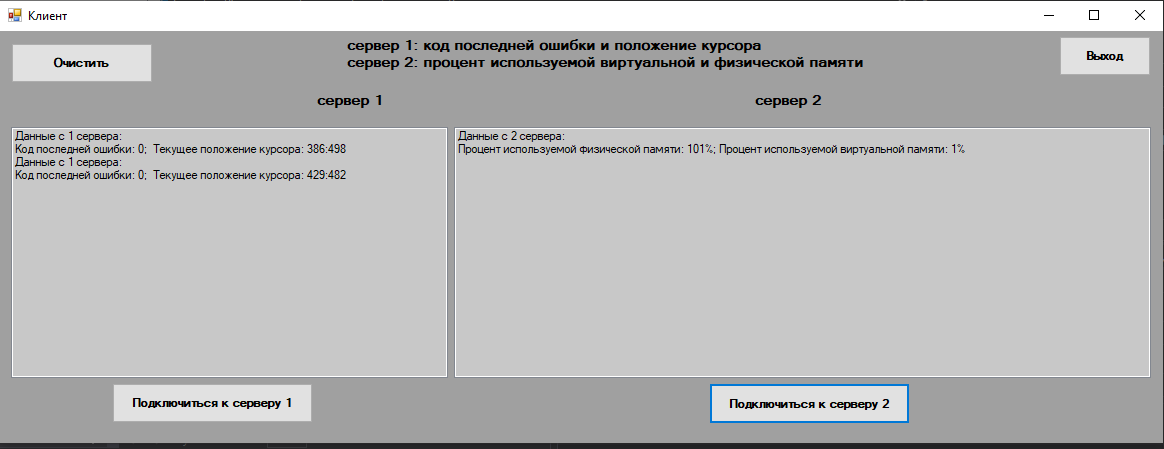


Рисунок 2 – Результат выполнения работы приложения на основе «клиент-сервер»

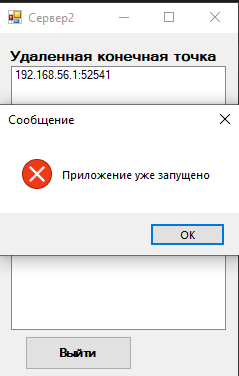


Рисунок 3 – Попытка повторного запуска сервера

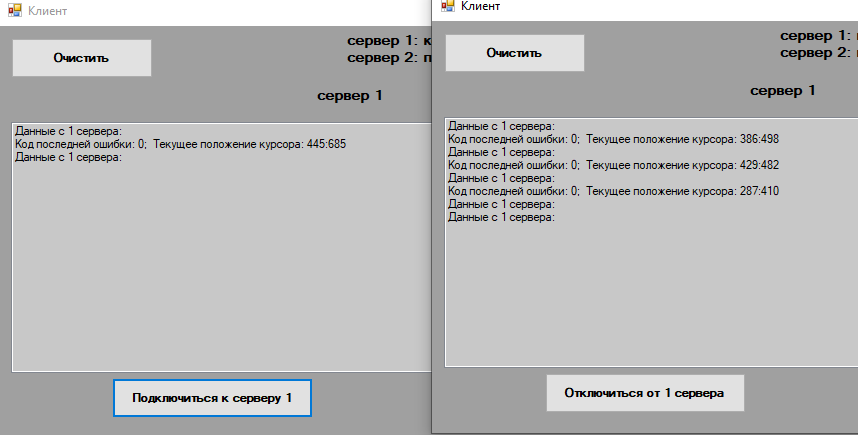


Рисунок 4 – Результат работы первого сервера при подключении двух клиентов.

# Заключение

В рамках пройденного материала за курс «Операционные системы» мы научились основам проектирования и работы операционных систем. В рамках курсовой работы мы реализовали приложение на основе технологии «клиент-сервер» с двумя серверами, каждый из которых отправляет системные данные клиенту, а также реализовали обновление информации при условии изменения информации сервера и клиента по таймеру, проверку на изменение информации.

# Список использованных источников

1. learn // Microsoft URL: learn.microsoft.com (дата обращения: 22.12.2022).
2. С.В. Назаров, А.И. Широков. «Современные операционные системы». — М., Бином, 2013
3. В.Г. Олифер, Н.А. Олифер. «Сетевые операционные системы». — СПб., Питер, 2009
4. 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
5. Д. Рихтер, К. Назар. Windows via C/C++. – СПб.: Питер, «Русская редакция», 2009

# Приложение

***Сервер 1***

SocketServer.cs

using System;

using System.Collections.Generic;

using System.Text;

using System.Net.Sockets;

using System.Threading;

using System.Net;

using System.Runtime.InteropServices;

using System.Windows.Forms;

namespace Server1

{

class SocketServer

{

public int a1;

public int a2;

public int a3;

public string messageA = "";

public Socket server;

public Form myf;

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>();

//Конструктор

public SocketServer(string ip, Int32 port, Form form)

{

this.max\_conn = 2;

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

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

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

this.server.Bind(this.ip);

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

myf = form;

}

//Запуск

public void Start()

{

for (int i = 0; i < this.max\_conn; i++)

{

Thread th = new Thread(Listening);

th.Start();

thread\_list.Add(th);

}

}

//закрытие подключений

public void Dispose()

{

foreach (Thread th in thread\_list)

{

th.Interrupt();

}

server.Close();

}

public void Listening()

{

while (true)

{

try

{

using (Socket client = this.server.Accept())

{

this.ClientList.Add(client.RemoteEndPoint.ToString());

if (client.Connected)

{

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

int error = Marshal.GetLastWin32Error();

int top = myf.Top;

int left = myf.Left;

int cursorX = Cursor.Position.X;

int cursorY = Cursor.Position.Y;

message.Add("Код последней ошибки: " + error + ";" + " " + " Текущее положение курсора: " + cursorX + ":" + cursorY);

message.Add("[end]");

if((error == a1) && (cursorX == a2) && (cursorY == a3))

{

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

{

message.RemoveAt(i);

}

}

else

{

a1 = error;

a2 = cursorX;

a3 = cursorY;

SendAll(client, message);

}

}

}

}

catch { }

}

}

public void SendAll(Socket handler, List<string> message)

{

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

{

Send(message[i], handler);

message.RemoveAt(i);

Thread.Sleep(500);

}

}

public void Send(string message, Socket handler)

{

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

try

{

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

}

catch

{

}

}

}

}

Form1.cs

using System;

using System.Windows.Forms;

using System.Threading;

using System.Net;

namespace Server1

{

public partial class Form1 : Form

{

SocketServer server;

public Form1()

{

InitializeComponent();

//запуск сервера

string MyIp = Dns.GetHostName();

server = new SocketServer(MyIp, 52000, this);

server.Start();

Thread list = new Thread(List); //если вдруг подключатся несколько клиентов

list.IsBackground = true;

list.Start();

this.Show();

}

private void List()

{

for (; ; Thread.Sleep(200))

if (server.ClientList.Count != listBox1.Items.Count)

{

server.messageA = "";

listBox1.Invoke(new MethodInvoker(delegate { listBox1.Items.Clear(); }));

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

{

listBox1.Invoke(new MethodInvoker(delegate { listBox1.Items.Add(server.ClientList[i]); }));

}

}

}

//если закрыть форму на крестик

private void Form1\_FormClosing(object sender, FormClosingEventArgs e)

{

server.Dispose();

Dispose();

Application.Exit();

}

//если закрыть форму по кнопке

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

{

server.Dispose();

Dispose();

Application.Exit();

}

}

}

Program.cs

using System;

using System.Windows.Forms;

using System.Threading;

namespace Server1

{

static class Program

{

/// <summary>

/// Главная точка входа для приложения.

/// </summary>

[STAThread]

static void Main()

{

Application.EnableVisualStyles();

Application.SetCompatibleTextRenderingDefault(false);

bool onlyInstance;

Mutex mtx = new Mutex(true, "Server1", out onlyInstance);

if (onlyInstance)

Application.Run(new Form1());

else

MessageBox.Show("Приложение уже запущено", "Сообщение", MessageBoxButtons.OK, MessageBoxIcon.Stop);

mtx.ReleaseMutex();

mtx.Close();

}

***Сервер 2***

Form1.cs

using System;

using System.Windows.Forms;

using System.Threading;

using System.Net;

namespace Server2

{

public partial class Form1 : Form

{

SocketServer server;

public Form1()

{

InitializeComponent();

//запуск сервера

string MyIp = Dns.GetHostName();

server = new SocketServer(MyIp, 53000, this);

server.Start();

Thread list = new Thread(List);

list.IsBackground = true;

list.Start();

this.Show();

}

private void List()

{

for (; ; Thread.Sleep(200))

if (server.ClientList.Count != listBox1.Items.Count)

{

listBox1.Invoke(new MethodInvoker(delegate { listBox1.Items.Clear(); }));

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

listBox1.Invoke(new MethodInvoker(delegate { listBox1.Items.Add(server.ClientList[i]); }));

}

}

private void Form1\_FormClosing(object sender, FormClosingEventArgs e)

{

server.Dispose();

this.Dispose();

Application.Exit();

}

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

{

server.Dispose();

this.Dispose();

Application.Exit();

}

}

}

SocketServer.cs

using System;

using System.Collections.Generic;

using System.Text;

using System.Net.Sockets;

using System.Threading;

using System.Net;

using System.Runtime.InteropServices;

using System.Windows.Forms;

namespace Server2

{

class SocketServer

{

public uint a;

public uint b;

public string messageA = "";

public Socket server;

public Socket client;

private IPEndPoint ip;

private List<Thread> thread\_list;

private int max\_conn;

public Form myf;

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

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

//Конструтор сервера

public SocketServer(string ip, Int32 port, Form form)

{

this.max\_conn = 3; //Количество клиентов

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

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

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

this.server.Bind(this.ip);

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

myf = form;

}

//Запуск

public void Start()

{

for (int i = 0; i < this.max\_conn; i++)

{

Thread th = new Thread(Listening);

th.Start();

thread\_list.Add(th);

}

}

//Разрыв подключения

public void Dispose()

{

foreach (Thread th in thread\_list)

{

th.Interrupt();//Прерывание потока

}

server.Close();

}

[DllImport("kernel32.dll")]

//выделение памяти

public static extern IntPtr GlobalAlloc(int con, int size);

[DllImport("kernel32.dll")]

//освобождение памяти

public static extern int GlobalFree(IntPtr start);

[DllImport("kernel32.dll")]

public static extern void GlobalMemoryStatus(ref MEMORYSTATUS lpBuffer);

public struct MEMORYSTATUS //Структура памяти

{

public uint dwMemoryLoad;

public uint dwTotalPhys;

public uint dwAvailPhys;

public uint dwTotalPageFile;

public uint dwAvailPageFile;

public uint dwTotalVirtual;

public uint dwAvailVirtual;

public uint dwAvailExtendedVirtual;

}

private void Listening()

{

while (true)

{

try

{

using (Socket client = this.server.Accept())

{

this.ClientList.Add(client.RemoteEndPoint.ToString());

while (client.Connected)

{

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

HashSet<IntPtr> handle\_list = new HashSet<IntPtr>();

HashSet<uint> potok\_list = new HashSet<uint>();

MEMORYSTATUS memStatus = new MEMORYSTATUS();

GlobalMemoryStatus(ref memStatus);

uint procphys = (memStatus.dwTotalPhys - memStatus.dwAvailPhys) \* 100 / (memStatus.dwTotalPhys);

uint procvirt = (memStatus.dwTotalVirtual - memStatus.dwAvailVirtual)\*100 / (memStatus.dwTotalVirtual);

message.Add("Процент используемой физической памяти: " + procphys + "%" + "; Процент используемой виртуальной памяти: " + procvirt + "%");

message.Add("[end]");

if ((memStatus.dwAvailPhys == a) && (memStatus.dwAvailVirtual == b))

{

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

{

message.RemoveAt(i);

}

}

else

{

a = memStatus.dwAvailPhys;

b = memStatus.dwAvailVirtual;

SendAll(client, message);

}

}

}

}

catch { }

}

}

//Работа с собщением

public void SendAll(Socket handler, List<string> message)

{

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

{

Send(message[i], handler);

message.RemoveAt(i);

Thread.Sleep(5);

}

}

//Побитовая отправка

public void Send(string message, Socket handler)

{

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

try

{

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

}

catch { }

}

}

}

Program.cs

using System;

using System.Windows.Forms;

using System.Threading;

namespace Server2

{

static class Program

{

/// <summary>

/// Главная точка входа для приложения.

/// </summary>

[STAThread]

static void Main()

{

Application.EnableVisualStyles();

Application.SetCompatibleTextRenderingDefault(false);

bool onlyInstance;

Mutex mtx = new Mutex(true, "Server2", out onlyInstance);

if (onlyInstance)

Application.Run(new Form1());

else

MessageBox.Show("Приложение уже запущено", "Сообщение", MessageBoxButtons.OK, MessageBoxIcon.Stop);

mtx.ReleaseMutex();

mtx.Close();

}

}

}

***Клиент***

Form1.cs

using System;

using System.Windows.Forms;

using System.Threading;

namespace Client

{

public partial class Form1 : Form

{

int k = 0, p = 0; //счетчики нажатий на кнопку

int port;

static SocketClient client;

public string rmes1, rmes2;// переменная для записи в листбокс

public Form1()

{

InitializeComponent();

this.Load += Form1\_Load;

}

//РАБОТА С 1 СЕРВЕРОМ

//запрос на подключениие

private void button4\_Click(object sender, EventArgs e)

{

//получить данные с сервера один

p++;//счетчик нажатия на кнопку

rmes1 = ""; //переменная для записи в листбокс

port = 52000; //Инициализируем порт подключения

if (p % 2 != 0)//Нажатие - запуск

{

Thread connect = new Thread(server1);//инициализируем поток на подключение

connect.Start();

connect.IsBackground = true;

timer1.Start();

button4.Text = "Отключиться от 1 сервера";

}

else //Нажатие - отключение

{

client.Disconnect();

button4.Text = "Подключиться к серверу 1";

}

}

//подключение

private void server1()

{

string MyIp = "192.168.56.1"; //инициализируем IP

client = new SocketClient(MyIp, this.port); //инициализация клиента

//Если есть подключение

if (client.Connect())

{

listBox1.Invoke(new MethodInvoker(delegate { listBox1.Items.Add("Данные с 1 сервера:"); })); //Информация о выводе

while (true)

{

client.Receive(); //получаем сообщение

if (client.ReceiveMessage.Count > 1) //проверка

{

if (client.ReceiveMessage[client.ReceiveMessage.Count - 1] == "[end]") { break; } //проверка

}

}

//запись сообщения в переменную

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

{

rmes1 += client.ReceiveMessage[i];

}

listBox1.Invoke(new MethodInvoker(delegate { listBox1.Items.Add(rmes1); })); //Добавление записи в листбокс

rmes1 = "";//очищение переменной

//очистка полученного сообщения

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

{

client.ReceiveMessage.RemoveAt(i);

}

}

}

//Отправление запроса по таймеру

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

{

if (p % 2 != 0)//если кнопка подключения нажата 1 раз, каждый тик (=1000) запрашиваем инфу от сервера

{

Thread connect = new Thread(server1);

connect.Start();

connect.IsBackground = true;

}

else//если нажали на кнопку подключения в 2 раз остановка связи с сервером

timer1.Stop();

}

//РАБОТА СО 2 СЕРВЕРОМ

//Запрос на подключение

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

{

//получить данные с сервера два

k++;//счетчик нажатия на кнопку

rmes2 = "";//Переменная для записи в листбокс

port = 53000;

if (k % 2 != 0)//Нажатие - запуск

{

Thread connect = new Thread(server2); //Инициализация потока

connect.Start();

connect.IsBackground = true;

button1.Text = "Отключиться от 2 сервера";

}

else //Нажатие - отключение

{

client.Disconnect();

button1.Text = "Подключиться к серверу 2";

}

}

//подключение

private void server2() //подключение к 2 серверу

{

string MyIp = "192.168.56.1"; //инициализация IP

client = new SocketClient(MyIp, this.port); //инициализация клиента

//Если есть подключение

if (client.Connect())

{

listBox2.Invoke(new MethodInvoker(delegate { listBox2.Items.Add("Данные с 2 сервера:"); }));//Информация о выводе

while (true)//Всегда получаем информацию от сервера

{

//получаем сообщения

while (true)

{

client.Receive(); //получаем сообщение

if (client.ReceiveMessage.Count > 1)//проверка

{

if (client.ReceiveMessage[client.ReceiveMessage.Count - 1] == "[end]") { break; }//Проверка

}

}

//Запись сообщения в переменную

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

{

rmes2 += client.ReceiveMessage[i];

}

listBox2.Invoke(new MethodInvoker(delegate { listBox2.Items.Add(rmes2); })); //Добавление записи в листбокс

rmes2 = "";//обнуление переменной для записи сообщения

//очистка полученного сообщения

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

{

client.ReceiveMessage.RemoveAt(i);

}

}

}

}

//Очитстка данных

private void button3\_Click(object sender, EventArgs e)

{

listBox1.Items.Clear();

listBox2.Items.Clear();

rmes1 = "";

rmes2 = "";

k = 0;

p = 0;

}

//Выход

private void button5\_Click(object sender, EventArgs e)

{

Application.Exit();

}

//Загрузка формы

private void Form1\_Load(object sender, EventArgs e)//определение таймера при загрузке формы

{

//Инициализация таймера

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

timer1.Interval = 5000;

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

}

}

}

SocketServer.cs

using System;

using System.Collections.Generic;

using System.Text;

using System.Net.Sockets;

using System.Net;

namespace Client

{

class SocketClient

{

public Socket client; //инициализация клиента

private IPEndPoint ip; //Инициализация IP

public List<string> ReceiveMessage = new List<string>(); //массив полученных сообщений

//Конструктор

public SocketClient(string ipAddress, Int32 port)

{

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

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

}

//Подключение

public bool Connect()

{

try

{

this.client.Connect(this.ip); //подключение по введенному ip

return true;

}

catch

{

return false;

}

}

//Получение сообщений

public void Receive()

{

string message = String.Empty; //задаем пустую строку для сообщения

byte[] GetBytes = new byte[1024]; //массив переданных байтов

try

{

int b = client.Receive(GetBytes); //сообщение по байтам

message = Encoding.UTF8.GetString(GetBytes, 0, b); //преобразуем полученые байты в строку

if (message != "")

this.ReceiveMessage.Add(message);

}

catch

{ }

}

//Отправление

public void Send(string message, Socket handler)

{

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

try

{

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

}

catch

{

}

}

public void Disconnect()

{

if (client.Connected)

this.client.Disconnect(false);

}

}

}

Program.cs

using System;

using System.Windows.Forms;

namespace Client

{

static class Program

{

/// <summary>

/// Главная точка входа для приложения.

/// </summary>

[STAThread]

static void Main()

{

Application.EnableVisualStyles();

Application.SetCompatibleTextRenderingDefault(false);

Application.Run(new Form1());

}

}

}