ЛАБОРАТОРНАЯ РАБОТА №6 “РАЗРАБОТКА GUI ПИЛОЖЕНИЯ В АРХИТЕКТУРЕ КЛИЕНТ-СЕРВЕР НА ЯЗЫКЕ C#”

1. Теоретическая часть

Сокеты появились тогда, когда начали создавать первые системы распределенных вычислений. И, скорее всего, о них еще не скоро забудут. Их любят многие за гибкость и простоту в использовании. На их основе можно очень быстро и просто сделать первый прототип любого распределенного приложения. Присутствуют они и в .NET (System.Net.Sockets). В следующем листинге представлен простейший пример взаимодействия “клиент-сервер”:

```
private static void  OnBeginAccept(IAsyncResult ar)
{
Socket listener = (Socket)ar.AsyncState
using  (Socket client = listener.EndAccept(ar))
{
byte [] buffer = new byte [_bufferSize];
// Получаем данные от клиента.
client.Receive(buffer);
buffer = Encoding.UTF8.GetBytes(string .Format("Hello, {0}" ,
Encoding.UTF8.GetString(buffer)));
// Отправляем клиенту ответ.
client.Send(buffer);
}
}
static void  Main(string [] args)
{
EndPoint endPoint = new  IPEndPoint(IPAddress.Loopback, 8320);
using  (Socket listener = new  Socket(endPoint.AddressFamily,
SocketType.Stream, ProtocolType.Tcp))
using  (Socket client = new  Socket(endPoint.AddressFamily,
SocketType.Stream, ProtocolType.Tcp))
{
// Сокет будет слушать порт 8320 на локальном адресе
listener.Bind(endPoint);
// Максимальный размер очереди подключений.
// Нам для теста достаточно одного.
listener.Listen(1);
// Запускаем поток, который ждет подключения клиента.
listener.BeginAccept(new  AsyncCallback(OnBeginAccept), listener);

// Подсоединяемся клиентом.
client.Connect(endPoint);
```

```
Console.Write("Введите сообщение: ");
string  request = Console.ReadLine();
// Отсылаем данные на сервер.
int  count = client.Send(Encoding.UTF8.GetBytes(request));
byte [] buffer = new byte [_bufferSize];
// Получаем данные с сервера.
client.Receive(buffer);
string  response = Encoding.UTF8.GetString(buffer);
int  index = response.IndexOf('\0');
Console.WriteLine(string .Format("Ответ сервера: {0}",
response.Remove(index)));
}
   }
```

В .NET есть и более высокоуровневые надстройки над сокетами. Классы `TcpClient` и `TcpListener` предоставляют функциональность, осуществляющую коммуникации не на уровне сокетов, а на уровне потока ввода/вывода. Для этого используется класс `NetworkStream`. Данный класс является наследником `System.IO.Stream`, но переопределенные им методы чтения и записи не записывают данные ни в какие внутренние хранилища (файлы, массивы байт), а передают прямо в сокет на сервер.

В .NET есть надстройка над сокетом, общающимся через UDP. Это класс `UdpClient`. Но пусть слово “Client” не вводит вас в заблуждение. UdpListener в .NET нет. `UdpClient` по сути является одновременно и клиентом, и сервером. Соответственно, его методы `Receive` и `Send` нужны именно для двунаправленного общения.

Теперь рассмотрим подробнее работу по созданию GUI приложений для платформы .NET. Создание приложений для Windows во многом упрощено благодаря использованию среды разработки Microsoft Visual Studio. Создание графических приложений на языке C# во многом аналогично созданию таковых на языке Visual C++;

Рассмотрим работу с формами Windows и сокетами на примере.

2. Практическая часть

Цель лабораторной работы – приобрести навыки разработки GUI приложений на языке C# в архитектуре клиент-сервер и на примере данной лабораторной работы ознакомиться с основными принципами разработки таких приложений для платформы .NET.

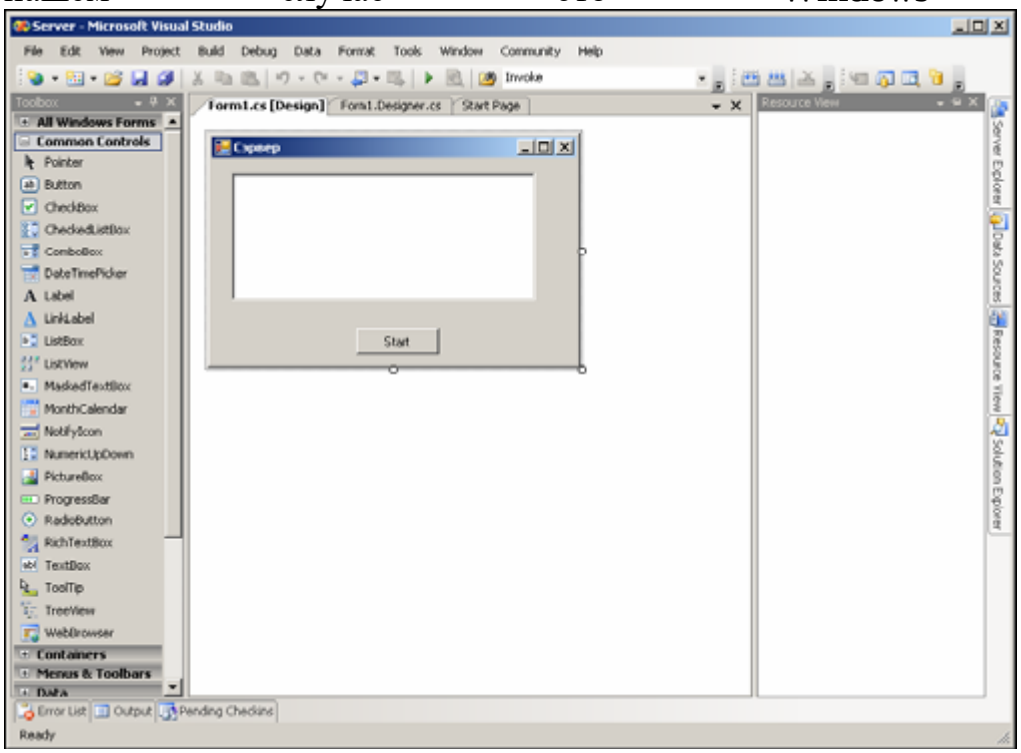
В данной лабораторной работе мы создадим два графических приложения для Windows(клиентское приложение и серверное приложение), которые будут взаимодействовать между собой по средством сокетов.

2.1. Серверная часть программы

Вначале рассмотрим пример реализации серверного приложения:

Запустим Microsoft Visual Studio и с помощью команды **File->New Project…** перейдем в диалоговое окно создания нового проекта.

В данном окне нам необходимо выбрать тип создаваемого приложения: в нашем случае это **Windows Application**.



После того как выбран тип приложения перед пользователем отобразиться следующее окно:

В данном окне среды Microsoft Visual Studio пользователю программы предоставлена возможность работы с формами Windows. По середине экрана мы видим главную форму нашего приложения.

На экране слева расположены элементы управления. Все элементы управления сгруппированы по своим функциям: например, базовые элементы управления, такие как Кнопка, Список, Поле Ввода Текста расположены в группе **Common Controls** (общие элементы управления), элементы для работы с данными – в группе **Data**(данные) и т.д. В нашей лабораторной работе мы будем использовать только общие элементы управления.

Добавим на нашу форму элементы управления **RichTextBox** и **Button**. В списке будут отображаться служебная информация, а Кнопка будет использоваться для инициализации сервера.

После того как элементы управления добавлены на форму, присвоим им соответствующие названия для дальнейшей наглядности и удобства. Делается это следующим образом: на форме выбирается необходимый элемент управления, затем правым щелчком мыши вызывается контекстное меню, в котором выбирается пункт **Properties** (Свойства). В окне свойств присвоим параметрам **Name (**имя**)** значения button1 и rtb1 для кнопки и поля для текста соответственно.

Далее приступим к непосредственному наполнению программы кодом. Начнем с обработки события нажатия кнопки. Наша кнопка будет предназначена для инициализации серверного приложения, поэтому в обработчике ее нажатия будет содержаться весь основной код приложения. Перейдем в окно редактирования кода обработчика событий для нашей кнопки путем выполнения двойного щелчка по ней. Однако перед началом написания кода нашего обработчика подключим необходимые нам библиотеки:

```
using System.Net;
using System.Net.Sockets;
using System.IO;
using System.Threading;
```

Первые две библиотеки библиотеки содержат необходимые классы для организации работы по сети, библиотека `System.IO` предназначена для работы с потоками ввода/вывода, а библиотека `System.Threading` – для работы с потоками.

Далее в конструкторе класса определим необходимые нам переменные:

```
//Путь к файлу, в котором будет храниться информация
String fileName = "e:\\file1.txt";
int fileCount = 0;
//Создание объекта класса TcpListener
TcpListener listener = null; //Создание объекта класса Socket
Socket socket = null;
//Создание объекта класса NetworkStream
NetworkStream ns = null;
//Создание объекта класса кодировки ASCIIEncoding
ASCIIEncoding ae = null;
```

Объекты класса `TcpListener` предназначены для прослушивания и принятия запросов от клиентских приложений. Для начала прослушивания используется метод `Start()`. Данный метод помещает входящие запросы на

соединение от клиентов в очередь. После этого, с помощью методов AcceptSocket() AcceptTcpClient() объекты класса принимают входящие запросы на соединение.

Объект класса Socket является стандартным объектом типа Socket.

Объект класса NetworkStream предоставляет методы для отправки и получения данных при помощи потоковых сокетов. Для использования объектов данного класса должен быть создан подсоединенный сокет. При закрытии объекта NetworkStream сокетное соединение не закрывается. Для отправки и получения данных с помощью объектов класса NetworkStream используются методы Write() Read() данного класса.

Перейдем к написанию обработчика нажатия кнопки запуска сервера. Рассмотрим следующий фрагмент кода:

```
// Создаем новый TCP_Listener который принимает запросы от любых IP адресов и слушает по порту 5555
listener = new TcpListener(IPAddress.Any, 5555);
     // Активация listen'ера
listener.Start();
socket = listener.AcceptSocket();
```

В данном фрагменте кода создается новый объект класса TcpListener, который принимает запросы от клиентов по любым IP адресам, и прослушивающий порт 5555 на предмет запросов на соединение. Как говорилось выше, с помощью метода Start() происходит активация данного объекта. После того как прослушивание началось, с помощью метода AcceptSocket() класса Socket мы ассоциируем созданный нами объект класса Socket с запросами на соединение от клиентских приложений.

Далее если сокет соединен, то нам необходимо создать новый сетевой поток для обмена информацией и новый поток для обработки нескольких клиентов. Выглядит это следующим образом:

```
if (socket.Connected)
{
ns = new NetworkStream(socket);
ae = new ASCIIEncoding();
//Создаем новый экземпляр класса ThreadClass
ThreadClass threadClass = new ThreadClass();
//Создаем новый поток
Thread thread = threadClass.Start(ns, fileName, fileCount, this);
}
```

Далее определим наш класс потока ThreadClass: данный класс будет содержать функцию запуска потока public Thread Start(), которая в свою очередь, будето содержать специальную определенную нами функцию

void ThreadOperations(), в которой будет производиться обработка входящих клиентских запросов.

Ключевым моментом при определении нашего класса потока явзяется создание экземпляра нашей формы в данномм классе. Это необходимо для доступа к элементам формы из определяемого нами класса.

```
Form1 form = null;
```

Рассмотрим создаваемую нами функцию запуска потока:

```
public Thread Start(NetworkStream ns, String fileName, int
fileCount, Form1 form)
{
this.ns = ns;
ae = new ASCIIEncoding();
this.fileName = fileName;

this.fileCount = fileCount;
this.form = form;
//Создание нового экземпляра класса Thread
Thread thread = new Thread(new ThreadStart(ThreadOperations));
//Запуск потока
thread.Start();
return thread;
}
```

При описании функции запуска потока в качестве параметров в неее передаются объект класса NetworkStream, имя файла, содержащего информацию, счетчик файлов и экземпляр нашей формы.

Далее при создании нового экземпляра класса Thread мы указываем, что при запуске нового потока в нем должна выполняться функция ThreadOperations(), которая будет определена нами ниже.

В функции ThreadOperations() определены все операции, которые будут работать в потоке, который будет создаваться для каждого из клиентских приложений. Данная функуия содержит ряд обработчиков клиентских запросов. Данные обработики функционально аналогичны, поэтому остановим наше внимание на некоторых из них. Работа нашей функции начинается с чтения запроса из потока:

```
//Создаем новую переменную типа byte[]
byte[] received = new byte[256]; //С помощью сетевого потока
считываем в переменную received данные от клиента
ns.Read(received, 0, received.Length);
String s1 = ae.GetString(received);
int i = s1.IndexOf("|",0);
String cmd = s1.Substring(0, i);
```

В данном фрагменте кода мы считываем получаемую от клиента информацию с помощью объекта класса `NetworkStream`, выделяем из полученных данных команду, которая отделена от остальных данных с помощью символа "|". После того как команда выделена, в программе идет обработка каждой полученной команды. Например, если от клиента поступает запрос на просмотр информации, то функция обработки данного запроса выглядит следующим образом:

```
if (cmd.CompareTo("view") == 0)
{
// Создаем переменную типа byte[] для отправки ответа клиенту
byte[] sent = new byte[256];
//Создаем объект класса FileStream для последующего чтения
информации из файла
FileStream fstr = new FileStream(fileName, FileMode.Open,
FileAccess.Read);

StreamReader sr = new StreamReader(fstr);
//Запись в переменную sent содержания прочитанного файла
sent = ae.GetBytes(sr.ReadToEnd());
sr.Close();
fstr.Close(); //Отправка информации клиенту
ns.Write(sent, 0, sent.Length);
}
```

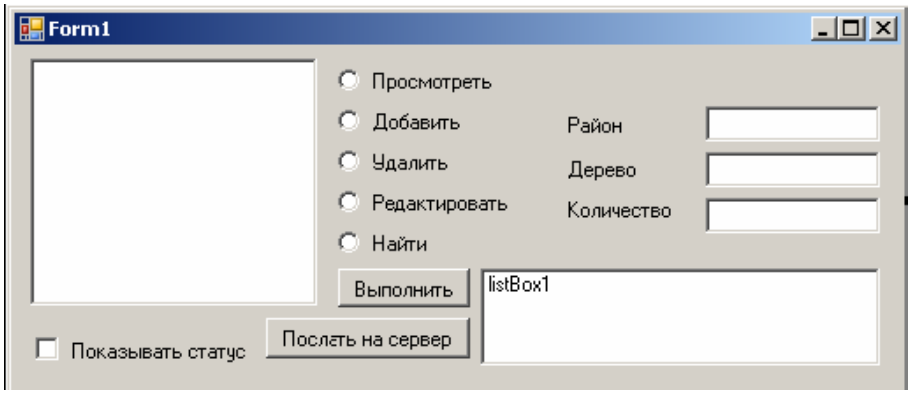
В данном фрагменте кода мы создаем переменную для хранения прочитанной информации в байтовом виде, записываем в нее содержание прочитанного нами файла, и отправляем данную информацию клиентскому приложению с помощью метода `Write()` класса `NetworkStream`.

Аналогично происходит обработка всех остальных клиентских запросов, с исходным кодом которых можно ознакомиться в Приложении 1.

2.2. Клиентская часть программы

Приступим к созданию клиентского приложения. Создадим новый проект типа Windows Application в Microsoft Visual Studio 2005. Наше клиентское приложение будет использовать основные типы для элементов управления для формирования запросов, отображения получаемых данных и служебной информации.

Добавим на нашу форму следующие элементы управления: элементы типа **TextBox** для ввода параметров запросов, элементы типа **RadioButton** для выбора операций, элемент типа **ListBox** для вывода служебной информации и элемент типа **CheckBox** , который будет отвечать за отображение/скрытие списка с служебной информации. После того, как все элементы добавлены наша форма должна иметь следующий вид:



Далее, для удобства понимания программы всем элементам управления можно присвоить свои специфические имена.

После окончания визуального формирования формы можно приступать к написанию исходного кода клиентской программы. Создадим объекты классов `TcpClient` и `ASIIEncoding`:

```
TcpClient tcp_client = new TcpClient("localhost", 5555);
ASCIIEncoding ae = new ASCIIEncoding();
```

Первый класс предоставляет простые методы для соединения, отправки и получения инфомации по сети в синхронном режиме. При создании объекта данного класса мы указываем, что соединение будет происходить с локальным сервером по порту 5555. Соединение будет происходить

Второй класс предназначен для для создания объектов кодировки, в котрой будут отправлятся и получаться данные.

После этого укажем начальное состояние определенных компонетов при загрузке нашей формы:

```
public Form1()
{
InitializeComponent();
region.Enabled = false;
kind.Enabled = false;
quant.Enabled = false;
listBox1.Visible = false;
}
```

Данные строки кода указывают, что определенные компоненты в момент инициализации формы будут недоступными.

Далее приступим к обработке событий нажатия кнопки выбора опреций и соответствующих RadioButtons. Принцип работы нашей программы будет следующим: сначала выбирается необходимый RadioButton, соответсвующий необходимой операции, затем по нажатии кнопки действия вызывается

обработчик соответсвуюшего события. Рассмотрим данный принцип на примере комманды просмотра информции, полчаемой с сервера:

```
private void operation_Click(object sender, EventArgs e)
{ //Если выбран RadioButton просмотра информции то…
if (radio_view.Checked == true)
{
//Создаем объект класса NetworkStream и ассоциируем его объектом класса TcpClient
NetworkStream ns = tcp_client.GetStream();
String command = "view";
String res = command + "|";

//Создаем переменные типа byte[] для отправки запроса и получения результата
byte[] sent = ae.GetBytes(res);
byte[] recieved = new byte[256];
//Отправляем запрос на сервер
ns.Write(sent, 0, sent.Length);
//Получаем результат выполнения запроса с сервера
ns.Read(recieved, 0, recieved.Length);
//Отображаем полученный результат в клиентском RichTextBox
richTextBox1.Text=ae.GetString(recieved);
String status = "=>Command sent:view data";
//Отображеем служебную информацию в клиентском ListBox
listBox1.Items.Add(status);
}
```

Прокомментируем данный фрагмент кода. В данном фрагменте происходит обработка сразу двух событий: нажатия кнопки действия и выбора соответствующего RadioButton. Первоначально при обработке данных событий создается объект класса `NetworkStream`, который с помощью метода `GetStream()` ассоциируется с созданным нами ранее объектом класса `TcpClient`. Метод `GetStream()` возвращает объект класса `NetworkStream`, который затем используется для отправки и получения данных. После создания данного объекта появляется возможность использовать методы `Write()` и `Read()` для отправки и получения данных соответственно. После того, как отправлен запрос и получен ответ, данные, содержащиеся в ответе, отображаются в форме путем их загрузки в элемент управления RichTextBox. Также в элементе управления ListBox отображается служебная информация об отправке запроса на сервер.

Остальные обработчики реализованы аналогичным образом, поэтому не будем останавливаться на их подробном описании, а при необходимости с исходным кодом всего приложения можно ознакомиться в Приложении 1.

Индивидуальные задания

В соответствии с заданием, полученным в лабораторной работе номер 5, разработать графическое приложение на языке C# в архитектуре клиент-сервер. В разрабатываемом приложении предусмотреть многопоточность, хранение информации в файле на серверной стороне, а со стороны клиентской части приложения предусмотреть возможность просмотра, добавления, удаления, редактирования и поиска информации хранящейся на сервере.

Вопросы для самопроверки

1. Что такое форма в .NET. Особенности ее создания
2. Для чего предназначен и как используется контрол RadioBox
3. Каким образом происходит заполнение и чтение данных из контролов RichTextBox и TextBox
4. Для чего предназначен класс TCP_Client
5. Что указывается в параметрах создаваемого экземпляра объекта TCP_Client
6. Последовательность действий при создании GUI приложения
7. Для чего предназначен и как используется контрол ListBox
8. Каким образом организуется работа контролов CheckBox и RadioBox
9. Для чего предназначен класс TCP_Listener
10. Что указывается в параметрах создаваемого экземпляра объекта TCP_Listener