Федеральное агенство образования

Южно-Уральский Государственный Университет

Факультет экономики и управления

Кафедра информатики

Курсовая работа

по дисциплине “Локальные сети”

СОЗДАНИЕ СЕТЕВОЙ ИГРЫ

“КОСТИ”

Выполнили:

Студенты группы ЭиУ-423

Захезин С.М.

Капацина Д.

Давидович И.

Проверил:

Сартасов Е.М.

Челябинск, 2006

**Аннотация**

Создание сетевой игры «Кости». Игра ведется до 21 очка, работа сделанна на 5 протоколах.

## **Введение**

**Правила игры**

Один из игроков является сервером, другой – клиентом. Для начала игры необходимо выбрать протокол, по которому будет установлена связь. Длительность игры зависит от набранных очков и желания играков.

**Протоколы**

**IPX**

Протокол IPX предоставляет возможность программам, запущенным на рабочих станциях, обмениваться пакетами данных без подтверждения. В сети Novell NetWare наиболее быстрая передача данных при наиболее экономном использовании памяти реализуется именно протоколом IPX. Протоколы SPX и NETBIOS сделаны на базе IPX и поэтому требуют дополнительных ресурсов.

Формат пакета IPX

Формат передаваемых по сети пакетов представлен на рис. 2.

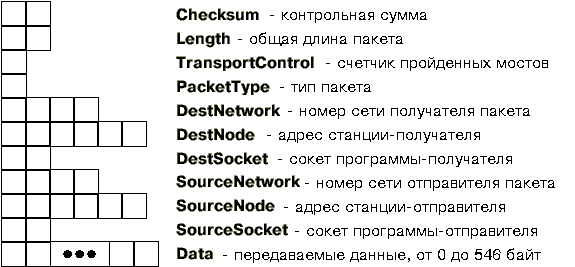


Рис. 2. Структура пакета IPX

Пакет можно разделить на две части - заголовок и передаваемые данные. Все поля, представленные на рис. 2, кроме последнего (Data), представляют собой заголовок пакета. Заголовок пакета выполняет ту же роль, что и конверт обычного письма - там располагается адрес назначения, обратный адрес и некоторая служебная информация.

Особенностью формата пакета является то, что все поля заголовка содержат значения в перевернутом формате, т. е. по младшему адресу записывается старший байт данных, а не младший, как это принято в процессорах фирмы Intel. Поэтому перед записью значений в многобайтовые поля заголовка необходимо выполнить соответствующее преобразование. Представление данных в заголовке пакета соответствует, например, формату целых числел в компьютере IBM-370 (серия ЕС ЭВМ). .

## 

## Работа с драйвером IPX/SPX

Первое, что должна сделать программа, желающая работать в сети с протоколом IPX или SPX, - проверить, установлен ли драйвер соответствующего протокола. Затем необходимо получить адрес вызова этого драйвера - точку входа API (Application Programm Interface - интерфейс для приложений). В дальнейшем программа вызывает драйвер при помощи команды межсегментного вызова процедуры по адресу точки входа API драйвера IPX/SPX.

#### Схема "клиент-сервер"

Обычно в сети одна из рабочих станций принимает запросы на выполнение каких-либо действий от других рабочих станций. Так как станция обслуживает запросы, она называется сервером (serve - обслуживать, server - обслуживающее устройство). Выполнив запрос, сервер посылает ответ в запросившую станцию, которая называется клиентом.

В сети может быть много серверов и много клиентов. Одни и те же клиенты могут посылать запросы разным серверам.

Говоря более строго, сервером или клиентом является не рабочая станция, а запущенная на ней программа. В мультизадачной среде разные программы, запущенные одновременно на одной и той же рабочей станции могут являться и клиентами, и серверами.

Программа-сервер, выполнив очередной запрос, переходит в состояние ожидания. Она ждет прихода пакета данных от программы-клиента. В зависимости от содержимого этого пакета программа-сервер может выполнять различные действия, в соответствии с логикой работы программы. Например, она может принять от программы-клиента дополнительные пакеты данных или передать свои пакеты.

Сервер и клиент при необходимости на какое-то время или навсегда могут поменяться местами, изменив свое назначение на противоположное.

Для того, чтобы создавать программы-серверы и программы-клиенты, нам необходимо научиться выполнять две задачи:

* инициализацию сервера и клиента;
  + прием и передачу пакетов данных.

#### Инициализация сервера и клиента

Для инициализации программ сервера и клиента, работающих на базе IPX, недостаточно убедиться в наличии соответствующего драйвера и получить точку входа в его API. Необходимо выполнить некоторые подготовительные действия для того, чтобы программа могла принимать и передавать пакеты данных. Прежде всего необходимо, чтобы программа-сервер или программа-клиент идентифицировали себя в сети при помощи механизма сокетов. Динамически распределяемые сокеты выдаются программам как бы во временное пользование (на время их работы) по специальному запросу. Перед началом работы программа должна запросить сокет у протокола IPX, а перед завершением - освободить его. При реализации схемы обмена данными "клиент-сервер" сервер обычно принимает пакеты на сокете, значение которого известно программам-клиентам. Сами же программы-клиенты могут использовать либо то же самое значение сокета, либо получать свой сокет динамически. Клиент может сообщить серверу свой сокет просто передав его в пакете данных (так как мы предполагаем, что сокет сервера известен программе-клиенту). После определения сокета необходимо узнать сетевой адрес станций-получателей. Для того чтобы клиент мог послать запрос серверу, необходимо кроме сокета сервера знать его сетевой адрес - номер сети и адрес рабочей станции в сети.

#### Прием и передача пакетов данных

#### Рассмотрим теперь процедуру приема пакетов данных средствами IPX.

#### Прием и передачу пакетов выполняет сетевой адаптер, работающий с использованием прерываний. Прикладные программы не работают напрямую с драйвером сетевого адаптера. Все свои запросы на прием и передачу пакетов они направляют драйверу IPX (программа ipx.exe или ipxodi.exe), который, в свою очередь, обращается к драйверу сетевого адаптера.

#### Для приема или передачи пакета прикладная программа должна подготовить пакет данных, сформировав его заголовок, и построить так называемый блок управления событием ECB (Event Control Block). В блоке ECB задается адресная информация для передачи пакета, адрес самого передаваемого пакета в оперативной памяти и некоторая другая информация.

#### Подготовив блок ECB, прикладная программа передает его адрес соответствующей функции IPX для выполнения операции приема или передачи пакета.

#### Функции IPX, принимающие или передающие пакет, не выполняют ожидания завершения операции, а сразу возвращают управление вызвавшей их программе. Прием или передача выполняются сетевым адаптером автономно и асинхронно по отношению к программе, вызвавшей функцию IPX для передачи данных. После того, как операция передачи данных завершилась, в соответствующем поле блока ECB устанавливается признак. Программа может периодически проверять ECB для обнаружения признака завершения операции.

**TCP/IP**

Рассмотрим работу семейства протоколов TCP/IP при обмене данными между двумя процессами telnet, выполняющимися на двух разных хостах, входящих в две разные сети, соединенные посредством маршрутизатора.

|  |
| --- |
| Хост А |
| telnet |
| TCP |
| IP |
| сетевой интерфейс 1 |

|  |  |
| --- | --- |
| Маршрутизатор | |
| IP | |
| сетевой интерфейс 1 | сетевой интерфейс 2 |

|  |
| --- |
| Хост B |
| telnet |
| TCP |
| IP |
| сетевой интерфейс 2 |

процесс- сервер

процесс- клиент

Уровень приложений/процессов

Транспортный уровень

Межсетевой уровень

Физически связаны

Физически связаны

## Работа протокола TCP

Протокол верхнего уровня (приложений/процессов) разделяет данные на кусочки (это процесс называется инкапсуляцией) и каждому кусочку добавляет заголовок. То, что получается в результате, называется TCP-сегментом.

|  |  |
| --- | --- |
| TCP-сегмент | |
| Заголовок  TCP-сегмента | Данные |

Модули протокола TCP обмениваются TCP-сегментами.

Протокол TCP обеспечивает надежную дуплексную передачу с предварительной установкой связи и с разрывом соединения.

Надежная – осуществляется контроль получения данных. Сегмент, на данные которого не пришло подтверждения через определенное время, посылается снова.

Дуплексная – значит, передача в две стороны, туда и обратно, передача только в дону сторону называется симплексной.

### Формат заголовка TCP-сегмента

Формат заголовка TCP-сегмента (заголовок состоит из 32-битных слов):

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 0 (бит) | 4 | 8 | 12 | 16 | 20 | 24 | | 28 31 |
| Source Port | | | | Destination Port | | | | |
| Sequence Number | | | | | | | | |
| Acknowledgement Number | | | | | | | | |
| Offset | Reserved | | Flags | Window | | | | |
| Checksum | | | | Urgent Point | | | | |
| Options | | | | | | | Padding | |
|  | | | | | | | | | |

* Source Port и Destination Port – это адреса процессов (отправителя и получателя соответственно). Грубо говоря, это просто числовые идентификаторы, которые присвоены процессам-протоколам верхнего уровня. Некоторые протоколы верхнего уровня имеют стандартные значения номеров портов:

|  |  |
| --- | --- |
| Номер порта | Процесс |
| 20 | ftp-data (передача данных по ftp) |
| 21 | fpt (команды) |
| 23 | telnet |
| 25 | smtp |
| 70 | gopher |
| 80 | www-http |

и т.д.

* Sequence Number – порядковый номер первого октета сегмента в потоке данных.
* Acknowledgement Number – количество полученных октетов данных
* Window – сколько октетов адресат готов принять
* Offset – начало данных сегмента
* Flags – управляющие флаги, используемые для установки и разрыва связи, для подтверждения получения данных, для передачи экстренных данных.
* Checksum – контрольная сумма: все байты заголовка суммируются отправителем и результат помещается в это поле. По получению адресат также суммирует все байты заголовка и сравнивает с этим числом. Если значения равны, значит, все в порядке.
* Urgent Point – определяет положение экстренных данных внутри сегмента.

**Pipe**

Канал – средство обмена информацией между процессами с одновременной синхронизацией, реализующей дисциплину FIFO (первый вошел – первый вышел), то есть прочитанные сообщения удаляются.

Канал – файл специального типа, особенности:

* время существования канала ограниченно временем работы процесса
* размер канала ограничен количеством блоков, доступных для прямой адресации (то есть на диске он не создается, а существует только в буферах ввода-вывода).
* Используется только для общения между родственными процессами (поскольку родственники наследуют все открытые файлы предка).

### Создание

Создает канал системный вызов pipe:

int fd[2];

pipe(int \*fd);

fd[0] - дескриптор файла для ввода

fd[1] - дескриптор файла для вывода

u-area ПР1

fdt:

|  |  |
| --- | --- |
|  |  |
|  |  |
|  |  |
| fd[0] |  |
| fd[1] |  |

u-area ПР2

fdt:

|  |  |
| --- | --- |
|  |  |
|  |  |
|  |  |
| fd[0] |  |
| fd[1] |  |

Общее пространство ядра

sft: vnode:

|  |  |  |
| --- | --- | --- |
| ре-жим доступа | указат. позиции чтения/  записи |  |
|  |  |  |
|  |  |  |
| R |  |  |
| W |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |

|  |
| --- |
|  |
|  |
|  |
|  |
|  |
|  |

Но файловый дескриптор – локальная для процесса характеристика, как могут два разных процесса использовать один файловый дескриптор? Для этого должна возникнуть такая ситуация

А это возможно только в том случае, если ПР1 и ПР2 являются родственниками – ведь потомок полностью наследует u-area предка, в том числе и таблицу файловых дескрипторов. Значит, либо ПР1 должен быть потомком ПР2, либо наоборот, либо они оба должны быть потомками третьего процесса

**NetBios**

Протокол NetBios работает на трех уровнях семиуровневой модели OSI: сетевом уровне, транспортном уровне и на уровне каналов связи. Уровень каналов связи обеспечивает механизм обмена сообщениями между программами, работающими на станциях в рамках канала связи или сессии. NETBIOS может обеспечить интерфейс более высокого уровня, чем протоколы IPX и SPX.

Протокол NETBIOS поддерживается в сетях IBM (IBM PC LAN), Novell NetWare, Microsoft Windows for Workgroups и в других сетях. К сожалению, нет единого стандарта на протокол NETBIOS, поэтому в сетевом программном обеспечении разных фирм используются разные интерфейсы для вызова команд NETBIOS.

С нашей точки зрения, наибольший интерес представляет применение NETBIOS в сетях Novell NetWare и Microsoft Windows for Workgroups. Мы рассмотрим основные возможности NETBIOS, связанные с передачей данных между рабочими станциями в пределах одного логического сегмента сети. Использовать NETBIOS проще, чем IPX или SPX. Однако, так как в среде Novell NetWare нужен специальный эмулятор NETBIOS, эффективность работы программы может снизиться. Кроме того, для эмулятора нужна дополнительная память, так как он реализован в виде резидентной программы.

Адресация станций и программ

Для идентификации рабочей станции протоколы IPX и SPX используют номер сети, адрес станции в сети и сокет. Адрес станции определяется на аппаратном уровне и представляет собой число длиной 6 байт. Номер сети занимает 4 байта. Сокеты выделяются динамически драйвером протокола IPX или могут быть получены в Novell.

Протокол NETBIOS использует другой механизм адресации станций и программ. Для адресации станции используются имена размером 16 байт. Каждая станция имеет одно *постоянное имя* (permanent name), которое образуется из аппаратного адреса добавлением к нему слева десяти нулевых байт. Кроме постоянного имени протокол NETBIOS позволяет добавлять (и удалять) *обычные имена и групповые имена*. Обычные имена служат для идентификации рабочей станции, групповые могут служить для посылки пакетов одновременно нескольким станциям в сети. Постоянное имя удалить нельзя, так как оно полностью определяется аппаратным обеспечением станции.

При добавлении обычного имени протокол NETBIOS опрашивает всю сеть для проверки уникальности имени. Групповое имя может быть одинаковое на нескольких станциях, поэтому при добавлении группового имени опрос сети не выполняется.

После добавления нового имени этому имени присваивается так называемый номер имени (name number), который используется для передачи данных по сети.

Сравнивая методы адресации, используемые протоколами IPX/SPX и NETBIOS, нетрудно заметить, что метод адресации протокола NETBIOS более удобен. Вы можете адресовать данные не только одной станции (как в IPX и SPX) или всем станциям сразу (как в IPX), но и группам станций, имеющим одинаковое групповое имя. Это может быть удобно, если в сети работают несколько групп пользователей, которые интенсивно обмениваются данными между собой.

Другим преимуществом схемы адресации протокола NETBIOS перед схемой адресации протоколов IPX/SPX можно считать отсутствие необходимости получать в фирме Novell свой собственный номер сокета для идентификации вашего программного обеспечения. Вы можете придумать свое собственное уникальное групповое имя, включающее, например, название программы и вашей фирмы, и использовать его для работы по схеме клиент-сервер.

## **MailSlot**

Канал Mailslot создается серверным процессом с помощью специально предназначенной для этого функции CreateMailslot. После создания серверный процесс получает идентификатор канала Mailslot. Пользуясь этим идентификатором, сервер может читать сообщения, посылаемые в канал клиентскими процессами. Однако сервер не может выполнять над каналом Mailslot операцию записи, так как этот канал предназначен только для односторонней передачи данных - от клиента к серверу. При ошибке функцией CreateMailslot возвращается значение INVALID\_HANDLE\_VALUE. Код ошибки можно определить при помощи функции GetLastError.

Прежде чем приступить к работе с каналом Mailslot, клиентский процесс должен его открыть. Для выполнения этой операции следует использовать функцию CreateFile. Запись сообщений в канал Mailslot выполняет клиентский процесс, вызывая для этого функцию WriteFile.

Серверный процесс может читать сообщения из созданного им канала Mailslot при помощи функции ReadFile. Заметим, что перед выполнением операции чтения следует проверить состояние канала Mailslot. Если в нем нет сообщений, то функцию ReadFile вызывать не следует. Для проверки состояния канала вы должны воспользоваться функцией GetMailslotInfo. С помощью функции SetMailslotInfo серверный процесс может изменить время ожидания для канала Mailslot уже после его создания.

**Исходники**

**Main.cpp**

#include <vcl.h>

#include <stdlib.h>

#include <StrUtils.hpp>

#pragma hdrstop

#include "Main.h"

#include "Podkluch.h"

#include "GameParam.h"

#include "About.h"

//---------------------------------------------------------------------------

#pragma package(smart\_init)

#pragma link "LMDCustomBevelPanel"

#pragma link "LMDCustomControl"

#pragma link "LMDCustomPanel"

#pragma link "LMDSimplePanel"

#pragma link "LMDBackPanel"

#pragma link "LMDCustomParentPanel"

#pragma link "LMDBaseControl"

#pragma link "LMDBaseGraphicControl"

#pragma link "LMDBaseLabel"

#pragma link "LMDControl"

#pragma link "LMDCustomSimpleLabel"

#pragma link "LMDSimpleLabel"

#pragma link "lmdcont"

#pragma link "LMDCustomComponent"

#pragma link "LMDCustomStatusBar"

#pragma link "LMDStatusBar"

#pragma link "LMDCustomPanelFill"

#pragma link "LMDPanelFill"

#pragma link "LMDCustomScrollBox"

#pragma link "LMDListBox"

#pragma link "LMDCustomFormFill"

#pragma link "LMDFormFill"

#pragma link "LMDWndProcComponent"

#pragma link "LMDBaseShape"

#pragma link "LMDShapeControl"

#pragma link "LMDCustomLabel"

#pragma link "LMDDrawEdge"

#pragma link "LMDLabel"

#pragma link "LMDBaseImage"

#pragma link "LMDCustomGroupBox"

#pragma link "LMDCustomLImage"

#pragma link "LMDGraphicControl"

#pragma link "LMDGroupBox"

#pragma link "LMDLImage"

#pragma link "LMDBaseGraphicButton"

#pragma link "LMDCustomShapeButton"

#pragma link "LMDShapeButton"

#pragma resource "\*.dfm"

TMainForm \*MainForm;

extern bool IsServer;

extern AnsiString Protocoli[];

extern int KolKub;

extern int MaxOch;

extern int choosenProt;

extern AnsiString ServerHostName;

extern AnsiString ServerHostIP;

extern bool IsHostName;

extern bool IsIgrayut;

extern int NomHoda;

extern int MyCurOch;

extern int MyWholeOch;

extern int HisWholeOch;

extern bool IsYaFixed;

extern bool IsHeFixed;

extern int HisWinPoPartiam;

extern int MyWinPoPartiam;

extern int NomPartia;

int \*MasPriem;

//---------------------------------------------------------------------------

\_\_fastcall TMainForm::TMainForm(TComponent\* Owner)

: TForm(Owner)

{

}

//---------------------------------------------------------------------------

void \_\_fastcall TMainForm::FormCreate(TObject \*Sender)

{

FirstTimeShow=true;

IsVisitApplicationBox=false;

}

//---------------------------------------------------------------------------

void \_\_fastcall TMainForm::FormActivate(TObject \*Sender)

{

if(FirstTimeShow)

{

FirstTimeShow=false;

InterfaceBezIgri();

menuBeginClick(this);

}

}

//---------------------------------------------------------------------------

void \_\_fastcall TMainForm::menuBeginClick(TObject \*Sender)

{

if (IsIgrayut)

{

menuStopClick(this);

if (IsIgrayut) return;

}

if (UstanParametrov()) //вызов 2-х форм для установки параметров

{

//----подключение----

switch(choosenProt)

{

case 0:

IPXConnect();

break;

case 1:

TCPIPConnect();

break;

case 2:

NetBiosConnect();

break;

case 3:

MailSlotConnect();

break;

case 4:

PipeConnect();

break;

}

if(IsServer) lbStat->Caption="Ожидается соединение с клиентом...";

}

}

//---------------------------------------------------------------------------

bool \_\_fastcall TMainForm::UstanParametrov(void)

{

int md1,md2;

for(;;)

{

md1=FormParamConnect->ShowModal();

FormParamConnect->Close();

if (md1==mrOk)

{

if (IsServer)

{

md2=FormGameParam->ShowModal();

if (md2==mrOk) return 1;

else continue;

}

else return 1;

}

else return 0;

}

}

void \_\_fastcall TMainForm::OtobrazhDannihSopernika(int \*MasPriem)

{

Graphics::TBitmap \*Bitmap2 = new Graphics::TBitmap();

AnsiString strCat;

if(IsServer)

{

strCat="\_2";

}

else

{

strCat="";

}

if(KolKub==1)

{

Bitmap2->LoadFromResourceName((unsigned int)HInstance,"KUB"+IntToStr(MasPriem[3])+strCat);

Kubiki2[0]->Picture->Bitmap=Bitmap2;

Kubiki2[0]->Left=MasPriem[4];

Kubiki2[0]->Top=MasPriem[5];

Kubiki2[0]->Visible=true;

}

else if (KolKub==2)

{

Bitmap2->LoadFromResourceName((unsigned int)HInstance,"KUB"+IntToStr(MasPriem[3])+strCat);

Kubiki2[0]->Picture->Bitmap=Bitmap2;

Kubiki2[0]->Left=MasPriem[4];

Kubiki2[0]->Top=MasPriem[5];

Kubiki2[0]->Visible=true;

Bitmap2->LoadFromResourceName((unsigned int)HInstance,"KUB"+IntToStr(MasPriem[6])+strCat);

Kubiki2[1]->Picture->Bitmap=Bitmap2;

Kubiki2[1]->Left=MasPriem[7];

Kubiki2[1]->Top=MasPriem[8];

Kubiki2[1]->Visible=true;

}

else if (KolKub==3)

{

Bitmap2->LoadFromResourceName((unsigned int)HInstance,"KUB"+IntToStr(MasPriem[3])+strCat);

Kubiki2[0]->Picture->Bitmap=Bitmap2;

Kubiki2[0]->Left=MasPriem[4];

Kubiki2[0]->Top=MasPriem[5];

Kubiki2[0]->Visible=true;

Bitmap2->LoadFromResourceName((unsigned int)HInstance,"KUB"+IntToStr(MasPriem[6])+strCat);

Kubiki2[1]->Picture->Bitmap=Bitmap2;

Kubiki2[1]->Left=MasPriem[7];

Kubiki2[1]->Top=MasPriem[8];

Kubiki2[1]->Visible=true;

Bitmap2->LoadFromResourceName((unsigned int)HInstance,"KUB"+IntToStr(MasPriem[9])+strCat);

Kubiki2[2]->Picture->Bitmap=Bitmap2;

Kubiki2[2]->Left=MasPriem[10];

Kubiki2[2]->Top=MasPriem[11];

Kubiki2[2]->Visible=true;

}

else

{

Bitmap2->LoadFromResourceName((unsigned int)HInstance,"KUB"+IntToStr(MasPriem[3])+strCat);

Kubiki2[0]->Picture->Bitmap=Bitmap2;

Kubiki2[0]->Left=MasPriem[4];

Kubiki2[0]->Top=MasPriem[5];

Kubiki2[0]->Visible=true;

Bitmap2->LoadFromResourceName((unsigned int)HInstance,"KUB"+IntToStr(MasPriem[6])+strCat);

Kubiki2[1]->Picture->Bitmap=Bitmap2;

Kubiki2[1]->Left=MasPriem[7];

Kubiki2[1]->Top=MasPriem[8];

Kubiki2[1]->Visible=true;

Bitmap2->LoadFromResourceName((unsigned int)HInstance,"KUB"+IntToStr(MasPriem[9])+strCat);

Kubiki2[2]->Picture->Bitmap=Bitmap2;

Kubiki2[2]->Left=MasPriem[10];

Kubiki2[2]->Top=MasPriem[11];

Kubiki2[2]->Visible=true;

Bitmap2->LoadFromResourceName((unsigned int)HInstance,"KUB"+IntToStr(MasPriem[12])+strCat);

Kubiki2[3]->Picture->Bitmap=Bitmap2;

Kubiki2[3]->Left=MasPriem[13];

Kubiki2[3]->Top=MasPriem[14];

Kubiki2[3]->Visible=true;

}

HisWholeOch=MasPriem[2];

lbHisWholeOch->Caption=IntToStr(MasPriem[2]);

delete Bitmap2;

}

//-----------пришли данные соперника--------------------------------------------

void \_\_fastcall TMainForm::Priem(int \*MasPriem)

{

if(MasPriem[0]==0) //первый ответ сервера клиенту

{

AnsiString mess;

mess="Ваш соперник выбрал следующие параметры игры:\nКоличество кубиков: "+IntToStr(MasPriem[1])+"\nМаксимальное количество очков: "+IntToStr(MasPriem[2]);

IsVisitApplicationBox=true;

Application->MessageBoxA(mess.c\_str(),"Параметры игры",MB\_OK);

KolKub=MasPriem[1];

MaxOch=MasPriem[2];

InterfaceBeginNewGame();

ChangeInterfacePerehodHoda(0);

IsVisitApplicationBox=false;

}

else if (MasPriem[0]==1) //соперник бросил

{

shapeBrosil->Brush->Color=0x0000CE00;

OtobrazhDannihSopernika(MasPriem);

if(!IsServer)

{

NomHoda++;

}

enum TypeDannihForTabl DannieForTabl;

DannieForTabl=hebrosil;

ZapolnTabl(DannieForTabl);

if(IsYaFixed && HisWholeOch>MyWholeOch && HisWholeOch<=MaxOch)

{

ViProigrali();

}

else if (HisWholeOch>MaxOch)

{

lbHisWholeOch->Font->Color=clRed;

ViViigrali();

}

else if(IsYaFixed && HisWholeOch<=MyWholeOch)

{

ChangeInterfacePerehodHoda(0);

}

else

{

ChangeInterfacePerehodHoda(1);

}

}

else if (MasPriem[0]==2) //соперник зафиксировал

{

IsHeFixed=true;

lbHisWholeOch->Font->Color=clRed;

shapeFixed->Brush->Color=0x000000CE;

if(!IsServer)

{

NomHoda++;

}

enum TypeDannihForTabl DannieForTabl;

DannieForTabl=hefixed;

ZapolnTabl(DannieForTabl);

if(IsYaFixed && MyWholeOch==HisWholeOch)

{

Nichia();

}

else if(MyWholeOch>HisWholeOch)

{

ViViigrali();

}

else

{

ChangeInterfacePerehodHoda(1);

}

}

else if (MasPriem[0]==3) //соперник разорвал соединение

{

IsVisitApplicationBox=true;

Application->MessageBoxA("Ваш соперник разорвал соединение.\nИгра окончена.","Разрыв соединения",MB\_OK);

switch(choosenProt)

{

case 0:

IPXCloseConnection();

break;

case 1:

TCPIPCloseConnection();

break;

case 2:

NetBiosCloseConnection();

break;

case 3:

MailSlotCloseConnection();

break;

case 4:

PipeCloseConnection();

break;

}

for (int i=0;i<KolKub;i++)

{

delete Kubiki1[i];

delete Kubiki2[i];

}

IsIgrayut=false;

InterfaceBezIgri();

IsVisitApplicationBox=false;

}

delete [] MasPriem;

}

//----------------процедура заполнения таблицы----------------------------------

void \_\_fastcall TMainForm::ZapolnTabl(enum TypeDannihForTabl DannieForTabl)

{

if(IsServer)

{

if(DannieForTabl==yabrosil)

{

listboxOchki->Items->Add(" "+IntToStr(NomHoda)+"; "+IntToStr(MyCurOch));

listboxOchki->TopIndex=NomHoda-1;

if(NomHoda==9)

{

listboxOchki->Width=236;

listboxOchki->Left=14;

}

}

else if (DannieForTabl==yafixed)

{

listboxOchki->Items->Add(" "+IntToStr(NomHoda)+"; 0");

listboxOchki->TopIndex=NomHoda-1;

if(NomHoda==9)

{

listboxOchki->Width=236;

listboxOchki->Left=14;

}

}

else if (DannieForTabl==hebrosil)

{

listboxOchki->SetItemPart(NomHoda-1,2," "+IntToStr(MasPriem[1]));

listboxOchki->TopIndex=NomHoda-1;

}

else

{

listboxOchki->SetItemPart(NomHoda-1,2," 0");

listboxOchki->TopIndex=NomHoda-1;

}

}

else

{

if(DannieForTabl==yabrosil)

{

listboxOchki->SetItemPart(NomHoda-1,1," "+IntToStr(MyCurOch));

listboxOchki->TopIndex=NomHoda-1;

}

else if (DannieForTabl==yafixed)

{

listboxOchki->SetItemPart(NomHoda-1,1," 0");

listboxOchki->TopIndex=NomHoda-1;

}

else if (DannieForTabl==hebrosil)

{

listboxOchki->Items->Add(" "+ IntToStr(NomHoda)+";; "+IntToStr(MasPriem[1]));

listboxOchki->TopIndex=NomHoda-1;

if(NomHoda==9)

{

listboxOchki->Width=236;

listboxOchki->Left=14;

}

}

else

{

listboxOchki->Items->Add(" "+ IntToStr(NomHoda)+";; 0");

listboxOchki->TopIndex=NomHoda-1;

if(NomHoda==9)

{

listboxOchki->Width=236;

listboxOchki->Left=14;

}

}

}

}

//------------------------------Результаты--------------------------------------

void \_\_fastcall TMainForm::ViProigrali(void)

{

IsVisitApplicationBox=true;

HisWinPoPartiam++;

Application->MessageBoxA("Вы проиграли!\nИграем дальше!","Поражение",MB\_OK+MB\_ICONEXCLAMATION);

InterfaceIgraemDalshe();

IsVisitApplicationBox=false;

}

//---------------------------------

void \_\_fastcall TMainForm::ViViigrali(void)

{

IsVisitApplicationBox=true;

MyWinPoPartiam++;

Application->MessageBoxA("Вы выиграли!\nИграем дальше!","Победа!!",MB\_OK+MB\_ICONEXCLAMATION);

InterfaceIgraemDalshe();

IsVisitApplicationBox=false;

}

//----------------------------------

void \_\_fastcall TMainForm::Nichia(void)

{

IsVisitApplicationBox=true;

MyWinPoPartiam++;

HisWinPoPartiam++;

Application->MessageBoxA("Вы сыграли вничью!\nИграем дальше!","Ничья",MB\_OK+MB\_ICONEXCLAMATION);

InterfaceIgraemDalshe();

IsVisitApplicationBox=false;

}

//----------------------------------------------------------------------------

void \_\_fastcall TMainForm::menuOProgrClick(TObject \*Sender)

{

FormAbout->ShowModal();

}

//----------------------------Интерфейсы---------------------------------------

void \_\_fastcall TMainForm::InterfaceBezIgri(void)

{

MyWholeOch=0;

HisWholeOch=0;

MyCurOch=0;

NomHoda=0;

IsYaFixed=false;

IsHeFixed=false;

NomPartia=0;

MyWinPoPartiam=0;

HisWinPoPartiam=0;

ServerHostName="";

ServerHostIP="";

IsIgrayut=false;

menuStop->Enabled=false;

panParam->Visible=false;

lbNomPartia->Visible=false;

panSchetPoPartiam->Visible=false;

gbPole1->Top=15;

gbPole2->Top=15;

gbTabl->Top=15;

this->Height=400;

this->Caption="Игровое поле";

btBrosit->Enabled=false;

btFix->Enabled=false;

btBrosit->Color=clCream;

btFix->Color=clCream;

shapeBrosil->Brush->Color=clSilver;

shapeFixed->Brush->Color=clSilver;

shapeBrosil->Caption->Font->Color=clWhite;

shapeFixed->Caption->Font->Color=clWhite;

lbMyWholeOch->Caption="0";

lbHisWholeOch->Caption="0";

lbMaxOch2->Caption="";

listboxOchki->Clear();

listboxOchki->Width=218;

listboxOchki->Left=23;

AnsiString tmp=AnsiQuotedStr("Игра->Начать новую игру",'"');

lbStat->Caption="Для начала новой игры выберете пункт меню "+tmp;

}

//--------------------------------------------------------------

void \_\_fastcall TMainForm::InterfaceBeginNewGame(void)

{

menuStop->Enabled=true;

if(IsServer)

{

Caption=Caption+" Сервер";

imStolServer->Parent=panPole1;

imStolClient->Parent=panPole2;

}

else

{

Caption=Caption+" Клиент";

imStolClient->Parent=panPole1;

imStolServer->Parent=panPole2;

}

for (int i=0;i<KolKub;i++)

{

Kubiki1[i] = new TImage(this);

Kubiki1[i]->Parent=panPole1;

Kubiki1[i]->Transparent=true;

Kubiki2[i] = new TImage(this);

Kubiki2[i]->Parent=panPole2;

Kubiki2[i]->Transparent=true;

}

panParam->Visible=true;

lbNomPartia->Visible=true;

panSchetPoPartiam->Visible=true;

gbPole1->Top=80;

gbTabl->Top=80;

gbPole2->Top=80;

this->Height=470;

lbKolKub->Caption=IntToStr(KolKub);

lbMaxOch->Caption=IntToStr(MaxOch);

lbMaxOch2->Caption=IntToStr(MaxOch);

lbNomPartia->Caption="Партия № 1";

lbSchetPoPartiam->Caption="0 : 0";

NomPartia=1;

IsIgrayut=true;

}

//---------------------------------------------------------

void \_\_fastcall TMainForm::ChangeInterfacePerehodHoda(bool HozhuYa)

{

if(HozhuYa)

{

btBrosit->Enabled=true;

btFix->Enabled=true;

btBrosit->Color=clCream;

btFix->Color=clCream;

shapeBrosil->Caption->Font->Color=clWhite;

shapeFixed->Caption->Font->Color=clWhite;

lbStat->Caption="Твой ход...";

if (MyWholeOch>MaxOch-KolKub)

{

IsVisitApplicationBox=true;

Application->MessageBoxA("Единственно верный вариант для Вас сейчас - зафиксировать очки!","Предупреждение",MB\_OK+MB\_ICONEXCLAMATION);

IsVisitApplicationBox=false;

}

}

else

{

btBrosit->Enabled=false;

btFix->Enabled=false;

shapeBrosil->Brush->Color=clSilver;

shapeFixed->Brush->Color=clSilver;

shapeBrosil->Caption->Font->Color=clBlack;

shapeFixed->Caption->Font->Color=clBlack;

lbStat->Caption="Ход соперника...";

}

}

//--------------------------------------------------------------------------

void \_\_fastcall TMainForm::InterfaceIgraemDalshe(void)

{

NomPartia++;

MyWholeOch=0;

HisWholeOch=0;

MyCurOch=0;

NomHoda=0;

IsYaFixed=false;

IsHeFixed=false;

IsIgrayut=true;

lbNomPartia->Caption="Партия № "+IntToStr(NomPartia);

lbSchetPoPartiam->Caption=IntToStr(MyWinPoPartiam)+" : "+IntToStr(HisWinPoPartiam);

listboxOchki->Clear();

listboxOchki->Width=218;

listboxOchki->Left=23;

lbMyWholeOch->Caption="0";

lbHisWholeOch->Caption="0";

lbMyWholeOch->Font->Color=clGreen;

lbHisWholeOch->Font->Color=clGreen;

btBrosit->Color=clCream;

btFix->Color=clCream;

shapeBrosil->Brush->Color=clSilver;

shapeFixed->Brush->Color=clSilver;

for (int i=0;i<KolKub;i++)

{

Kubiki1[i]->Visible=false;

Kubiki2[i]->Visible=false;

}

if(IsServer)

{

ChangeInterfacePerehodHoda(1);

}

else

{

ChangeInterfacePerehodHoda(0);

}

}

//-----------------------Завершение игры---------------------------------------

void \_\_fastcall TMainForm::menuStopClick(TObject \*Sender)

{

IsVisitApplicationBox=true;

if(Application->MessageBoxA("Вы действительно хотите завершить игру?","Завершение игры",MB\_YESNO)==IDYES)

{

for (int i=0;i<KolKub;i++)

{

delete Kubiki1[i];

delete Kubiki2[i];

}

switch(choosenProt)

{

case 0:

IPXSend("3;");

IPXCloseConnection();

break;

case 1:

TCPIPSend("3;");

TCPIPCloseConnection();

break;

case 2:

NetBiosSend("3;");

NetBiosCloseConnection();

break;

case 3:

MailSlotSend("3;");

MailSlotCloseConnection();

break;

case 4:

PipeSend("3;");

PipeCloseConnection();

break;

}

IsIgrayut=false;

InterfaceBezIgri();

}

IsVisitApplicationBox=false;

}

//---------------------------------------------------------------------------

void \_\_fastcall TMainForm::menuQuitClick(TObject \*Sender)

{

Close();

}

//---------------------------------------------------------------------------

void \_\_fastcall TMainForm::FormCloseQuery(TObject \*Sender, bool &CanClose)

{

if (IsIgrayut)

{

menuStopClick(this);

if(IsIgrayut) CanClose=false;

}

}

//-------------------служебные функции преобразования типов---------------------

char\* \_\_fastcall TMainForm::DannieToChar(int Type, int MyCurOch, int MyWholeOch,

int ZnachKub1,int XKub1,int YKub1,

int ZnachKub2,int XKub2,int YKub2,

int ZnachKub3,int XKub3,int YKub3,

int ZnachKub4,int XKub4,int YKub4

)

{

AnsiString strTmp=IntToStr(Type)+";"+IntToStr(MyCurOch)+";"+IntToStr(MyWholeOch)+";"+

IntToStr(ZnachKub1)+";"+IntToStr(XKub1)+";"+IntToStr(YKub1)+";"+

IntToStr(ZnachKub2)+";"+IntToStr(XKub2)+";"+IntToStr(YKub2)+";"+

IntToStr(ZnachKub3)+";"+IntToStr(XKub3)+";"+IntToStr(YKub3)+";"+

IntToStr(ZnachKub4)+";"+IntToStr(XKub4)+";"+IntToStr(YKub4)+";";

return strTmp.c\_str();

}

//---------------------------------------------------------------------------

int\* \_\_fastcall TMainForm::CharToMas(char\* charPriem)

{

AnsiString strPriem=charPriem;

MasPriem=new int[15];

int y=0;

int PrevDelim=0;

int CurDelim;

for(int i=1;i<=strPriem.Length();i++)

{

if(IsDelimiter(";",strPriem,i))

{

MasPriem[y++]=StrToInt(MidStr(strPriem,PrevDelim+1,i-PrevDelim-1));

PrevDelim=i;

}

}

return MasPriem;

}

//Бросить---------------------------------------------------------------------------

//Закогнчить--------------------------------------------------------------------------

void \_\_fastcall TMainForm::btBrosit2Click(TObject \*Sender)

{

Graphics::TBitmap \*Bitmap1 = new Graphics::TBitmap();

char strSend[70];

AnsiString strCat;

if(IsServer)

{

NomHoda++;

strCat="";

}

else

{

strCat="\_2";

}

if(KolKub==1)

{

int ZnachKub, XKub, YKub ;

randomize();

ZnachKub = random(6)+1;

XKub=random(126)+14;

YKub=random(126)+14;

Bitmap1->LoadFromResourceName((unsigned int)HInstance,"KUB"+IntToStr(ZnachKub)+strCat);

Kubiki1[0]->Picture->Bitmap=Bitmap1;

Kubiki1[0]->Left=XKub;

Kubiki1[0]->Top=YKub;

Kubiki1[0]->Visible=true;

MyCurOch=ZnachKub;

MyWholeOch=MyWholeOch+MyCurOch;

strcpy(strSend,DannieToChar(1, MyCurOch, MyWholeOch, ZnachKub, XKub, YKub));

}

else if (KolKub==2)

{

int ZnachKub1, XKub1, YKub1, ZnachKub2, XKub2, YKub2;

randomize();

ZnachKub1 = random(6)+1;

XKub1=random(126)+14;

YKub1=random(52)+14;

Bitmap1->LoadFromResourceName((unsigned int)HInstance,"KUB"+IntToStr(ZnachKub1)+strCat);

Kubiki1[0]->Picture->Bitmap=Bitmap1;

Kubiki1[0]->Left=XKub1;

Kubiki1[0]->Top=YKub1;

Kubiki1[0]->Visible=true;

ZnachKub2 = random(6)+1;

XKub2=random(126)+14;

YKub2=random(52)+88;

Bitmap1->LoadFromResourceName((unsigned int)HInstance,"KUB"+IntToStr(ZnachKub2)+strCat);

Kubiki1[1]->Picture->Bitmap=Bitmap1;

Kubiki1[1]->Left=XKub2;

Kubiki1[1]->Top=YKub2;

Kubiki1[1]->Visible=true;

MyCurOch=ZnachKub1+ZnachKub2;

MyWholeOch=MyWholeOch+MyCurOch;

strcpy(strSend,DannieToChar(1, MyCurOch, MyWholeOch, ZnachKub1, XKub1, YKub1, ZnachKub2, XKub2, YKub2));

}

else if (KolKub==3)

{

int ZnachKub1, XKub1, YKub1, ZnachKub2, XKub2, YKub2, ZnachKub3, XKub3, YKub3;

randomize();

ZnachKub1 = random(6)+1;

XKub1=random(126)+14;

YKub1=random(27)+14;

Bitmap1->LoadFromResourceName((unsigned int)HInstance,"KUB"+IntToStr(ZnachKub1)+strCat);

Kubiki1[0]->Picture->Bitmap=Bitmap1;

Kubiki1[0]->Left=XKub1;

Kubiki1[0]->Top=YKub1;

Kubiki1[0]->Visible=true;

ZnachKub2 = random(6)+1;

XKub2=random(126)+14;

YKub2=random(27)+63;

Bitmap1->LoadFromResourceName((unsigned int)HInstance,"KUB"+IntToStr(ZnachKub2)+strCat);

Kubiki1[1]->Picture->Bitmap=Bitmap1;

Kubiki1[1]->Left=XKub2;

Kubiki1[1]->Top=YKub2;

Kubiki1[1]->Visible=true;

ZnachKub3 = random(6)+1;

XKub3=random(126)+14;

YKub3=random(28)+112;

Bitmap1->LoadFromResourceName((unsigned int)HInstance,"KUB"+IntToStr(ZnachKub3)+strCat);

Kubiki1[2]->Picture->Bitmap=Bitmap1;

Kubiki1[2]->Left=XKub3;

Kubiki1[2]->Top=YKub3;

Kubiki1[2]->Visible=true;

MyCurOch=ZnachKub1+ZnachKub2+ZnachKub3;

MyWholeOch=MyWholeOch+MyCurOch;

strcpy(strSend,DannieToChar(1, MyCurOch, MyWholeOch, ZnachKub1, XKub1, YKub1, ZnachKub2, XKub2, YKub2, ZnachKub3, XKub3, YKub3));

}

else

{

int ZnachKub1, XKub1, YKub1, ZnachKub2, XKub2, YKub2, ZnachKub3, XKub3, YKub3, ZnachKub4, XKub4, YKub4;

randomize();

ZnachKub1 = random(6)+1;

XKub1=random(52)+14;

YKub1=random(52)+14;

Bitmap1->LoadFromResourceName((unsigned int)HInstance,"KUB"+IntToStr(ZnachKub1)+strCat);

Kubiki1[0]->Picture->Bitmap=Bitmap1;

Kubiki1[0]->Left=XKub1;

Kubiki1[0]->Top=YKub1;

Kubiki1[0]->Visible=true;

ZnachKub2 = random(6)+1;

XKub2=random(52)+88;

YKub2=random(52)+14;

Bitmap1->LoadFromResourceName((unsigned int)HInstance,"KUB"+IntToStr(ZnachKub2)+strCat);

Kubiki1[1]->Picture->Bitmap=Bitmap1;

Kubiki1[1]->Left=XKub2;

Kubiki1[1]->Top=YKub2;

Kubiki1[1]->Visible=true;

ZnachKub3 = random(6)+1;

XKub3=random(52)+14;

YKub3=random(52)+88;

Bitmap1->LoadFromResourceName((unsigned int)HInstance,"KUB"+IntToStr(ZnachKub3)+strCat);

Kubiki1[2]->Picture->Bitmap=Bitmap1;

Kubiki1[2]->Left=XKub3;

Kubiki1[2]->Top=YKub3;

Kubiki1[2]->Visible=true;

ZnachKub4 = random(6)+1;

XKub4=random(52)+88;

YKub4=random(52)+88;

Bitmap1->LoadFromResourceName((unsigned int)HInstance,"KUB"+IntToStr(ZnachKub4)+strCat);

Kubiki1[3]->Picture->Bitmap=Bitmap1;

Kubiki1[3]->Left=XKub4;

Kubiki1[3]->Top=YKub4;

Kubiki1[3]->Visible=true;

MyCurOch=ZnachKub1+ZnachKub2+ZnachKub3+ZnachKub4;

MyWholeOch=MyWholeOch+MyCurOch;

strcpy(strSend,DannieToChar(1, MyCurOch, MyWholeOch, ZnachKub1, XKub1, YKub1, ZnachKub2, XKub2, YKub2,ZnachKub3, XKub3, YKub3, ZnachKub4, XKub4, YKub4));

}

switch(choosenProt)

{

case 0:

IPXSend(strSend);

break;

case 1:

TCPIPSend(AnsiString(strSend));

break;

case 2:

NetBiosSend(strSend);

break;

case 3:

MailSlotSend(strSend);

break;

case 4:

PipeSend(strSend);

break;

}

enum TypeDannihForTabl DannieForTabl;

DannieForTabl=yabrosil;

ZapolnTabl(DannieForTabl);

lbMyWholeOch->Caption=MyWholeOch;

if (MyWholeOch>MaxOch)

{

lbMyWholeOch->Font->Color=clRed;

ViProigrali();

}

else if(IsHeFixed && MyWholeOch>HisWholeOch && MyWholeOch<=MaxOch)

{

ViViigrali();

}

else if(IsHeFixed && MyWholeOch<=HisWholeOch)

{

ChangeInterfacePerehodHoda(1);

}

else

{

ChangeInterfacePerehodHoda(0);

}

delete Bitmap1;

}

//---------------------------------------------------------------------------

void \_\_fastcall TMainForm::btFix2Click(TObject \*Sender)

{

IsYaFixed=true;

if(IsServer)

{

NomHoda++;

}

btFix->Color=clRed;

btBrosit->Color=clCream;

lbMyWholeOch->Font->Color=clRed;

switch(choosenProt)

{

case 0:

IPXSend("2;");

break;

case 1:

TCPIPSend("2;");

break;

case 2:

NetBiosSend("2;");

break;

case 3:

MailSlotSend("2;");

break;

case 4:

PipeSend("2;");

break;

}

enum TypeDannihForTabl DannieForTabl;

DannieForTabl=yafixed;

ZapolnTabl(DannieForTabl);

if(IsHeFixed && MyWholeOch==HisWholeOch)

{

Nichia();

}

else if(MyWholeOch<HisWholeOch)

{

ViProigrali();

}

else

{

ChangeInterfacePerehodHoda(0);

}

}

//---------------------------------------------------------------------------

**Ipx.cpp**

//---------------------------------------------------------------------------

#pragma hdrstop

#include <winsock.h>

#include <wsipx.h>

#include "IPX.h"

#include "Main.h"

#include "Podkluch.h"

//---------------------------------------------------------------------------

#pragma package(smart\_init)

#define IPX\_SERVER\_SOCKET 0x4504

#define IPX\_CLIENT\_SOCKET 0x4514

#define WSA\_NETEVENT (WM\_USER+2)

SOCKET IPXSocket;

sockaddr\_ipx CallAddress;

sockaddr\_ipx OurAddress;

extern bool IsServer;

extern int KolKub;

extern int MaxOch;

//-----------------------Подключение--------------------------------------------

void \_\_fastcall TMainForm::IPXConnect(void)

{

WSADATA WSAData;

int rc, OptVal=1;

rc = WSAStartup(MAKEWORD(2,0), &WSAData);

if (rc != 0)

{

Application->MessageBox("Ошибка инициализации WSAStartup","Ошибка",MB\_OK+MB\_ICONHAND);

Application->Terminate();

}

IPXSocket = socket(AF\_IPX, SOCK\_DGRAM, NSPROTO\_IPX);

if (IPXSocket == INVALID\_SOCKET)

{

Application->MessageBox("Протокол IPX не установлен","Ошибка",MB\_OK+MB\_ICONHAND);

Application->Terminate();

}

memset(&OurAddress, 0, sizeof(OurAddress));

OurAddress.sa\_family = AF\_IPX;

if(IsServer)

OurAddress.sa\_socket = htons((unsigned short)(IPX\_SERVER\_SOCKET));

else

OurAddress.sa\_socket = htons((unsigned short)(IPX\_CLIENT\_SOCKET));

if (bind(IPXSocket, (LPSOCKADDR)&OurAddress, sizeof(sockaddr\_ipx)) == SOCKET\_ERROR)

{

Application->MessageBox("Адресная ошибка IPX","Ошибка",MB\_OK+MB\_ICONHAND);

closesocket(IPXSocket);

IPXSocket = INVALID\_SOCKET;

Application->Terminate();

}

rc = setsockopt(IPXSocket, SOL\_SOCKET, SO\_BROADCAST, (char \*)&OptVal, sizeof(OptVal));

if (rc == SOCKET\_ERROR)

{

rc = WSAGetLastError();

Application->MessageBox("Ошибка setsockopt","Ошибка",MB\_OK+MB\_ICONHAND);

closesocket(IPXSocket);

IPXSocket = INVALID\_SOCKET;

Application->Terminate();

}

if(!IsServer)

{

memset(&CallAddress, 0, sizeof(CallAddress));

CallAddress.sa\_family = AF\_IPX;

CallAddress.sa\_socket = htons(IPX\_SERVER\_SOCKET);

CallAddress.sa\_nodenum[0] = 0xff;

CallAddress.sa\_nodenum[1] = 0xff;

CallAddress.sa\_nodenum[2] = 0xff;

CallAddress.sa\_nodenum[3] = 0xff;

CallAddress.sa\_nodenum[4] = 0xff;

CallAddress.sa\_nodenum[5] = 0xff;

rc = sendto(IPXSocket, "connect", 8, 0, (PSOCKADDR)&CallAddress, sizeof(SOCKADDR));

if (rc == SOCKET\_ERROR)

{

rc = WSAGetLastError();

Application->MessageBox("Ошибка sendto IPX","Ошибка",MB\_OK+MB\_ICONHAND);

closesocket(IPXSocket);

IPXSocket = INVALID\_SOCKET;

Application->Terminate();

}

}

rc = WSAAsyncSelect(IPXSocket, Handle, WSA\_NETEVENT, FD\_READ);

if (rc != 0)

{

Application->MessageBox("Ошибка WSAAsyncSelect","Ошибка",MB\_OK+MB\_ICONHAND);

closesocket(IPXSocket);

IPXSocket = INVALID\_SOCKET;

Application->Terminate();

}

WindowProc = IPXWndProc;

}

//------------------------------прием---------------------------------------

void \_\_fastcall TMainForm::IPXWndProc(Messages::TMessage &Message)

{

int rc, l=sizeof(CallAddress);

char Buf[80];

if (Message.Msg == WSA\_NETEVENT)

{

if (Message.LParam == FD\_READ)

{

rc = recvfrom((SOCKET)Message.WParam, Buf, sizeof(Buf)-1, 0, (PSOCKADDR)&CallAddress, &l);

if (rc == SOCKET\_ERROR)

{

rc = WSAGetLastError();

Application->MessageBox("Ошибка recvfrom IPX","Ошибка",MB\_OK+MB\_ICONHAND);

closesocket(IPXSocket);

IPXSocket = INVALID\_SOCKET;

Application->Terminate();

}

if (rc >= 1)

{

if(!strcmp(Buf,"connect"))

{

char \*strPered=DannieToChar(0,KolKub,MaxOch);

IPXSend(strPered);

InterfaceBeginNewGame();

ChangeInterfacePerehodHoda(1);

}

else

{

if(IsVisitApplicationBox) return;

Priem(CharToMas(Buf));

}

}

}

}

WndProc(Message);

}

//-------------------------отправка---------------------------------------------

void \_\_fastcall TMainForm::IPXSend (char\* Buf)

{

int rc;

rc = sendto(IPXSocket, Buf, strlen(Buf)+1, 0, (PSOCKADDR)&CallAddress, sizeof(SOCKADDR));

if (rc == SOCKET\_ERROR)

{

rc = WSAGetLastError();

Application->MessageBox("Ошибка sendto IPX","Ошибка",MB\_OK+MB\_ICONHAND);

closesocket(IPXSocket);

IPXSocket = INVALID\_SOCKET;

Application->Terminate();

}

}

//-------------------------разрыв---------------------------------------------

void \_\_fastcall TMainForm::IPXCloseConnection(void)

{

closesocket(IPXSocket);

WSACleanup();

}

**TCP.cpp**

//---------------------------------------------------------------------------

#pragma hdrstop

#include "TCPIP.h"

#include "Main.h"

#include "Podkluch.h"

//---------------------------------------------------------------------------

#pragma package(smart\_init)

extern bool IsServer;

extern bool IsHostName;

extern AnsiString ServerHostName;

extern AnsiString ServerHostIP;

extern int KolKub;

extern int MaxOch;

//----------Подключение---------------------------------------------------

void \_\_fastcall TMainForm::TCPIPConnect(void)

{

if (IsServer==true)

{

ClientSocket1->Close();

ServerSocket1->Open();

}

else

{

ServerSocket1->Close();

if (IsHostName)

ClientSocket1->Host=ServerHostName;

else

ClientSocket1->Host=ServerHostIP;

ClientSocket1->Open();

}

}

//---------Accept---------------------------------------------------------------

void \_\_fastcall TMainForm::ServerSocket1Accept(TObject \*Sender,

TCustomWinSocket \*Socket)

{

char \*strPered=DannieToChar(0,KolKub,MaxOch);

TCPIPSend(AnsiString(strPered));

InterfaceBeginNewGame();

ChangeInterfacePerehodHoda(1);

}

//----------Отправка------------------------------------------------------------

void \_\_fastcall TMainForm::TCPIPSend(AnsiString strSend)

{

if(IsServer)

ServerSocket1->Socket->Connections[0]->SendText(strSend);

else

ClientSocket1->Socket->SendText(strSend);

}

//-------------Получение--------------------------------------------------------

void \_\_fastcall TMainForm::ServerSocket1ClientRead(TObject \*Sender,

TCustomWinSocket \*Socket)

{

if(IsVisitApplicationBox) return;

AnsiString strPriem=Socket->ReceiveText();

Priem(CharToMas(strPriem.c\_str()));

}

//---------------------------------------------------------------------------

void \_\_fastcall TMainForm::ServerSocket1ClientConnect(TObject \*Sender,

TCustomWinSocket \*Socket)

{

//

}

//---------------------------------------------------------------------------

void \_\_fastcall TMainForm::ClientSocket1Connect(TObject \*Sender,

TCustomWinSocket \*Socket)

{

//StatusBar1->Panels->Items[0]->Text = "Connect to: " + Socket->RemoteHost;

}

//---------------------------------------------------------------------------

void \_\_fastcall TMainForm::ServerSocket1Listen(TObject \*Sender,

TCustomWinSocket \*Socket)

{

//StatusBar1->Panels->Items[0]->Text="Listening...";

}

//--------------Разрыв соединения-------------------------------------------

void \_\_fastcall TMainForm::TCPIPCloseConnection(void)

{

ServerSocket1->Close();

ClientSocket1->Close();

}

//-----------Обработка ошибок---------------------------------------------------

void \_\_fastcall TMainForm::ClientSocket1Error(TObject \*Sender,

TCustomWinSocket \*Socket, TErrorEvent ErrorEvent, int &ErrorCode)

{

if(ErrorEvent==eeConnect)

{

ErrorCode=0;

Application->MessageBox("Не удалось осуществить соединение с сервером.\nУбедитесь, что сервер ожидает запроса клиента.","Ошибка",MB\_OK+MB\_ICONHAND);

}

else

{

Application->MessageBox("Ошибка TCP/IP","Ошибка",MB\_OK+MB\_ICONHAND);

Application->Terminate();

}

}

void \_\_fastcall TMainForm::ServerSocket1ClientError(TObject \*Sender,

TCustomWinSocket \*Socket, TErrorEvent ErrorEvent, int &ErrorCode)

{

//

}

//-----------------------------------------------------------------------------

**Netbios.cpp**

//---------------------------------------------------------------------------

#pragma hdrstop

#include <winsock.h>

#include <wsnetbs.h>

#include <stdio.h>

#include "NetBios.h"

#include "Main.h"

#include "Podkluch.h"

#define NETBIOS\_SERVER\_NAME "Kosti Server "

#define NETBIOS\_CLIENT\_NAME "Kosti Client "

#define WSA\_ACCEPT (WM\_USER + 1)

#define WSA\_NETEVENT (WM\_USER+2)

//---------------------------------------------------------------------------

#pragma package(smart\_init)

SOCKET NetBiosSocket;

sockaddr\_nb OurAddress;

sockaddr\_nb CallAddress;

extern bool IsServer;

extern int KolKub;

extern int MaxOch;

//----------------------------Подключение---------------------------------------

void \_\_fastcall TMainForm::NetBiosConnect(void)

{

WSADATA WSAData;

int rc;

rc = WSAStartup(MAKEWORD(2,0), &WSAData);

if (rc != 0)

{

Application->MessageBox("Ошибка инициализации WSAStartup","Ошибка",MB\_OK+MB\_ICONHAND);

Application->Terminate();

}

int Lana;

for (Lana=0;Lana<1000;Lana++)

{

NetBiosSocket = socket(AF\_NETBIOS, SOCK\_SEQPACKET, -Lana);

if (NetBiosSocket == INVALID\_SOCKET)

{

continue;

}

memset(&OurAddress, 0, sizeof(OurAddress));

OurAddress.snb\_family = AF\_NETBIOS;

OurAddress.snb\_type = NETBIOS\_UNIQUE\_NAME;

if(IsServer)

strcpy(OurAddress.snb\_name,NETBIOS\_SERVER\_NAME);

else

strcpy(OurAddress.snb\_name,NETBIOS\_CLIENT\_NAME);

if (bind(NetBiosSocket, (LPSOCKADDR)&OurAddress, sizeof(sockaddr\_nb)) == SOCKET\_ERROR)

{

closesocket(NetBiosSocket);

NetBiosSocket = INVALID\_SOCKET;

continue;

}

else break;

}

if(Lana==1000)

{

Application->MessageBox("Не определена Lana","Ошибка",MB\_OK+MB\_ICONHAND);

Application->Terminate();

}

if(IsServer)

{

if(listen(NetBiosSocket, 1) == SOCKET\_ERROR )

{

Application->MessageBox("Listen Error","Ошибка",MB\_OK+MB\_ICONHAND);

closesocket(NetBiosSocket);

Application->Terminate();

}

rc = WSAAsyncSelect(NetBiosSocket, Handle, WSA\_ACCEPT, FD\_ACCEPT);

if (rc != 0)

{

Application->MessageBox("Ошибка WSAAsyncSelect","Ошибка",MB\_OK+MB\_ICONHAND);

closesocket(NetBiosSocket);

NetBiosSocket = INVALID\_SOCKET;

Application->Terminate();

}

}

else

{

memset(&CallAddress, 0, sizeof(CallAddress));

CallAddress.snb\_family = AF\_NETBIOS;

CallAddress.snb\_type = NETBIOS\_UNIQUE\_NAME;

strcpy(CallAddress.snb\_name,NETBIOS\_SERVER\_NAME);

if(connect(NetBiosSocket , (PSOCKADDR )&CallAddress, sizeof(CallAddress)) < 0)

{

Application->MessageBox("Connect Error","Ошибка",MB\_OK+MB\_ICONHAND);

closesocket(NetBiosSocket);

Application->Terminate();

}

rc = WSAAsyncSelect(NetBiosSocket, Handle, WSA\_NETEVENT, FD\_READ);

if (rc != 0)

{

Application->MessageBox("Ошибка WSAAsyncSelect","Ошибка",MB\_OK+MB\_ICONHAND);

closesocket(NetBiosSocket);

NetBiosSocket = INVALID\_SOCKET;

Application->Terminate();

}

}

WindowProc = NetBiosWndProc;

}

//-------------------Прием и accept---------------------------------------------

void \_\_fastcall TMainForm::NetBiosWndProc(Messages::TMessage &Message)

{

int rc;

char Buf[80];

if (Message.Msg == WSA\_NETEVENT)

{

if (Message.LParam == FD\_READ)

{

rc = recv((SOCKET)Message.WParam, Buf, sizeof(Buf)-1, 0);

if (rc == SOCKET\_ERROR)

{

rc = WSAGetLastError();

Application->MessageBox("Ошибка recv NetBios","Ошибка",MB\_OK+MB\_ICONHAND);

closesocket(NetBiosSocket);

NetBiosSocket = INVALID\_SOCKET;

Application->Terminate();

}

if (rc >= 1)

{

if(IsVisitApplicationBox) return;

Priem(CharToMas(Buf));

}

}

}

else if(Message.Msg == WSA\_ACCEPT)

{

if(Message.LParam == FD\_ACCEPT)

{

int sizeCallAddress = sizeof(CallAddress);

NetBiosSocket = accept (NetBiosSocket, (LPSOCKADDR )&CallAddress, (int FAR \*)&sizeCallAddress);

if(NetBiosSocket == INVALID\_SOCKET)

{

Application->MessageBox("Ошибка Accept","Ошибка",MB\_OK+MB\_ICONHAND);

Application->Terminate();

}

char \*strPered=DannieToChar(0,KolKub,MaxOch);

NetBiosSend(strPered);

InterfaceBeginNewGame();

ChangeInterfacePerehodHoda(1);

rc = WSAAsyncSelect(NetBiosSocket, Handle, WSA\_NETEVENT, FD\_READ);

if (rc != 0)

{

Application->MessageBox("Ошибка WSAAsyncSelect","Ошибка",MB\_OK+MB\_ICONHAND);

closesocket(NetBiosSocket);

NetBiosSocket = INVALID\_SOCKET;

Application->Terminate();

}

}

}

WndProc(Message);

}

//-------------------отправка---------------------------------------------------

void \_\_fastcall TMainForm::NetBiosSend(char \*Buf)

{

int rc;

rc = send(NetBiosSocket, Buf, strlen(Buf)+1, 0);

if (rc == SOCKET\_ERROR)

{

rc = WSAGetLastError();

Application->MessageBox("Ошибка sendto NetBios","Ошибка",MB\_OK+MB\_ICONHAND);

closesocket(NetBiosSocket);

NetBiosSocket = INVALID\_SOCKET;

Application->Terminate();

}

}

//----------------разрыв-------------------------------------------------------

void \_\_fastcall TMainForm::NetBiosCloseConnection(void)

{

closesocket(NetBiosSocket);

WSACleanup();

}

//---------------------------------------------------------------------------

**Pipe.cpp**

//---------------------------------------------------------------------------

#pragma hdrstop

#include <stdio.h>

#include <stdlib.h>

#include <StrUtils.hpp>

#include "Pipe.h"

#include "Main.h"

#include "Podkluch.h"

//---------------------------------------------------------------------------

#pragma package(smart\_init)

HANDLE PipeReceiveHandle, PipeSendHandle;

extern bool IsServer;

extern AnsiString ServerHostName;

extern int KolKub;

extern int MaxOch;

//-----------------------подключение--------------------------------------------

void \_\_fastcall TMainForm::PipeConnect(void)

{

if(IsServer)

{

PipeReceiveHandle = CreateNamedPipe("\\\\.\\pipe\\pipe\_s", PIPE\_ACCESS\_INBOUND,

PIPE\_TYPE\_BYTE|PIPE\_NOWAIT, PIPE\_UNLIMITED\_INSTANCES,

1000, 1000, 5000, NULL);

if (PipeReceiveHandle == INVALID\_HANDLE\_VALUE)

{

Application->MessageBox("Ошибка при создании pipe на сервере","Ошибка",MB\_OK+MB\_ICONHAND);

TimerPipe->Enabled = false;

Application->Terminate();

}

TimerPipe->Enabled = true;

}

else

{

PipeReceiveHandle = CreateNamedPipe("\\\\.\\pipe\\pipe\_c", PIPE\_ACCESS\_INBOUND,

PIPE\_TYPE\_BYTE|PIPE\_NOWAIT, PIPE\_UNLIMITED\_INSTANCES,

1000, 1000, 5000, NULL);

if (PipeReceiveHandle == INVALID\_HANDLE\_VALUE)

{

Application->MessageBox("Ошибка при создании pipe на клиенте","Ошибка",MB\_OK+MB\_ICONHAND);

TimerPipe->Enabled = false;

Application->Terminate();

}

TimerPipe->Enabled = true;

char Buf[100];

sprintf(Buf, "\\\\%s\\pipe\\pipe\_s", ServerHostName.c\_str());

PipeSendHandle = CreateFile(Buf, GENERIC\_WRITE, FILE\_SHARE\_READ, NULL, OPEN\_EXISTING, 0, NULL);

if (PipeSendHandle == INVALID\_HANDLE\_VALUE)

{

int rc;

rc=GetLastError(); //5 51

Application->MessageBox("Ошибка при подключении к pipe на сервере","Ошибка",MB\_OK+MB\_ICONHAND);

Application->Terminate();

}

char CompName[50];

unsigned long cnl = 50;

GetComputerName(CompName, &cnl);

int i=StrLen(CompName);

if(i>=50)

{

CompName[49]='\0';

}

unsigned long cbWritten;

char BufExchange[100];

strcpy(BufExchange,"connect;");

strcat(BufExchange,CompName);

WriteFile(PipeSendHandle, BufExchange, i+9, &cbWritten, NULL);

}

}

//---------------прием и accept--------------------------------------

void \_\_fastcall TMainForm::TimerPipeTimer(TObject \*Sender)

{

int Rc;

DWORD cbRead;

DWORD cbWritten;

char Buf[100];

ConnectNamedPipe(PipeReceiveHandle, NULL);

Rc = GetLastError();

if (Rc == ERROR\_PIPE\_CONNECTED) //535

{

if (ReadFile(PipeReceiveHandle, Buf, sizeof(Buf)-1, &cbRead, NULL))

{

if(LeftStr(AnsiString(Buf),7)=="connect")

{

char \*Buftmp;

char BufConnect[150];

Buftmp=AnsiStrScan(Buf,';');

Buftmp++;

sprintf(BufConnect, "\\\\%s\\pipe\\pipe\_c", Buftmp);

PipeSendHandle = CreateFile(BufConnect, GENERIC\_WRITE, FILE\_SHARE\_READ, NULL,

OPEN\_EXISTING, 0, NULL);

if (PipeSendHandle == INVALID\_HANDLE\_VALUE)

{

Application->MessageBox("Ошибка при подключении к pipe на клиенте","Ошибка",MB\_OK+MB\_ICONHAND);

Application->Terminate();

}

char \*strPered=DannieToChar(0,KolKub,MaxOch);

PipeSend(strPered);

InterfaceBeginNewGame();

ChangeInterfacePerehodHoda(1);

}

else

{

if(IsVisitApplicationBox) return;

Priem(CharToMas(Buf));

}

}

}

}

//-----------------отправка-----------------------------------------------------

void \_\_fastcall TMainForm::PipeSend (char\* Buf)

{

unsigned long cbWritten=0;

WriteFile(PipeSendHandle, Buf, strlen(Buf)+1, &cbWritten,NULL);

}

//-------------------разрыв соединения-----------------------------------------

void \_\_fastcall TMainForm::PipeCloseConnection(void)

{

CloseHandle(PipeSendHandle);

CloseHandle(PipeReceiveHandle);

TimerPipe->Enabled = false;

}

**Mailslot.cpp**

//---------------------------------------------------------------------------

#pragma hdrstop

#include <stdio.h>

#include <stdlib.h>

#include <string.h>

#include "MailSlot.h"

#include "Podkluch.h"

#include "Main.h"

//------------------------------------------------------------------------------

#pragma package(smart\_init)

HANDLE ReceiveMailslotHandle, SendMailslotHandle;

int HisPrevNomMessage=0;

int MyCurrNomMessage=0;

extern bool IsServer;

extern AnsiString ServerHostName;

extern int KolKub;

extern int MaxOch;

//-----------------------подключение--------------------------------------------

void \_\_fastcall TMainForm::MailSlotConnect(void)

{

if(IsServer)

{

ReceiveMailslotHandle = CreateMailslot("\\\\.\\mailslot\\msserver", 0, MAILSLOT\_WAIT\_FOREVER, NULL);

if (ReceiveMailslotHandle == INVALID\_HANDLE\_VALUE)

{

Application->MessageBox("Ошибка при создании mailslot","Ошибка",MB\_OK+MB\_ICONHAND);

Application->Terminate();

}

TimerMailSlot->Enabled = true;

}

else

{

ReceiveMailslotHandle = CreateMailslot("\\\\.\\mailslot\\msclient", 0, MAILSLOT\_WAIT\_FOREVER, NULL);

if (ReceiveMailslotHandle == INVALID\_HANDLE\_VALUE)

{

Application->MessageBox("Ошибка при создании mailslot","Ошибка",MB\_OK+MB\_ICONHAND);

Application->Terminate();

}

TimerMailSlot->Enabled = true;

char BufConnect[70];

sprintf(BufConnect, "\\\\%s\\mailslot\\msserver" , ServerHostName.c\_str());

SendMailslotHandle = CreateFile(BufConnect, GENERIC\_WRITE, FILE\_SHARE\_READ, NULL, OPEN\_EXISTING, 0, NULL);

if (SendMailslotHandle == INVALID\_HANDLE\_VALUE)

{

Application->MessageBox("Ошибка при открытии mailslot","Ошибка",MB\_OK+MB\_ICONHAND);

Application->Terminate();

}

char CompName[50];

unsigned long cnl = 50;

GetComputerName(CompName, &cnl);

int i=StrLen(CompName);

if(i>=50)

{

CompName[49]='\0';

}

unsigned long cbWritten;

char BufExchange[50];

MyCurrNomMessage=1;

strcpy(BufExchange,"1;");

strcat(BufExchange,CompName);

WriteFile(SendMailslotHandle,BufExchange, strlen(BufExchange)+1, &cbWritten, NULL);

}

}

//-----------------------прием и accept-----------------------------------------

void \_\_fastcall TMainForm::TimerMailSlotTimer(TObject \*Sender)

{

if(IsVisitApplicationBox) return;

int rc;

unsigned long cbMessages, cbMsgNumber, cbRead;

char \*BufExchange;

rc = GetMailslotInfo(ReceiveMailslotHandle, NULL, &cbMessages, &cbMsgNumber, NULL);

if (!rc)

{

Application->MessageBox("Ошибка GetMailslotInfo","Ошибка",MB\_OK+MB\_ICONHAND);

Application->Terminate();

}

if (cbMsgNumber == 0) return;

BufExchange = new char[cbMessages];

ZeroMemory(BufExchange, cbMessages);

if (ReadFile(ReceiveMailslotHandle, BufExchange, cbMessages, &cbRead, NULL))

{

int HisCurrNomMessage;

char Buftmp[150];

strcpy(Buftmp,BufExchange);

HisCurrNomMessage=atoi(strtok(BufExchange,";"));

if(HisCurrNomMessage>HisPrevNomMessage)

{

HisPrevNomMessage=HisCurrNomMessage;

char \*Buf;

Buf=AnsiStrScan(Buftmp,';');

Buf++;

if(IsServer && HisCurrNomMessage==1)

{

char BufConnect[100];

sprintf(BufConnect, "\\\\%s\\mailslot\\msclient" , Buf);

SendMailslotHandle = CreateFile(BufConnect, GENERIC\_WRITE, FILE\_SHARE\_READ, NULL, OPEN\_EXISTING, 0, NULL);

if (SendMailslotHandle == INVALID\_HANDLE\_VALUE)

{

Application->MessageBox("Ошибка при открытии mailslot","Ошибка",MB\_OK+MB\_ICONHAND);

Application->Terminate();

}

char \*strPered=DannieToChar(0,KolKub,MaxOch);

MailSlotSend(strPered);

InterfaceBeginNewGame();

ChangeInterfacePerehodHoda(1);

}

else

{

Priem(CharToMas(Buf));

}

}

}

delete [] BufExchange;

}

//------------------------отправка----------------------------------------------

void \_\_fastcall TMainForm::MailSlotSend(char \*Buf)

{

unsigned long cbWritten;

char BufExchange[100];

MyCurrNomMessage++;

itoa(MyCurrNomMessage,BufExchange,10);

strcat(BufExchange,";");

strcat(BufExchange,Buf);

WriteFile(SendMailslotHandle, BufExchange, strlen(BufExchange)+1, &cbWritten, NULL);

}

//----------------------разрыв соединения---------------------------------------

void \_\_fastcall TMainForm::MailSlotCloseConnection(void)

{

CloseHandle(SendMailslotHandle);

CloseHandle(ReceiveMailslotHandle);

TimerMailSlot->Enabled = false;

MyCurrNomMessage=0;

HisPrevNomMessage=0;

}