ТРОИЦКИЙ АВИАЦИОННЫЙ ТЕХНИЧЕСКИЙ КОЛЛЕДЖ
– ФИЛИАЛ МГТУ ГА

МЕТОДИЧЕСКОЕ ПОСОБИЕ

по практическим работам по предмету

«Инфокоммуникационные системы и сети»

Раздел:

«СЕТЕВОЕ ПРОГРАММИРОВАНИЕ В СРЕДЕ DELPHI с использованием библиотеки Internet Direct (Indy)»

Специальность 09.02.03
«Программирование в компьютерных системах»

г.Троицк, 2015 г.

Литература

1. Абрамян М. Э., Брагилевский В. Н. Сетевое программирование в Delphi
2. Архангельский А. Я. Приемы программирования в Delphi на основе VCL Издательство: Бином-Пресс, 2009 г.
3. Гагарина Л.Г. и др. Основы технологии разработки программных продуктов: учеб. пособие.-М.:ФОРУМ:ИНФРА-М,2006.-192 с.-(Профессиональное образование)
4. Рудаков А.В. Технология разработки рекламных продуктов: учеб. пособие для студ. сред. спец. учеб. заведений.-М.:ИЦ «Академия»,2005.-208 с.
5. Фленов М.Е. Программирование в Delphi глазами хакера. – СПб.: БХВ-Петербург, 2004.
6. Хандадашева Л.Н., Истомина И.Г. Программное обеспечение. Вычислительные сети:-М.:Ростов н/Д:ИЦ «МарТ),2005.-320 с.(серия «Среднее профессиональное образование»)
7. Чад Хувер (Chad Z. Hower). Indy in Depth (Глубины Indy)

Пособие составлено преподавателем ЦК ПКС Орловым С.В.

Обсуждено
На заседании ЦК ПКС
«____»_____201___г.

Практическая работа № 1

Создание программы обмена текстовыми сообщениями между двумя узлами

Цель работы:

повторить свойства сетевых протоколов UDP и IP, изучить методы использования компонент idUDPServer и idUDPClient.

Основные сетевые компоненты:

idUDPServer, idUDPclient

1. Краткие сведения из теории

UDP (англ. *User Datagram Protocol* — протокол пользовательских датаграмм) — один из ключевых элементов TCP/IP, набора сетевых протоколов для Интернета. С UDP компьютерные приложения могут посылать сообщения (в данном случае называемые датаграммами) другим хостам по IP-сети без необходимости предварительного сообщения для установки специальных каналов передачи или путей данных. Протокол был разработан Дэвидом П. Ридом в 1980 году и официально определён в RFC 768.

Протокол UDP не устанавливает соединения и не поддерживает целостности передаваемых данных. Протокол просто открывает порт, выкидывает туда порцию данных и даже не волнуется о том, дошли они до получателя, или нет. Поэтому UDP работает намного быстрей, чем TCP.

Если вы захотите работать с этим протоколом, то проверку правильности получения данных придется реализовывать самим. Поэтому для передачи файлов или другой информации большого размера вы должны выбрать TCP, потому что если хоть один маленький кусочек от файла будет потерян, то его уже будет не восстановить. Ну а для чата, который мы сегодня напишем, более удобным вариантом

будет UDP. Он очень быстрый и при маленьких размерах сообщений очень эффективен.

2. Порядок выполнения работы

2.1. Проектирование приложения

В Delphi для работы с UDP-протоколом хорошо подходит библиотека Indy. В частности компоненты IdUDPClient (умеет отсылать данные) с закладки Indy Clients и IdUDPServer (умеет получать данные) с закладки Indy Servers.

Основные принципы работы нашего приложения заключаются в следующем:

1. Пользователь определяет текст сообщения.
2. Пользователь указывает IP-адрес другого узла.
3. Пользователь инициирует отправку сообщения.
4. Компоненту IdUDPClient передаются значения параметров:
   – Host – IP-адрес назначения
   – Port – номер порта UDP для работы приложения
5. Компонент IdUDPClient посылает указанные данные с помощью метода Send.
6. На активном компоненте IdUDPServer удаленного узла при получении сообщения выполняется событие OnUDPRead, обрабатывающее входной поток и отображающее IP-адрес отправителя и полученный текст.

2.2. Создание программы

1. Создать новый проект.
2. Поместить на форму проекта экземпляр компонента IdUDPServer с палитры Indy Servers и IdUDPClient с палитры Indy Clients.

3. Определить значение свойства Port компонентов IdUDPClient1.

4. Определить значение свойства DefaultPort компонентов IdUDPServer1.

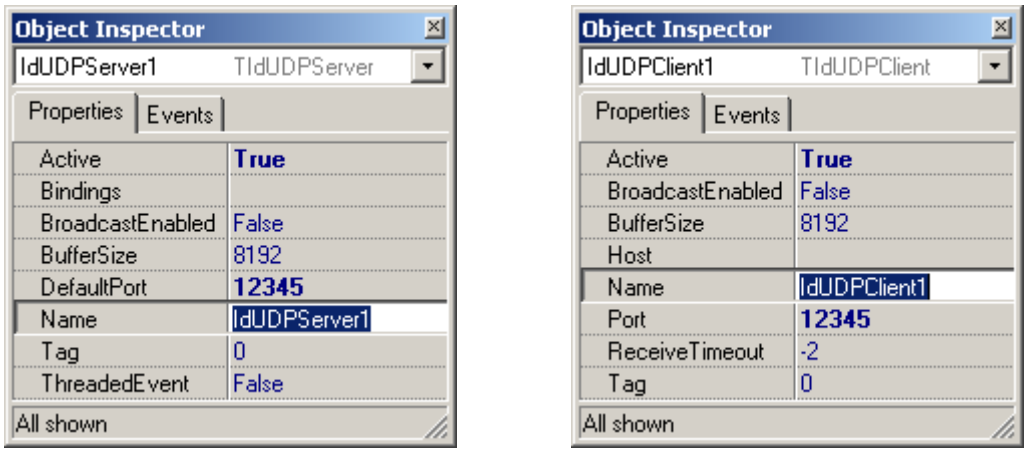

Рис.2 Параметры компонентов IdUDPServer1 и IdUDPClient

Номер порта у обоих компонентов должен быть одинаковым. Это заставит клиента и сервер работать на одном и том же порту, что необходимо для корректной работы. Номер порта, через который будет происходить связь, может быть от 1 до 65000 (рекомендуется присваивать большие значения, чтобы избежать дублирования номеров портов у различных приложений). Порты протокола UDP не пересекаются с портами TCP. Это значит, что TCP-порт 80 не равен UDP-порту 80.

5. Свойству IdUDPServer1.Active и IdUDPClient1.Active присвоить значение True.

6. Поместить на главную форму проекта два компонента Memo:

– первый для отображения получаемых сообщений

– второй для подготовки нового сообщения

7. Поместить на форму метку для отображения IP-адреса отправителя.

8. Поместить на форму поле ввода для указания IP-адреса получателя.

9. Поместить на форму кнопку отправки сообщения.

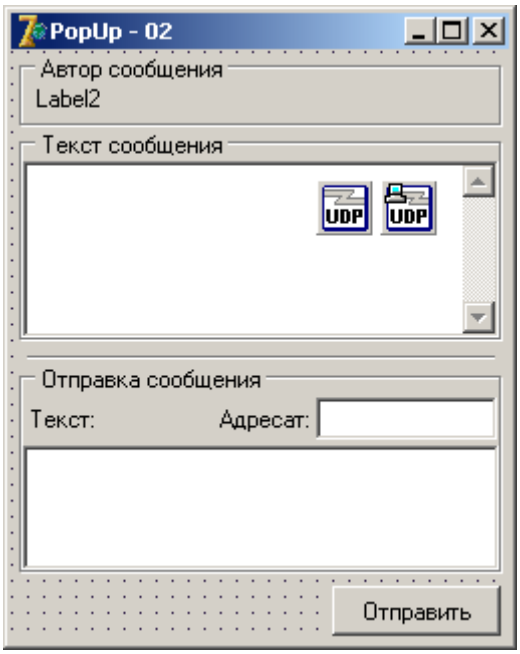

Рис. 3. Форма приложения

10. Создать обработчик события OnClick кнопки и написать там следующий код:

```
procedure TForm1.BitBtn1Click(Sender: TObject);
  begin
    IdUDPClient1.Host := LabeledEdit2.Text;
    IdUDPClient1.Send(Memo2.Text);
  end;
```

Программный код этих срок выполняет:

– определение свойства Host компонента IdUDPClient1, которое обозначает IP-адрес удаленного узла

– отправка с помощью метода Send компонента IdUDPClient1 содержимое второго Memo-поля на удаленный узел.

11. Теперь нужно научить UDP-сервер получать эту информацию. Для этого необходимо создать обработчик события onUDPRead для компонента IdUDPServer1. В нем написать следующий код:

```
procedure TForm1.IdUDPServer1UDPRead(Sender: TObject; AData:
TStream; ABinding: TIdSocketHandle);
```

```
var S :TStringStream;
begin
  // Инициализация
  S := TStringStream.Create('');
  //Копирование из простого потока в строковый
  S.CopyFrom(AData,AData.Size);
  //Вывод полученного сообщения
  Memo1.Text := S.DataString;
  ......
  //Освобождение строкового потока
  S.Free;
end;
```

У процедуры-обработчика события есть три параметра, из которых для нас интерес представляют два последних:

- AData – данные, которые получены из сети
- ABinding - содержит информацию о том, откуда пришли данные.

AData хранит данные, как простой неформатированный поток TStream, позволяющий передавать данные любого типа: текст, графику и другое. Чтобы удобней было работать с данными, в нашем случае, их лучше переконвертировать в строковый поток TStringStream.

В обработчике объявлена одна переменная S типа TStringStream (строковый поток). Первой строкой кода она инициализируется. Во второй строке данные из простого неформатированного потока копируются в строковый поток. Теперь переданный текст находится в свойстве DataString строкового потока StringFormatedStream. После этого можно выводить этот результат в компоненте Memo.

3. Задание на самостоятельную работу

1. Отобразить IP-адрес отправителя сообщения, например, используя свойство PeerIP параметра ABinding.

2. Защитить приложение от изменения полученного сообщения получателем и от неправильного ввода IP-адреса получателя.

3. Реализовать накопление сообщений для последующего просмотра в массиве.

4. Реализовать просмотр истории сообщений, отображая их по одному, например, используя кнопки [ВПЕРЕД]/[НАЗАД].

5. Реализовать отображение в строке статуса информации о количестве принятых сообщений и номер текущего сообщения.

4. Контрольные вопросы

1. Основные функции транспортных протоколов.

2. Основные функции протоколов маршрутизации.

3. Отличия UDP от TCP.

4. Основные свойства, методы и события компонент idUDPClient и idUDPServer.

5. Назначение параметров AData, ABinding, PeerIP.

Практическая работа № 2

Создания программы обмена текстовыми сообщениями между несколькими узлами

Цель работы:

повторить адресацию протокола IP, понятия широковещательный запрос, закрепить методы использования компонент idUDPServer и idUDPClient.

Основные сетевые компоненты:

idUDPServer, idUDPclient

1. Краткие сведения из теории

В основе адресации протокола IP версии 4 лежит понятие класса сети, который определяется значением первого байта адреса и который определяет границу между номером сети и номером узла в этой сети.

Для удобства администрирования IP-адрес представляют четырьмя десятичными числами, разделенных точками, например «192.168.1.100».

Существует три основных класса сетей A, B и C. Для адресации узлов локальных сетей выделены диапазоны, не использующиеся в глобальной сети, например: 192.168.x.x или 10.x.x.x

Сетевая часть номера должна быть одинаковой для всех узлов логической сети, номер узла должен быть уникальным в пределах одной логической сети.

Можно использовать широковещательный IP-адрес для обращения ко всем узлам данной сети. В таком адресе вместо конкретного номера узла используется адрес, где все битовые разряды равны 1 (в десятичном представлении 255).

Так для сети класса «C» 192.168.1.x (граница между номером

сети и номером узла в этой сети за третьим байтом) в качестве широковещательного адреса используется последний байт равный 255.

2. Порядок выполнения работы

2.1. Проектирование приложения

В программе мы будем использовать те же сетевые компоненты с панели Indy Clients и Indy Servers, а конкретно: компоненты IdUDPClient (умеет отсылать данные) с закладки Indy Clients и IdUDPServer (умеет получать данные) с закладки Indy Servers.

Основные принципы работы нашего приложения заключаются в следующем:

1. Пользователь определяет текст сообщения.
2. Пользователь указывает IP-адрес другого узла, в качестве которого может выступать конкретный адрес какого либо узла или специальный широковещательный адрес, определяющий все узлы данной сети.
3. Пользователь инициирует отправку сообщения.
4. Компоненту IdUDPClient передаются значения параметров:

– Host – IP-адрес назначения

– Port – номер порта UDP для работы приложения

5. Компонент IdUDPClient посылает указанные данные с помощью метода Send.
6. На активном компоненте IdUDPServer удаленного узла (или удаленных узлов) при получении сообщения выполняется событие OnUDPRead, обрабатывающее входной поток и последовательно отображающее атрибуты отправителя и полученный текст в окне.

2.2. Создание программы

1. Создать новый проект.
2. Поместить на главную форму проекта экземпляр компонента IdUDPServer с палитры Indy Servers и IdUDPClient с

палитры Indy Clients.

3. Определить значение свойства Port компонентов IdUDPClient1.

4. Определить значение свойства DefaultPort компонентов IdUDPServer1.

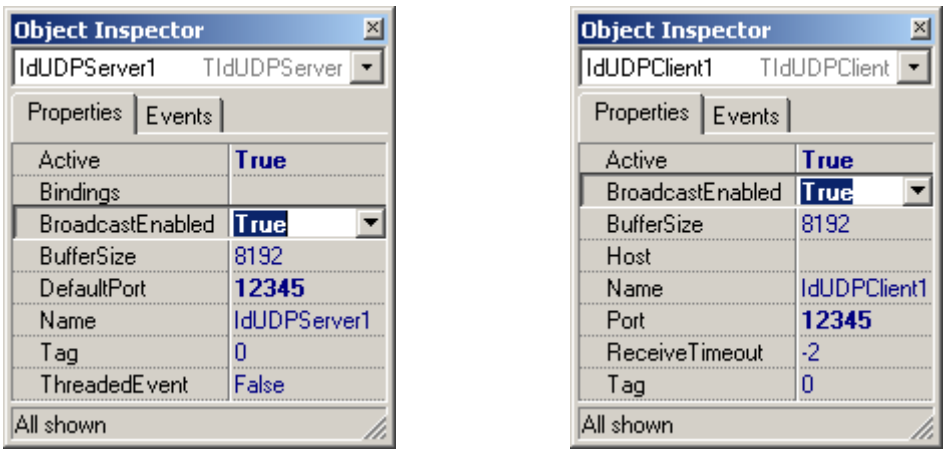

Рис. 2 Параметры компонентов IdUDPServer1 и IdUDPClient

5. Свойству IdUDPServer1.Active и IdUDPClient1.Active присвоить значение True.

6. Свойству IdUDPServer1.BroadcastEnabled и IdUDPClient1.BroadcastEnabled присвоить значение True.

7. Поместить на форму проекта компонент Memo для отображения получаемых сообщений.

8. Поместить на форму проекта компонент Edit для подготовки нового сообщения.

9. Поместить на форму кнопку отправки сообщения.

10. Создать обработчик события OnClick кнопки и написать там следующий код:

```
procedure TForm1.BitBtn1Click(Sender: TObject);
  begin
    IdUDPClient1.Host := '192.168.1.255';
    IdUDPClient1.Send(LabeledEdit1.Text);
  end;
```

Основным отличием от предыдущей программы является то, что

сообщение должно быть отправлено нескольким компьютерам сети (практически всем, но должным образом примут его только те, на которых запущен соответствующий сервер с открытым портом).

Для этого необходимо установить у обоих сетевых компонентов свойство BroadcastEnabled равным TRUE. А вместо конкретного IP-адреса использовать широковещательный. Если в сети используются адреса типа 192.168.1.x, то для такой сети широковещательный адрес будет 192.168.1.255 (последний байт меняем на 255).

11. Серверная часть приложения практически повторяет серверную часть предыдущей программы

```
procedure TForm1.IdUDPServer1UDPRead(Sender: TObject; AData:
TStream; ABinding: TIdSocketHandle);
  var S :TStringStream;
  begin
    S := TStringStream.Create('');
    S.CopyFrom(AData,AData.Size);
    //Вывод полученного сообщения в общее поле
    Memo1.Lines.Add(ABinding.PeerIP + ': ' + S.DataString);
    S.Free;
end;
```

Принятые сообщения добавляются в Memo-поле, предваряясь IP-адресом отправителя.

Memo1.Lines.Add(ABinding.PeerIP + ': ' + S.DataString);

3. Задание на самостоятельную работу

1. Реализовать возможность отправки сообщения по выбору пользователя: всем узлам или конкретному узлу.

2. Реализовать накопление IP-адресов клиентов в списке и выбор адреса для отправки из списка (например, используя компонент ComboBox). Добавление в список адресов (ComboBox) должно выполняться только для новых клиентов.

Вариант решения:

- При запуске клиента на новом узле автоматически выполнять широковещательную рассылку специальной кодовой строки, например: #new
- На всех узлах сети обрабатывать серверным компонентом этот код, как команду добавить нового клиента в список (не выводя на экран!).
- В ответ на #new, все работающие узлы посылают служебный код, например #online на адрес нового узла, то есть узла, с которого пришёл код #new.
- При получении сообщения с префиксом #online, адрес отправителя добавляется в список, работающих в сети узлов.

3. Реализовать удаление IP-адреса клиента из списка выбора адреса (ComboBox), при его отключении (завершении работы). Аналогично предыдущему пункту.

4. Реализовать идентификацию пользователей по именам

- Работа с приложением (передача и прием данных) возможна только после ввода пользователем своего имени.
- При получении сообщения отображать имя пользователя, IP-адрес и сообщение.
- В списке адресов клиентов отображать имя пользователя и IP-адрес.

4. Контрольные вопросы

1. Структура IP-адреса.
2. Основные классы IP-адресов.
3. Понятие широковещательного адреса.
4. Назначение параметров AData, ABinding, PeerIP.

Практическая работа № 3

Проверка связи (PING)

Цель работы:

повторить настройку и контроль стека TCP/IP, рассмотреть методы использования компоненты idICMPClient для контроля доступности сетевого узла.

Основные сетевые компоненты:

idICMPclient

1. Краткие сведения из теории

Простейшим сетевым клиентом является программа, посылающая пакет на другой узел. Для обеспечения некоторого контроля за передачей пакета, например, время отклика, можно обработать ответ от «сервера». Таким образом поступает команда PING, существующая в любой современной сетевой операционной системе.

2. Порядок выполнения работы

Для реализации аналога программы PING можно воспользоваться компонентом IdICMPClient из состава Indy.

1. С палитры компонентов Indy Clients поместить компонент IdICMPClient

2. Поместить на форму кнопку (TButton)

3. Поместить на форму поле ввода (TEdit)

4. Поместить на форму список (TListBox)

5. В обработчик события кнопки OnClick записать код:

```
IdIcmpClient1.Host := Edit1.Text;
IdIcmpClient1.Ping;
```

6. Выставить соответствующий интервал ожидания ответа, по завершении которого (или при получении данных от пингуемого) вызывается обработчик: TidIcmpClient.OnReply(Sender:TComponent; const AReplyStatus:TReplyStatus);

в котором реализуется вывод данных пинга на экран:

```
procedure TForm1.IdIcmpClient1Reply(ASender: TComponent;
const AReplyStatus: TReplyStatus);
  begin
    ListBox1.Items.Add ('Время ответа (мс): ' +
                        IntToStr(AreplyStatus.MsRoundTripTime));
  end;
```

3. Задание на самостоятельную работу

1. В случае отсутствия связи с проверяемым узлом, вывести соответствующее сообщение.

2. Реализовать автоматическую отправку нескольких пакетов (количество задается пользователем с помощью SpinEdit). Следующий пакет должен посылаться после обработки предыдущего пакета.

3. В конце цикла вывести среднее, минимальное и максимальное время отклика.

4. Контрольные вопросы

1. Использование стандартной утилиты PING.
2. Что означает превышение времени ожидания времени возврата пакета?
3. Какова продолжительность времени возврата пакета в сети Интернет?
4. Основные свойства и события компонента idICMPClient.

Практическая работа № 4.

Сканирование сетевых портов

Цель работы:

повторить назначение и принципы использования сетевых портов транспортных протоколов (TCP и UDP); рассмотреть методы использования компонентов idTCPClient и idTCPServer для обнаружения работающих сетевых программ конкретного узла; познакомиться с методами обработки исключительных ситуаций.

Основные сетевые компоненты:

idTCPclient, idTCPServer, idAntiFreeze

1. Краткие сведения из теории

Каждая сетевая программа при старте открывает для себя свободный порт. Есть программы, которые открывают заведомо определенный порт, например, для FTP это 21-й порт, HTTP — 80-й порт и т. д. Если на сервере запущено два сервиса: FTP и WEB, значит на сервере работают две программы, к которым можно присоединиться по сети. Таким образом, сетевые порты – это число, по которому программы и ОС определяют, кому пришли данные по сети.

Если знать, какие порты открыты, то можно понять, какие программы запущены на удаленном компьютере. Так, например, если на компьютере открыт 21-й порт, то значит, на нем работает FTP-сервер, и к нему можно попытаться присоединиться с помощью программы FTP Client.

Как сканируются порты? Для понимания этого нужно представлять процесс соединения двух компьютеров. Когда двое в сети хотят соединиться, то один из них посылает другому запрос с номером порта, на котором должно произойти соединение. По этому порту другая сторона определяет, к какой программе хотят

подключиться. Если какое-то приложение действительно открыло нужный порт и ожидает соединения, то запрашиваемый получит ответ об успешности попытки. Все проверки пароля и прочие защитные механизмы происходят уже после соединения с портом удаленного компьютера, поэтому мы можем произвести соединение и узнать о доступности порта, даже если программа требует пароля.

2. Порядок выполнения работы

2.1. Создание сервера

1. Создать новый проект.
2. Поместить на главную форму проекта экземпляр компонента IdTCPServer с палитры Indy Servers.
3. Поместить на главную форму проекта компонент Label.
4. Определить значение свойства DefaultPort компонента IdTCPServer равным 6000.
5. Свойству Active компонента IdTCPServer присвоить значение True.
6. Добавить следующий текст в обработчик события OnExecute:

```
ShowMessage('Пришел запрос!');
AThread.Connection.Disconnect;
```

7. Сохранить и скомпилировать проект, запустите его как автономную программу

2.2. Создание клиента

1. Запустить Delphi.
2. Создать новый проект.
3. Перенести на форму следующие компоненты:
   а) одну кнопку;

б) две метки;

в) три поля ввода;

4. У кнопки поменять свойство Caption на «Сканировать», у первой метки на «Начальный порт», а у второй метки на «Конечный порт». После нажатия кнопки мы будем сканировать порты, начиная от номера, указанного в первом поле ввода, до номера, указанного во втором поле ввода.

5. Перенести на форму компонент idTcpClient.

6. Установить у Edit1 свойство Text равным 5998, а у Edit2 – 6003. Таким образом мы задаем значения по умолчанию для начального и конечного порта.

7. Перенести на форму компонент TMemo. Здесь будет отображаться состояние сканирования.

8. В обработчике события OnClick кнопки «Сканировать» написать:

```
idTcpClient1.Host := Edit3.Text;
for i := StrToInt(Edit1.Text) to StrToInt(Edit2.Text) do
  begin
    idTcpClient1.Port := i;
    try
      idTcpClient1.Connect;
      if idTcpClient1.Connected then
        begin
          Memo1.Lines.Add(IntToStr(i) + ' открыт');
          idTcpClient1.Disconnect;
        end
    except // E.Message
      on E : Exception do Memo1.Lines.Add(IntToStr(i) +
                                          ' закрыт');
    end;
  end;
```

В Windows сканирование 1000 портов проходит очень долго, поэтому лучше сканировать маленькими порциями – не более 10

портов. Созданный сканер – вполне рабочая программа, и его можно использовать даже в реальных условиях. Для создания более быстрого сетевого сканера, который будет сканировать 1000 портов практически мгновенно, необходимо использовать низкоуровневые библиотеки WinSock.

3. Задание на самостоятельную работу

1. Добавить в приложение «Сервер» ведение лога подключений, например в компоненте Memo. Отображать дату, время получения запроса и IP-адрес клиента.

2. На «Клиенте» отображать начальное, конечное время и продолжительность сканирования диапазона портов.

3. Обосновать использование и добавить компонент idAntiFeeze.

3. Объединить программу проверки связи и сканирования портов в одно приложение.

4. Контрольные вопросы

1. Что позволяет узнать процедура сканирования портов?

2. Почему некоторые сетевые порты зарезервированы?

3. Можно ли на одном узле одновременно запустить сервер на порту 3000 протокола TCP и на порту 3000 протокола UDP?

4. Основные свойства и события компонента idTCPClient и idTCPSever.

5. Для чего используется механизм обработки исключительных ситуаций?

6. Для чего используется компонент idAntiFreeze?

Практическая работа № 5

Передача двоичных файлов

Цель работы:

повторить назначение и принципы использования транспортного протокола TCP, методы использования компонентов idTCPClient и idTCPServer для передачи данных, требующих достоверности и разбиения на пакеты; закрепить практические навыки использования методов обработки исключительных ситуаций.

Основные сетевые компоненты:

idTCPclient, idTCPServer, idAntiFreeze

1. Краткие сведения из теории

Текстовый файл является частным случаем двоичного файла. Поэтому для передачи любого файла с соблюдением требований достоверности и доступности узлов к среде передачи информации при передаче больших объемов данных (временное разделение среды) лучше использовать более медленный, но надежный протокол TCP.

2. Порядок выполнения работы

Для упрощения задачи создадим две отдельные программы: клиент и сервер. Сервер будет загружаться, определять файл для передачи, открывать порт и ожидать соединения. Клиент соединяется с сервером и запрашивает у сервера файл. Сервер отсылает его клиенту.

2.1. Создание сервера

Создать новый проект и перенесите на форму следующие компоненты.

- поле ввода Edit, в которое будет отображаться имя файла.
- кнопку Button, с помощью которой можно будет выбирать отправляемый файл.
- компонент idTCPServer, с его помощью мы будет отправлять данные.
- компонент OpenDialog, с помощью которого мы будем открывать файл.

У компонента idTCPServer нужно установить свойство Port равным какому-нибудь реальному значению порта, который будет открываться для ожидания подключения. Например, номер 6001. Свойство Active установить равным true, чтобы сервер автоматически активизировался при старте.

В обработчик нажатия кнопки можно вставить следующий код:

```
if OpenDialog1.Execute then Edit1.Text:=OpenDialog1.FileName;
```

При выборе пользователем файла, помещаем путь в строку ввода.

Сервер ожидает прихода определенной команды, и если она пришла, то отправляет файл. Для этого создадим обработчик события OnExecute, который вызывается каждый раз, когда данные приходят из сети.

```
procedure TfrmServer.IdTCPServerExecute(AThread:
TIdPeerThread);
  var fStream : TFileStream;
  begin
    fStream := TFileStream.Create(Edit1.Text,fmOpenRead);
    AThread.Connection.OpenWriteBuffer;
    AThread.Connection.WriteStream(fStream);
  AThread.Connection.CloseWriteBuffer;
```

```
  AThread.Connection.Disconnect;
  fStream.Free;
  ShowMessage('Файл передан');
end;
```

Прежде чем отсылать данные, файл нужно открыть и загрузить. Для этого используется объект файлового потока (переменная fStream типа TFileStream). Сначала эта переменная инициализируется:

fStream := TFileStream.Create(Edit1.Text,fmOpenRead);

В качестве параметров конструктора передается имя файла Editl.Text и режим, в котором будет подключен файл. Нам достаточно использовать режим чтения, поэтому указан флаг fmOpenRead.

После открытия текущая позиция в файле должна быть установлено на самое начало, но для уверенности, сделаем это самостоятельно:

fStream.Position:= 0;

Событие onExecute, возникающее при соединении с клиентом, возвращает структуру AThread — указатель на, так называемую «нить», протянутую как бы между «клиентом» и «сервером».

Одним из свойств «нити» является свойство Connection, используемое для доступа к свойствам, методам и событиям серверного компонента и управления передаваемыми данными.

Метод OpenWriteBuffer использует внутренний буфер Indy-компонента для временной накопления и хранения данных, предназначенных для передачи.

Метод WriteStream() посылает заготовленный поток данных, указанный в качестве параметра.

Метод CloseWriteBuffer прекращает кэширование данных после окончания передачи.

После передачи файла разрываем соединение с клиентом и очищаем память от файлового потока.

2.2. Создание клиента

1. Создать новый проект.
2. Поместить на форму следующие компоненты:
   - поле для ввода IP-адреса сервера;
   - кнопку «Соединиться и запросить файл», при нажатии которой будет запрашиваться файл;
   - компонент idTCPClient, с помощью которого будем присоединяться к серверу, запрашивать и получать файл.
3. У компонента idTCPClient установить свойство Port равным тому же значению, что и у сервера.
4. В обработчике события onClick кнопки «Соединиться и запросить файл» написать следующий код:

```
if IdTCPClient.Connected then IdTCPClient.DisConnect;
IdTCPClient.Host := edtServerHost.text;
IdTCPClient.Port := StrToInt(edtServerPort.text);
IdTCPClient.Connect;
fStream := TFileStream.Create(ExtractFileDir(ParamStr(0)) +
  '\file_name.tmp',fmCreate);
while IdTCPClient.connected do
  IdTCPClient.ReadStream(fStream,-1,true);
IdTCPClient.Disconnect;
fStream.Free;
```

В отличие от «сервера», «клиент» в конкретный момент времени может взаимодействовать только с одним «сервером». Поэтому если сеанс связи открыт, закрываем его:

if IdTCPClient.Connected then IdTCPClient.DisConnect;

Затем определяем основные параметры «клиента»: адрес компьютера, где расположен сервер, с которым надо соединиться и

порт и пробуем соединиться с «сервером»:

```
IdTCPClient.Host := edtServerHost.text;
IdTCPClient.Port := StrToInt(edtServerPort.text);
IdTCPClient.Connect;
```

Инициализируем файловый поток для входящих данных и соединяем его с файлом:

```
fStream := TFileStream.Create(ExtractFileDir(ParamStr(0)) +
'\file_name.tmp',fmCreate);
```

Принимаем входящие данные, пока «сервер» не разорвет связь:

```
while IdTCPClient.connected do IdTCPClient.ReadStream(fStream,-1,true);
```

Подтверждаем разрыв сеанса связи и очищаем память:

```
IdTCPClient.Disconnect;
fStream.Free;
```

3. Задание на самостоятельную работу

1. Добавить механизм обработки исключительной ситуации.
2. Реализовать сохранение файла с исходным именем.
3. Реализовать передачу и прием изображений, используя потоки типа TMemoryStream и компоненты Image
4. Обрабатывая события «серверного» компонента onConnect и onDisconnect, сохранять в логе (файле) информацию об успешном соединении и разъединении с «клиентским» компонентом.

4. Контрольные вопросы

1. Почему для передачи файлов желательно использовать TCP?
2. Для чего у компонента idTCPClient метод Disconnect?
2. Что выполняют методы Create и Free?

Практическая работа № 6

Использование сервисов электронной почты

Цель работы:

повторить назначение и принципы использования протоколов SMTP и POP3, методы использования компонентов idSMTP и idPOP3 для взаимодействия с почтовыми серверами.

Основные сетевые компоненты:

idSMTP, idPOP3, idMessage

1. Краткие сведения из теории

Одна из наиболее частых операций в Интернете — это отправка и прием сообщений электронной почты. Кроме написания многофункциональных программ можно делать некоторые интересные вещи с почтовыми компонентами и протоколами. Условно разделим эти возможности на две группы:

- Автоматическое создание почтовых сообщений. Можно написать приложение, содержащее окно «О программе», чтобы отправить уведомление о регистрации или запрос в службу технической поддержки.
- Использование почтовых протоколов для общения с пользователями, которые редко подключаются к сети. Использовать имеющийся почтовый сервер и написать две программы, основанные на почтовых протоколах. При таком способе передачи данные, как правило, представлены в специальном формате, поэтому удобнее создать для таких сообщений отдельный почтовый ящик. Основное преимущество такого подхода в том, что можно работать через брандмауэры, и информация сохраняется на почтовом сервере.

1.1. Протокол SMTP

SMTP - почтовый протокол, при помощи которого почта пересылается пользователем на сервер для дальнейшей пересылки, а также между серверами. В этом случае клиентом является очередной промежуточный почтовый сервер (relay-сервер), получивший сообщение. Он и должен инициировать отправку сообщения далее для достижения конечной цели, а сервером будет следующий промежуточный или уже конечный пункт.

Стандартный порт SMTP-сервера - 25.

В протоколе SMTP не предусмотрена аутентификация, т.е. отправить сообщение от имени какого-либо пользователя можно, не зная его пароль.

Сообщения, отсылаемые клиентом серверу (отправителем получателю), называют командами; сообщения, отсылаемые сервером клиенту (получателем отправителю), называют ответами. Среди команд клиента различают команды без параметров, команды с обязательными параметрами и команды с опциональными параметрами. Ответ сервера (получателя) всегда начинается с трехзначного числа - кода ответа. После кода ответа следует текстовое пояснение.

Полное описание протокола SMTP приведено в спецификации RFC 821.

1.1.1. Некоторые команды протокола SMTP.

HELO (обязательный параметр) - данная команда является необязательной, клиент посылает ее серверу после получения приветствия для того, чтобы убедиться, что взаимодействуют именно те хосты, которые и должны взаимодействовать. Здесь в качестве обязательного параметра domain клиент должен указать свое доменное имя.

QUIT - данная команда инициирует закрытие соединения.

В SMTP-протоколе отправка одного почтового сообщения

сопоставляется одной транзакции. За один сеанс связи может быть отправлено несколько сообщений, т. е. выполнено несколько транзакций. В SMTP- транзакции выделяют три шага (этапа). Начинается транзакция с команды MAIL, идентифицирующей отправителя. Затем следует серия из одной или нескольких команд RCPT, содержащих адрес получателя (получателей). Затем следует команда DATA, с помощью которой передается тело сообщения. Как только завершена передача тела сообщения, SMTP-транзакция также считается завершенной.

MAIL FROM:<адрес отправителя> (обязательный параметр) - данная команда инициирует новую SMTP-транзакцию, в ней в качестве обязательного параметра указывается адрес отправителя, который будет использован для отправления сообщения об ошибке. Если указанный адрес отправителя допустим, то сервер посылает ответ "250 OK". Для того, чтобы определить, является ли команда допустимой, сервер пытается преобразовать доменное имя в IP-адрес, посылая запрос DNS-серверу.

RCPT TO:<адрес получателя> (обязательный параметр) - данная команда идентифицирует одного получателя при помощи обязательного параметра "адрес получателя". Эта команда может повторяться любое количество раз, т. к. в спецификации количество получателей не ограничивается каким-либо конкретным значением.

DATA (без параметров) - данная команда означает начало передачи почтового сообщения (письма). В SMTP окончание почтового сообщения помечается при помощи точки в пустой строке. Почтовое сообщение должно содержать такую заголовочную информацию, как Date, Subject, To, Cc, From.

1.2. Протокол POP3

POP3 - почтовый протокол, с помощью которого пользователи могут забирать свою почту с почтовых серверов. Клиентская часть протокола выполняется на компьютере пользователя. Серверная часть

протокола выполняется на почтовом сервере.

Стандартный порт POP3 сервера - 110.

Начало взаимодействия клиента и сервера символизирует приветствие, которое сервер отсылает клиенту при успешном установлении соединения.

Сообщения, посылаемые клиентом серверу, называют командами; сообщения, посылаемые сервером клиенту, называют ответами. Среди команд клиента различают команды без параметров, команды с обязательными параметрами и команды с опциональными (не обязательными) параметрами.

Ответ сервера всегда начинается с одного из двух идентификаторов: «+OK», что означает успешное выполнение команды клиента, или «-ERR», что означает, что команда не была выполнена. После идентификатора выполнения команды может следовать более подробное пояснение. Окончанием команды, а также ответа, состоящего из одной строки, служит возврат каретки и перевод строки. Окончанием многострочного ответа служит точка в пустой строке.

После того, как клиент получил приветствие сервера, должна быть выполнена авторизация (состояние AUTHORIZATION). При успешной авторизации между клиентом и сервером начинается обмен сообщениями (состояние TRANSACTION). После команды QUIT и ответа сервера «+OK», сервер освобождает все ресурсы, которые были задействованы в состоянии TRANSACTION (состояние UPDATE).

Под допустимостью команд клиента при определенном состоянии протокола мы понимаем, что лишь эти команды сервер, возможно, выполнит успешно, а про все другие команды нам заведомо известно, что будет сгенерирован отрицательный ответ сервера. При успешном выполнении команд протокол может перейти из состояния AUTHORIZATION в состояние TRANSACTION, из состояния TRANSACTION в состояние UPDATE.

Полное описание протокола POP3 приведено в спецификации RFC 1460.

1.2.1. Некоторые команды протокола POP3.

Состояние AUTHORIZATION: допустимы команды USER, PASS, QUIT.

USER user_name (обязательный параметр) - с помощью этой команды клиент передает серверу имя пользователя.

PASS user_password (обязательный параметр) - с помощью этой команды клиент передает серверу пароль пользователя.

QUIT - с помощью этой команды клиент сообщает серверу о намерении разорвать соединение.

Состояние TRANSACTION: допустимы команды STAT, LIST, RETR, DELE, RSET, QUIT.

STAT - с помощью этой команды клиент запрашивает у сервера общую информацию о почтовом ящике пользователя (количество писем, суммарный размер).

LIST [number_of_msg] - с помощью этой команды клиент запрашивает у сервера более подробную информацию о почтовом ящике пользователя, а при задании параметра - непосредственно информацию об указанном письме.

Если задан номер несуществующего (например, отмеченного как удаленное) сообщения, то сервер отвечает "-ERR":

Если опциональный параметр не задан, то после идентификатора "+OK" сервер выдаст информацию о всех письмах в том же формате (номер письма, его размер), о каждом письме - в новой строке. Окончанием такого ответа будет служить точка в пустой строке.

RETR number_of_msg (обязательный параметр) - с помощью этой команды клиент запрашивает у сервера письмо с номером number of msg.

Если задан номер несуществующего сообщения, то сервер отвечает "-ERR".

DELE number_of_msg (обязательный параметр) - с помощью этой команды клиент информирует сервер о том, что следует удалить

письмо с номером number of msg.

Если в качестве параметра задан номер существующего сообщения, то сервер отвечает "+OK" и помечает письмо как удаленное.

RSET - с помощью этой команды клиент запрашивает восстановление всех писем, отмеченных как удаленные.

Письмо, отмеченное как удаленное, будет удалено в состоянии UPDATE (наступает после того, как сервер получит команду QUIT) - в этом состоянии сервер освобождает все ресурсы, которые были задействованы в состоянии TRANSACTION, удаляет письма, отмеченные как удаленные, посылает POP3-клиенту положительный ответ на команду QUIT. Затем TCP-соединение будет закрыто.

2. Порядок выполнения работы

Мы будем создавать программы работающие на стороне клиента и общающиеся с уже работающим почтовым сервером (процедуры выбора, запуска, настройки и контроля работы почтового сервера смотри в соответствующем методическом пособии.)

Для отправки сообщения на почтовый сервер при помощи компонент Indy, необходимо заполнить соответствующими данными компонент IdMessage, а затем передать его на сервер при помощи компонента IdSMTP.

Чтобы получить сообщение из почтового ящика, используется компонент IdPOP3, который возвращает объект IdMessage.

Для того чтобы получить общее представление о работе подобных приложений, рассмотрим некоторые процедуры.

2.1. Отправка сообщения:

```
procedure TForm1.btnSendClick(Sender: TObject);
  begin
    IdMessage1.Body.Assign(MemoOut.Lines);
    IdMessage1.From.Text := 'user1@server.ru';
    IdMessage1.ReplyTo.EMailAddresses := 'user1@server.ru';
    IdMessage1.Recipients.EMailAddresses := EditTo.Text;
    IdMessage1.Subject := EditSubject.Text;
    IdSMTP1.Username := 'user1';
    IdSMTP1.Password := 'user1';
    IdSMTP1.Host := '127.0.0.1';
    IdSMTP1.Port := 25;
    IdSMTP1.Connect;
    try
      IdSMTP1.Send(IdMessage);
    finally
      IdSMTP1.Disconnect;
    end;
  end;
```

2.2. Получение списка сообщений:

```
procedure TForm1.btnGetClick(Sender: TObject);
  var iMsgCount,i :integer;
begin
  if IdPOP3.Connected then
     begin
        IdPOP3.Disconnect;
     end;
  IdPOP3.Host := EditPOP3Host.Text;
  IdPOP3.Port := StrToInt(EditPOP3Port.Text);
  IdPOP3.Username := EditUserName.Text;
  IdPOP3.Password := EditUserPassword.Text;

  IdPOP31.Connect;
  iMsgCount := IdPOP31.CheckMessages;
  if iMsgCount > 0 then
    begin
      sgMes.Rows[1].Clear;
      for i := 1 to iMsgCount do
        begin
         IdMessage1.Clear;
         IdPOP3.RetrieveHeader(i, IdMessage);
         // Add info to StringGrid
         sgMes.Cells[0,i] := IntToStr(i);
         sgMes.Cells[1,i] := IdMessage.Subject;
         sgMes.Cells[2,i] := IdMessage.From.Text;
         sgMes.Cells[3,i] := DateToStr(IdMessage.Date);
         sgMes.Cells[4,i] :=
                   IntToStr(IdPOP3.RetrieveMsgSize(i));
      end;
   end
  else ShowMessage('Нет новых сообщений!');
end;
```

2.2. Получение конкретного сообщения:

```
procedure TForm1.Button1Click(Sender: TObject);
begin
  MemoIn.Clear;
  IdMessage.Clear;
  IdPOP3.Retrieve(sgMes.Row,IdMessage);
  MemoIn.Lines := IdMessage.Body;
end;
```

2.4. Разрыв связи с POP3-сервером:

```
procedure TForm1.Button2Click(Sender: TObject);
begin
  IdPOP31.Disconnect;
end;
```

3. Задание на самостоятельную работу

1. Реализовать программу-клиент, выполняющую запросы на прием почтовых сообщений от сервера по протоколу POP3. При входе программа должна запрашивать у пользователя адрес POP3-сервера, имя пользователя и пароль. Затем, в случае успешного соединения и авторизации, программа должна выдавать пронумерованный список сообщений с указанием размера каждого сообщения и далее выводить на экран выбранное пользователем сообщение.

2. Реализовать программу-клиент, формирующую запросы на передачу почтовых сообщений серверу по протоколу SMTP. При входе программа должна запрашивать у пользователя имя SMTP-сервера, имя пользователя, посылающего сообщение, и имя пользователя, для кого предназначается сообщение. Затем введенное пользователем с клавиатуры сообщение отсылается на сервер.

4. Контрольные вопросы

1. В чём преимущества обмена данными через почтовые службы?
2. Можно ли использовать нестандартные номера портов в программах-клиентах, взаимодействующих с почтовыми серверами?
3. Для чего используется компонент idMessage?
4. Как аутентифицировать пользователя при попытке отправить сообщение на почтовый сервер по протоколу SMTP?
5. Какие состояния клиента существуют при работе с почтовым сервером по протоколу POP3?
6. Как типы данных можно передавать с помощью протоколов SMTP и POP3?

Оглавление