Шарук Алексей Андреевич

Магистрант НАЧОУ ВПО СГА

Направление: Информатика и вычислительной техники

Магистерская программа: Распределенные автоматизированные системы

Разработка клиент-серверного чата средствами C++ Builder

Аннотация. Работая сотрудником одного из коммерческих банков, мне понадобился сетевой чат, найдя в сети десяток, я понял, что не один из них не устраивает меня в полной мере, и я решил написать свой. Просмотрев кучу форумов, я выделил для себя главные моменты и в итоге, за пару дней написал чат, устраивавший меня на все 100%. В данной статье выложены основы создания клиент-серверного чата, для создания которого вам хватит и десяти минут, ну а его доработка, ограничена только вашей фантазией и знанием языка программирования.

Ключевые слова: сетевой чат, клиент-сервер, C++ Builder, сокет.

Работая в сети, мы часто пользуемся многопользовательскими приложениями – почтовые клиенты, форумы, чаты, файловые клиенты и т.п. Все эти приложения используют для своей работы разные протокола, но базовым для них является единый протокол – TCP/IP. Типичное клиент-серверное приложение построенное на протоколе TCP/IP является чат реального времени. В этой статье мы создадим сетевой чат средствами C++ [1]. Для этой цели нам необходимы компоненты TClientSocket и TServerSocket. Эти компоненты при работе используют интерфейс сокетов.

Сокеты – это интерфейс прикладного программирования для сетевых приложений TCP/IP. В переводе с английского «sockets» – гнезда, т.е. сетевые приложения используют виртуальные разъемы для обмена данными между собой. Сокеты бывают трех видов: клиентские сокеты устанавливают связь с

сервером и обмениваются с ним данными, включен в компонент TClientSocket; слушающий сокет принимает запрос на соединение от клиентского сокета и соединяет сервер с клиентом, содержится в компоненте TServerSocket; серверный сокет обменивается данными с клиентом по уже установленному (слушающим сокетом) соединению.

Общий алгоритм работы сетевого чата очень прост: сервер открывает порт для прослушивания, клиент соединяется с сервером, клиент посылает сообщение серверу, сервер считывает данные с сокета, сервер отправляет сообщение клиенту, клиент и сервер разрывают соединение.

Приложение будет разработано с использованием среды разработки C++ Builder 2010 и одно и то-же приложение может быть как клиентом, так и сервером.

Итак, запускаем C++ Builder и создаем новый проект File – New – VCL Forms Application – C++ Builder

Добавляем на формы следующие компоненты и изменяем свойства:

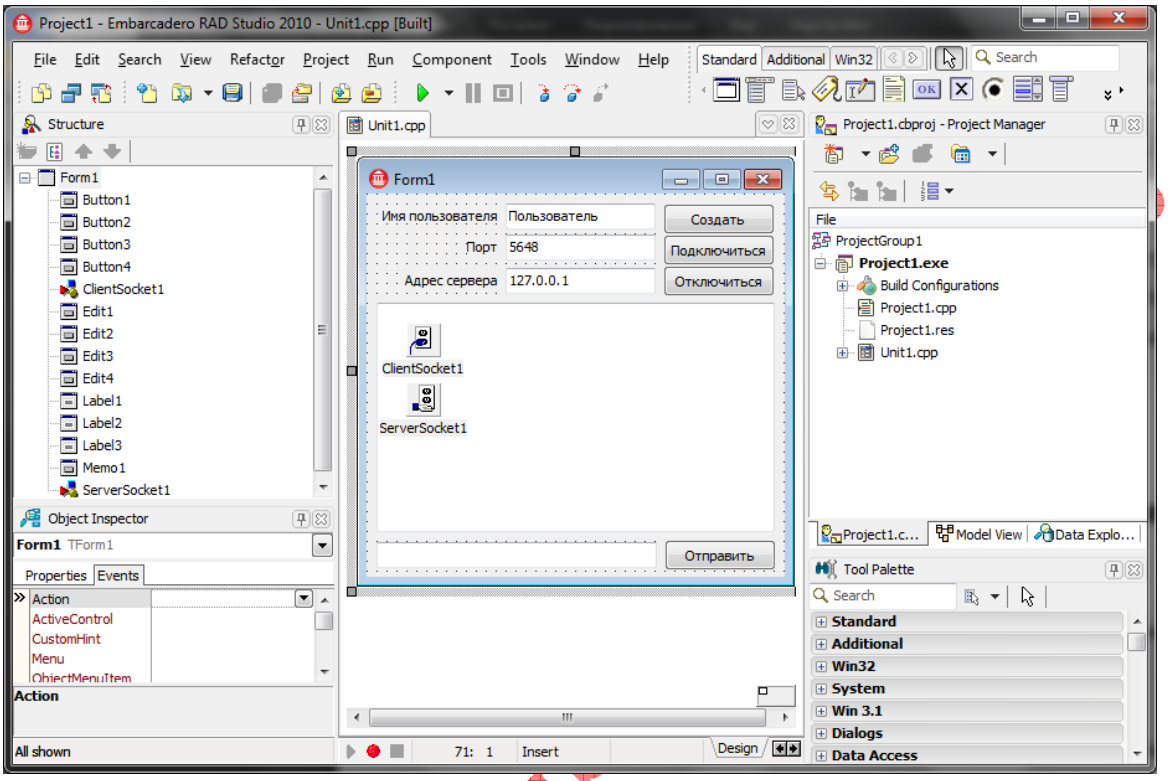
Label1 – в свойстве Caption пишем «Имя пользователя», Label2 – в свойстве Caption пишем «Порт» и в Label3 – в свойстве Caption пишем «Адрес сервера».

Edit1 – в свойстве Text пишем «Пользователь», здесь будет указано имя пользователя, Edit2 – в свойстве Text укажем «5648» – это будет наш порт для подключения, можно задать любой другой. В Edit3 – в свойстве Text пишем «127.0.0.1» – необходимо указать другой, но мы будем проверять работоспособность на локальном компьютере, Edit4 – в свойстве Text стираем все символы, – это окно предназначена для ввода отправляемого сообщения.

Button1 – в свойстве Caption пишем «Создать» – кнопка создания сервера, Button2 – в свойстве Caption пишем «Подключиться» – кнопка подключения к серверу, Button3 – в свойстве Caption пишем «Отключиться» – кнопка разрыва соединения, Button4 – в свойстве Caption пишем «Отправить» – кнопка отправки сообщения.

Memo1 – В свойстве Lines удаляем все строки.

Добавляем на форму элемент TClientSocket и TServerSocket с вкладки Internet.



Начиная с 7-й версии Builder-а, элементы TClientSocket и TServerSocket отсутствуют или при компиляции выдают ошибку. Например в C++ Builder 2010, необходимо войти в меню Component выбрать пункт Install Packages нажать кнопку Add, зайти в папку C:\Program Files (x86)\Embarcadero\RAD Studio\7.0\bin и выбрать файл dclsockets140.bpl, после этого с этими компонентами можно работать.

Начинаем писать код для компонентов с добавления универсальной функции по настройке кнопок интерфейса, например, без активного соединения кнопка «Отправить» должна быть неактивной:

```
void LockGUI (void){
if (Form1->ClientSocket1->Active == true || Form1->ServerSocket1->Active == true) {
//Отключаем поля ввода имени, порта, адреса и управляющие кнопки
Form1->Edit1->Enabled = false;
```

```
Form1->Edit2->Enabled = false;
Form1->Edit3->Enabled = false;
Form1->Button1->Enabled = false;
Form1->Button2->Enabled = false;
Form1->Button3->Enabled = true;
Form1->Button4->Enabled = true;
} else {
// Меняем все наоборот
Form1->Edit1->Enabled = true;
Form1->Edit2->Enabled = true;
Form1->Edit3->Enabled = true;
Form1->Button1->Enabled = true;
Form1->Button2->Enabled = true;
Form1->Button3->Enabled = false;
Form1->Button4->Enabled = false;}}
```

Сразу используем эту функцию в методе OnCreate элемента Form1:

```
LockGUI(); //Отключаем неактивными элементы управления.
```

Теперь создадим универсальную функцию отправки сообщения, функция будет проверять активен сервер или клиент, и в зависимости от результата выполнять определенные действия, назовем функцию SendingMessage, а отправляемое сообщение Message:

```
void SendingMessage (AnsiString Message){
//Проверяем если активен клиент
if (Form1->ClientSocket1->Active == true) {
//Отсылаем на сервер Имя пользователя и текст сообщения
Form1->ClientSocket1->Socket->SendText(Message);}
if (Form1->ServerSocket1->Active == true) {
//Если сервер активен, добавляем сообщение на экран
Form1->Memo1->Lines->Add(TimeToStr(Time()) + " – " + Message);
//Отправляем сообщение всем подключенным пользователям
```

for(int i = 0; i < Form1->ServerSocket1->Socket->ActiveConnections; i++) {

//Добавляем к сообщению время сервера и отправляем Клиентам

Form1->ServerSocket1->Socket->Connections[i]->SendText(TimeToStr(Time()) + " – " + Message); }}}

Начинаем писать код для визуальных компонентов. Кнопка «Создать»:

ServerSocket1->Port = Edit2->Text.ToInt(); //Задаем серверу порт

ServerSocket1->Active = true; //Включаем сервер

LockGUI(); //Отключаем неактивные элементы управления

Кнопка «Подключиться».

ClientSocket1->Port = Edit2->Text.ToInt(); //Даем клиенту порт

ClientSocket1->Host = Edit3->Text; //Даем клиенту адрес сервера

ClientSocket1->Active = true; //Подключаем клиента

LockGUI(); //Отключаем неактивные элементы управления

Кнопка «Отключиться».

ServerSocket1->Active = false; //Отключаем сервер

ClientSocket1->Active = false; //Отключаем клиента

LockGUI(); //Отключаем неактивные элементы управления

Метод OnError компонента ClientSocket1, метод начинает работать при возникновении ошибки на стороне клиента.

Memo1->Lines->Add("Ошибка подключения к " + Edit3->Text);

ClientSocket1->Active = false; //Отключаем клиента

LockGUI(); //Отключаем неактивные элементы управления

ErrorCode = 0; //Возвращаем код ошибки

Метод OnRead компонента ClientSocket1.

Memo1->Lines->Add(Socket->ReceiveText());

//Выводим на экран принятое сообщение

Метод OnClientRead компонента ServerSocket1.

//Читаем сообщения и рассылаем их через функцию SendingMessages()

SendingMessage(Socket->ReceiveText());

И наконец, действия при нажатии кнопки «Отправить»:

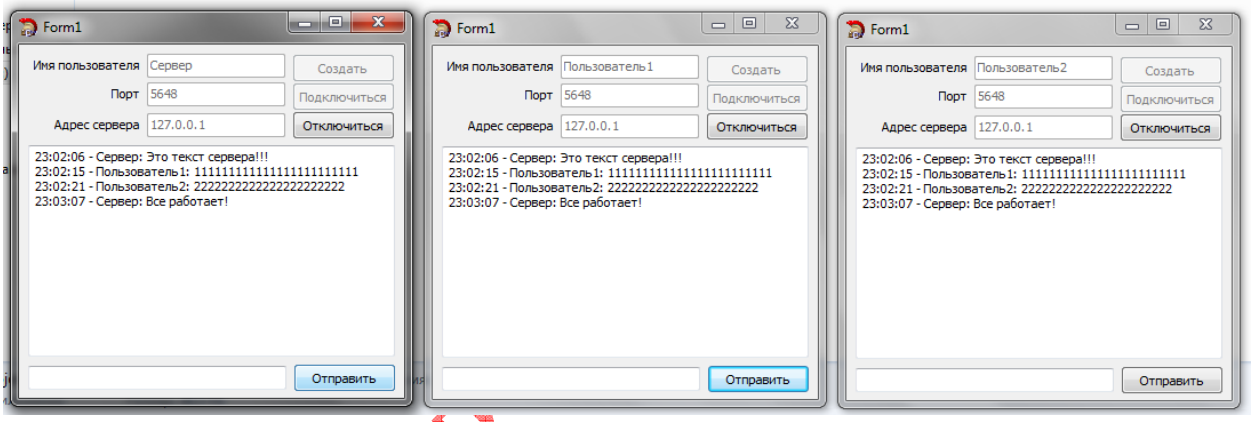
//Отправляем сообщение добавив к нему имя отправителя

SendingMessage(Edit1->Text + ": " + Edit4->Text);

Edit4->Text = ""; //Очищаем поле ввода

Сохраняем. Компилируем (клавиша F9, по умолчанию запускает проект и в папке Debug создается уже откомпилированная версия программы). Тестируем.

Поскольку сети у нас нет, мы используем локальный IP адрес для подключения. Запускаем 3 экземпляра программы, первый создает сервер чата, два другие – являются клиентами.



При незначительной доработке, возможно, отличать пользователя по цветам и создавать списки пользователей с возможностью отправки приватных сообщений и пересылкой файлов. На этом все, удачных разработок.

Литература

1. Архангельский А.Я. Программирование в C++ Builder 6. М.: БИНОМ, 2010.