UNIVERSIDADE TECNOLÓGICA FEDERAL DO PARANÁ

PROGRAMA DE ENSINO TUTORIAL DE ENGENHARIA ELETRÔNICA

Tutorial

PROGRAMAÇÃO C++ TCP/IP COM API

WINSOCKS 2

Autor: *Supervisor:*

Marcelo Hiroshi Sugita Dr. Fábio Schneider

07 de fevereiro de 2011

Sumário

1. INTRODUÇÃO

Com o crescimento acentuado na utilização dos computadores, e com a cada vez maior utilização da internet, surge a necessidade de se interligar as máquinas, formando-se uma grande rede de dados. Essas redes são possíveis através das conexões em LAN (locais) ou através da própria internet (globais). Para se administrar os dados compartilhados, é necessário se utilizar certos protocolos e sistemas.

1.1. Objetivo

Neste tutorial será explicado a utilização de um desses protocolos que possibilita a troca de dados entre computadores. O protocolo TCP/IP, que será o alvo do estudo, possibilita tanto a troca de dados utilizando-se conexões locais sem a necessidade da internet como a troca de dados através da internet. No tutorial será focado a troca de dados entre computadores conectados por meio de uma conexão local.

1.2. Condições de Trabalho

O tutorial utilizará a linguagem de programação orientada a objetos C++, no ambiente Microsoft Visual Studio 2008 Express Edition, com a API do Windows para programação de aplicativos TCP/IP utilizando Winsocks 2.

1.3. Pré-Requisitos

Para se tirar o máximo proveito deste tutorial, é necessário ter um conhecimento em linguagem de programação C++.

2. CONCEITOS NECESSÁRIOS

2.1. Servidor e Cliente

Para todo tipo de conexão existente entre dois ou mais computadores existe ao menos um servidor e um cliente.

Servidor em geral é quem recebe as conexões. Há diferentes tipos de servidor, sendo possível receber, enviar, armazenar dados.

Cliente em geral é quem se conecta. Ele pode requisitar dados ao servidor, enviar, solicitar arquivos.

No exemplo deste tutorial, o servidor será quem irá receber dados e o cliente quem os enviará.

É importante frisar que numa conexão nem sempre as funções são fixas. Um computador pode num dado momento ser um servidor, e num outro um cliente (cliente-servidor). Para motivo de teste, um computador pode também ser ao mesmo tempo um servidor e um cliente, utilizando o seu próprio endereço.

2.2. Endereço IP

O endereço IP (Internet Protocol) de uma forma geral é o endereço que indica o local de uma determinada máquina numa rede, ou seja, como uma máquina enxerga a outra. Esse endereço é utilizado para estabelecer uma conexão remota entre computadores.

2.3. Protocolos de Transporte

O protocolo de transporte de uma forma geral é o modo como serão trocados os dados entre os computadores em uma rede. Os mais conhecidos são TCP (Transmission Control Protocol) e UDP (User Datagram Protocol).

O TCP é utilizado em protocolos de alto nível como HTTP e FTP, e é nele que se faz a maior parte de transferência de dados na internet. Ele é mais seguro pois verifica se a seqüência de dados enviada foi de forma correta, na seqüência apropriada e sem erros. Porém, isso o torna mais lento que o UDP.

O UDP é utilizado em situações em que a velocidade se sobrepõe a qualidade (os dados chegarem certo), como é o caso de aplicações em streaming, como vídeos ao vivo, chamadas de voz. Ele também permite a um único cliente enviar o mesmo pacote de dados para vários outros. Apesar de ser mais rápido, ele não dá a garantia que os pacotes de dados chegaram ao destinatário, se chegaram na ordem correta, ou se dados foram perdidos.

2.4. Porta

Em um computador (endereço IP, para uma rede) há vários processos (programas) sendo executados em um momento. Quando um pacote de dados é recebido, é a porta que direciona tal pacote de dados ao processo certo. A porta é, então, um número que associa os pacotes recebidos a um dado processo em execução numa máquina

2.5. Socket

O socket é a combinação do endereço IP e a porta de determinado processo executado no endereço IP. O socket é o necessário para se transmitir dados através dos protocolos de transporte (TCP e UDP).

3. CLASSES, VARIÁVEIS E FUNÇÕES BÁSICAS DA API

3.1. Classes, Variáveis e Estruturas Básicas

Para programar utilizando a API do Windows, é interessante conhecer as variáveis e estruturas que são utilizadas durante a programação, nas funções que serão utilizadas.

- sockaddr_in – estrutura derivada da sockaddr especializada para trabalhar com o protocolo TCP/IP.

```
struct sockaddr_in
{
   short   sin_family;
   u_short sin_port;
   struct  in_addr sin_addr;
   char    sin_zero[8];
};
```

sin_family – define o protocolo de transporte, como no caso é TCP ou UDP o utilizado é *"AF_INET";*

sin_port – o número da porta a ser utilizada nesse processo;

sin_addr – o endereço IP;

sin_zero – não é utilizado;

- SOCKET – estrutura que receberá o socket criado pela função *socket(...)*. Posteriormente através da função *bind(...)* receberá um endereço IP e uma porta. Será utilizada em quase todas as funções de troca de dados.

- WSADATA – estrutura que contém os detalhes da implementação de socket no Windows

Também serão utilizadas algumas variáveis já conhecidas, tanto para fazer o controle como também para receber o endereço de IP e porta.

3.2. Funções Básicas

3.2.1. WSAStartup (...)

A função WSAStartup(...) inicializa o winsocks (implementação de sockets do Windows) e verifica se o computador suporta a versão do winsocks a ser utilizada.

```
int WSAStartup(
                    WORD wVersionRequested,
                    LPWSADATA lpWSAData
              );
```

No primeiro parâmetro deve ser fornecida a versão do winsocks a ser utilizada. Para se utilizar a versão 2.2 (ultima) é feito: *MAKEWORD(2,2)*

No segundo parâmetro é enviado um ponteiro para uma estrutura *WSADATA*, a qual receberá os detalhes da implementação do socket.

A função retornará 0 se não ocorrer nenhum erro na inicialização. Caso contrario, para cada erro ele retornará um valor diferente.

3.2.2. WSACleanup()

A função WSACleanup(...) finaliza o uso do winsocks.

```
int WSACleanup( );
```

3.2.3. socket(...)

A função socket(...) criará um socket para determinado tipo de protocolo de transferência. A função retornará o socket criado, que será necessário para maioria das funções do tutorial.

```
SOCKET socket(
                        int af,
                        int type,
                        int protocol
            );
```

No primeiro parâmetro deve ser fornecido o tipo de endereço utilizado. Para protocolo TCP ou UDP utilizamos *"AF_INET"*

No segundo parâmetro define-se o tipo de socket a ser criado. "*SOCK_STREAM"* para criar um socket TCP, e "*SOCK_DGRAM"* para um socket UDP.

No ultimo parâmetro define-se o protocolo a ser utilizado no socket. "*IPPROTO_TCP"* para utilizar o protocolo TCP.

3.2.4. closesocket (...)

A função closesocket(...) é utilizada para encerrar um socket criado pela função *socket*(...). Todos os sockets criados pela função devem ser encerrados utilizando o *closesocket(...)*.

```
int closesocket(SOCKET s);
```

O único parâmetro a ser enviado é o próprio socket a ser encerrado.

3.2.5. bind (...)

A função bind(...) atribui a um socket um endereço IP e uma porta que foram anteriormente definidas numa estrutura *sockaddr_in*. Essa função só é necessária na inicialização do socket do servidor.

```
int bind(
                SOCKET s,
                const struct sockaddr *name,
                int namelen
        );
```

No primeiro parâmetro coloca-se o socket a receber o endereço IP e a porta.

No segundo parâmetro coloca-se o *sockaddr_in* criado anteriormente. O único detalhe é que se faz necessário fazer uma conversão para *sockaddr*.

```
reinterpret_cast < SOCKADDR* > ( &name )
```

No ultimo parâmetro coloca-se o tamanho da estrutura *sockaddr.*

```
sizeof ( meu_end )
```

3.2.6. listen (...)

A função listen(...) coloca o socket (já configurado) em estado de escuta. Dessa forma, ele pode receber a conexão de outro socket (de outra máquina) a fim de estabelecer uma comunicação. Essa função só é necessária no servidor.

```
int listen(SOCKET s, int backlog);
```

O primeiro parâmetro é o socket que será colocado em estado de escuta. Ou seja, o socket do seu próprio programa.

O outro parâmetro é a quantidade de sockets que poderão se conectar ao seu sistema.

3.2.7. accept (...)

A função accept(...) é utilizada após um socket ser colocada em estado de escuta e há uma conexão pendente nele. Essa função aceita a conexão feita por um outro computador (socket remoto) e retorna o socket com as informações do computador remoto.

```
SOCKET accept(SOCKET s, struct sockaddr *addr, int *addrlen);
```

O primeiro parâmetro é o socket que está em estado de escuta.

O segundo parâmetro é um ponteiro para uma estrutura addr_in, contendo as informações do computador remoto.

O ultimo parâmetro é o tamanho da estrutura addr_in presente no segundo parâmetro dessa função.

3.2.8. connect (...)

A função connect(…) é o casal das funções listen(...) e accept(…). Isso porque enquanto as duas ocorrem em um lado da conexão (servidor), a connect(...) acontece do outro lado (cliente). É através dessa função que um computador se conecta a outro. Porém, para que isso aconteça, é necessário saber o endereço IP e a porta de acesso para o outro computador.

```
int connect(SOCKET s, const struct sockaddr *name, int namelen);
```

O primeiro parâmetro é o socket criado através da função socket(...) para um determinado protocolo de transferência. Apesar de ser o socket criado na máquina cliente, não é necessário fazer o bind(...).

O segundo parâmetro é o *sockaddr_in* que deve ser criado e possuir o endereço IP e porta do servidor ao qual se quer conectar.

O ultimo parâmetro é o tamanho da estrutura presente no segundo parâmetro dessa função.

3.2.9. send (...)

A função send(…) é utilizada para enviar os dados de um computador ao outro. Após realizar a conexão através dos métodos citados acima, as duas máquinas são capazes de estabelecer uma troca de dados, que se dá por essa função em conjunto à recv(...). A função send(...) pode ser utilizada tanto pelo cliente como pelo servidor, o detalhe é que para enviar algo, o outro precisa receber.

```
int send(SOCKET s, const char *buf, int len, int flags);
```

O primeiro parâmetro é o socket que contém as informações do computador remoto a que se deseja enviar os dados.

O segundo parâmetro é um ponteiro do tipo char (ou seja, qualquer dado que será enviado deve ser anteriormente convertido em chars) cujo conteúdo será o que será mandado ao outro computador.

O terceiro parâmetro é um inteiro com o tamanho do buffer a ser enviado.

O ultimo parâmetro é o modo como a função se comporta, e dificilmente será utilizado um valor diferente de 0.

3.2.10. recv (...)

A função recv(...) é o par da função send(...). Para cada send(...) deve haver um recv(...). Essa função recebe no buffer toda informação procedente do socket apontado.

```
int recv(SOCKET s, char *buf, int len, int flags);
```

O primeiro parâmetro é o socket que contém as informações do computador remoto do qual se deseja receber os dados.

O segundo parâmetro é um ponteiro do tipo char que receberá o conteúdo procedente do outro computador.

O terceiro parâmetro é um inteiro com o tamanho do buffer recebido.

O ultimo parâmetro é o modo como a função se comporta, e dificilmente será utilizado um valor diferente de 0.

4. A PROGRAMAÇÃO

Para programar utilizando redes é necessário primeiramente acrescentar ao projeto a biblioteca que é responsável pelas funções que gerenciam os sockets e, assim, a rede. Essa biblioteca faz parte da API (Application programming interface) do Windows e é incluída utilizando o seguinte comando:

```
#include <winsock2.h>
```

Dessa forma, já estamos aptos a utilizar todas as funções que foram explicadas nesse tutorial, a fim de se estabelecer uma conexão entre computadores.

É importante salientar que é necessário programar tanto a parte do servidor quanto a do cliente, sendo duas partes distintas e especificas do código.

A primeira função a ser chamada deve ser a WSAStartup(...) de inicialização do winsocks. Essa função pode ser chamada na própria construtora do seu programa, uma vez que quando iniciada, pode ser utilizada até o seu encerramento.

A seguir deve-se criar os sockets a serem usados, tanto no servidor quanto no cliente. A peculiaridade é que no servidor deve ser criado um socket com o

endereço IP da sua própria máquina – seja na rede interna ou visto pela internet, pode-se utilizar o valor 0 por padrão-, enquanto no cliente o socket deve possuir o endereço IP da máquina remota a que se quer conectar.

A fim de se descobrir o endereço IP de sua máquina numa rede, pode ser usado o próprio DOS do Windows, por meio do prompt de comando.

Abra menu iniciar do Windows e em seguida execute o "cmd". Digitando "ipconfig" no prompt de comando, o seu endereço IP será mostrado na tela. Outro comando interessante é o "netstat" que exibe uma lista de todas as conexões feitas com o computador.

Sabendo o endereço IP necessário, segue-se distintamente a parte do cliente e do servidor. Enquanto o servidor amarra o endereço IP e a porta no socket criado (função *bind(...)*), e a seguir entra em fase de escuta e aceitação de conexões (funções *listen(...)* e *accept(...)*), o cliente deve tentar se conectar com o computador remoto (função *connect(...)*).

Então, quando a conexão estiver estabelecida, basta uma das partes enviar dados pela função *send(...)* enquanto a outra estiver recebendo pela função *recv(...)*. Para tanto, deve-se fazer algum controle para saber quando enviar e quando receber, ou até mesmo utilizar threads para administrar essa parte.

No código exemplo que segue há exemplos de controles utilizados para enviar dados ou até mesmo classes (convertendo anteriormente para bytes/chars para ser enviados), e saber que tipo de dado está sendo recebido no momento.

5. PROGRAMA EXEMPLO

O programa exemplo pode ser baixado no site do Professor Jean Marcelo Simão, cujo endereço está indicado nas referências.

O código foi desenvolvido por André de Castilho Costa Pinto, em 2007, então aluno do Professor Simão.

6. REFERENCIAS BIBLIOGRÁFICAS

SIMÃO, J. M. Página de Internet do Prof. Simão – Disponível em <http://pessoal.utfpr.edu.br/jeansimao/> Acessado em 6 de Fevereiro de 2011.

Slides de aula do Professor Jean Marcelo Simão. Disponível em <http://pessoal.utfpr.edu.br/jeansimao/> Acessado em 6 de Fevereiro de 2011.

Slide TCP/IP em Winsocks 2 por de Castilho Costa Pinto. Disponível em <http://pessoal.utfpr.edu.br/jeansimao/Fundamentos2/APITCPIP/Fundamentos2-SlidesC++20-2009-11-16.pdf > Acessado em 6 de Fevereiro de 2011.

Códigos do programa exemplo por de Castilho Costa Pinto. Disponível em <http://pessoal.utfpr.edu.br/jeansimao/Fundamentos2/APITCPIP/codigoapitcpip.zip> Acessado em 6 de Fevereiro de 2011.

MICROSOFT. Windows Sockets 2. Disponível em <http://msdn.microsoft.com/en-us/library/ms740673(v=vs.85).aspx> Acessado em 6 de Fevereiro de 2011.