

[Home](#)[Arquivo](#)[Colunas](#)[Colaboradores](#)[Cursos](#)[Enviar Dica](#)[F.A.Q](#)[Cadastro](#)[Contato](#)[RSS](#)

JÚLIO CEZAR NEVES

PAPAI DO SHELL



COM O  
SHELL DÁ  
PARA  
FAZER  
TUDO,  
TUDO!



você está aqui: [Home](#) → [Arquivo de Mensagens](#)



## Programando Socket em C++ sem segredo

**Colaboração: Alessandro de Oliveira Faria**

**Data de Publicação: 19 de Dezembro de 2004**

As informações contidas neste texto foram baseadas em um artigo escrito por Rob Tougher (<rtougher (a) yahoo com>), publicado na LinuxGazette em Janeiro de 2002. Inclusive os fontes-exemplo são apontados para o link original.

O motivo principal da elaboração deste texto, pois a dificuldade de encontrar algo simples para iniciantes que desejam programar socket em C++.

Sockets são mecanismos usados para a troca de dados entre processos, que podem estar todos em uma máquina local ou em diversas máquinas.

Estou desenvolvendo uma aplicação, na qual diversos clientes em diferentes plataformas e linguagens precisam enviar comandos para o software executar determinadas tarefas.

Este texto contém fundamentos básicos para a criação de aplicações que se comunicam via socket, assim aprenderemos a usar as classes ClientSocket e ServerSocket em C++.

## Visão geral sobre comunicação Cliente-Servidor

---

Antes de partirmos para o código-fonte, vamos entender o funcionamento dos processos Cliente e Servidor:

- Servidor: Estabiliza o modo ouvinte, aguardando a conexão com um processo cliente
- Cliente: Cria um socket e tenta conectar-se com um servidor
- Servidor: Aceita uma conexão cliente
- Cliente: Envia e recebe dados
- Servidor: Envia e recebe dados
- Cliente: Fecha conexão
- Servidor: Fecha conexão

Este é o funcionamento básico de uma comunicação via socket. Primeiro, o servidor entra em modo escuta, aguardando um processo cliente solicitá-lo.

Um processo cliente ao criar um socket tenta imediatamente conectar-se com o processo SERVER. Assim que a conexão foi estabelecida, ambos os processos começam a trocar dados. Qualquer um dos lados pode encerrar a conexão, fechando o socket.

## Executando um simples Server e Client

---

Vamos agora colocar a mão na massa! Criaremos simples aplicações cliente-servidor suficientes para o aprendizado e para fundamentarmos conceitos de comunicação via socket.

## Servidor - Entrando em modo ouvinte

---

A primeira ação de uma aplicação Server é de entrar em modo escuta por meio de uma

porta especificada, por onde a aplicação cliente solicitará a conexão.

### Listagem 1: Criando um servidor socket ( parte do fonte simple\_server\_main.cpp )

---

```
#include "ServerSocket.h"
#include "SocketException.h"
#include

int main ( int argc, int argv[] )
{
    try
    {
        // Criação do server socket na porta
        30000
        ServerSocket server ( 30000 );

        // insira aqui o restante do código

    }
    catch ( SocketException& e )
    {
        // Tratamento de erro
        std::cout << "Exceção foi capturada:"
        << e.description() << " Finalizando. ";
    }

    return 0;
}
```

O mais importante na listagem acima é a instanciação da Classe ServerSocket que coloca a aplicação em modo escuta. Se algum método dessa classe falhar, o tratamento de erro invocará o método description(), pertencente à classe SocketException.

### Cliente - Conectando com o server

---

O segundo passo é criar uma típica conexão cliente-servidor. A função da aplicação cliente é de tentar se conectar com a aplicação server. Vejamos como efetuar esse procedimento na listagem abaixo:

Listagem 2 : criando um cliente socket ( parte do fonte simple\_client\_main.cpp )

```
#include "ClientSocket.h"
#include "SocketException.h"
#include
#include
```

```
int main ( int argc, int argv[] )
{
    try
    {
        // Criando um client socket e
        solicitando a conexão na porta 30000 e IP
        local.
        ClientSocket client_socket (
            "localhost", 30000 );

        // O restante do código vem aqui.

    }
    catch ( SocketException& e )
    {
        std::cout << "Exceção foi capturada:"
        << e.description() << " ";
    }

    return 0;
}
```

Resumindo na instanciação da classe ClientSocket, a aplicação tenta conectar no IP e na porta especificada. Se o construtor falhar, o tratamento de erro invoca o método description() conforme mencionado no exemplo do Server.

### Server - Aceitando a conexão cliente

---

O próximo passo é o servidor estabilizar a conexão solicitada pela aplicação Client. Após estabilizar a conexão, um canal de comunicação entre os sockets é criado.

Listagem 3: Estabilizando a conexão com a aplicação cliente (parte do fonte simple\_server\_main.cpp)

```
#include "ServerSocket.h"
#include "SocketException.h"
#include

int main ( int argc, int argv[] )
{
    try
    {
        // Criação do server socket na porta
        30000
        ServerSocket server ( 30000 );

        while ( true )
        {
            // Aceitando a conexão solicitada
            ServerSocket new_sock;
            server.accept ( new_sock );

            // O restante do código vem aqui
```

```
}  
}  
catch ( SocketException& e )  
{  
    // Tratamento de erro  
    std::cout << "Exceção foi capturada:"  
<< e.description() << " Finalizando. ";  
}  
  
return 0;  
}
```

A conexão é estabelecida justamente na chamada do método `accept`, que aceita a conexão e preenche em `new_sock` as informações referentes à conexão atual.

### Cliente e Server - Enviando e recebendo dados

---

Agora que estabelecemos conexão, vamos para a parte de envio e recebimento de dados através do socket.

Uma das grandes características do C++ é a habilidade de sobrecarga de operadores. Usando a sobrecarga de com os operadores `<<` e `>>`, podemos escrever e ler dados nas classes `ClientSocket` e `ServerSocket`.

Listagem 4: Implementações no programa fonte ( `simple_server_main.cpp` )

```
#include "ServerSocket.h"  
#include "SocketException.h"  
#include  
  
int main ( int argc, int argv[] )  
{  
    try  
    {  
        // Criação do server socket na porta  
30000  
        ServerSocket server ( 30000 );  
  
        while ( true )  
        {  
            // Aceitando a conexão solicitada  
            ServerSocket new_sock;  
            server.accept ( new_sock );  
  
            try  
            {  
                while ( true )  
                {  
                    std::string data;  
                    // Recebendo dados  
                    new_sock >> data;
```

```
// Enviando dados
new_sock << data;
}
catch ( SocketException& ) {}
}
catch ( SocketException& e )
{
// Tratamento de erro
std::cout << "Exceção foi capturada:"
<< e.description() << " Finalizando. ";
}

return 0;
}
```

O `new_sock` é uma variável que contém todas as informações usadas para a troca de dados com o client. O comando "`new_sock >> data;`" armazena os dados recebidos na variável `data`.

A próxima linha é muito parecida, mas executa a tarefa inversa, ou seja, transmite o conteúdo da variável `data` para o Client.

Resumindo: o programa Server ecoa os dados recebidos pelo Client. Para comprovarmos essa funcionalidade, o fonte abaixo envia uma string ao cliente e aguarda o recebimento do Server, imprimindo o resultado no vídeo.

Listagem 5: implementações no client ( `simple_client_main.cpp` )

```
#include "ClientSocket.h"
#include "SocketException.h"
#include
#include

int main ( int argc, int argv[] )
{
try
{
// Criando um client socket e
solicitando a conexão na porta 30000 e IP
local.
ClientSocket client_socket (
"localhost", 30000 );

std::string reply;
try
{
// Envia a string "Test message" ao
server
client_socket << "Test message.";

// Recebe dados do server
client_socket >> reply;
```

```
std::cout< catch ( SocketException& )
{}

}
catch ( SocketException& e )
{
std::cout << "Exceção foi capturada:"
<< e.description() << " ";
}

return 0;
}
```

A aplicação acima, após enviar a string "Test Message" para o Server, aguarda a resposta do Server e ao recebê-la, a string é mostrada no vídeo.

## Compilando e testando os fontes

---

Como mencionei no início desse texto, em [www.linuxgazette.com](http://www.linuxgazette.com), podemos encontrar todos os fontes citados neste documento. Segue abaixo a relação dos arquivos fontes:

- [Makefile](#): O arquivo Makefile para teste projeto
- [Socket.h e Socket.cpp](#) : As classes de objetos Socket, implementadas nos testes deste documento e <http://www.linuxgazette.com/issue74/misc/tougher/Socket.cpp.txt>
- [simple\\_server\\_main.cpp](#): Arquivo server principal
- [ServerSocket.h e ServerSocket.cpp](#): As classes de SocketServer e <http://www.linuxgazette.com/issue74/misc/tougher/ServerSocket.cpp.txt>
- [simple\\_client\\_main.cpp](#): Arquivo client principal
- [ClientSocket.h e ClientSocket.cpp](#): As Classes de ClientSocket e <http://www.linuxgazette.com/issue74/misc/tougher/ClientSocket.cpp.txt>

## Compilando e Testando

---

A compilação é simples. Primeiro, salve todos os projeto em um diretório. Logo após, entre com o seguinte comando:

```
[root@athlon socket]# make
```

Este comando compilará todos os arquivos do projeto e criará o binário `simple_server` e `simple_client`. Para testá-los, devemos

[Dicas-L] Programando Socket em C++ sem segredo  
executá-los um em cada terminal (primeiro o  
server e depois o cliente).

```
[root@athlon socket]# cd
/mnt/D/linux/fontes/c++/tcp-
socket/socket/
[root@athlon socket]# ./simple_server
running...
Test message.

[root@athlon root]# cd
/mnt/D/linux/fontes/c++/tcp-
socket/socket/
[root@athlon socket]# ./simple_client
We received this response from the
server:
"Test message."
[root@athlon socket]#
```

O cliente envia a string para o server e aguarda o recebimento dos dados. Após o retorno do Server, a resposta é mostrada no vídeo. Nota-se que o client termina assim que a mensagem é retornada.

Mas o server ficará no modo escuta até que seja interrompido com o CTRL+C.

O Socket é uma maneira simples e eficiente para troca de dados entre processos. Com esse documento, você poderá implementar os recursos sockets em suas aplicações.

Ressalto que este documento é uma versão traduzida/adaptação do artigo LinuxGazette elaborada por Rob Tougher.

Alessandro de Olivera Faria (Cabelo)  
<alessandrofaria (a) netitec com br>

**Veja a relação completa dos artigos de  
Alessandro de Oliveira Faria**