Departamento de Ciência da Computação

2o. Semestre de 2016

Trabalho Prático I de Redes de Computadores:

Um Servidor e Cliente de arquivos em IPV6

Eugênio Pacceli Reis da Fonseca, Otávio Augusto de O. Souza

Universidade Federal de Minas Gerais (UFMG)

{eugenio.pacceli,oaugusto}@dcc.ufmg.br

1. Introdução

Neste trabalho implementamos uma aplicação capaz de listar e transferir arquivos remotamente de um nó servidor da internet para outro nó cliente. As requisições são tratadas pelo servidor, podendo estas serem exibir o conteúdo de um diretório ou enviar um arquivo pertencente ao diretório.

Todo projeto foi implementado usando a linguagem C++ e máquinas Linux como cliente/servidor. Além disso, a aplicação do servidor é capaz de atender várias requisições simultaneamente utilizando de threads para cumprir as tarefas. Como especificado na documentação, foi utilizado o protocolo Ipv6 para transferência dos dados.

A figura abaixo mostra o escopo da aplicação cliente/servidor para o trabalho:

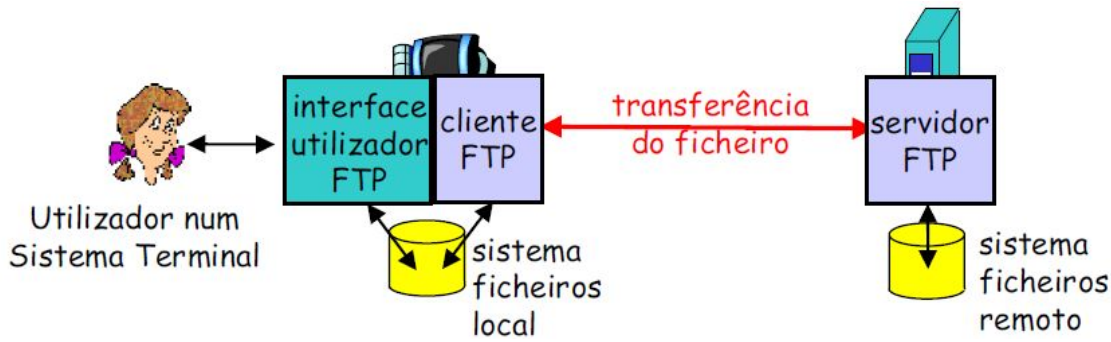

Figura 1: Aplicação FTP Cliente/Servidor

2. Modelagem

A linguagem C++, por implementar o paradigma orientado por objetos, permite uma modularização mais completa do código, sem perder os poderes que herdou da linguagem C.

Fazer o servidor e cliente divididos por módulos (descritos pelas classes), significou reduzir o problema em vários pedaços menores. O controle de acesso por meio das definições de público e privado de cada classe, permite à cada módulo garantir seu bom funcionamento, sem perder a capacidade de interagir com os outros. O conceito de instanciar objetos das classes (módulos) permite que condições iniciais sejam satisfeitas na criação de cada instância de um módulo, e o reaproveitamento de código torna a tarefa de programar mais produtiva.

A divisão por módulos também permitiu que a equipe trabalhasse em trechos de códigos diferentes, por integrante, e através da definição de interfaces e boas normas, foi possível produzir código válido sem precisar saber de todos os detalhes das outras partes do programa, dado que essas também devem funcionar como unidades separadas e manter suas condições de validez.

Basicamente, o código fonte responsável pelo servidor tem seus módulos a parte, assim como o código fonte responsável pelo cliente, mas ambos compartilham a API de Sockets, definida na pasta "./socketAPI/".

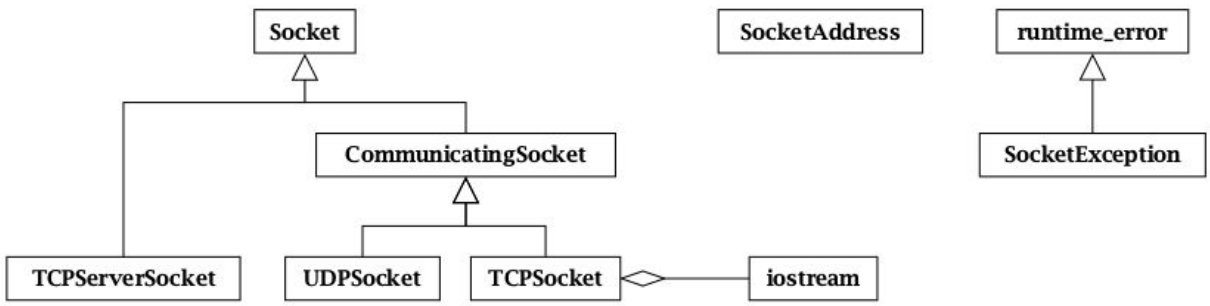

Figura 2. Modelo UML das classes implementadas em socketAPI

3. Implementação

O problema foi modelado no paradigma orientado por objetos, em C++ 2011. Criamos uma API para mexer com os sockets IPV6 de C, que chamamos de SocketAPI, além de classes para tratar dos problemas do servidor e do cliente.

A aplicação se divide basicamente em dois programas: O ClientFTP, que faz queries por arquivos e listas, e o servidorFTP, que atende às queries dos clientes. O servidorFTP utiliza a biblioteca pthreads, pois dispara uma thread por solicitação a ser atendida.

Abaixo uma breve descrição das responsabilidades e atributos de cada classe:

3.1 FileDeliveryIPV6Client:

Implementa a lógica de manipulação dos sockets para que o programa cliente receba a lista de arquivos em um diretório, do servidor, ou o arquivo propriamente dito (e salve-o no disco). Como utilizar o binário está definido na documentação inicial do Trabalho Prático I.

Função list: Solicita, recebe e exibe na tela a lista dos arquivos disponíveis em um diretório para download. Durante o recebimento dos pacotes, cujo conteúdo são strings puras, o cliente também os imprime imediatamente, na ordem. Após a solicitação por "list", o cliente houve o servidor até receber o último pacote.

Função get: Solicita ao servidor e recebe um arquivo. Recebe todos os pacotes do servidor, que contêm os bytes (na ordem de aparecimento) do arquivo, e grava o conteúdo, na ordem de recebimento, imediatamente no disco, formando, até o fim da recepção, um clone do arquivo original do servidor. Os pacotes contêm apenas chunks de bytes do arquivo original.

3.2 FileDeliveryIPV6Server:

Implementa a lógica de manipulação dos sockets para ouvir por requisições de programas clientes e atendê-las. Como utilizar o binário está definido na documentação inicial do Trabalho Prático I. O tamanho do buffer de leitura de arquivo do disco foi

definido como tendo o mesmo tamanho do buffer a ser enviado em cada pacote, tornando buffers maiores ainda mais atrativos.

3.3 Socket, ChannelSocket, TCPSocket, TCPServerSocket:

Implementam os sockets das bibliotecas de sockets padrão da linguagem C, em um paradigma orientado por objetos. Provêm serviços definidos no diagrama de Berkeley, como connect(), bind(), accept(), send() e recv(). Todas as funções de rede desse trabalho, tanto do cliente quanto do servidor, foram definidas utilizando objetos dessas classes.

3.4 OSServices:

A classe OSServices é responsável por pegar a lista de arquivos em um diretório, com auxílio do sistema operacional, além de também prover um serviço de saneamento de strings de diretório, para uso no programa servidor. A lista de arquivos em um diretório é obtida com uma consulta pelo comando “dir”, com alguns parâmetros desejados, nativo dos sistemas operacionais UNIX.

3.5 GlobalErrorTable:

Essa classe tabela e provêm todas as mensagens de erros, assim como definidas na tabela de falhas (Figura 3).

Código de Erro	Descrição do erro
-1	Erros nos argumentos de entrada
-2	Erro de criação de socket
-3	Erro de bind
-4	Erro de listen
-5	Erro de accept
-6	Erro de connect
-7	Erro de comunicação com servidor/cliente
-8	Arquivo solicitado não encontrado
-9	Erro em ponteiro
-10	Comando de clienteFTP não existente
-999	Outros erros (não listados)

Figura 3.Tabela de erros

3.6 SocketException, CommunicatingService:

Classes auxiliares, usadas nos serviços providos pelas classes de Socket.

4. Testes

Os testes foram realizados buscando cobrir todas requisições do trabalho. Com o desenvolvimento modular do programa cada parte foi testada em desenvolvimento, garantindo a funcionalidade e corretude do código. Além disso, procurou-se testar o programa em diferentes máquinas a fim de cumprir com as requisições e protocolo.

Os testes principais foram:

1. Listagem remota de diretório vazio e não vazio;
2. Requisição de arquivo não existente;
3. Erros de paramêtros passados para aplicação;
4. Servidor desconectado;
5. Conexão falha com o servidor.

Para todos os testes listados o programa terminou corretamente e para os casos de erro todos foram reportados seguindo as tabela da especificação abaixo. Erros além dos especificados foram impressos na saída padrão de erro (“cerr“ C++).

5. Resultados

Para avaliar o tempo de resposta em função do tamanho do buffer foram feitos diversos testes com cliente/servidor, o cliente estava em Belo Horizonte, Minas Gerais, Brasil durante o teste, e a máquina servidora era um máquina virtual, da Digital Ocean, hosteada em Nova Iorque (a cidade norte americana). A banda máxima entre o cliente e Nova Iorque, de acordo com o SpeedTest.net, era 29.75Mbps, e a latência 190ms. Tanto cliente e servidor rodavam a última versão do Ubuntu (16.10) x64.

Para diferentes tamanhos de buffer, foi requisitado um arquivo padrão para avaliar o tempo de resposta. Os dados obtidos foram plotados no gráfico a seguir:

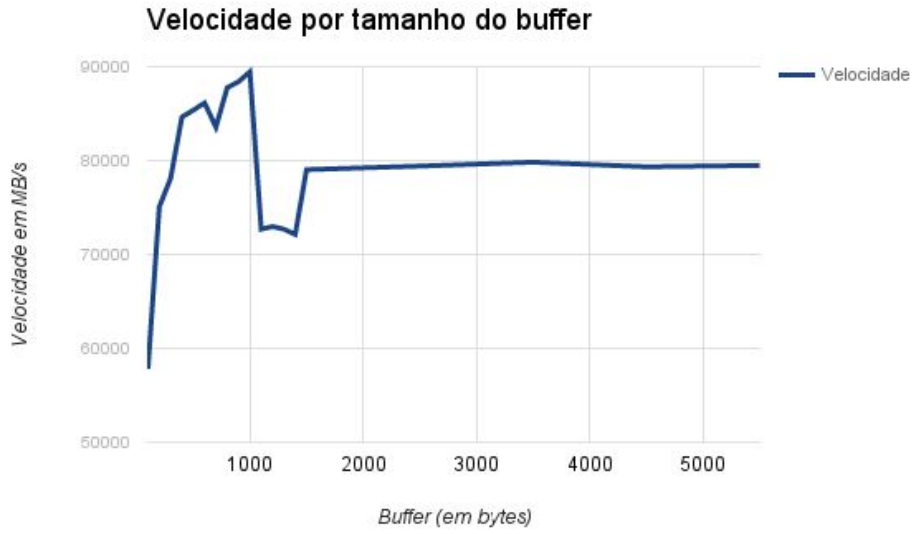

Gráfico 1: Velocidade por tamanho do buffer

Devido a alta velocidade oferecida e a falta de máquinas com ipv6, os testes apresentaram uma variação irregular com a variação do tamanho do buffer. A utilização de um servidor na Digital Ocean foi tentar suprir essa irregularidade. Nos testes realizados, mediu-se a latência e a velocidade numa taxa de 100 bytes acrescidos no buffer (tanto do cliente quanto do servidor).

6. Compilação e Execução

Para compilar o programa basta executar o comando "make" no dentro do diretório do programa. Antes compilação é necessário criar uma pasta "bin" caso ainda não exista. Após a compilação são gerados dois arquivos executáveis: **ServidorFTP** e **ClienteFTP**.

Para execução é necessário a listagem correta dos parâmetros durante. Caso não corresponda aos parâmetros corretos a uma mensagem de erro e as seguintes informações aparecerá no terminal.

Para listar os parâmetros fora de ordem é necessário utilizar a flag **"-o"** seguido das flags listadas na mensagem de ajuda. Para imprimir as mensagens de ajuda basta utilizar somente a flag **"-h"** .

No caso de erro na execução as mensagens de erro aparecerá no console junto com o tipo de erro ocasionado.

7. Conclusão

A variação do tamanho do buffer não influencia tanto na velocidade de transferência quanto a banda máxima e a latência da rede, mas buffers muito pequenos produzem trabalho a mais desnecessário para a máquina (os overheads de leitura/escrita do disco, encapsulamento, e envio desses dados para as camadas abaixo da TCP/IP, ocorrerão mais frequentemente quanto menor for o buffer escolhido pelo usuário).

O trabalho foi muito produtivo, a API de Sockets em C++ ficou bem completa, e os programas cliente e servidor totalmente operacionais. Através do código gerado nesse trabalho, montamos um servidor de arquivos em uma máquina em outro país, e os arquivos foram servidos para as máquinas clientes com sucesso. Aprender Sockets IPV6 também foi uma experiência interessante, foram requeridas várias horas de estudo e experimentação para a produção de todo o código, e, para a equipe, foi muito satisfatório concluir tudo e ver funcionando.

8. Referências

Donahoo, M. J. and Calvert, K. L. TCP/IP sockets in C: practical guide for programmers.

Morgan Kaufmann. 2009

W. Richard Stevens, Bill Fenner, and Andrew M. Rudoff. 2003. UNIX Network Programming, Vol. 1 (3 ed.). Pearson Education.

Douglas E. Comer. 1991. Internetworking with TCP/IP (2nd Ed.), Vol. I. Prentice-Hall, Inc., Upper Saddle River, NJ, USA.

Michael Kerrisk. 2010. The Linux Programming Interface: A Linux and UNIX System Programming Handbook (1st ed.). No Starch Press, San Francisco, CA, USA.

Bjarne Stroustrup. The C++ Programming Language. Addison-Wesley, 2013, 4 th Edition.

BARNEY, Blaise. POSIX Threads Programming. Lawrence Livermore National Laboratory. Disponível em: < https://computing.llnl.gov/tutorials/pthreads/ >. Acesso em: novembro 2016.

Linux Man Pages. The Linux Foundation. Disponível em: < https://www.kernel.org/doc/man-pages/ >. Acesso em: novembro 2016.