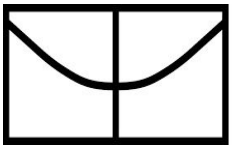

Universidade de Brasília

Aplicação distribuída usando sockets

Programação para Sistemas Paralelos e Distribuídos

Ícaro Pires de Souza Aragão - 15/0129815
Prof. Fernando W. Cruz

Gama, Março de 2018

Introdução

Foi proposto que fosse construída uma aplicação distribuída simples, usando sockets na linguagem C. Na primeira versão seria usado UDP como protocolo de comunicação e na segunda TCP. A aplicação deveria apenas realizar uma operação matemática simples num servidor e retornar o resultado ao cliente.

Solução

Foram escritos dois servidores e dois clientes, um para cada protocolo, seguindo a seguinte estrutura de pastas:

- ***sockets***: com arquivo de cabeçalho e código fonte, contendo funções para o envio e recebimento de mensagens e gerenciamento das conexões dos sockets;

- ***utils***: com funções que seriam úteis tanto a versão UDP, quanto a TCP da aplicação;

- ***src***: códigos fontes dos clientes e servidores;

- ***obj***: criada na primeira compilação, contém os arquivos objeto;

- ***bin***: criada na primeira compilação, contém os executáveis dos clientes e servidores.

Para facilitar o processo de compilação, foi adicionado também um script que gera cada um dos executáveis, esse script é o *make_all*.

Todos os executáveis são executados passando o IP e a porta como parâmetros.

Para a passagem de informação entre o cliente e o servidor foi criado a tipo *Package* a partir da *struct pdu*, que contém os operandos, o operador da operação aritmética e um espaço para o resultado. Então o cliente captura os parâmetros passados pelo usuário em tempo de execução, empacota a informação num *Package* o envia para o servidor. O servidor então recebe o pacote, calcula o que foi solicitado, preenche o campo resultado e devolve o pacote ao cliente. O cliente então recebe o pacote e exibe o resultado.

Foi dado a cliente o direito de pedir 5 operações matemáticas por conexão ao servidor. Se necessário mais, teria que se conectar novamente ao servidor.

Alguns elementos de boas práticas também foram adicionados, como o retorno de mensagens de erro personalizadas para o projeto junto a mensagens de erro do sistema para todas as funções que usam a variável *errno*. O código também foi todo modularizado e funções reaproveitadas entre os arquivos. Além disso, também há o fechamento da conexão por parte do servidor TCP em caso de sinais de interrupção ou de término enviados ao servidor e a interação com o usuário através de mensagens a cada ação realizada.

A solução foi feita para suportar várias conexões, mas poderia ser facilmente escalada a esse ponto, seria necessário apenas deixar o processo pai aceitando

conexões e utilizando da função *fork()* para que os filhos pudessem atender aos clientes em processos separados. Isso não chegou a ser implementado porque não seria possível estabilizar o funcionamento dentro do prazo de entrega.

Considerações

As soluções adotadas utilizando sockets foram bastante eficientes até certo ponto, mas para se ter um código consistente e com certa resistência a eventuais problemas, é necessário que se escreva muito código e consequentemente mais tempo é demandado.

O tempo para construção de uma aplicação simples é notável, mesmo usando-se das facilidades oferecidas pelas bibliotecas já implementadas e pelo serviço prestado pelos protocolos. Isso se deve ao fato de que mesmo com todos esses recursos, ainda existem muitas correções a serem feitas.

Utilizando dessa abordagem, muitas questões ficam pendentes de solução, como por exemplo: como será o formato da mensagem que será enviada? Quais tipos de variáveis? Quanto do código do servidor a pessoa que escreverá o cliente tem que conhecer? A arquitetura do computador que receberá o resultado foi prevista? Todas essas questões podem gerar problemas que deverão ter uma solução em correspondente em código.

Esses aspectos podem tornar essa abordagem inviável na maioria dos projetos um pouco mais complexos e então ferramentas mais elaboradas são necessárias.