Programação Distribuída: Cliente e Servidor usando *sockets*

Filipe Franciel Utzig
Engenharia de Computação
Pontifícia Universidade Católica do Rio Grande do Sul - PUCRS
Porto Alegre 90619-900
Email: filipeutzig *at* gmail.com

Abstract*—*The present work presents the implementation of sorting algorithm using distributed computing in a model client-server, this we'll be made with help of sockets to communicate all proccess, so that, at the end, we can have a better knowledge regarding the troubles of this kind of programming model.

Keywords*—*distributed computing, sort, sockets, select.

I. Descrição do problema

O problema consiste em implementar uma versão cliente-servidor [1] utilizando *sockets* para ordenar um vetor de tamanho *x*, o ordenamento pode ser realizando utilizando qualquer algoritmo de ordenação.

O cliente possui um conjunto de valores a ser ordenado. Este cliente sabe que existem X servidores de ordenação que recebem um conjunto de valores inteiros, ordena estes valores e devolve um conjunto de valores inteiros ordenados (ordem crescente). O cliente quebra seu vetor em diversas partes (M) e envia estas partes para os diversos servidores. O cliente sabe o endereço onde estes servidores estão executando.

O servidor recebe um conjunto de valores inteiros para ser ordenado, ordena estes valores e devolve os valores ordenados para quem havia enviado. O servidor pode atender a diversos clientes ao mesmo tempo.

A. Descrição do Cliente

O cliente inicia sua computação lendo os valores que devem ser ordenados em memória e analisando os argumentos passados pelo usuário, para após carregar a lista de servidores disponíveis e tentar estabelecer a conexão. Caso um servidor esteja indisponível ou apresente problemas na conexão, este não é adicionado a lista de servidores utilizados pelo cliente, forçando assim a execução com um número menor de servidores do que o requisitado pelo usuário. Caso nenhum servidor esteja disponível o cliente aborta a execução informando o usuário.

Após todas as conexões serem estabelecidas o cliente então divide o vetor de entrada em vetores menores que serão enviados para os servidores disponíveis ordenarem. A cada mensagem é enviado um valor inteiro de 4 bytes contendo o número de elementos que serão enviado na sequencia, e então é enviado o vetor correspondente ao servidor.

Ao enviar todos os vetores para os servidores o cliente então fica monitorando os *sockets* abertos utilizando multiplexação de I/O (*Input/Output*, Entrada/Saída) com um *select()* nos descritores [2]. Ao receber a resposta de um servidor o cliente então organiza o vetor recebido junto aos valores já recebidos previamente utilizando uma técnica de *merge* e volta a monitorar os *sockets* abertos. Ao final é gerado um arquivo 'sorted_vector.txt' contendo os valores ordenados de forma crescente.

A execução do cliente requer como entrada os parâmetros: o número de servidores disponíveis, o número de fatias de que o vetor será dividido, o arquivo que contém os valores para ser adicionado no vetor. Para configurar os servidores é necessário criar um arquivo chamado 'servers.txt' no diretório onde se está executando o programa cliente, este arquivo deve conter os endereços IPs dos servidores e a porta TCP onde os servidores estão rodando, como no exemplo abaixo:

```
::1 33333
172.16.32.12 33333
172.16.32.64 33333
172.16.32.64 42578
201.224.76.120 8080
```

B. Descrição do Servidor

O servidor tem como único parâmetro opcional a porta TCP onde quer que seja iniciado o servidor, caso nenhuma porta seja fornecida ele automáticamente utiliza a porta 33333. Após definida a porta a ser utilizada o programa cria um *s*ocket e fica aguardando conexão dos clientes, quando uma conexão é efetuada com sucesso o servidor então obtém um novo descritor que é armazenado na lista de descritores a serem multiplexados, permitindo que o servidor trate de diversas conexões sem abrir novos processos.

Ao receber um evento em algum descritor o servidor então para para descobrir qual o *socket* que gerou o evento e então realiza a leitura dos dados na rede. O primeiro valor recebido é um inteiro de 4 bytes, contendo a quantidade de elementos que devem ser recebidos na sequencia. A partir do momento em que o servidor sabe a quantidade de elementos que devem ser recebidos ele fica preso recebendo os dados pelo *socket* até que todos os elementos sejam recebidos.

Ao terminar de receber todos os dados o servidor então adiciona uma mensagem na fila de requisições que devem ser tratadas e verifica por outras requisições de outros clientes. Após verificar todas as requisições o servidor então começa a processar as mensagens e ordena os valores recebidos utilizando o método *qsort()* presente na *libc* [3] do sistema,

então, é gerada uma mensagem que é adicionada na fila de mensagens a serem enviadas e as outras mensagens recebidas são tratadas realizando o mesmo processo.

Após terminar de tratar todas as mensagens o servidor então começa a processar a fila de mensagens a serem enviadas e devolve os resultados para os clientes que requisitaram. Terminando de esvaziar a fila o servidor retorna ao estado de multiplexação de I/O esperando por novos eventos nos descritores monitorados.

II. Conclusão

O trabalho possibilitou a implementação de um modelo cliente-servidor utilizando o conceito de sockets, para verificação foi criado vários cenários como diversos servidores e um cliente, mais de um cliente acessando o mesmo servidor e em todos os casos o cliente envia mais de uma vez para um servidor, em todos os testes.

References

[1] Springer, *Modelling for Distributed Network Systems: The Client-Server Model*.

[2] IBM, "Using select() for i/o multiplexing," 2015. [Online]. Available: http://publib.boulder.ibm.com/html/as400/v4r5/ic2979/info/RZAB6IOMULTI.HTM

[3] L. P. Manual, "libc - overview of standard c libraries on linux," 2015. [Online]. Available: http://man7.org/linux/man-pages/man7/libc.7.html