Documentação TP1
Sockets, medição de desempenho

Luciano Otoni Milen - [2012079754]

1- Introdução

Neste trabalho prático deve-se implementar dois programas que estabelecem uma conexão entre si via TCP. O servidor cria uma porta disponível que será acessada pelo cliente, que requisita um arquivo a ser transferido pelo servidor, similar ao FTP. O acesso deve ser possível utilizando IPv4 ou IPv6, através das bibliotecas do Linux.

2- Metodologia

Os experimentos foram realizados da seguinte forma:

a) Foi criado um arquivo `teste` que contém um texto qualquer;
b) O servidor é iniciado em uma porta aleatória (9000, por exemplo) e um *tamanho_de_buffer*, como 4;
c) O cliente é iniciado com o endereço do *host*, como *localhost* por exemplo. Os parâmetros de chamada são *hostname*, *porta*, *nome_do_arquivo* (`teste`, no caso) e *tamanho_de _buffer* (4). A conexão então é estabelecida;
d) O servidor recebe o *nome_do_arquivo* e o abre no diretório em modo de leitura;
e) Cada bloco de tamanho *tamanho_de_buffer* é lido e escrito no socket criado entre cliente e servidor;
f) A conexão é encerrada quando a leitura do arquivo pelo servidor é finalizada, onde a cópia feita pelo cliente é salva;
g) Imprime os resultados do teste na tela.

O programa foi elaborado e testado em uma máquina Ubuntu GNOME Linux 17.04 Intel® Core ™ i7-3610QM CPU @ 2.30GHz × 8 com 8gb de memória RAM, em uma rede banda larga *wireless* de 15MB/s de download (2MB/s de upload). As medições foram feitas utilizando o `gettimeofday` para calcular o tempo decorrido na transferência. O teste principal foi executado 15 vezes e diversos testes menores foram feitos durante o desenvolvimento dos programas. O teste principal é composto por um arquivo sem formato específico contendo um texto *Lorem Ipsum* gerado aleatoriamente com 10 mil palavras.

O código está todo documentado, com o passo a passo do que é feito na execução. Para rodá-lo, basta compilar os arquivos `clienteFTP.c` e `servidorFTP.c` utilizando o GCC no Linux e executá-los na ordem servidor -> cliente, passando informações como *hostname*, porta, tamanho do *buffer*, arquivo da transferência. Após a transferência ser completada, um arquivo `result.md` é gerado.

3- Resultados

Seguem abaixo informações gráficas a respeito do desempenho dos programas. A tabela identifica alguns resultados obtidos nos testes realizados. Cada experimento foi executado 10 vezes e o valor identificado representa a média obtida.

Experimento	Tamanho Mensagem (B)	Tamanho *Buffer* (B)	Número de Mensagens	Tempo Decorrido (s)	*Throughtput* (kbps)
1	100	5	21	0.000675	245.5445
2	100	60	2	0.000657	158.2952
3	100	700	1	0.000542	191.8819
4	1500	5	301	0.001232	1262.175
5	1500	60	26	0.000840	2790.740
6	1500	700	3	0.000546	3378.923
7	30000	5	6010	0.009525	3160.000
8	30000	60	501	0.001218	24672.41
9	30000	700	43	0.000931	41109.43

4- Análise

Experimento 1: com uma mensagem pequena e *buffer* pequeno, o tempo fica bem próximo do obtido mesmo aumentando o tamanho do *buffer*.

Experimento 2: a diferença de 12 vezes mais *buffers* não gerou impacto significativo no tempo.

Experimento 3: como já observado, ainda que seja somente 1 pacote a ser transferido o tempo não se altera tanto. Provavelmente em decorrência do gasto ser maior para estabelecer a conexão do que transferir os arquivos.

Experimento 4: aqui se vê que o tempo aumentou consideravelmente, mas ainda foi pouco: 15 vezes mais Bytes impactou em somente aproximadamente o dobro do tempo de 100 Bytes.

Experimento 5: as taxas de transferência começam a ficar bem maiores. Demorou um pouco mais do que o experimento 1 para executar, logo o aumento no número de *buffers* é muito impactante.

Experimento 6: com uma mensagem 15 vezes maior que o experimento 3, o tempo foi praticamente o mesmo.

Experimento 7: o mais longo de todos os experimentos. A diferença no número de *buffers* se mostra crucial neste caso.

Experimento 8: muito mais rápido que o 7. Aparentemente o efeito do número maior de *buffers* se revela quando as mensagens são bem maiores.

Experimento 9: com um *buffer* tão grande uma mensagem pequena de 100 Bytes não apresentou resultados tão diferentes de uma mensagem de 30000 Bytes.

Os resultados foram bem interessantes. Não esperava que a relação tamanho do *buffer* e tamanho da mensagem fosse ter tal comportamento.
Quanto maior a mensagem sem alterar o tamanho do *buffer*, logicamente mais tempo demora para concluir a transferência. Entretanto, se o *buffer* é absurdamente grande fica difícil notar esta diferença. Os experimentos 6, 7 e 8 foram os mais diferenciadores de todos.

5- Conclusão

Com esse trabalho foi possível implementar um par de programas que operam no modelo cliente-servidor e exercitar a comunicação do tipo requisição-resposta sobre o protocolo TCP. Além disso, foi possível praticar a programação com a biblioteca de sockets do Unix para ter uma noção de como funciona o princípio da computação em nuvem. Os resultados obtidos com esse trabalho estava de acordo com o esperado, exceto pelo quanto o tamanho maior de *buffers* "ignora" o tamanho da mensagem no resultado do tempo.
A maior dificuldade foi o uso da linguagem C, que demanda muita atenção nos detalhes na hora de lidar com strings. Entretanto, o aprendizado foi grande: agora sabe-se como é feita uma comunicação simples cliente-servidor.

6- Referências

- Slides do professor Marcos Vieira (http://homepages.dcc.ufmg.br/~mmvieira/redes/sockets.pdf)
- http://lipsum.com (gerador de texto)
- https://www.tutorialspoint.com/c_standard_library/
- https://stackoverflow.com
- http://www.geeksforgeeks.org/socket-programming-cc/
- http://www.linuxhowtos.org/C_C++/socket.htm