

Redes de Computadores

Trabalho Prático 1

Carolina Coimbra Vieira

Setembro de 2017

1 Introdução

O presente trabalho objetiva apresentar e analisar a metodologia e os resultados da implementação de um par de programas que operam no modelo cliente-servidor e exercitam tanto a transmissão unidirecional quanto a comunicação do tipo requisição-resposta sobre o protocolo TCP. Para isso, faz-se necessário utilizar a biblioteca de sockets do Unix (Linux).

A tecnologia cliente-servidor é uma arquitetura na qual o processamento da informação é dividido em módulos ou processos distintos. Um processo é responsável pela manutenção da informação (servidores) e outros responsáveis pela obtenção dos dados (os clientes). Os processos cliente enviam pedidos para o processo servidor, e este por sua vez processa e envia os resultados dos pedidos [2]. Nesse caso, em particular, o cliente será responsável por enviar o nome do arquivo que o servidor deverá enviar para ele, dividindo em buffers de tamanho determinado a priori.

Os Sockets são uma abstração de endereços de comunicação que consistem de um nome de host e um número de porta. Os sockets surgiram no sistema operacional BSD Unix e, atualmente, são responsáveis pela interação e comunicação de programas ao longo da Internet. Além disso, os sockets são a interface padrão para a comunicação entre processos em redes TCP/IP [3], sobretudo por fornecer uma interface para a camada de aplicação se comunicar com a camada de transportes dessa arquitetura. Ou seja, os sockets UDP e TCP são a interface provida pelos respectivos protocolos na interface da camada de transporte [1] e, programar com sockets pode ser visto como desenvolver um protocolo de aplicação.

A comunicação entre cliente-servidor se dá sobre o protocolo TCP (Transmission Control Protocol - Protocolo de Controle de Transmissão). Esse protocolo possui características muito importantes, pois é um protocolo fim-a-fim, orientado a conexão e fornece uma transferência confiável de dados entre aplicações parceiras. Ou seja, durante uma comunicação através do protocolo TCP, as duas máquinas devem estabelecer uma conexão. Além disso, as mensagens trocadas são verificadas de forma a garantir seu recebimento.

Dessa forma, a implementação do programa deve levar em consideração todas as especificações acima. Primeiro dois programas principais são criados, o cliente e o servidor, de forma que antes da troca de dados, há o estabelecimento da conexão via funções da biblioteca sockets. Após isso, ocorre a troca de dados de forma confirmada. E, por fim, várias análises e testes são realizados para avaliar o desempenho dessa comunicação variando, principalmente, o tamanho do buffer que será utilizado para trocar dados.

O trabalho se organiza da seguinte forma. Primeiro, na Seção 2 é apresentada a metodologia contendo dados sobre os experimentos como a configuração da máquina utilizada, a localização da mesma na rede além da descrição de como foram realizadas as medições e número de execuções. Então, na Seção 3 são apresentados os resultados obtidos após a realização de diversos experimentos. A análise mais aprofundada sobre cada um dos experimentos é descrita na Seção 4. Finalmente, na Seção 5, é feita uma breve conclusão sobre o trabalho.

2 Metodologia

A metodologia apresentada se refere à implementação e testes de um par de programas que operam no modelo cliente-servidor com comunicação via protocolo TCP, utilizando uma biblioteca de sockets do Unix. Inicialmente, ao se pensar em comunicação, é importante ressaltar que para que dois processos comuniquem entre si, é necessário estabelecer uma conexão. Sendo assim, os programas cliente e servidor são implementados seguindo um padrão simples, conforme descrito a seguir:

Cliente:

```
processa argumentos da linha de comando:
host_do_servidor porto_servidor nome_arquivo tam_buffer
chama gettimeofday para tempo inicial
faz abertura ativa a host_do_servidor : porto_servidor
via string com nome do arquivo (terminada em zero)
abre arquivo que vai ser gravado - pode ser fopen(nome,"w+")
loop recv buffer até que receba zero bytes ou valor negativo
escreve bytes do buffer no arquivo (fwrite)
atualiza contagem de bytes recebidos
fim loop
fecha conexão e arquivo
chama gettimeofday para tempo final e calcula tempo gasto
imprime resultado:
"Buffer = %5u byte(s), %10.2f kbps (%u bytes em %3u.%06u s)
fim cliente.
```

Servidor:

processa argumentos da linha de comando:
porto_servidor tam_buffer
faz abertura passiva e aguarda conexão
recebe, byte a byte até o zero, o string com nome do arquivo
abre arquivo que vai ser lido – pode ser fopen(nome,"r")
se deu erro, fecha conexão e termina
loop lê o arquivo, um buffer por vez até fread retornar zero
envia o buffer lido
se quiser, contabiliza bytes enviados
fim loop
fecha conexão e arquivo
chama gettimeofday para tempo final e calcula tempo gasto
se quiser, imprime nome arquivo e no. de bytes enviados
fim servidor.

De forma resumida, o cliente deve se conectar ao servidor, enviar um string com o nome do arquivo desejado, receber o arquivo um buffer de cada vez e salvar os dados no disco à medida que eles chegam. Quando não houver mais bytes para ler o cliente fecha a conexão e o arquivo e gera uma linha com os dados da execução. O servidor por sua vez deve operar de forma complementar, conforme exemplificado na Figura 1.

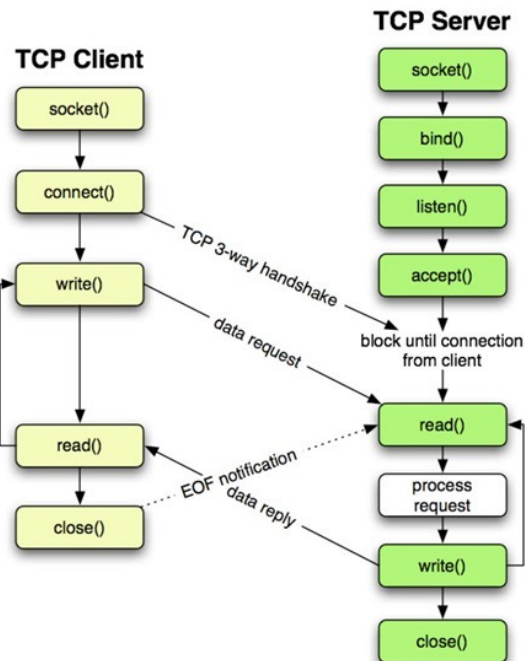


Figura 1: Diagrama cliente-servidor TCP

A conexão que deve ser estabelecida por esses processos está representada, de forma geral, na Figura 1. Nela, é possível perceber que, as estruturas são criadas e, em seguida ocorre a abertura passiva da conexão, ou seja, um processo está pronto para receber conexões apesar de não necessariamente haver a identificação do outro. Em seguida, ocorre, de fato, a abertura ativa da conexão, através da qual um dos processos já sabe como alcançar o interlocutor e o mesmo está preparado para recebê-lo. Após isso, ocorre a comunicação propriamente dita até que a conexão seja encerrada de forma explícita ou não.

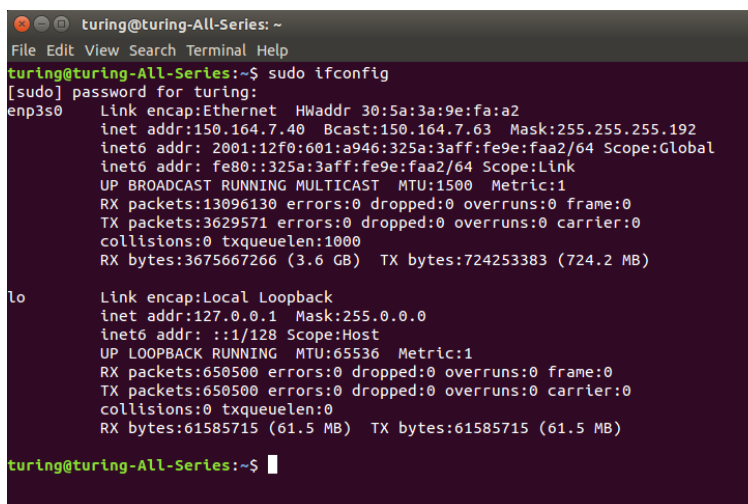
Todos os testes a serem descritos foram executados utilizando a mesma máquina, cujas informações principais referentes ao poder de processamento, memória e placa de rede são descritas a seguir:

Mémoire RAM: 16GB

Processador: Intel(R) Core(TM) i5-4440 CPU @ 3.10GHz

Placa de rede: 03:00:0 Ethernet controller: Realtek Semiconductor Co., Ltd. RTL8111/8168/8411 PCI Express Gigabit Ethernet Controller (rev 0c)

E, por fim, a máquina está localizada no laboratório *WISEMAP* do Departamento de Ciência da Computação na UFMG e possui o seguinte endereço IP: 150.164.7.40. Na Figura 2 é possível visualizar algumas configurações de rede da máquina.

A terminal window titled 'turing@turing-All-Series: ~' with a menu bar (File, Edit, View, Search, Terminal, Help). The user has run 'sudo ifconfig' and entered the password. The output shows details for the 'enp3s0' (Ethernet) and 'lo' (Loopback) interfaces. For 'enp3s0', it shows link encap: Ethernet, HWaddr 30:5a:3a:9e:fa:a2, inet addr: 150.164.7.40, Bcast: 150.164.7.63, Mask: 255.255.255.192, and various statistics. For 'lo', it shows link encap: Local Loopback, inet addr: 127.0.0.1, Mask: 255.0.0.0, and statistics.

```
turing@turing-All-Series:~$ sudo ifconfig
[sudo] password for turing:
enp3s0    Link encap:Ethernet  HWaddr 30:5a:3a:9e:fa:a2
          inet addr:150.164.7.40  Bcast:150.164.7.63  Mask:255.255.255.192
          inet6 addr: 2001:12f0:601:a946:325a:3aff:fe9e:faa2/64  Scope:Global
          inet6 addr: fe80::325a:3aff:fe9e:faa2/64  Scope:Link
          UP BROADCAST RUNNING MULTICAST  MTU:1500  Metric:1
          RX packets:13096130 errors:0 dropped:0 overruns:0 frame:0
          TX packets:3629571 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:1000
          RX bytes:3675667266 (3.6 GB)  TX bytes:724253383 (724.2 MB)

lo        Link encap:Local Loopback
          inet addr:127.0.0.1  Mask:255.0.0.0
          inet6 addr: ::1/128  Scope:Host
          UP LOOPBACK RUNNING  MTU:65536  Metric:1
          RX packets:650500 errors:0 dropped:0 overruns:0 frame:0
          TX packets:650500 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:0
          RX bytes:61585715 (61.5 MB)  TX bytes:61585715 (61.5 MB)

turing@turing-All-Series:~$
```

Figura 2: Especificações da rede do computador utilizado nos testes

O servidor irá rodar na máquina, ou seja, ele usa o IP da máquina e, é necessário definir a porta através da qual o cliente e o servidor irão se comunicar. Já para o cliente, é necessário especificar o IP onde o servidor está rodando e a porta correspondente. Dado essas especificações, a comunicação pode ser exemplificada conforme mostra a figura 3.

O servidor deve ser o primeiro a ser inicializado e, a partir de então, ele estará sempre "escutando" a porta. Quando o cliente se conecta, ocorre o esta-

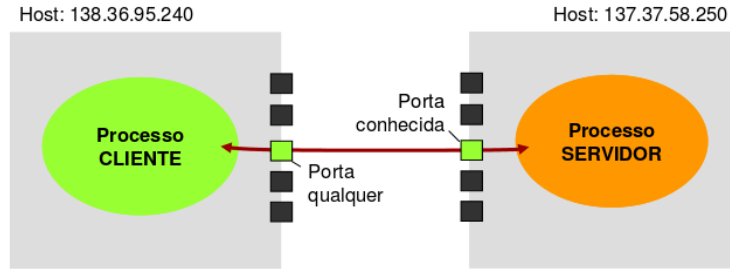


Figura 3: Diagrama representando como se dá a comunicação entre o cliente e servidor dado as especificações do IP e a porta de comunicação com o servidor

belecimento da conexão e, a partir desse momento, o cliente pode iniciar sua comunicação com o servidor. Os parâmetros que o cliente recebe são: host do servidor (`argv[1]`), número da porta (`argv[2]`), nome do arquivo (`argv[3]`) e o tamanho do buffer (`argv[4]`) enquanto que o servidor recebe o número da porta (`argv[1]`) e o tamanho do buffer (`argv[2]`).

Os testes sobre essa comunicação cliente-servidor, foram executados medindo o tempo decorrido desde o estabelecimento da conexão até a finalização da mesma, após o servidor enviar todo o arquivo solicitado pelo cliente. Sendo assim, o mesmo código foi executado repetidas vezes variando alguns parâmetros como o tamanho do arquivo e tamanho do buffer. Para cada uma dessas variações, foi calculada a média e desvio padrão conforme será melhor apresentado na Seção 3.

3 Resultados

Os principais experimentos executados após a implementação do par cliente-servidor envolveram testes de desempenho medindo o tempo que leva a comunicação e completa transferência do arquivo. As hipóteses levantadas sugerem que esse desempenho é afetado principalmente com a variação do tamanho do arquivo que se deseja transferir e o tamanho do buffer através do qual o mesmo será transferido.

No que diz respeito à relação entre o tamanho do buffer e o tempo de comunicação entre cliente-servidor, foram realizados testes variando o tamanho do buffer de 2^1 até 2^{16} bytes. Nesse teste, o tamanho do arquivo considerado era de 3MB e para cada tamanho de buffer, 10 execuções foram feitas. A Figura 4 representa o gráfico obtido após fazer a média e o desvio padrão¹ dos tempos encontrados para cada tamanho de buffer usado. O eixo y representa o tempo médio da troca de dados, medido em microsegundos, e o eixo x representa o tamanho do buffer utilizado. Note que a escala utilizada no eixo x está em log a fim de facilitar a visualização.

¹Intervalo de confiança de 95%

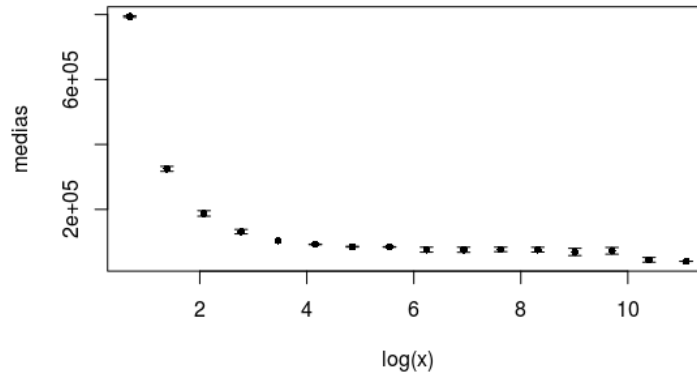


Figura 4: Gráfico representando o tempo médio de comunicação dado o tamanho do buffer usado

Similarmente ao gráfico representado na Figura 4, a Figura 5 apresenta um gráfico com o *throughput* médio da comunicação, utilizando os dados do teste anterior. Porém, o eixo y representa agora a razão entre o número total de bytes enviados pelo tempo medido no cliente. Nesse gráfico, é possível visualizar o crescimento da taxa de envio com o aumento do tamanho do buffer. Note que a escala utilizada no eixo x está em log a fim de facilitar a visualização.

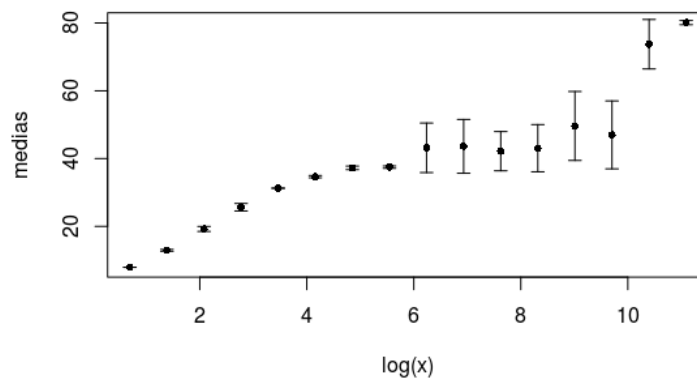


Figura 5: Gráfico representando o *throughput* dado o tamanho do buffer usado

Outro fator que também influencia no tempo de comunicação é o tamanho do arquivo que se deseja enviar. Nesse caso, fixado um tamanho de buffer igual a 2, escolhido de forma arbitrária, foram realizados testes para verificar a influência dos arquivos de tamanho 3MB, 30MB e 300MB no tempo de comunicação. A Figura 6 mostra o gráfico resultante desses testes realizados. Nela é possível se ter uma noção do quanto o tamanho do arquivo contribui para o aumento do tempo de troca de dados entre o cliente e o servidor. O eixo y representa o tempo médio da troca de dados, medido em microsegundos, e o eixo x representa o tamanho do arquivo enviado, dado um buffer constante igual a 2 bytes. Note que a escala utilizada no eixo x está em log a fim de facilitar a visualização.

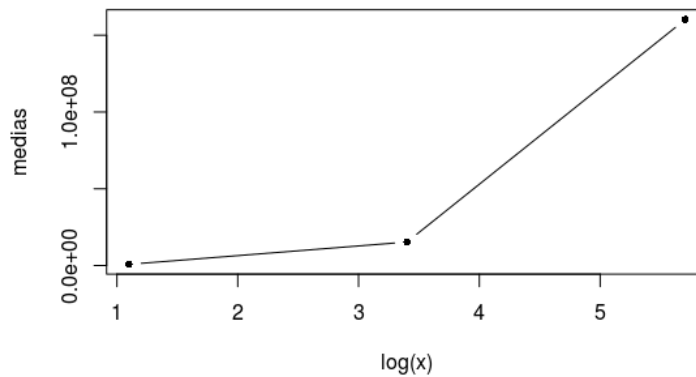


Figura 6: Gráfico representando o tempo médio de comunicação dado o tamanho do arquivo, usando um buffer de tamanho fixo igual a 2 bytes

E, por fim, é possível comparar a influência combinada do tamanho do buffer e do tamanho do arquivo no tempo de comunicação, conforme mostra a tabela na Figura 7. Nesse caso, cada uma das linhas da tabela representam um tamanho de buffer diferente (2^1 , 2^4 , 2^7 , 2^{10} , 2^{13} e 2^{16} bytes) e cada coluna representa o tamanho do arquivo (3MB, 6MB, 9MB, 12MB e 15MB). É possível perceber que, de fato, o aumento no tamanho do arquivo bem como a diminuição no tamanho do buffer influenciam muito no aumento do tempo de comunicação.

Além da tabela apresentada, é possível visualizar os resultados dos testes através da Figura 8. Cada uma das barras coloridas representam os respectivos tamanhos de arquivos testados (3MB, 6MB 9MB, 12MB e 15MB). O eixo y representa o tempo médio da troca de dados, medido em microsegundos, e o eixo x representa o tamanho do buffer. Note que a escala utilizada no eixo x está em log a fim de facilitar a visualização.

Buffer (bytes)	log(buffer)	3MB	6MB	9MB	12MB	15MB
2	0,3010299957	1.437.223,80	2.767.902,60	4.413.244,70	6.055.336,30	9.043.251,20
16	1,204119983	186.129,50	277.861,30	382.681,10	467.641,70	555.814,10
128	2,10720997	108.234,20	121.012,60	133.777,90	143.709,90	152.656,10
1024	3,010299957	93.196,60	74.246,60	67.930,30	114.738,40	89.507,00
8192	3,913389944	73.411,30	62.578,70	65.025,30	70.637,30	82.421,80
65536	4,816479931	52.432,80	57.679,80	61.299,70	116.428,10	87.715,60

Figura 7: Tabela representando o tempo médio de comunicação dado o tamanho do arquivo, usando buffers de tamanho 2^1 , 2^4 , 2^7 , 2^{10} , 2^{13} e 2^{16}

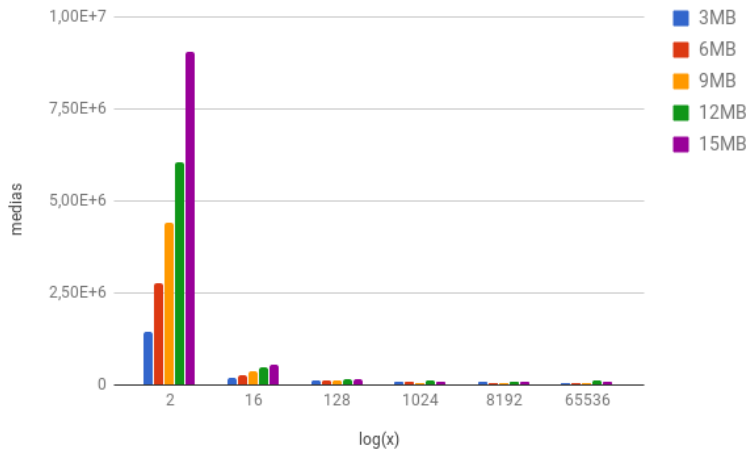


Figura 8: Gráfico representando o tempo médio de comunicação dado o tamanho do arquivo, usando buffers de tamanho 2^1 , 2^4 , 2^7 , 2^{10} , 2^{13} e 2^{16} . Deve ser visualizado em cores.

4 Análise

Os resultados encontrados na Seção 3 refletem bem o que era esperado sobre essa comunicação. O fato do tempo médio de comunicação aumentar com a diminuição do tamanho do buffer mostra que, de fato, foi necessário fazer mais envios ao cliente, aumentando o tempo de comunicação. Esse resultado é sustentado também pelos testes em que mostram o *throughput* da comunicação, isto é, a taxa de transferência obtida entre cliente e servidor (basicamente, o número total de bytes enviados dividido pelo tempo medido no cliente) com a variação do tamanho do buffer.

Assim como o tamanho do buffer, o tamanho do arquivo que se deseja enviar ao cliente influencia no tempo de comunicação. Entretanto, essa relação é direta, uma vez que o aumento no tamanho do arquivo provoca um aumento no tempo que leva a fase de troca de dados. Esse resultado também foi comprovado pelos testes realizados.

5 Conclusão

O presente trabalho foi desenvolvido com o objetivo de implementar um par de cliente-servidor que se comunicam via sockets TCP. Todo o código foi desenvolvido em linguagem C, usando a biblioteca de sockets.

Os testes realizados comprovam hipóteses formuladas anteriormente, referentes à influência do tamanho do buffer e o tamanho do arquivo nessa comunicação. Nesse caso, conforme esperado, o aumento no tamanho do buffer reduz significativamente o tempo que demora a comunicação. Assim como o aumento no tamanho do arquivo está diretamente relacionado ao aumento no tempo necessário para a completa transferência do mesmo.

Conforme mencionado várias vezes pelo professor, o presente trabalho trata-se de uma barreira transponível, uma vez que está relacionado a sockets, tema pouco trabalhado na sala de aula. Entretanto, foi uma oportunidade de aprender e conviver com problemas práticos que são vividos no dia-a-dia e enfrentados no mercado de trabalho e que nos ensinam a ser autodidatas e transpor barreiras por necessidade e desejo próprio. E, de fato, após muita pesquisa e aprendizado, é possível desenvolver um trabalho bem estruturado e com resultados interessantes e consistentes.

Referências

- [1] Cesar Augusto Tacla. Sockets udp, tcp e multicast. Slides de aula.
- [2] UFRGS. Protocolos cliente/servidor. Página Web.
- [3] Marcos Augusto M. Vieira. Sockets. Slides de aula.