Projeto 1 - Cliente e Servidor - TCP

Gabriel Affonso - 167710 e Luiz Eduardo Cartolano - 183012
MC833 - Turma A

Abril de 2019

Resumo

Análise de tempo de comunicação da camada de transporte em protocolo TCP no contexto de um sistema de banco de dados de perfis pessoais com objetivo de comparar eficiência e confiabilidade com outros protocolos. O TCP mostrou-se bastante confiável, pelo menos em testes na rede interna do Instituto de Computação Unicamp. Testes externos foram impossíveis devido a falta de acesso aos roteadores da equipe para realizar *port-forwarding*. O tempo de transporte foi menor do que o esperado provavelmente pela proximidade *cliente-servidor*.

1 Introdução

O modelo de arquitetura de computadores *cliente-servidor* é um dos mais populares no mundo. Usado em praticamente todos os sistemas automatizados de bibliotecas, ele separa a aplicação em duas partes distintas. O *cliente* - responsável por fazer requisições - e o *servidor* - responsável por prover as informações requisitadas. Uma exemplificação do modelo de funcionamento pode ser vista na Figura 1.

A comunicação entre ambas as partes é regida por protocolos padrões que são mundialmente conhecidos, como *UDP* e *TCP*. Os protocolos de transporte fornecem uma interface para as camadas superiores de abstração trocarem dados sem se preocupar com as complexidades inerentes ao transporte físico dos dados, que é *unreliable*. Assim, é garantido que as aplicações seguirão padrões mínimos de qualidade.

2 Sistema

2.1 Descrição Geral

O modelo de arquitetura *cliente-servidor* descreve como um servidor provê recursos e serviços para um ou mais clientes. Quando um cliente realiza uma requisição de conexão com o servidor, esta pode ou não ser aceita. Se aceita, o servidor irá estabelecer e manter uma conexão com o cliente sob a tutela de um protocolo específico. No caso deste trabalho, o servidor irá usar o protocolo *TCP*.

O *TCP* é um protocolo de transporte orientado à conexão em que o cliente estabelece uma conexão com o servidor e ambos trocam múltiplas mensagens de tamanhos variados, sendo a aplicação do cliente quem termina a sessão. Algumas características importantes desse tipo de protocolo são: confiabilidade, entrega ordenada e controle de fluxo.

2.2 Casos de Uso

Muitas aplicações do dia a dia fazem uso da arquitetura *cliente-servidor*, como a troca de e-mails, acesso à internet ou acesso a um banco de dados. O modelo cliente-servidor, tornou-se uma das ideias centrais de computação de rede. A maioria dos servidores possui uma relação *um para muitos(1:n)* com o cliente, ou seja, um único servidor é capaz de prover recursos para muitos clientes ao mesmo tempo.

Aplicações que mais beneficiam-se do protocolo TCP são aquelas orientadas a conexão e que precisam de confiabilidade. Por exemplo, quase todos os programas de transmissão de arquivos de dados utilizam o TCP pois devem garantir a integridade da chegada e recebimento dos dados em ordem. Outros exemplos clássicos são os programas *SSH (Secure Shell)* e o *TELNET (Telecomunications Network)*.

3 Implementação

Para realizar a implementação da aplicação usou-se [3] como modelo prático, uma vez que o livro apresenta as implementações de *hand-shaking* desejadas de comunicação por TCP já incluindo tratamento de erros na linguagem C. Já os conceitos teóricos que auxiliaram, sobretudo, no entendimento do funcionamento da aplicação podem ser vistos com mais detalhes em [4].

3.1 Cliente

O *cliente*, possui uma implementação mais simples que o *servidor*, visto que, sua única função é fazer uma requisição e aguardar um retorno. Uma vez que trabalhou-se com protocolo *TCP*, em um primeiro momento o *cliente* avisa ao *servidor* que irá abrir uma conexão, o que chamamos de *hand-shaking*. Após estabelecida essa conexão inicial, os *sockets* estarão se comunicando e ambos podem trocar informações.

No cliente, criou-se uma estrutura na qual o usuário pode escolher qual ação ele deseja executar e então informar os parâmetros necessários para que a execução seja bem sucedida. Então, usando a função $ssize\_tsend(intsocket, constvoid*buffer, size_tlength, intflags)$ da biblioteca *sys/socket.h*, o *cliente* envia ao *servidor* a ação que deseja ser executada. Para isso é preciso informar o socket que está sendo usado, a mensagem a ser enviada e o tamanho da mensagem. Uma nuância, porém, é que não utilizamos diretamente a função *send()*, pois ela não garante o envio de todos os bytes: chamamos a função *send_all()*, que nada mais é que uma repetição de *send()* até todos os bytes serem enviados.

Após requisitar a ação do *servidor*, basta ao *cliente* esperar o processamento dela acontecer, e então receber a resposta. Para receber a mensagem usa-se também uma função da biblioteca *sys/socket.h*, chamada $ssize_trecv(intsocket, void*buffer, size_tlength, intflags)$, cujos parâmetros são o socket utilizado na conexão, um *buffer* na memória no qual será salva a mensagem recebida e o tamanho dele.

É importante ressaltar que após realizada a comunicação, é o *cliente* o responsável por encerrar a conexão entre ambas as partes.

3.2 Servidor

A implementação do *servidor* é mais complexa e exige um pouco mais de atenção do que a do *cliente*. A explicação de tal complexidade dá-se pois, antes de tratar a requisição do cliente, primeiro é preciso criar um mecanismo que o permita atender a vários *clientes* ao mesmo tempo, ou seja, liberando o *socket* de *hand-shaking*. Além disso, ele é o responsável por administrar as informações salvas dos perfis e manusear operações com elas.

Para resolver o primeiro problema e criar um servidor concorrente, usou-se a função $pid_tfork(void)$ da biblioteca *sys/types.h*, ela é responsável por criar um processo filho, semelhante ao processo pai. Desse modo, o processo pai fica responsável por apenas aceitar as conexões que chegam ao *servidor*, e então direcioná-las para um novo processo que irá de fato lidar com ela.

Para lidar com o armazenamento dos perfis de usuários optou-se por usar um arquivo texto no qual eram armazenadas as informações. Este, por sua vez, era atualizado ao fim de todo o processo filho, garantindo que informações alteradas neles ficassem acessíveis aos demais. Uma limitação da abordagem escolhida acontece caso dois clientes diferentes requisitem modificações para um mesmo perfil, visto que a solução não lida com esse tipo de concorrência. Em uma implementação mais robusta, poderia ser utilizado um *Sistema Gerenciador de Banco de Dados (SGBD)* que garantisse operações *ACID (Atômicas, Consistentes, Isoladas e Duráveis)*.

Assim como acontecia na implementação do *cliente*, vide secção 3.1, um novo socket era estabelecido para cada nova conexão, e as funções *send()* e *recv()* foram usadas para enviar e receber arquivos entre ambos os lados da aplicação.

4 Experimento

4.1 Metodologia

Foram executadas 90 iterações de um script em Python que realizava uma de cada requisição possível do sistema de perfis:

1. Listar todas as pessoas formadas em um determinado curso
2. Listar as habilidades dos perfis que moram em uma determinada cidade
3. Acrescentar uma nova experiência em um perfil
4. Dado o email do perfil,retornar sua experiência
5. Listar todas asinformações de todos os perfis
6. Dado o email de um perfil, retornar suas informações.

Medimos o tempo de execução do programa do cliente como o tempo a partir do primeiro byte enviado ao servidor até o último byte recebido. Medimos o tempo de processamento do servidor como sendo o tempo do primeiro byte recebido até o último enviado. Assim, o tempo de comunicação será o tempo do cliente subtraído do tempo do servidor.

Para fins de comparação, executamos o mesmo *script* também em comunicação com *localhost*.

4.2 Resultados

Os resultados de média, desvio padrão e intervalos de confiança do tempo de execução do cliente podem ser vistos nas Tabelas 1 e 2. Já os valores para os tempos de comunicação se encontram nas Tabelas 3 e 4. A fim de obter os resultados apresentados, usando conceitos apresentados em [2], a partir do qual foi possível obter a fórmula para o cálculo do desvio padrão, que pode ser vista na Equação 1 e também para o cálculo do intervalo de confiança, que para um valor de 95%, é dada pela Equação 2.

Já os histogramas obtidos para as execuções junto ao *localhost* podem ser vistos na Figura 2, que mostra os tempos de execução do cliente, e na Figura 5, que mostra o tempo de comunicação. Enquanto que os histogramas para execução em duas máquinas diferentes se encontram na Figura 3, que mostra o tempo de execução do cliente, e na Figura 4, que mostra o tempo de comunicação.

5 Discussão

A implementação do sistema nos fez lidar com camadas de "baixo nível"de abstração no *stack de rede*. Tivemos de tratar buffers, sockets e particularidades de confiabilidade do protocolo, algo que normalmente fica escondido para camadas de aplicação. Isto fez nosso código tomar um tamanho considerável mesmo para um projeto simples, como podemos observar na Tabela 5.

Encontramos um tempo de conexão bastante pequeno nos quatro testes. A explicação mais plausível do motivo é porque estamos em máquinas adjacentes no Instituto de Computação, diminuindo a latência e perdas de pacote e garantindo excelente confiabilidade. A variância das amostras foi grande no teste de duas máquinas, acreditamos que pelo mesmo motivo, pois a maior componente do atraso não seria devido a aspectos estruturais da rede ou protocolo, mas sim flutuações aleatórias. Em comparação com *localhost*, o tempo de comunicação das duas máquinas foi significativamente maior, a média do segundo variava entre o dobro e o quadruplo da média do primeiro.

Referências

[1] MADEIRA Edmundo. Projeto 1. Disponível em: http://www.ic.unicamp.br/~edmundo/MC833/turma1s2019/proj18331s19.pdf, Acesso em: 20-03-2019.

[2] Charles Grinstead and J Laurie Snell. Introduction to probability / charles m. grinstead, j. laurie snell. *SERBIULA (sistema Librum 2.0)*, 04 2019.

[3] Brian Hall. Beej's guide to network programming. using internet sockets. published online by author, 2007.

[4] James F. Kurose and Keith W. Ross. *Computer Networking: A Top-Down Approach (6th Edition)*. Pearson, 6th edition, 2012.

ANEXOS

$$\sigma = \sqrt{\frac{\sum_{i=1}^{n}(x_i - \overline{x})^2}{n-1}} \tag{1}$$

(1) Fórmula para o cálculo do desvio padrão de um espaço amostral.

$$(\overline{x} - 1.96 \cdot \frac{\sigma}{\sqrt{n}}, \overline{x} + 1.96 \cdot \frac{\sigma}{\sqrt{n}}) \tag{2}$$

(2) Fórmula para o cálculo do intervalo de confiança de 95%.

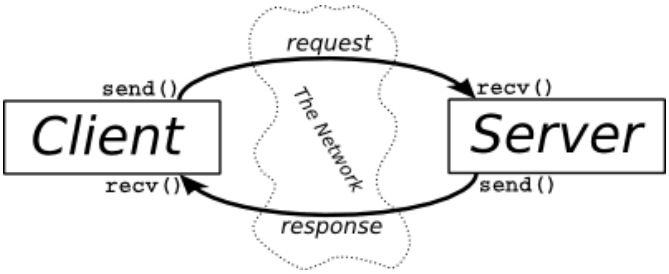
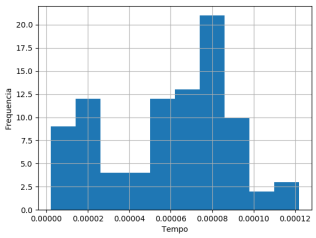
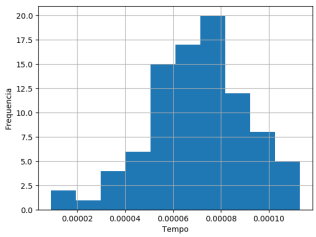


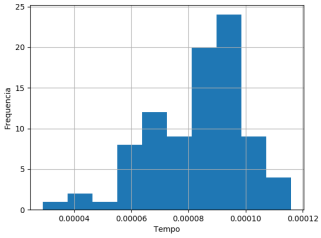
Figura (1) Diagrama cliente servidor.



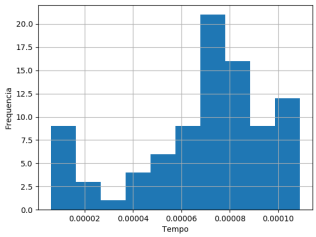
(a) Tempo quando executada a opção 1



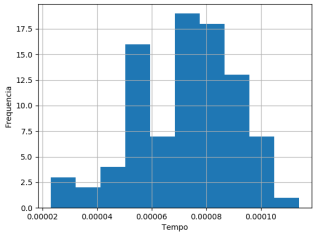
(b) Tempo quando executada a opção 2



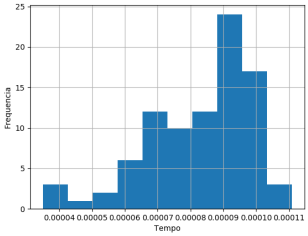
(c) Tempo quando executada a opção 3



(d) Tempo quando executada a opção 4

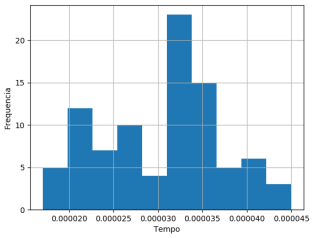


(e) Tempo quando executada a opção 5

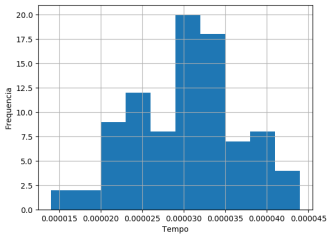


(f) Tempo quando executada a opção 6

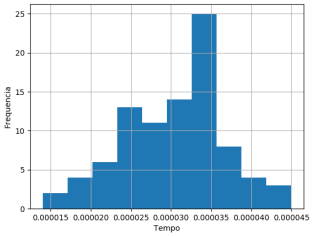
Figura (3) Tempo total na execução com dois computadores diferentes.



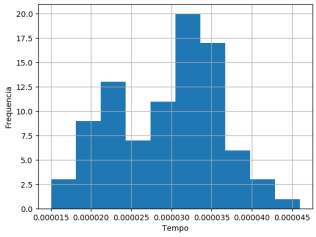
(a) Tempo quando executada a opção 1



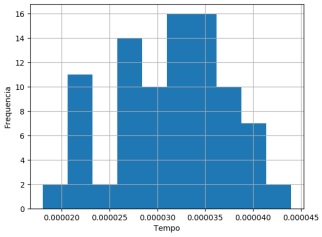
(b) Tempo quando executada a opção 2



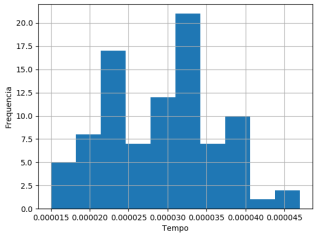
(c) Tempo quando executada a opção 3



(d) Tempo quando executada a opção 4

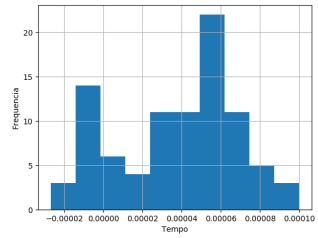


(e) Tempo quando executada a opção 5

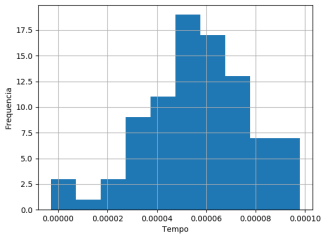


(f) Tempo quando executada a opção 6

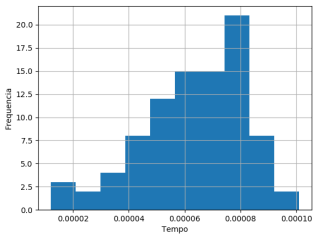
Figura (2) Tempo total na execução com o localhost.



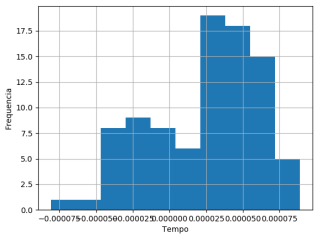
(a) Tempo quando executada a opção 1



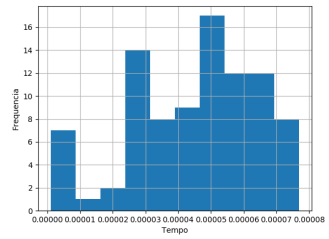
(b) Tempo quando executada a opção 2



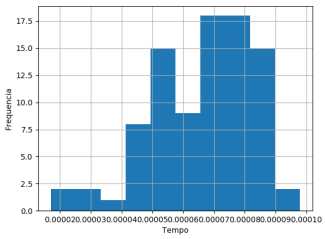
(c) Tempo quando executada a opção 3



(d) Tempo quando executada a opção 4



(e) Tempo quando executada a opção 5



(f) Tempo quando executada a opção 6

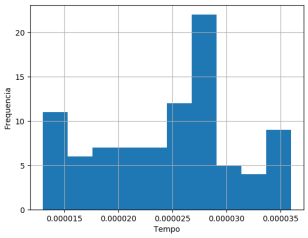
Figura (4) Tempo de comunicação na execução com dois computadores diferentes.

Tabela (1) Tempos de cliente para experimento em localhost

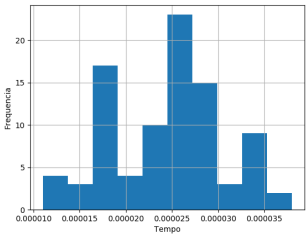
Opção de Exec.	Média(μ s)	Desvio(μ s)	Confiança .95(μ s)
1	30.3	6.78	28.91 - 31.71
2	29.9	6.33	28.57 - 31.18
3	30.4	6.13	29.10 - 31.63
4	29.8	6.27	28.60 - 31.05
5	31.3	5.79	30.11 - 32.50
6	29.2	7.07	27.72 - 30.65

Tabela (2) Tempo de cliente para experimento em dois computadores

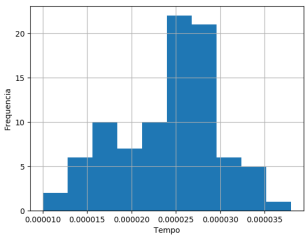
Opção de Exec.	Média(μ s)	Desvio(μ s)	Confiança .95(μ s)
1	58.8	30.2	52.56 - 65.03
2	70.9	20.7	66.58 - 75.12
3	82.8	16.4	79.42 - 86.21
4	68.3	27.3	62.69 - 73.99
5	72.7	18.2	68.94 - 76.48
6	82.9	16.0	79.61 - 86.24



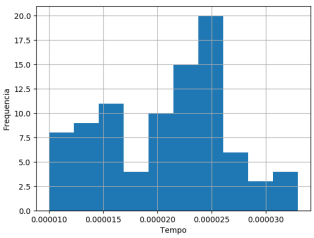
(a) Tempo quando executada a opção 1



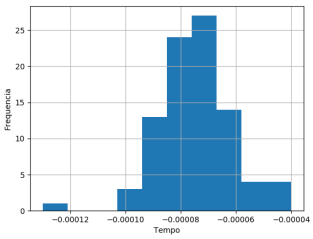
(b) Tempo quando executada a opção 2



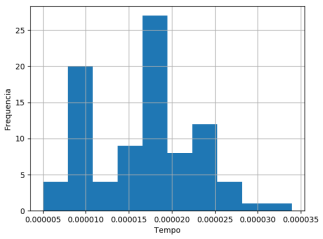
(c) Tempo quando executada a opção 3



(d) Tempo quando executada a opção 4



(e) Tempo quando executada a opção 5



(f) Tempo quando executada a opção 6

Figura (5) Tempo de comunicação na execução com o localhost.

Tabela (3) Tempo de comunicação para experimento em localhost

Opção de Exec.	Média(μ s)	Desvio(μ s)	Confiança .95(μ s)
1	24.5	6.31	23.24 - 25.84
2	24.6	6.03	23.35 - 25.84
3	24.3	5.76	23.06 - 25.44
4	20.8	5.70	19.66 - 22.02
5	-74.8	1.35	-77.59 - -72.00
6	16.7	6.38	15.42 - 18.06

Tabela (4) Tempos de comunicação para experimento em dois computadores

Opção de Exec.	Média(μ s)	Desvio(μ s)	Confiança .95(μ s)
1	37.8	30.8	31.42 - 44.15
2	56.3	21.1	51.90 - 60.62
3	63.8	19.1	59.84 - 67.75
4	23.8	37.1	16.08 - 31.42
5	45.0	19.2	41.02 - 48.97
6	66.1	16.4	62.72 - 69.51

Tabela (5) Tamanho dos códigos usados

Arquivo	Tamanho de Linhas
cliente.c	158
server.c	164
funcoes.c	343
funcoes.h	100
script de testes	126
TOTAL	891