Servidor Concorrente TCP

Leandro Kümmel Tria Mendes RA033910
Fernando Teixeira Barros RA085858

18 de abril de 2013

Sumário

Lista de Figuras

Lista de Tabelas

1 Introdução

O objetivo desse projeto é implementar um sistema cliente/servidor concorrente, com operações para o gerenciamento de livros em uma livraria. Na comunicação entre cliente e servidor utilizou-se o protocolo TCP[1], da camada de transporte, e a partir da execução de testes podemos avaliar alguns aspectos desse protocolo e posteriormente compará-lo com outro(s).

2 Desenvolvimento

Linux foi o sistema operacional utilizado para o desenvolvimento (distribuição 2.6.43.8-1.fc15.i686). Igualmente, para os testes utilizou-se duas máquinas com Linux, porém com distribuições diferentes.

2.1 Protocolo TCP - Transmission Control Protocol

O protocolo TCP foi escrito de modo a garantir que os dados enviados (pelo servidor) e recebidos (pelo cliente) de forma correta, na sequência adequada e sem erros, pela rede.
As características fundamentais do TCP são:

- *Orientado à conexão*: necessidade de uma conexão.
- *Ponto a ponto*: conexão é estabelecida entre dois pontos.
- *Confiabilidade*: Permite a recuperação de arquivos perdidos, elimina arquivos duplicados, recupera dados corrompidos, entrega na ordem do envio e pode recuperar o "link" entre cliente e servidor, caso esse, por algum motivo, seja perdido.
- *Full duplex*: Possível transferência simultânea, entre cliente e servidor.
- *Handshake*: Mecanismo de estabelecimento e finalização de conexão. O TCP garante que, no final da conexão, **todos** os pacotes sejam entregues ou recebidos.
- *Entrega ordenada*: A aplicação entrega ao TCP blocos de dados de tamanho variável. Esse protocolo divide estes dados em segmentos de tamanho especificado (valor MTU). Sabe-se que camadas inferiores à de transporte podem fazer com que os pacotes não cheguem na ordem em que foram enviados. Porém, o TCP garante a reconstrução dos segmentos no cliente (TCP utiliza um número de sequência).
- *Controle de fluxo*: O protocolo em estudo utiliza-se de um campo denominado **janela** para controlar o fluxo

[1] Mais informações sobre TCP http://www.linktionary.com/t/tcp.html

2.2 Implementação

O projeto conta com cinco diretórios, cada um com seu Makefile (exceto relatorio e estat), arquivos principal (main.c e main.h) e um README.md[2] para instruções adicionais. Há também um Makefile no diretório raiz, o qual compila todo o sistema.
Os diretório são:

I. *common*: Contém arquivos de uso comum, tanto pelo servidor quanto pelo cliente, inclusive o arquivo que calcula a média dos testes executados.

II. *server*: Contém os arquivos que preparam uma porta para esperar conexões e manipulam o sistema de livraria.

III. *client*: Funções que provém conexão com um servidor, envio das opções escolhidas pelo cliente e interface para as respostas do sistema de livraria (servidor).

IV. *estat*: Medidas de tempo efetuadas pelo teste.

V. *relatorio*: Arquivos de criação do relatório em LaTeX.

2.2.1 Manipulação de dados

Todas as estruturas utilizadas para leitura/escrita dos livros são dinâmicas.
Arquivos presentes no diretório *common*[I]:

I. *error.c[4.1.1] error.h*: Gerencia erros que eventualmente podem ocorrer.

II. *common.c[4.1.2] common.h*: Funções de uso comum.

III. *avl.c[4.1.3] avl.h*: Gerencia a estrutura básica da livraria, utiliza-se árvore AVL[3], pois a busca, inserção/atualização têm complexidade $\boxed{O(logN)}$, sendo N o número de elementos na árvore, no caso a quantidade de livros diferentes.

IV. *archives.c[4.1.4] archives.h*: Manipula arquivos. Faz a leitura do arquivo da livraria[4].

V. *tcp.c[4.1.5] tcp.h*: Contém apenas algumas constantes.

VI. *books.c books.h*: Gerencia a estrutura básica de um livro e seus autores.

VII. *tempo.c[4.1.7] tempo.h*: Gerencia a estrutura de testes, lê e escreve em arquivos localizados no diretório *estat*[IV].

VIII. *livros/livros*: Arquivo contendo os livros[5].

[2] Leia esse arquivo antes de executar o sistema
[3] http://pages.cs.wisc.edu/~ealexand/cs367/NOTES/AVL-Trees/index.html
[4] Ver README.md para mais detalhes do arquivo da livraria
[5] Ver README.md para mais detalhes do arquivo da livraria

2.2.2 Conexão Servidor/Cliente

Compreende dois diretórios *server*[II] e *client*[III]
Servidor:

I. *server.c[4.2.1] server.h*: Apenas inicia o servidor dada um número de uma porta.

II. *tcp_server.c[4.2.2] tcp_server.h*: Gerencia tanto as conexões com os clientes quanto a comunicação, em outras palavras, o tcp_server.c recebe um stream do tcp_client.c[II] e envia uma resposta adequada ao mesmo.

III. *login.c[4.2.3] login.h*: Gerencia o login necessário para editar a quantidade de um livro.

Cliente:

I. *client.c client.h*: Apenas inicia a comunicação com um servidor dado o endereço IP e um número de uma porta.

II. *tcp_client.c tcp_client.h*: Gerencia tanto a criação de uma conexão entre o cliente e servidor quanto a leitura, da entrada dada pelo usuário do sistema, e a comunicação entre hospedeiro e cliente.

2.3 Coleta e gerência de dados para testes

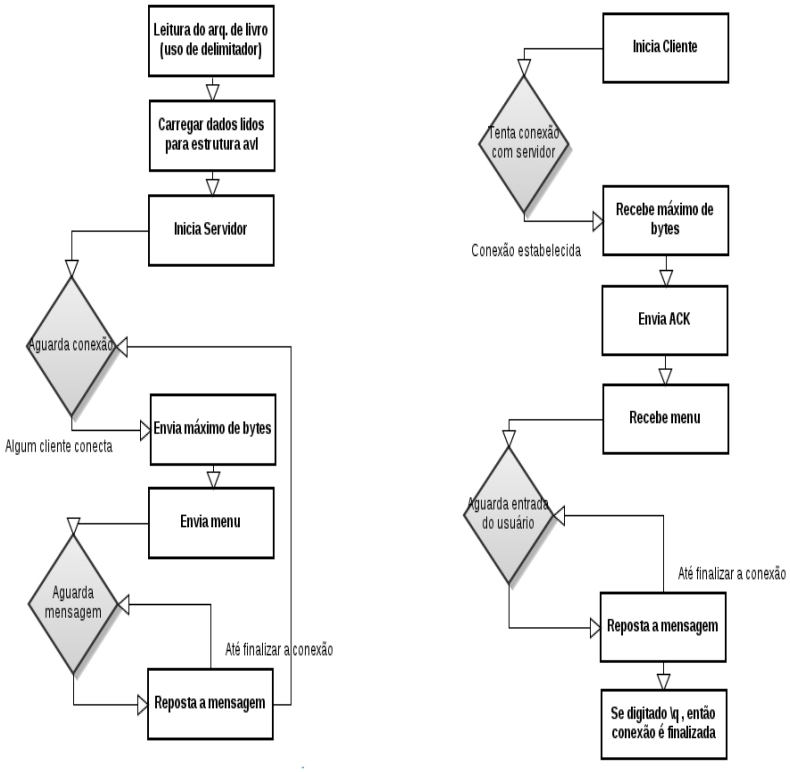
Para realizar os testes implementou-se alguns arquivos adicionas[VII]. Uma constante, denominada NUM_TESTES[6] , contém o número de testes a serem realizados, ou seja, cada opção do *menu*[II] é executada NUM_TESTE vezes. Todos os dados são salvos no diretório *estat*[IV].

2.4 Vantagens da implementação

O sistema de livraria é um sistema robusto e com baixa complexidade de tempo. A escolha da estrutura de árvore *AVL*[III], possibilitou em boa performance em questão de tempo de processamento, uma vez que, essa estrutura mostrou-se eficaz para o problema e possui melhor complexidade de tempo com relação a outras estruturas, além da implementação e manutenção serem relativamente simples.
Com relação às conexões e comunicações entre cliente/servidor, vale ressaltar que há uma troca de mensagens[7] inicial entre os dois, na qual o conteúdo do stream é o número de bytes da maior mensagem possível a ser enviada pelo servidor. Como a função $\boxed{rcv(int sockfd, void * buf, size_t len, int flags)}$[8] começa a ler o buffer, o qual é escrito o stream enviado pelo servidor, antes do mesmo estar com ele completo, em outras palavras, antes de toda mensagem enviada pelo servidor estar escrita no buffer então, o número de bytes das mensagens mostra-se necessário, uma vez que, podemos controlar a função $\boxed{rcv(int sockfd, void * buf, size_t len, int flags)}$, junto ao envio de mensagens de

[6] Nesse sistema consideramos NUM_TESTES igual a 100
[7] Sabe-se que o TCP envia/recebe streams
[8] http://linux.die.net/man/2/recv

controle (ACK[9]), afim de ler o buffer apenas quando o mesmo estiver completo.[10]



(a) Fluxograma Servidor (b) Fluxograma Cliente

Figura 1: Fluxogramas

[9] Veja tcp.h[V]

[10] The receive calls normally return any data available, up to the requested amount, rather than waiting for receipt of the full amount requested.

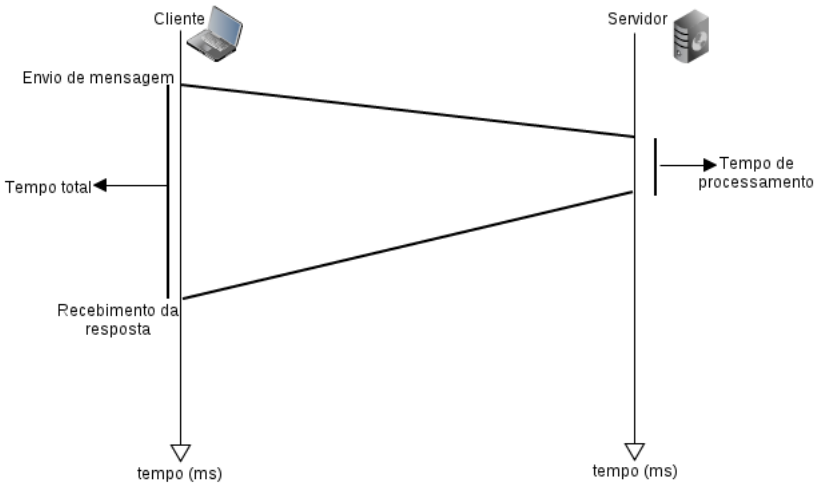


Figura 2: Definição do cálculo dos tempos

3 Resultados e discussões

Os testes foram efetuados em duas máquinas, ambas conectadas à rede porém, não localmente. Denominaremos a máquina servidor como [S] e a cliente como [C]. [S] e [C] estão em continentes diferentes.
Foram efetuadas 100 medições para cada opção do menu[II], ao todo foram 600 medições de tempo. Dividiu-se o tempo em tempo de processamento e tempo de comunicação, sendo o último a diferença entre o tempo total e o tempo de processamento.

3.1 Tabelas e gráficos

3.1.1 Tempo total

Representaremos todas as 100 medidas de tempo total das 6 opções do menu[II]

Opção 1 [ms]	Opção 2 [ms]	Opção 3 [ms]	Opção 4 [ms]	Opção 5 [ms]	Opção 6 [ms]
186134.000000	379959.000000	376944.000000	387384.000000	825847.000000	399407.000000
187389.000000	375949.000000	400865.000000	374474.000000	790328.000000	373987.000000
187232.000000	370050.000000	370365.000000	427843.000000	756741.000000	377698.000000
186872.000000	376848.000000	378147.000000	382172.000000	751569.000000	390804.000000
184125.000000	371970.000000	383610.000000	378222.000000	803135.000000	370656.000000
186449.000000	378102.000000	372725.000000	376055.000000	790978.000000	397889.000000
186252.000000	384827.000000	406249.000000	376315.000000	765067.000000	371087.000000
189478.000000	389313.000000	385968.000000	194417.000000	969968.000000	375421.000000
183976.000000	383991.000000	399733.000000	380161.000000	817787.000000	405330.000000
208636.000000	405586.000000	395936.000000	379678.000000	781436.000000	391226.000000
201516.000000	412777.000000	410493.000000	405222.000000	823250.000000	405368.000000
208826.000000	391916.000000	376920.000000	388630.000000	783629.000000	375988.000000
199975.000000	396349.000000	372669.000000	378038.000000	775647.000000	399652.000000
187522.000000	372619.000000	386818.000000	372268.000000	753483.000000	379150.000000
187232.000000	388165.000000	396910.000000	382632.000000	746090.000000	386807.000000
186570.000000	385603.000000	387320.000000	197574.000000	989000.000000	374798.000000
184801.000000	371246.000000	370559.000000	580704.000000	777143.000000	371615.000000
187593.000000	390852.000000	369696.000000	380162.000000	791327.000000	374636.000000
187224.000000	372110.000000	388092.000000	404885.000000	747343.000000	368790.000000
199953.000000	374939.000000	380271.000000	385934.000000	789833.000000	377796.000000
184321.000000	372986.000000	383898.000000	377787.000000	766127.000000	378929.000000
184627.000000	372867.000000	385548.000000	381415.000000	789028.000000	401560.000000
185264.000000	386347.000000	370293.000000	648557.000000	973036.000000	373652.000000
182234.000000	380072.000000	372772.000000	374264.000000	742324.000000	375463.000000
185997.000000	374853.000000	374805.000000	388695.000000	795654.000000	383348.000000
183630.000000	392627.000000	371155.000000	384329.000000	780909.000000	393012.000000
195198.000000	374530.000000	372819.000000	207618.000000	954986.000000	375117.000000
185220.000000	407452.000000	376015.000000	379876.000000	775651.000000	371329.000000
184872.000000	370111.000000	369152.000000	376061.000000	784223.000000	379558.000000
184440.000000	371786.000000	384647.000000	213692.000000	954416.000000	371527.000000
215691.000000	383864.000000	379819.000000	376815.000000	752007.000000	372939.000000
185690.000000	380330.000000	387667.000000	226705.000000	978050.000000	389300.000000
185212.000000	412519.000000	393443.000000	219818.000000	974375.000000	382385.000000
184893.000000	379579.000000	374329.000000	557030.000000	784645.000000	373910.000000
184262.000000	379550.000000	368171.000000	373069.000000	806536.000000	396343.000000
187784.000000	374553.000000	387358.000000	382635.000000	818931.000000	384945.000000
190727.000000	381625.000000	388207.000000	382833.000000	809578.000000	372323.000000
186847.000000	372065.000000	382376.000000	384977.000000	800117.000000	374682.000000
187853.000000	382966.000000	374341.000000	383732.000000	763441.000000	390692.000000
187898.000000	387376.000000	372352.000000	391522.000000	791098.000000	373671.000000
184341.000000	375648.000000	376884.000000	207540.000000	999633.000000	382487.000000
192566.000000	379688.000000	379830.000000	574534.000000	759718.000000	380988.000000
192088.000000	375459.000000	376261.000000	375237.000000	778494.000000	397173.000000
195817.000000	376471.000000	378325.000000	195837.000000	973809.000000	369877.000000
189500.000000	375927.000000	375789.000000	600420.000000	748282.000000	370903.000000
187443.000000	392176.000000	390918.000000	377294.000000	763193.000000	384424.000000
197499.000000	383317.000000	375919.000000	195515.000000	973658.000000	396167.000000
188974.000000	386863.000000	375807.000000	378985.000000	780145.000000	371593.000000
187302.000000	377841.000000	373075.000000	379188.000000	815248.000000	379885.000000
187996.000000	411911.000000	398038.000000	208805.000000	1031502.000000	411339.000000
207983.000000	394753.000000	414173.000000	570363.000000	758345.000000	373207.000000
183661.000000	372805.000000	375157.000000	372253.000000	775336.000000	368434.000000
188076.000000	384455.000000	383204.000000	211366.000000	967862.000000	371541.000000
193033.000000	376016.000000	390993.000000	388508.000000	792555.000000	374001.000000
203251.000000	377246.000000	371334.000000	392704.000000	797030.000000	375507.000000
191111.000000	381236.000000	426817.000000	495126.000000	823025.000000	405569.000000
208720.000000	406373.000000	411600.000000	407102.000000	772950.000000	435929.000000
191228.000000	381707.000000	389590.000000	211494.000000	1006556.000000	397694.000000
208735.000000	405468.000000	404575.000000	384187.000000	747142.000000	369092.000000
192248.000000	377373.000000	374398.000000	382093.000000	786838.000000	377644.000000
188302.000000	377298.000000	375974.000000	211832.000000	963735.000000	375433.000000
199366.000000	373149.000000	368093.000000	372351.000000	787738.000000	371240.000000
185200.000000	374735.000000	368255.000000	375555.000000	818610.000000	387515.000000
189354.000000	376027.000000	372728.000000	572923.000000	768839.000000	375768.000000
184946.000000	388418.000000	375615.000000	386961.000000	791648.000000	387758.000000
212453.000000	375709.000000	389035.000000	386677.000000	795321.000000	372274.000000
186990.000000	368840.000000	395126.000000	392726.000000	796099.000000	400387.000000
187158.000000	388926.000000	387413.000000	380430.000000	798079.000000	372894.000000
186925.000000	368673.000000	370925.000000	384439.000000	799194.000000	370114.000000
187869.000000	371949.000000	371555.000000	604662.000000	749932.000000	388944.000000
190985.000000	376248.000000	428680.000000	389692.000000	812855.000000	412465.000000
192168.000000	414355.000000	372998.000000	398815.000000	785395.000000	381061.000000
188490.000000	379056.000000	375593.000000	562676.000000	773942.000000	376122.000000
198904.000000	376609.000000	374853.000000	372192.000000	780995.000000	387099.000000
211250.000000	401945.000000	369829.000000	375258.000000	838770.000000	376507.000000
190856.000000	397610.000000	372959.000000	196175.000000	990778.000000	397232.000000
186611.000000	386879.000000	383926.000000	418567.000000	740847.000000	374729.000000
184224.000000	400721.000000	374829.000000	372079.000000	780270.000000	388227.000000
188581.000000	374753.000000	373974.000000	194860.000000	964569.000000	367928.000000
200660.000000	371650.000000	379977.000000	422544.000000	781779.000000	395918.000000
187730.000000	370857.000000	394231.000000	377544.000000	801518.000000	372307.000000
183613.000000	367969.000000	371856.000000	374463.000000	798388.000000	374634.000000
182749.000000	369355.000000	373896.000000	376970.000000	830862.000000	371330.000000
184443.000000	372903.000000	379262.000000	376326.000000	791705.000000	386736.000000
191332.000000	370708.000000	388287.000000	400862.000000	796368.000000	376339.000000
186974.000000	370738.000000	381855.000000	391890.000000	804335.000000	371677.000000
183429.000000	381375.000000	402648.000000	206752.000000	965144.000000	371026.000000
188809.000000	383981.000000	382801.000000	412661.000000	755087.000000	388206.000000
187492.000000	380894.000000	380474.000000	390881.000000	785804.000000	387494.000000
189442.000000	385549.000000	403718.000000	223366.000000	967479.000000	376311.000000
188319.000000	386792.000000	367633.000000	373351.000000	803788.000000	372622.000000
212336.000000	382847.000000	385087.000000	400206.000000	797278.000000	411486.000000
184941.000000	374897.000000	380782.000000	575655.000000	743384.000000	368429.000000
199550.000000	383519.000000	419549.000000	388674.000000	772949.000000	370838.000000
188052.000000	384007.000000	376016.000000	383672.000000	795339.000000	411770.000000
184290.000000	367800.000000	372644.000000	198639.000000	958940.000000	377237.000000
189488.000000	372088.000000	387662.000000	377000.000000	763934.000000	387930.000000
187303.000000	375202.000000	368928.000000	372877.000000	787705.000000	384990.000000
187554.000000	375090.000000	424597.000000	387913.000000	812114.000000	371655.000000
184799.000000	376349.000000	375084.000000	435362.000000	755337.000000	386787.000000

Tabela 1: Tabela de tempo total

3.1.2 Tempo de processamento

Representaremos todas as 100 medidas de tempo de processamento das 6 opções do menu[II]

Opção 1 [ms]	Opção 2 [ms]	Opção 3 [ms]	Opção 4 [ms]	Opção 5 [ms]	Opção 6 [ms]
124.000000	187864.000000	188315.000000	555.000000	587293.000000	399407.000000
58.000000	188106.000000	212262.000000	545.000000	564101.000000	373987.000000
57.000000	184076.000000	183957.000000	554.000000	568421.000000	377698.000000
54.000000	187812.000000	188119.000000	577.000000	555928.000000	390804.000000
58.000000	187866.000000	188052.000000	550.000000	572451.000000	370656.000000
57.000000	190486.000000	188099.000000	547.000000	556187.000000	397889.000000
59.000000	200030.000000	216218.000000	543.000000	564367.000000	371087.000000
68.000000	191954.000000	188029.000000	550.000000	564026.000000	375421.000000
57.000000	188039.000000	183860.000000	569.000000	600243.000000	405330.000000
55.000000	203982.000000	200087.000000	545.000000	564148.000000	391226.000000
59.000000	207885.000000	215889.000000	560.000000	560172.000000	405368.000000
56.000000	204139.000000	188095.000000	569.000000	560065.000000	375988.000000
57.000000	212125.000000	185139.000000	552.000000	588025.000000	399652.000000
57.000000	183855.000000	184042.000000	531.000000	568018.000000	379150.000000
56.000000	200041.000000	187957.000000	543.000000	560106.000000	386807.000000
62.000000	188081.000000	184264.000000	541.000000	582428.000000	374798.000000
50.000000	183926.000000	183808.000000	536.000000	591891.000000	371615.000000
58.000000	188064.000000	184054.000000	561.000000	571918.000000	374636.000000
48.000000	184483.000000	199956.000000	545.000000	559977.000000	368790.000000
62.000000	188110.000000	184008.000000	543.000000	568044.000000	377796.000000
57.000000	188397.000000	188036.000000	550.000000	576041.000000	378929.000000
58.000000	187930.000000	184344.000000	554.000000	560274.000000	401560.000000
59.000000	198271.000000	184166.000000	556.000000	560677.000000	373652.000000
61.000000	195834.000000	188164.000000	551.000000	556033.000000	375463.000000
61.000000	188065.000000	188141.000000	563.000000	572326.000000	383348.000000
58.000000	195857.000000	184088.000000	581.000000	560592.000000	393012.000000
59.000000	187831.000000	183927.000000	586.000000	564051.000000	375117.000000
59.000000	196151.000000	187987.000000	565.000000	587935.000000	371329.000000
58.000000	183849.000000	183537.000000	548.000000	558832.000000	379558.000000
59.000000	184247.000000	200472.000000	551.000000	564474.000000	371527.000000
58.000000	184117.000000	188142.000000	575.000000	563958.000000	372939.000000
58.000000	188044.000000	200219.000000	552.000000	604257.000000	389300.000000
59.000000	224286.000000	188142.000000	557.000000	580073.000000	382385.000000
60.000000	195970.000000	189240.000000	547.000000	596047.000000	373910.000000
59.000000	184066.000000	184175.000000	553.000000	584255.000000	396343.000000
56.000000	185871.000000	187648.000000	556.000000	592302.000000	384945.000000
57.000000	191902.000000	200239.000000	555.000000	588301.000000	372323.000000
59.000000	187992.000000	185829.000000	543.000000	604191.000000	374682.000000
59.000000	195889.000000	188046.000000	591.000000	576113.000000	390692.000000
60.000000	187754.000000	188027.000000	582.000000	564300.000000	373671.000000
60.000000	188167.000000	187906.000000	546.000000	604714.000000	382487.000000
58.000000	191822.000000	187851.000000	545.000000	569860.000000	380988.000000
58.000000	188031.000000	188179.000000	581.000000	555921.000000	397173.000000
58.000000	188003.000000	191983.000000	550.000000	572221.000000	369877.000000
62.000000	189106.000000	188221.000000	556.000000	559985.000000	370903.000000
75.000000	196087.000000	188579.000000	596.000000	575807.000000	384424.000000
58.000000	199696.000000	187994.000000	542.000000	572493.000000	396167.000000
53.000000	187931.000000	187963.000000	551.000000	559996.000000	371593.000000
57.000000	193525.000000	188035.000000	591.000000	576394.000000	379885.000000
68.000000	216357.000000	203946.000000	549.000000	604193.000000	411339.000000
58.000000	195860.000000	215849.000000	549.000000	572449.000000	373207.000000
59.000000	188330.000000	188099.000000	547.000000	559829.000000	368434.000000
58.000000	195911.000000	196222.000000	565.000000	567938.000000	371541.000000
60.000000	188164.000000	200387.000000	567.000000	556361.000000	374001.000000
61.000000	192030.000000	183926.000000	570.000000	573747.000000	375507.000000
58.000000	192066.000000	204376.000000	622.000000	620089.000000	405569.000000
56.000000	200254.000000	207862.000000	547.000000	579521.000000	435929.000000
56.000000	188240.000000	199903.000000	553.000000	585557.000000	397694.000000
57.000000	203828.000000	188165.000000	587.000000	559933.000000	369092.000000
58.000000	188029.000000	188186.000000	570.000000	562390.000000	377644.000000
78.000000	187562.000000	188052.000000	555.000000	560382.000000	375433.000000
60.000000	188118.000000	184142.000000	553.000000	564387.000000	371240.000000
56.000000	188206.000000	184236.000000	542.000000	592450.000000	387515.000000
47.000000	188048.000000	186500.000000	551.000000	583946.000000	375768.000000
57.000000	200284.000000	187825.000000	565.000000	604141.000000	387758.000000
54.000000	188029.000000	188401.000000	551.000000	569690.000000	372274.000000
61.000000	184438.000000	208265.000000	559.000000	580572.000000	400387.000000
56.000000	183981.000000	199942.000000	560.000000	576014.000000	372894.000000
59.000000	183958.000000	183889.000000	570.000000	560146.000000	370114.000000
64.000000	184077.000000	184155.000000	547.000000	562996.000000	388944.000000
59.000000	187871.000000	200217.000000	561.000000	564238.000000	412465.000000
61.000000	187792.000000	187861.000000	557.000000	560074.000000	381061.000000
57.000000	188182.000000	188080.000000	561.000000	587962.000000	376122.000000
57.000000	187823.000000	187817.000000	558.000000	556263.000000	387099.000000
53.000000	211905.000000	184627.000000	540.000000	588097.000000	376507.000000
56.000000	192265.000000	188196.000000	566.000000	564699.000000	397232.000000
59.000000	197549.000000	196189.000000	531.000000	553780.000000	374729.000000
57.000000	188045.000000	188115.000000	545.000000	556397.000000	388227.000000
57.000000	187903.000000	190450.000000	576.000000	563761.000000	367928.000000
56.000000	184213.000000	184440.000000	562.000000	592242.000000	395918.000000
58.000000	183973.000000	207926.000000	552.000000	568260.000000	372307.000000
59.000000	184288.000000	188008.000000	560.000000	564199.000000	374634.000000
58.000000	184355.000000	183775.000000	589.000000	594113.000000	371330.000000
57.000000	187898.000000	188201.000000	548.000000	568013.000000	386736.000000
59.000000	183910.000000	188028.000000	546.000000	573846.000000	376339.000000
59.000000	184344.000000	184124.000000	546.000000	580129.000000	371677.000000
58.000000	196219.000000	216001.000000	554.000000	564235.000000	371026.000000
60.000000	184028.000000	196161.000000	573.000000	555879.000000	388206.000000
60.000000	185273.000000	196133.000000	550.000000	552448.000000	387494.000000
61.000000	186253.000000	188227.000000	552.000000	564098.000000	376311.000000
56.000000	199852.000000	184153.000000	555.000000	553049.000000	372622.000000
59.000000	195951.000000	201407.000000	566.000000	569645.000000	411486.000000
60.000000	187867.000000	196270.000000	549.000000	556187.000000	368429.000000
57.000000	199919.000000	209297.000000	558.000000	564195.000000	370838.000000
61.000000	196153.000000	188059.000000	557.000000	568313.000000	411770.000000
61.000000	184016.000000	187982.000000	600.000000	556060.000000	377237.000000
59.000000	187972.000000	188077.000000	553.000000	571894.000000	387930.000000
60.000000	187983.000000	184149.000000	545.000000	561857.000000	384990.000000
58.000000	187858.000000	227777.000000	546.000000	572339.000000	371655.000000
55.000000	188121.000000	188091.000000	547.000000	569532.000000	386787.000000

Tabela 2: Tabela de tempo de processamento

3.2 Médias

Devido ao grande número de dados decidimos por representar apenas as tabelas e desvios de cada opção do menu[II].

3.2.1 Cálculos

A média calculada entre os pontos foi a média simples, em outras palavras, denominamos a média como $\boxed{\mu}$ então,

$$\mu = \frac{1}{N}\sum_{i=1}^{N}(x_i)\,,$$

onde N é o número de pontos, no caso desse projeto temos N=100 medidas para cada uma das opções de menu[II].
Já o desvio padrão amostral, denominado $\boxed{\sigma}$ é calculado como

$$\sigma = \sqrt{\frac{1}{N}\sum_{i=1}^{N}(x_i-\mu)^2}.$$

Seja $\boxed{\mu_t}$, $\boxed{\mu_p}$, $\boxed{\mu_c}$, a média do tempo total, média do tempo de processamento e média do tempo de comunicação, respectivamente, e o desvio padrão do tempo de comunicação, denominado $\boxed{\sigma_c}$.

Menu	$\boxed{\mu_t}$	$\boxed{\mu_p}$	$\boxed{\mu_c}$	$\boxed{\pm\sigma_c}$
1	190559,24	59,13	190500,11	7720,62 (4,05%)
2	381804,32	191211,66	190592,66	3446,65 (1,80%)
3	383214,91	191636,56	191578,35	4483,76 (4,89%)
4	374492,29	557,72	373934,57	99632,35 (26,64%)
5	820829,07	571827,67	249001,40	64343,48 (25,84%)
6	382556,63	191372,81	191183,82	12192.48 (6,37%)

Tabela 3: Tabela com médias e desvios

4 Códigos Fonte *(.c)*

4.1 Diretório *commons*

4.1.1 *error.c*

```
1 #include "error.h"
2
3 void printError(int e){
4   switch(e){
5   case ERROR_SOCKET:
6     perror("socket:: nao foi possivel criar um novo socket\n");
7     break;
8   case ERROR_SOCKET_CLIENT:
9     perror("socket:: nao foi possivel conectar ao cliente\n");
```

```
10      break;
11    case ERROR_SOCKET_SEND:
12      perror("socket:: nao foi possivel enviar mensagem para o
            servidor\n");
13      break;
14    case ERROR_SOCKET_SERVER_ERROR:
15      perror("send_msg:: ocorreu um erro na comunicacao com o
            servidor, tente novamente");
16      break;
17    case ERROR_SOCKET_CLIENT_CLOSED:
18      perror("receive_msg:: cliente fechou a conexao");
19      break;
20    case ERROR_SOCKET_SERVER_CLOSED:
21      perror("send_msg:: servidor fechou a conexao");
22      break;
23    case ERROR_SOCKET_SERVER_BIND:
24      perror("start_server:: ocorreu um erro com o bind");
25      break;
26    case ERROR_SOCKET_SERVER_LISTEN:
27      perror("start_server:: ocorreu um erro com o listen");
28      break;
29    case ERROR_SOCKET_SERVER_ACCEPT:
30      perror("start_server:: ocorreu um erro com o accept");
31      break;
32    case ERROR_USAGE_MAIN:
33      printf("erro de entrada: tente ./main <num_porta> (para iniciar
              servidor) ou ./main <endereco_ip> <porta> para iniciar o
            cliente\n");
34      break;
35    case ERROR_USAGE_TEST:
36      printf("erro de entrada: tente ./test <endereco_ip> <porta>
            para iniciar o cliente (o servidor deve estar up!)\n");
37      break;
38    case ERROR_FILE_OPEN:
39      perror("archives:: Erro ao abrir arquivo\n");
40      break;
41    case ERROR_NUM_COLUNAS_BOOKS:
42      printf("archives:: Arquivo com conteudo de livros corrompido.
            VEJA README PARA MAIS DETALHES\n");
43      break;
44    }
45  }
```

4.1.2 *common.c*

```
1  #include "common.h"
2
3  char* my_itoa(int val, int base){
4    static char buf[32] = {0};
5    int i = 30;
6    for(; val && i ; --i, val /= base)
7      buf[i] = "0123456789abcdef"[val % base];
8    return &buf[i+1];
9  }
10 void poscpy(char *t,char *s,int a,int b){
11    while ( a < b ) *(t++) = *(s + (a++));
12    *t = 0x0;
13 }
14
15 char **split(char *str,char n,int *length){
16    register int i, j;
```

```
17      /* control */
18      int len=strlen(str);
19      int elements = 0, elpos = 0;
20      char **array;
21      /* number of new itens in array */
22      for(i=0;i<len;i++) if(str[i]==n) elements++;
23      /* get the memory */
24      array=(char **)calloc(elements,sizeof(char *));
25      if(!array){
26          printf("# Error in malloc(*).\n");
27          return NULL;
28      }
29      /* the number of elements for the caller */
30      *length=elements;
31      /* lvl1
32       *
33       * @i = will be the start point for copy
34       */
35      for(i=0;i<len;i++)
36        /* lvl2
37         *
38         * @j = will be end point for copy
39         */
40        for(j=i;j<=len;j++)
41          /* found splitChar or EoL */
42          if(str[j]==n){
43      /*
44       * @i has start point
45       * @j has end point
46       */
47      array[elpos]=(char *)malloc((j-i+1)*sizeof(char));
48      if ( !array[ elpos ] ){
49        printf("# lvl2\n");
50        printf(" # Error in malloc().\n");
51        return NULL;
52      }
53      /* copy the string into the array */
54      poscpy(array[elpos],str,i,j);
55      /* increment array position */
56      elpos++;
57      /* after the copy is done,
58       * @i must be equal to @j
59       */
60      i=j;
61      /* end loop lvl2 */
62      break;
63          }
64      /* return array */
65      return array;
66  }
```

4.1.3 *avl.c*

```
1  #include "avl.h"
2
3  void newAVLTree(AVL *avl){
4    *avl=NULL;
5  }
6
7  int altura(AVL avl){
8    if (avl == NULL)
```

```
 9      return 0;
10    int hesq = altura(avl->esq);
11    int hdir = altura(avl->dir);
12    return hesq > hdir ? hesq + 1 : hdir + 1;
13  }
14
15  int verifica_AVL(AVL avl){
16    if (avl==NULL)
17      return 1;
18    return abs(altura(avl->dir) - altura(avl->esq)) <= 1;
19  }
20
21  void LL(AVL *avl){
22    AVL_NO *b = *avl;
23    AVL_NO *a = b->esq;
24    b->esq = a->dir;
25    a->dir = b;
26    a->bal = 0;
27    b->bal = 0;
28    *avl = a;
29  }
30
31  void RR(AVL *avl){
32    AVL_NO *a = *avl;
33    AVL_NO *b = a->dir;
34    a->dir = b->esq;
35    b->esq = a;
36    a->bal = 0;
37    b->bal = 0;
38    *avl = b;
39  }
40  void LR(AVL *avl){
41    AVL_NO *c = *avl;
42    AVL_NO *a = c->esq;
43    AVL_NO *b = a->dir;
44    c->esq = b->dir;
45    a->dir = b->esq;
46    b->esq = a;
47    b->dir = c;
48    switch(b->bal) {
49    case -1:
50      a->bal = 0;
51      c->bal = 1;
52      break;
53    case 0:
54      a->bal = 0;
55      c->bal = 0;
56      break;
57    case +1:
58      a->bal = -1;
59      c->bal = 0;
60      break;
61    }
62    b->bal = 0;
63    *avl = b;
64  }
65
66  void RL(AVL *avl){
67    AVL_NO *a = *avl;
68    AVL_NO *c = a->dir;
69    AVL_NO *b = c->esq;
70    c->esq = b->dir;
```

```
71    a->dir = b->esq;
72    b->esq = a;
73    b->dir = c;
74    switch(b->bal) {
75    case -1:
76      a->bal = 0;
77      c->bal = 1;
78      break;
79    case 0:
80      a->bal = 0;
81      c->bal = 0;
82      break;
83    case +1:
84      a->bal = -1;
85      c->bal = 0;
86      break;
87    }
88    b->bal = 0;
89    *avl = b;
90  }
91
92  int aux_insereAVL(AVL *avl, Livro *l, int *cresceu){
93    if (*avl == NULL) {
94      AVL_NO *no = (AVL_NO *)malloc(sizeof(struct avl));
95      //     Livro *temp = (Livro *)malloc(sizeof(struct book));
96      no->id = (char *)malloc(sizeof(char)*strlen(l->isbn)+1);
97      memset(no->id,'\0',strlen(l->isbn)+1);
98      memmove(no->id,l->isbn,strlen(l->isbn)+1);
99      //     memmove(temp,l,sizeof(l));
100     no->bal = 0;
101     no->esq = NULL;
102     no->dir = NULL;
103     no->dado=(void *)l;
104     *avl = no;
105     *cresceu = 1;
106     return 1;
107   }
108   if(memcmp(l->isbn,((Livro *)(*avl)->dado)->isbn,strlen(l->isbn))
          >0){
109     if(aux_insereAVL(&(*avl)->esq,l,cresceu)) {
110       if(*cresceu) {
111   switch((*avl)->bal) {
112   case -1:
113     if((*avl)->esq->bal == -1)
114       LL(avl);
115     else
116       LR(avl);
117     *cresceu = 0;
118     break;
119   case 0:
120     (*avl)->bal = -1;
121     *cresceu = 1;
122     break;
123   case +1:
124     (*avl)->bal = 0;
125     *cresceu = 0;
126     break;
127   }
128       }
129       return 1;
130     }
131     else
```

```
132         return 0;
133     }
134     if(aux_insereAVL(&(*avl)->dir,l,cresceu)) {
135       if(*cresceu) {
136         switch((*avl)->bal) {
137         case -1:
138     (*avl)->bal = 0;
139     *cresceu = 0;
140     break;
141         case 0:
142     (*avl)->bal = +1;
143     *cresceu = 1;
144     break;
145         case +1:
146     if ((*avl)->dir->bal == +1)
147       RR(avl);
148     else
149       RL(avl);
150     *cresceu = 0;
151     break;
152         }
153       }
154       return 1;
155     }
156     else
157       return 0;
158 }
159
160 int insereAVL(AVL *avl, Livro *l) {
161     int cresceu;
162     return aux_insereAVL(avl,l,&cresceu);
163 }
164
165 /**
166  * @function : Grava todos os ids separados pelo delimitador em uma
          string
167  * @param AVL avl : estrutura avl
168  * @param char **str : string q contera os ids
169  * @param char *del : delimitador entre ids
170  * @example : str tera o formato delstrdelstr....
171  **/
172 void avlIdToStr(AVL avl, char **str, char *del){
173     if(avl!=NULL){
174       avlIdToStr(avl->esq,str,del);
175       strcat(*str,del);
176       strcat(*str,avl->id);
177       avlIdToStr(avl->dir,str,del);
178     }
179 }
180
181 void avlToStr(AVL avl, char **str, char *del){
182     if(avl!=NULL){
183       avlToStr(avl->esq,str,del);
184       strcat(*str,bookNodeToStr((Livro *)avl->dado));
185       strcat(*str,"\n");
186       avlToStr(avl->dir,str,del);
187     }
188 }
189
190 int totAVLchar(AVL avl){
191     if(avl==NULL)
192       return 0;
```

```
193   int e=totAVLchar(avl->esq);
194   int d=totAVLchar(avl->dir);
195   return e+d+getBookNumBytes((Livro *)avl->dado);
196 }
197
198 int totISBNchar(AVL avl){
199   if(avl==NULL)
200     return 0;
201   int e = totISBNchar(avl->esq);
202   int d = totISBNchar(avl->dir);
203   return (e+d+strlen(avl->id));
204 }
205
206 int totElemAVL(AVL avl){
207   if(avl==NULL)
208     return 0;
209   int e = totElemAVL(avl->esq);
210   int d = totElemAVL(avl->dir);
211   return (e+d+1);
212 }
213
214 void freeAVL(AVL_NO *avl){
215   if(avl!=NULL){
216     freeAVL(avl->esq);
217     freeAVL(avl->dir);
218     free(avl);
219   }
220 }
221
222 AVL_NO *getAVLElemById(AVL_NO *avl, char *id){
223   if(&(*avl) != NULL){
224     AVL_NO *aux = avl;
225     if(strcmp(aux->id,id)==0)
226       return aux;
227     else if(strcmp(aux->id,id)<0)
228       return getAVLElemById(aux->esq,id);
229     return getAVLElemById(aux->dir,id);
230   }
231   return NULL;
232 }
```

4.1.4 *archives.c*

```
1 #include "archives.h"
2
3 FILE *openFile(char *nome, char *param){
4   FILE *f;
5   //abre em modo leitura
6   f = fopen(nome, param);
7   if(f == NULL){
8       printError(ERROR_FILE_OPEN);
9       exit(0);
10  }
11  return f;
12 }
13
14 void readFileBooks(FILE *f, AVL *avl){
15  char del='#';
16  char delAutores=';';
17  char **arr=NULL;//array
18  char **arrAutores=NULL;//array p/ split de autores
```

```
19    char buffer[ARCHIVES_TAM_BUFFER];
20    int col=0; //coluna que esta sendo lida
21    int len=0;
22    int i=0;
23    Autores autores;
24    while(fgets(buffer, ARCHIVES_TAM_BUFFER, f)){
25      arr=split(buffer,del,&col);
26      if(col!=NUM_COLUNAS_BOOKS){
27        printError(ERROR_NUM_COLUNAS_BOOKS);
28        exit(1);
29      }
30      else{
31        Livro *l = (Livro *)malloc(sizeof(struct book));
32        arrAutores=split(arr[AUTORES],delAutores,&col);//split
              autores
33        newAutoresList(&autores);
34        for(i=0;i<col;i++)
35    insertAutoresList(&autores,arrAutores[i]);
36        l->autores = autores;//(Autor *)malloc(sizeof(struct autor));
37        //       memmove(l->autores,autores,sizeof(autores));
38        len=strlen(arr[ISBN]);
39        l->isbn=(char *)malloc(sizeof(char)*len+1);
40        memmove(l->isbn,arr[ISBN],len+1);
41        len=strlen(arr[TITULO]);
42        l->titulo=(char *)malloc(sizeof(char)*len+1);
43        memmove(l->titulo,arr[TITULO],len+1);
44        len=strlen(arr[ANO]);
45        l->ano=(char *)malloc(sizeof(char)*len+1);
46        memmove(l->ano,arr[ANO],len+1);
47        len=strlen(arr[DESCRICAO]);
48        l->desc=(char *)malloc(sizeof(char)*len+1);
49        memmove(l->desc,arr[DESCRICAO],len+1);
50        len=strlen(arr[EDITORA]);
51        l->editora=(char *)malloc(sizeof(char)*len+1);
52        memmove(l->editora,arr[EDITORA],len+1);
53        l->estoque=atoi(arr[ESTOQUE]);
54        insereAVL(avl,l);
55      }
56    }
57  }
```

4.1.5 *tcp.c*

```
1  #include "tcp.h"
```

4.1.6 *books.c*

```
1  #include "books.h"
2
3  void newAutoresList(Autores *a){
4    *a=NULL;
5  }
6
7  void insertAutoresList(Autores *a, char *nome){
8    Autor *autor=(Autor *)malloc(sizeof(struct autor));
9    autor->nome=(char *)malloc(sizeof(char)+strlen(nome));
10   autor->prox=NULL;
11   memmove(autor->nome,nome,strlen(nome)+1);
```

```
12    if(*a==NULL)
13      *a = autor;
14    else{
15      Autor *aux = *a;
16      while(aux->prox!=NULL)
17        aux=aux->prox;
18      aux->prox=autor;
19    }
20  }
21
22  void freeAutoresList(Autores *a){
23    Autor *aux=*a;
24    while(aux!=NULL){
25      Autor *r=aux;
26      aux=aux->prox;
27      free(r);
28    }
29  }
30  int getAutoresNum(Autor *a){
31    if(a==NULL)
32      return 0;
33    return 1+getAutoresNum(a->prox);
34  }
35  int getAutoresNumBytes(Autor *a){
36    Autor *aux = a;
37    int len=0;
38    while(aux!=NULL){
39      len+=strlen(aux->nome)+1;
40      aux=aux->prox;
41    }
42    return len;
43  }
44  int getBookNumBytes(Livro *l){
45    int len=0;
46    len=strlen(l->isbn)+strlen(l->titulo)+strlen(l->desc)+strlen(l->
          editora);
47    len+=strlen(l->ano)+sizeof(int)+6;
48    len+=getAutoresNumBytes(l->autores);
49    return len;
50  }
51  char *autoresNodeToStr(Autor *n,char *del){
52    Autor *a=n;
53    int len=getAutoresNumBytes(n)+getAutoresNum(n)*strlen(del)+1;
54    char *str=(char *)malloc(len);
55    memset(str,'\0',len);
56    while(a!=NULL){
57      strcat(str,a->nome);
58      strcat(str,del);
59      a=a->prox;
60    }
61    return str;
62  }
63  int getBookNodeSize(){
64    char *isbn="\nISBN: ";
65    char *tit="\nTITULO: ";
66    char *desc="\nDESCRICAO: ";
67    char *autores="\nAutor(es): ";
68    char *edit="\nEditora: ";
69    char *ano="\nAno: ";
70    char *estoque="\nQuantidade: ";
71    return strlen(isbn)+strlen(tit)+strlen(desc)+strlen(autores)+
          strlen(edit)+strlen(ano)+strlen(estoque)+10;
```

```
72 }
73 char *bookNodeToStr(Livro *l){
74   int len = getBookNumBytes(l);
75   char *isbn="\nISBN: ";
76   char *tit="\nTITULO: ";
77   char *desc="\nDESCRICAO: ";
78   char *autores="\nAutor(es): ";
79   char *edit="\nEditora: ";
80   char *ano="\nAno: ";
81   char *estoque="\nQuantidade: ";
82   len+=strlen(isbn)+strlen(tit)+strlen(desc)+strlen(autores)+strlen
         (edit)+strlen(ano)+strlen(estoque);
83   len+=4;
84   //agora ja temos o tamanho total da msg len+4 (\n\0)
85   char *msg=(char *)malloc(len);//\n
86   memset(msg,'\0',len);
87   strcat(msg,isbn);
88   strcat(msg,l->isbn);
89   strcat(msg,tit);
90   strcat(msg,l->titulo);
91   strcat(msg,desc);
92   strcat(msg,l->desc);
93   strcat(msg,autores);
94   strcat(msg,autoresNodeToStr(l->autores,", "));
95   strcat(msg,edit);
96   strcat(msg,l->editora);
97   strcat(msg,ano);
98   strcat(msg,l->ano);
99   strcat(msg,estoque);
100  strcat(msg,my_itoa(l->estoque,10));
101  strcat(msg,"\n");
102  return msg;
103 }
```

4.1.7 *tempo.c*

```
1  #include "tempo.h"
2
3  void newMedia(Media **m, char *nome){
4    *m=(Media *)malloc(sizeof(struct headTempo));
5    (*m)->medida=(char *)malloc(sizeof(char)*strlen(nome));
6    memset((*m)->medida,'\0',strlen(nome));
7    memmove((*m)->medida,nome,strlen(nome));
8    (*m)->media=00.00;
9    (*m)->desvio=00.00;
10   (*m)->length=0;
11 }
12 void calcMedia(Media **m){
13   Tempo *t=(*m)->dados;
14   double c=00.00;
15   while(t!=NULL){
16     c+=t->decorrido;
17     t=t->prox;
18   }
19   (*m)->media=c/(double)(*m)->length;
20   //desvio
21   t=(*m)->dados;
22   c=00.00;
23   while(t!=NULL){
24     c+=((t->decorrido-(*m)->media)*(t->decorrido-(*m)->media));
25     t=t->prox;
```

```
26    }
27    (*m)->desvio=sqrt((1.00/((double)(*m)->length)*c));
28  }
29  void insertTempo(Media **m,double i,double f){
30    Tempo *t=(Tempo *)malloc(sizeof(struct tempo));
31    t->inicio=i;
32    t->fim=f;
33    t->decorrido=f-i;
34    t->prox=NULL;
35    Tempo *aux=(*m)->dados;
36    if(aux==NULL){
37      (*m)->dados = t;
38    }
39    else{
40      while(aux->prox!=NULL)
41        aux=aux->prox;
42      aux->prox=t;
43    }
44    (*m)->length++;
45  }
46
47  void readFileMedia(Media **m, char *fname){
48    FILE *f=openFile(fname,"r");
49    char buffer[ARCHIVES_TAM_BUFFER];
50    memset(buffer,'\0',ARCHIVES_TAM_BUFFER);
51    double val;
52    while(fgets(buffer, ARCHIVES_TAM_BUFFER, f)){
53      val=atof(buffer);
54      insertTempo(m,0,val);
55    }
56  }
57
58  void writeFileMedia(Media **m){
59    char *str = (char *)malloc(sizeof(char)*(strlen("./estat/")+
          strlen((*m)->medida)));
60    memset(str,'\0',strlen(str));
61    strcat(str,"./estat/");
62    strcat(str,((*m)->medida));
63    FILE *f=openFile(str,"w+");
64    fprintf(f,"%lf (+-) %lf\n",(*m)->media, (*m)->desvio);
65    fclose(f);
66  }
67
68  int main(int argc, char *argv[]){
69    Media *m[NUM_OPCOES_MENU*2];
70    char files[NUM_OPCOES_MENU*2][10];
71    strcpy(files[0],"mtt_1\0");
72    strcpy(files[1],"mtt_2\0");
73    strcpy(files[2],"mtt_3\0");
74    strcpy(files[3],"mtt_4\0");
75    strcpy(files[4],"mtt_5\0");
76    strcpy(files[5],"mtt_6\0");
77    strcpy(files[6],"mtp_1\0");
78    strcpy(files[7],"mtp_2\0");
79    strcpy(files[8],"mtp_3\0");
80    strcpy(files[9],"mtp_4\0");
81    strcpy(files[10],"mtp_5\0");
82    strcpy(files[11],"mtp_6\0");
83    char rfiles[NUM_OPCOES_MENU*2][15];
84    strcpy(rfiles[0],"./estat/tt_1\0");
85    strcpy(rfiles[1],"./estat/tt_2\0");
86    strcpy(rfiles[2],"./estat/tt_3\0");
```

```
87    strcpy(rfiles[3],"./estat/tt_4\0");
88    strcpy(rfiles[4],"./estat/tt_5\0");
89    strcpy(rfiles[5],"./estat/tt_6\0");
90    strcpy(rfiles[6],"./estat/tp_1\0");
91    strcpy(rfiles[7],"./estat/tp_2\0");
92    strcpy(rfiles[8],"./estat/tp_3\0");
93    strcpy(rfiles[9],"./estat/tp_4\0");
94    strcpy(rfiles[10],"./estat/tp_5\0");
95    strcpy(rfiles[11],"./estat/tp_6\0");
96    int i=0;
97    for(i=0;i<NUM_OPCOES_MENU*2;i++){
98      newMedia(&m[i],files[i]);
99      readFileMedia(&m[i],rfiles[i]);
100     calcMedia(&m[i]);
101     writeFileMedia(&m[i]);
102   }
103   return 0;
104 }
```

4.2 Diretório *server*

4.2.1 *server.c*

```
1  #include "server.h"
2  void start_server(char *argv[]){
3    AVL l;
4    newAVLTree(&l);
5    load_books(&l);
6    bindNlisten(atoi(argv[1]),&l);
7    //  freeAVL(l);
8  }
9
10 void load_books(AVL *l){
11   readFileBooks(openFile(FILE_BOOK,"r"),l);
12 }
```

4.2.2 *tcp_server.c*

```
1  #include "tcp_server.h"
2
3  void bindNlisten(int port, AVL *l){
4    int sDesc;//socket descriptor
5    int conn; //nova conexao de um cliente
6    int started;//bind,listen
7    char str[32];
8    socklen_t len = sizeof(struct sockaddr);//evitar: error:
         conversion to non-scalar type requested
9    struct sockaddr_in *s = (struct sockaddr_in*)malloc(sizeof(struct
         sockaddr_in));//servidor
10   struct sockaddr_in *c = (struct sockaddr_in*)malloc(sizeof(struct
         sockaddr_in));//cliente
11   struct in_addr *addr = (struct in_addr *)malloc(sizeof(struct
         in_addr));
12   sDesc = socket(AF_INET,SOCK_STREAM,0);//IPv4 e TCP (sequencial,
         confiavel,2-way,byte-stream)
13   if(sDesc<0){
14     printError(ERROR_SOCKET);
15     exit(1);
```

```
16    }
17    memset(str,'\0',sizeof(str));
18    memset(s,'0',sizeof(s));
19    memset(addr,'0',sizeof(addr));//write zero-valued bytes
20    addr->s_addr = htons(INADDR_ANY);//bind all local interfaces
21    s->sin_family = AF_INET;//tcp
22    s->sin_port = htons(port);//convert to network byte order
23    s->sin_addr = *addr;
24    started = bind(sDesc,(struct sockaddr*)s,len);//assigns the
          address specified to by addr to the socket
25    if(started<0){
26      printError(ERROR_SOCKET_SERVER_BIND);
27      close(sDesc);//lembrar de fechar o socket aberto
28      exit(1);
29    }
30    started = listen(sDesc,TCP_MAX_CONN);
31    if(started<0){
32      printError(ERROR_SOCKET_SERVER_LISTEN);
33      close(sDesc);//lembrar de fechar o socket aberto
34      exit(1);
35    }
36    //  writeLog("TCP_SERVER::iniciado porta",my_itoa(port,10),"");
37    //conexao? wait
38    while(1){
39      conn=accept(sDesc,(struct sockaddr *)c,&len);
40      if(!fork()){
41        close(sDesc);
42        send_max_size(conn,l);
43        receive_msg(conn,l);
44      }
45      else{
46        close(conn);
47      }
48    };
49    close(sDesc);
50    free(s);
51    free(c);
52    free(addr);
53  }
54
55  void receive_msg(int conn, AVL *l){
56    char msg[TCP_BUF_SIZE];//msg transmit
57    int b;//byte received
58    bool transmit=false;
59    double ini;
60    int i=0;
61    struct timeval tvini; //apenas assim para pegar microsegundos
62    do{
63      b = recv(conn,msg,TCP_BUF_SIZE,0);
64      gettimeofday(&tvini, NULL);
65      msg[b] = '\0';
66      transmit=false;
67      for(i=0;i<b;i=i+3){
68        if(b-i >= 3){
69    if(msg[i]!='A' || msg[i+1]!='C' || msg[i+2] != 'K'){
70      transmit=false;
71      i=b;
72    }
73    else{
74      transmit=true;
75    }
76        }
```

```
77          else{
78    transmit=false;
79    i=b;
80          }
81        }
82        if(transmit){
83          for(i=3;i<TCP_BUF_SIZE;i++)
84    msg[i]='\0';
85        }
86        printf("%s\n",msg);
87        if(strcmp(msg,TCP_MSG_ACK)==0){
88          //do nothing
89          memset(msg,'\0',TCP_BUF_SIZE);
90        }
91        else if(strlen(msg)>5){
92          send_menu(conn,true,NULL);
93        }
94        else if(strcmp(msg,TCP_COMMAND_MENU)==0){
95          send_menu(conn,false,NULL);
96        }
97        else if (strcmp(msg,TCP_COMMAND_CLOSE_CONNECTION) == 0){
98          send(conn,TCP_MSG_BYE,strlen(TCP_MSG_BYE), 0);
99        }
100       else{
101         ini=DOUBLE_MILHAO*(double)(tvini.tv_sec)+(double)(tvini.
              tv_usec);
102         read_menu(conn,l,msg,ini);
103       }
104       transmit=true;
105   }while(transmit);
106   close(conn);
107 }
108
109 void send_menu(int conn,bool alert, TimeVal *tvend){
110   char *msg[TAM_MENU];
111   msg[0] = "*** MENU (tecle \\m para visualizar o menu) *********\n
          \0";
112   msg[1] = "*** (1) Listar ISBN                                  *\n
          \0";
113   msg[2] = "*** (2) Ver descricacao por ISBN (entrada: ISBN)     *\n
          \0";
114   msg[3] = "*** (3) Ver info. completa (entrada: ISBN)          *\n
          \0";
115   msg[4] = "*** (4) Ver info. completa (todos os livros)        *\n
          \0";
116   msg[5] = "*** (5) Alterar estoque (apenas adm)                *\n
          \0";
117   msg[6] = "*** (6) Ver quantidade em estoque (entrada: ISBN)   *\n
          \0";
118   msg[7] = "*** (\\q) Fechar conexao e sair                       *\n
          \0";
119   msg[8] = "************************************************\n
          \0";
120   int i=0,j=0;
121   int last=0,tam=0;
122   for(i=0;i<TAM_MENU;i++)
123     tam+=strlen(msg[i]);
124   if(alert)
125     tam+=strlen(TCP_MSG_COMMAND_NOT_FOUND);
126   char aux[tam];
127   memset(aux,'\0',tam+1);
128   if(alert){
```

```
129       strcat(aux,TCP_MSG_COMMAND_NOT_FOUND);
130       last=strlen(TCP_MSG_COMMAND_NOT_FOUND);
131     }
132     for(i=0;i<TAM_MENU;i++){
133       for(j=0;j<strlen(msg[i]);j++){
134         aux[last+j]=msg[i][j];
135       }
136       last+=j;
137     }
138     last=strlen(aux);
139     if(tvend != NULL)
140       gettimeofday(tvend, NULL);
141     send(conn,aux,last, 0);//chegara em ordem, tcp
142   }
143
144   bool read_menu(int conn, AVL *l, char opt[], double ini){
145     double end;
146     struct timeval tvend; //apenas assim para pegar microsegundos?
147     int i=0;
148     for(i=0;i<strlen(opt);i++){
149       if(strcmp(opt,TCP_MSG_ACK)==0)
150         return false;
151       if(!isdigit(opt[i])){
152         send_menu(conn,true,NULL);
153         return false;
154       }
155     }
156     i=atoi(opt);
157     switch(i){
158     case 1:
159       send_all_ids(conn,l,&tvend);
160       end=DOUBLE_MILHAO*(double)(tvend.tv_sec)+(double)(tvend.tv_usec
              );
161       writeDoubleToFile("./estat/tp_1","a+",end-ini);
162       return true;
163     case 2:
164       send_desc_byId(conn,l,&tvend);
165       end=DOUBLE_MILHAO*(double)(tvend.tv_sec)+(double)(tvend.tv_usec
              );
166       writeDoubleToFile("./estat/tp_2","a+",end-ini);
167       return true;
168     case 3:
169       send_book_info(conn,l,&tvend);
170       end=DOUBLE_MILHAO*(double)(tvend.tv_sec)+(double)(tvend.tv_usec
              );
171       writeDoubleToFile("./estat/tp_3","a+",end-ini);
172       return true;
173     case 4:
174       send_all_books_info(conn,l,&tvend);
175       end=DOUBLE_MILHAO*(double)(tvend.tv_sec)+(double)(tvend.tv_usec
              );
176       writeDoubleToFile("./estat/tp_4","a+",end-ini);
177       return true;
178     case 5:
179       edit_estoque(conn,l,&tvend);
180       end=DOUBLE_MILHAO*(double)(tvend.tv_sec)+(double)(tvend.tv_usec
              );
181       writeDoubleToFile("./estat/tp_5","a+",end-ini);
182       return true;
183     case 6:
184       send_estoque_byId(conn,l,&tvend);
```

```
185       end=DOUBLE_MILHAO*(double)(tvend.tv_sec)+(double)(tvend.tv_usec
              );
186       writeDoubleToFile("./estat/tp_6","a+",end-ini);
187       return true;
188     default:
189       send_menu(conn,true,&tvend);
190       end=DOUBLE_MILHAO*(double)(tvend.tv_sec)+(double)(tvend.tv_usec
              );
191       writeDoubleToFile("./estat/tp_menu","a+",end-ini);
192       return false;
193     }
194     return true;
195 }
196
197 void send_all_ids(int conn, AVL *l, TimeVal *tvend){
198     int len=totISBNchar(*l);
199     int tot=totElemAVL(*l);
200     char *del="\n ISBN: ";
201     char *ids=(char *)malloc(len+tot*strlen(del)+2);
202     memset(ids,'\0',len+tot*strlen(del)+2);
203     avlIdToStr(*l,&ids,del);
204     strcat(ids,"\n");
205     gettimeofday(tvend,NULL);
206     send(conn,ids,strlen(ids),0);
207     free(ids);
208 }
209
210 void send_desc_byId(int conn, AVL *l, TimeVal *tvend){
211     char msg[TCP_BUF_SIZE];//msg transmit
212     send(conn,TCP_MSG_ISBN_REQUIRED,strlen(TCP_MSG_ISBN_REQUIRED),0);
213     int b=recv(conn,msg,TCP_BUF_SIZE,0);//isbn?
214     msg[b] = '\0';
215   printf("%s\n",msg);
216     AVL_NO *livro=getAVLElemById(*l,msg);
217     if(livro!=NULL){
218       Livro *temp=(Livro *)livro->dado;
219       char str[strlen(temp->desc)+4];
220       memset(str,'\0',strlen(temp->desc)+4);
221       strcat(str,temp->desc);
222       strcat(str,"\n");
223       gettimeofday(tvend,NULL);
224       send(conn,str,strlen(str),0);
225     }
226     else{
227       gettimeofday(tvend,NULL);
228       send(conn,BOOK_NOT_FOUND,strlen(BOOK_NOT_FOUND),0);
229     }
230 }
231
232 void send_book_info(int conn, AVL *l, TimeVal *tvend){
233     char msg[TCP_BUF_SIZE];//msg transmit
234     send(conn,TCP_MSG_ISBN_REQUIRED,strlen(TCP_MSG_ISBN_REQUIRED),0);
235     int b=recv(conn,msg,TCP_BUF_SIZE,0);//isbn?
236     msg[b] = '\0';
237     AVL_NO *livro=getAVLElemById(*l,msg);
238   printf("%s\n",msg);
239     if(livro!=NULL){
240       Livro *temp=(Livro *)livro->dado;
241       char *str = bookNodeToStr(temp);
242       gettimeofday(tvend,NULL);
243       send(conn,str,strlen(str),0);
244     }
```

```
245     else{
246       gettimeofday(tvend,NULL);
247       send(conn,BOOK_NOT_FOUND,strlen(BOOK_NOT_FOUND),0);
248     }
249   }
250
251   void send_all_books_info(int conn, AVL *l, TimeVal *tvend){
252     int len=totAVLchar(*l);
253     int tot=totElemAVL(*l);
254     int bsize=getBookNodeSize();
255     char *del="\n";
256     int lentot=(len+tot*bsize+tot*strlen(del)*2+2);
257     char *allin=(char *)malloc(lentot);
258     memset(allin,'\0',lentot);
259     avlToStr(*l,&allin,del);
260     strcat(allin,"\n");
261     gettimeofday(tvend,NULL);
262     send(conn,allin,lentot,0);
263     free(allin);
264   }
265
266   void edit_estoque(int conn, AVL *l, TimeVal *tvend){
267     char msg[TCP_BUF_SIZE];//msg transmit
268     send(conn,TCP_MSG_ISBN_REQUIRED,strlen(TCP_MSG_ISBN_REQUIRED),0);
269     int b=recv(conn,msg,TCP_BUF_SIZE,0);//isbn?
270     msg[b] = '\0';
271    printf("%s\n",msg);
272     AVL_NO *livro=getAVLElemById(*l,msg);
273     if(livro!=NULL){
274       send(conn,TCP_MSG_LOGIN_REQUIRED,strlen(TCP_MSG_LOGIN_REQUIRED)
              ,0);
275       b=recv(conn,msg,TCP_BUF_SIZE,0);//senha?
276       msg[b] = '\0';
277    printf("%s\n",msg);
278       if(doLogin(msg)){
279         send(conn,TCP_MSG_QTD_REQUIRED,strlen(TCP_MSG_QTD_REQUIRED)
                ,0);
280         b=recv(conn,msg,TCP_BUF_SIZE,0);//senha?
281         msg[b] = '\0';
282    printf("%s\n",msg);
283         int q=atoi(msg);
284         if(q>=0){
285     Livro *aux = (Livro *)livro->dado;
286     aux->estoque=q;
287     livro->dado = (void *)aux;
288     gettimeofday(tvend,NULL);
289     send(conn,TCP_MSG_SUCESS_EDIT,strlen(TCP_MSG_SUCESS_EDIT),0);
290         }
291         else{
292     gettimeofday(tvend,NULL);
293     send(conn,TCP_MSG_NUM_NATURAL_REQUIRED,strlen(
            TCP_MSG_NUM_NATURAL_REQUIRED),0);
294         }
295       }
296       else{
297         gettimeofday(tvend,NULL);
298         send(conn,MSG_SENHA_INVALIDA,strlen(MSG_SENHA_INVALIDA),0);
299       }
300     }
301     else{
302       gettimeofday(tvend,NULL);
303       send(conn,BOOK_NOT_FOUND,strlen(BOOK_NOT_FOUND),0);
```

```
304     }
305 }
306
307 void send_estoque_byId(int conn, AVL *l, TimeVal *tvend){
308   char msg[TCP_BUF_SIZE]; //msg transmit
309   send(conn,TCP_MSG_ISBN_REQUIRED,strlen(TCP_MSG_ISBN_REQUIRED),0);
310   int b=recv(conn,msg,TCP_BUF_SIZE,0); //isbn?
311   msg[b] = '\0';
312   printf("%s\n",msg);
313   AVL_NO *livro=getAVLElemById(*l,msg);
314   if(livro!=NULL){
315     Livro *aux = (Livro *)livro->dado;
316     gettimeofday(tvend,NULL);
317     send(conn,my_itoa(aux->estoque,10),strlen(my_itoa(aux->estoque
          ,10)),0);
318   }
319   else{
320     gettimeofday(tvend,NULL);
321     send(conn,BOOK_NOT_FOUND,strlen(BOOK_NOT_FOUND),0);
322   }
323 }
324
325 void send_max_size(int conn, AVL *l){
326   int len=totAVLchar(*l);
327   int tot=totElemAVL(*l);
328   int bsize=getBookNodeSize();
329   char *del="\n";
330   int lentot=(len+tot*bsize+tot*strlen(del)*2+2);
331   char str[10];
332   memset(str,'\0',10);
333   memmove(str,my_itoa(lentot,10),strlen(my_itoa(lentot,10)));
334   send(conn,str,strlen(str),0);
335 }
```

4.2.3 *login.c*

```
1 #include "login.h"
2
3 bool doLogin(char *senha){
4   if(memcmp(senha,SENHA,strlen(senha))!=0)
5     return false;
6   return true;
7 }
```

4.3 Diretório *client*

4.3.1 *client.c*

```
1 #include "client.h"
2
3 void start_client(int argc, char *argv[]){
4   int socket;
5   struct in_addr *addr = (struct in_addr *)malloc(sizeof(struct
        in_addr));
6   addr->s_addr = inet_addr(argv[1]);
7   socket = conn(*addr,atoi(argv[2]));
8   if(argc==3)
9     send_msg(socket,*addr,atoi(argv[2]));
```

```
10    else if(argc==4 && strcmp(argv[3],OPT_TESTE)==0)
11      build_teste(socket,*addr,atoi(argv[2]));
12    close(socket);
13    free(addr);
14  }
```

4.3.2 *tcp_client.c*

```
1  #include "tcp_client.h"
2
3  int conn(struct in_addr addr, int port){
4    int sDesc;//socket descriptor
5    int conn;//conexao
6    struct sockaddr_in *s = (struct sockaddr_in*)malloc(sizeof(struct
         sockaddr_in));//servidor
7    socklen_t len = sizeof(struct sockaddr);//evitar: error:
         conversion to non-scalar type requested
8    sDesc = socket(AF_INET,SOCK_STREAM,0);//IPv4 e TCP (sequencial,
         confiavel,2-way,byte-stream)
9    if(sDesc<0){
10     printError(ERROR_SOCKET);
11     exit(1);
12   }
13   memset(s,0,sizeof(s));//write zero-valued bytes
14   s->sin_family = AF_INET;
15   s->sin_port = htons(port);//eh short nao long...nao funciova sem
         htons :(
16   s->sin_addr = addr;
17   //conexao
18   conn = connect(sDesc,(struct sockaddr *)s,len);
19   if(conn<0){
20     printError(ERROR_SOCKET_CLIENT);
21     close(sDesc);//lembrar de fechar o socket aberto
22     exit(1);
23   }
24   free(s);
25   return sDesc;
26 }
27
28 void send_msg(int sDesc,struct in_addr addr, int port){
29   char msgMaxTCPBuffSize[10];
30   memset(msgMaxTCPBuffSize,'\0',10);
31   int max = recv(sDesc,msgMaxTCPBuffSize,10,0);//-1 error, 0 closed
32   if(max<0){
33     printError((max==-1)?ERROR_SOCKET_SERVER_ERROR:
           ERROR_SOCKET_SERVER_CLOSED);
34     close(sDesc);
35     exit(1);
36   }
37   msgMaxTCPBuffSize[max]='\0';
38   send(sDesc,TCP_MSG_ACK,strlen(TCP_MSG_ACK),0);//send message
39   int tcpBuffSize=atoi(msgMaxTCPBuffSize);
40   char msg[TCP_BUF_SIZE];//msg a ser transmitida
41   memset(msg,'\0',TCP_BUF_SIZE);
42   char recvMsg[tcpBuffSize];//received msg
43   bool transmit=true;//var de saida
44   printf("Para sair tecle %s\n",TCP_COMMAND_CLOSE_CONNECTION);
45   int rc=0;
46   do{
47     if(strcmp(msg,"4")==0){
48       do{
```

```
49    max = recv(sDesc,(void *)recvMsg,tcpBuffSize,0);//-1 error, 0
          closed
50    if(max<0){
51      printError((max==-1)?ERROR_SOCKET_SERVER_ERROR:
            ERROR_SOCKET_SERVER_CLOSED);
52      close(sDesc);
53      exit(1);
54    }
55    recvMsg[max]='\0';
56    printf("%s",recvMsg);
57    if(rc<tcpBuffSize)
58      send(sDesc,TCP_MSG_ACK,strlen(TCP_MSG_ACK),0);//send message
59    rc+=max;
60        }while(rc<tcpBuffSize);
61      }
62      else{
63        if(strlen(msg)==0)
64    send(sDesc,TCP_COMMAND_MENU,strlen(TCP_COMMAND_MENU),0);
65        max = recv(sDesc,(void *)recvMsg,tcpBuffSize,0);//-1 error, 0
              closed
66        recvMsg[max]='\0';
67        printf("%s",recvMsg);
68      }
69      rc=0;
70      if(strcmp(TCP_COMMAND_CLOSE_CONNECTION,msg)==0){//saida?
71        transmit=false;//nao transmitir mais
72      }
73      else{
74        do{
75    fgets(msg,TCP_BUF_SIZE-1,stdin);//lendo input
76    msg[strlen(msg)-strlen(CHAR_NEW_LINE)]='\0';//remove new line
77    if(strlen(msg)==0)
78      printf("Digite um comando valido!\n");
79        }while(strlen(msg)==0);
80        //temos que transmitir a msg de fechar conexao
81        max = send(sDesc,msg,strlen(msg),0);//send message
82        if(max<0){
83    close(sDesc);//close socket
84    if(max==-1){
85      exit(1);
86      sDesc = conn(addr,port);
87    }
88    else{
89      printError(ERROR_SOCKET_SERVER_ERROR);
90      exit(1);
91    }
92        }
93      }
94    }while(transmit);
95  }
96
97  void build_teste(int sDesc,struct in_addr addr, int port){
98    char msgMaxTCPBuffSize[10];
99    memset(msgMaxTCPBuffSize,'\0',10);
100   int max = recv(sDesc,msgMaxTCPBuffSize,10,0);//-1 error, 0 closed
101   if(max<0){
102     printError((max==-1)?ERROR_SOCKET_SERVER_ERROR:
            ERROR_SOCKET_SERVER_CLOSED);
103     close(sDesc);
104     exit(1);
105   }
106   msgMaxTCPBuffSize[max]='\0';
```

```
107     send(sDesc,TCP_COMMAND_MENU, strlen(TCP_COMMAND_MENU),0);//send
            message
108     int tcpBuffSize=atoi(msgMaxTCPBuffSize);
109     char recvMsg[tcpBuffSize];//received msg
110     struct timeval tvini, tvend;
111     double ini=00.00,end=00.00;
112     int *msgT=getTesteMsg();
113     char testes[7][18];
114     strcpy(testes[0],"./estat/tt_menu\0");
115     strcpy(testes[1],"./estat/tt_1\0");
116     strcpy(testes[2],"./estat/tt_2\0");
117     strcpy(testes[3],"./estat/tt_3\0");
118     strcpy(testes[4],"./estat/tt_4\0");
119     strcpy(testes[5],"./estat/tt_5\0");
120     strcpy(testes[6],"./estat/tt_6\0");
121     char isbn[14][13];
122     strcpy(isbn[0],"978853526-0\0");
123     strcpy(isbn[1],"978052007-4\0");
124     strcpy(isbn[2],"857302773-8\0");
125     strcpy(isbn[3],"850109104-9\0");
126     strcpy(isbn[4],"843760494-X\0");
127     strcpy(isbn[5],"207036002-4\0");
128     strcpy(isbn[6],"081297215-5\0");
129     strcpy(isbn[7],"978858041-1\0");//errado
130     //  strcpy(isbn[7],"850197213-4");//errado
131     strcpy(isbn[8],"978858041-1\0");
132     strcpy(isbn[9],"978857679-9\0");
133     strcpy(isbn[10],"978857608-5\0");
134     strcpy(isbn[11],"978853990-1\0");
135     strcpy(isbn[12],"978857521-2\0");
136     strcpy(isbn[13],"978850220-0\0");
137     int i=0,random=0;
138     int rc=0;
139     max = recv(sDesc,recvMsg,tcpBuffSize,0);//menu
140     recvMsg[max]='\0';
141     printf("%s",recvMsg);
142     for(i=0;i<NUM_TESTES*NUM_OPCOES_MENU;i++){
143       rc=0;
144       gettimeofday(&tvini, NULL);//marca envio
145       printf("%d",msgT[i]);
146       max = send(sDesc,my_itoa(msgT[i],10),strlen(my_itoa(msgT[i],10)
              ),0);
147       if(msgT[i]==2 || msgT[i]==3 || msgT[i]==5 || msgT[i]==6){//
              precisa de isbn
148         max = recv(sDesc,recvMsg,tcpBuffSize,0);
149         recvMsg[max]='\0';
150         printf("%s",recvMsg);
151         if(strcmp(recvMsg,TCP_MSG_ISBN_REQUIRED)==0){//caso receba
              req. isbn
152     random=rand()%13;
153     max = send(sDesc,isbn[random],strlen(isbn[random]),0);
154     printf(" SEND %s+ ",isbn[random]);
155     if(msgT[i]==5){//precisa de senha && qntidade
156       max = recv(sDesc,recvMsg,tcpBuffSize,0);
157       recvMsg[max]='\0';
158       printf("%s",recvMsg);
159       if(strcmp(recvMsg,TCP_MSG_LOGIN_REQUIRED)==0){//caso receba req
              . login
160         max = send(sDesc,SENHA,strlen(SENHA),0);
161         printf(" SEND %s+ ",SENHA);
162         max = recv(sDesc,recvMsg,tcpBuffSize,0);
163         recvMsg[max]='\0';
```

```
164         printf("%s",recvMsg);
165         if(strcmp(recvMsg,TCP_MSG_QTD_REQUIRED)==0){//qntidade
166           random=rand()%1000;
167           max = send(sDesc,my_itoa(random,10),strlen(my_itoa(random
                ,10)),0);
168           printf(" SEND %s+ ",my_itoa(random,10));
169           max = recv(sDesc,recvMsg,tcpBuffSize,0);//opcao
170           recvMsg[max]='\0';
171         }//qntd
172       }//ogin
173     }
174     else{
175       max = recv(sDesc,recvMsg,tcpBuffSize,0);//opcao
176       recvMsg[max]='\0';
177     }
178         }//isbn
179         printf("@%s@",recvMsg);
180       }
181       else if(msgT[i]==4){
182         do{
183     max = recv(sDesc,(void *)recvMsg,tcpBuffSize,0);//-1 error, 0
          closed
184     if(max<0){
185       printError((max==-1)?ERROR_SOCKET_SERVER_ERROR:
            ERROR_SOCKET_SERVER_CLOSED);
186         close(sDesc);
187         exit(1);
188     }
189     recvMsg[max]='\0';
190     printf("@%s@",recvMsg);
191     if(rc==0)
192       send(sDesc,TCP_MSG_ACK,strlen(TCP_MSG_ACK),0);//send message
193     rc+=max;
194         }while(rc<tcpBuffSize);
195         }
196       else{
197         max = recv(sDesc,(void *)recvMsg,tcpBuffSize,0);//-1 error, 0
              closed
198         recvMsg[max]='\0';
199         printf("@%s@",recvMsg);
200       }
201       gettimeofday(&tvend, NULL);//marca recebimento
202       ini=DOUBLE_MILHAO*(double)(tvini.tv_sec)+(double)(tvini.tv_usec
            );
203       end=DOUBLE_MILHAO*(double)(tvend.tv_sec)+(double)(tvend.tv_usec
            );
204       writeDoubleToFile(testes[msgT[i]],"a+",end-ini);
205     }
206 }
207 int *getTesteMsg(){
208     int *msg=(int *)malloc(sizeof(int)*NUM_OPCOES_MENU*NUM_TESTES);
209     int i=0,j=0,c=0;
210     for(i=0;i<NUM_TESTES;i++){
211       for(j=0;j<NUM_OPCOES_MENU;j++){
212         *(msg+c)=j+1;
213         c++;
214       }
215     }
216     return msg;
217 }
```

5 Conclusão

Nota-se que as opções 1, 2, 3 e 6 têm o tempo de comunicação parecidos, uma vez que a quantidade de dados transmitida não difere muito um do outro. Já a opção 4 e 5 possuem um alto tempo de comunicação (consequentemente alto tempo total) devido, respectivamente, ao grande volume de dados transmitidos e a quantidade de mensagens de comunicação (o menu 5 requer ISBN, senha e quantidade em estoque).