Servidor iterativo UDP

Leandro Kümmel Tria Mendes RA033910
Fernando Teixeira RA085858

8 de maio de 2013

Sumário

Lista de Figuras

Lista de Tabelas

1 Introdução

O objetivo desse projeto é implementar um sistema cliente/servidor iterativo, com operações para o gerenciamento de livros em uma livraria. Na comunicação entre cliente e servidor utilizou-se o protocolo UDP[1], da camada de transporte, e a partir da execução de testes podemos avaliar alguns aspectos desse protocolo e posteriormente compará-lo com o projeto anterior, no qual foi utilizado o TCP como protocolo.

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 UDP - User Datagram Protocol

Diferentemente do protocolo TCP, o UDP é simplificado, ou seja, possui características diferentes do primeiro protocolo, logo foi escrito de modo a não garantir que os dados enviados (pelo servido) e recebidos (pelo cliente) de forma correta, na sequência adequada, pela rede. O protocolo UDP, User Datagram Protocol, não envia dados em s̈treamsim em d̈atagrams[2]:. No primeiro caso, a comunicação têm dois pontos, o dado é colocado em um final e vêem do outro ponto, não há dados duplicados, descartados, ou reorganizados. Já o d̈atagramss̈ão bem menores do que o primeiro, há uma fonte e destino para envio dos dados, mas não pode ser chamada de conexão, um d̈atagramn̈ão possui relacionamento com nenhum outro, sendo assim não há garantia de entrega do pacote, porém arregura-se que os pacotes não serão enviados fragmentados.
As características fundamentais do UDP são:

- *Orientado à transação*: não há necessidade de uma conexão.
- *Simples*: utiliza-se de datagramas, sem confiabilidade de entrega.
- *Stateless*: não mantém o estado sobre uma conexão, visto que não existe uma.

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[3] para instruções adicionais. Há também um Makefile, o qual compila e executa 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.

[1] Mais informações sobre UDP http://tools.ietf.org/html/rfc768
[2] Fonte: http://www.freesoft.org/CIE/Course/Section3/5.htm
[3] Leia esse arquivo antes de executar o sistema

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

III. *client*: Funções que provêm comunicação 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*

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 error.h*: Gerencia erros que eventualmente podem ocorrer.

II. *common.c common.h*: Funções de uso comum.

III. *avl.c avl.h*: Gerencia a estrutura básica da livraria, utiliza-se árvore AVL[4], 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 archives.h*: Manipula arquivos. Faz a leitura do arquivo da livraria[5].

V. *tcp.c tcp.h*: Contém apenas algumas constantes, utilizadas tanto pelo protocolo TCP quanto pelo UDP.

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

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

VIII. *livros/livros*: Arquivo contendo os livros[6].

2.2.2 Conexão Servidor/Cliente

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

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

II. *udp_server.c udp_server.h*: Gerencia a comunicação com o cliente, em outras palavras, o udp_server.c recebe um datagram do udp_client.c[II] e envia uma resposta adequada ao mesmo. Como mencionado anteriormente, o UDP não garante a entrega de um datagrama, sendo assim, espera de um pacote, que é uma chamada do sistema (syscall)[7], é bloqueante. Portanto, garante-se o envio/recebimento de pelo menos 1 byte para o cliente/servidor, previnindo um bloqueio por ambas as partes envolvidas.

[4] http://pages.cs.wisc.edu/~ealexand/cs367/NOTES/AVL-Trees/index.html
[5] Ver README.md para mais detalhes do arquivo da livraria
[6] Ver README.md para mais detalhes do arquivo da livraria
[7] Mais em: http://www.tldp.org/LDP/khg/HyperNews/get/syscall/syscall86.html

III. *login.c 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. *udp_client.c udp_client.h*: Gerencia leitura, da entrada dada pelo usuário do sistema, e a comunicação entre hospedeiro e cliente. Também, garante-se o envio de pelo menos 1 byte para o servidor [II].

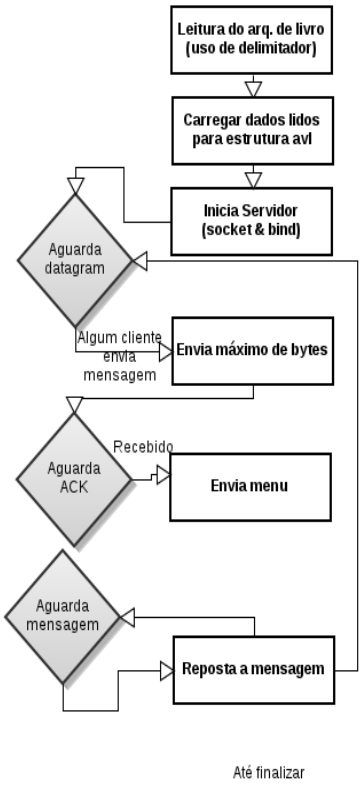
2.3 Coleta e gerência de dados para testes

Para realizar os testes implementou-se alguns arquivos adicionas[VII]. Uma constante, denominada NUM_TESTES[8] , 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].

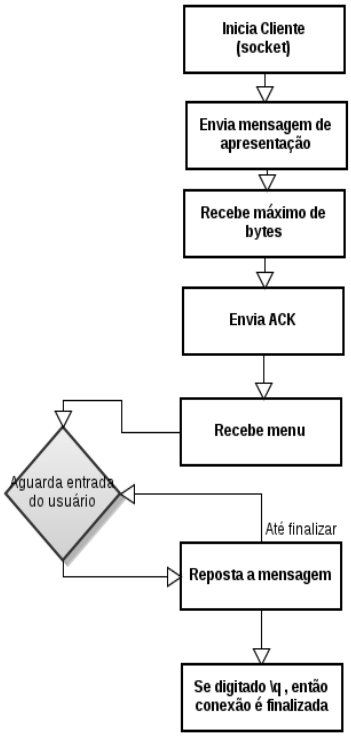
2.4 Vantagens da implementação

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 tem melhor complexidade de tempo com relação a outras estruturas, além de ser da implementação e manuntenção serem simples.
Com relação a comunicação entre cliente/servidor, vale ressaltar que há uma troca de mensagens[9] inicial entre os dois, na qual o cliente inicia, enviando um pacote apenas para conhecimento do servidor de sua existência, já o servidor responde com um pacote cujo o conteúdo é o número de bytes da maior mensagem possível a ser enviada pelo servidor, seguido de ACK (reconhecimento), enviado pelo cliente. Como a função $recvfrom(intsockfd, void * buf, size_tlen, intflags, structsockaddr * src_addr, $ começa a ler o buffer, o qual é escrito o datagram enviado pelo servidor, antes do mesmo estar com 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, junto ao envio de mensagens de controle (ACK[11]), afim de ler o buffer apenas quando o mesmo estiver completo.[12]

[8] Nesse sistema consideramos NUM_TESTES igual a 100
[9] Sabe-se que o UDP envia/recebe datagrams
[10] http://linux.die.net/man/2/recvfrom
[11] Veja tcp.h[V]
[12] The receive calls normally return any data available, up to the requested amount, rather than waiting for receipt of the full amount requested.



(a) Fluxograma Servidor



(b) Fluxograma Cliente

Figura 1: Fluxogramas

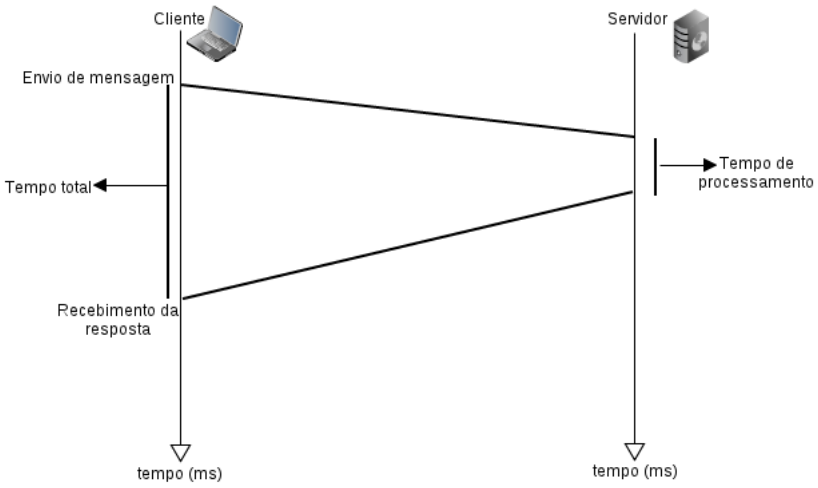


Figura 2: Definição do cálculo dos tempos

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.
Abaixo demostramos o traceroute, o qual mostra a rota percorrida por um pequeno pacote, marcando assim os roteadores por qual passa. Vale ressaltar que os teste foram realizados em uma rede sem fio, sendo assim há uma perda maior de pacotes, e quanto maior a rota a ser percorrida, também espera-se uma perda maior.

```
traceroute to 64.90.43.176 (64.90.43.176), 30 hops max, 60 byte packets
 1  c9528601.virtua.com.br (201.82.134.1)  26.304 ms  26.319 ms
30.156 ms
 2  c952005e.virtua.com.br (201.82.0.94)  20.162 ms  20.168 ms
20.187 ms
 3  embratel-T0-4-0-0-uacc01.cas.embratel.net.br (200.213.139.1)
19.337 ms  20.442 ms  25.724 ms
 4  ebt-T0-8-0-0-tcore01.cas.embratel.net.br (200.230.162.22)
38.070 ms ebt-T0-8-0-1-tcore02.cas.embratel.net.br (200.230.162.42)
34.997 ms ebt-T0-8-0-3-tcore02.cas.embratel.net.br (200.230.162.54)
35.304 ms
 5  ebt-Bundle-POS11932-intl03.mianap.embratel.net.br (200.230.220.54)
146.580 ms ebt-Bundle-POS11931-intl03.mianap.embratel.net.br (200.230.220.2)
149.048 ms  156.461 ms
 6  ae99.edge4.Miami1.Level3.net (4.59.90.9)  153.509 ms
134.693 ms  134.205 ms
 7  ae-2-52.edge2.Miami2.Level3.net (4.69.138.109)  140.293 ms ae-1-51.edge2.M
140.607 ms  142.033 ms
 8  ge-2-0-0.mpr1.mia1.us.above.net (64.125.14.89)  125.945 ms
125.854 ms  128.335 ms
 9  ge-1-0-0.mpr2.mia1.us.above.net (64.125.30.194)  135.120 ms
135.575 ms  135.136 ms
10  xe-4-0-0.cr2.iah1.us.above.net (64.125.30.202)  167.099 ms
161.208 ms  161.525 ms
11  xe-2-0-0.cr2.lax112.us.above.net (64.125.25.18)  190.919 ms
197.641 ms  197.113 ms
12  xe-3-2-0.mpr1.lax12.us.above.net (64.125.21.126)  200.062 ms
200.292 ms  200.553 ms
13  xe-0-1-0.mpr1.lax103.us.above.net (64.125.30.45)  204.362 ms
225.417 ms  225.202 ms
14  64.125.187.174 (64.125.187.174)  196.712 ms  200.319 ms
200.532 ms
15  ip-66-33-201-126.dreamhost.com (66.33.201.126)  199.449 ms
194.863 ms  195.811 ms
```

16 ps59582.dreamhost.com (64.90.43.176) 194.074 ms 192.464 ms 193.108 ms

3.1 Tabelas e gráficos

3.1.1 Tempo total

[2.4] Representaremos todas as 100 medidas das 6 opções do menu[II]

Opção 1 [ms]	Opção 2 [ms]	Opção 3 [ms]	Opção 4 [ms]	Opção 5 [ms]	Opção 6 [ms]
202832.000000	401895.000000	401923.000000	234172.000000	798799.000000	395717.000000
200577.000000	411700.000000	395513.000000	236042.000000	809658.000000	394135.000000
197890.000000	397754.000000	409410.000000	245335.000000	806619.000000	442529.000000
199016.000000	405479.000000	407760.000000	232236.000000	807654.000000	400506.000000
200978.000000	397303.000000	399229.000000	249373.000000	841950.000000	397816.000000
198565.000000	423318.000000	424794.000000	691800.000000	804294.000000	399382.000000
201203.000000	395594.000000	397743.000000	208758.000000	794001.000000	407256.000000
208996.000000	394777.000000	412663.000000	207192.000000	803674.000000	391731.000000
195891.000000	394066.000000	410704.000000	209402.000000	823421.000000	397586.000000
196933.000000	395442.000000	396442.000000	227592.000000	836823.000000	404601.000000
197299.000000	425744.000000	423432.000000	219610.000000	812663.000000	400127.000000
209918.000000	420353.000000	397797.000000	210569.000000	827006.000000	399239.000000
196991.000000	406967.000000	407510.000000	210354.000000	802566.000000	407265.000000
197843.000000	394869.000000	421449.000000	221175.000000	200806.000000	197058.000000
2590.000000	195847.000000	197219.000000	208901.000000	791945.000000	393372.000000
199915.000000	401820.000000	395579.000000	231461.000000	810600.000000	409742.000000
197283.000000	405074.000000	395842.000000	210949.000000	829574.000000	399366.000000
199574.000000	396137.000000	395147.000000	207355.000000	819217.000000	395139.000000
197009.000000	400958.000000	405829.000000	226757.000000	806333.000000	414520.000000
196346.000000	395740.000000	397434.000000	212092.000000	790672.000000	420302.000000
199322.000000	395872.000000	417248.000000	209784.000000	815474.000000	406576.000000
228425.000000	395555.000000	396113.000000	209417.000000	860111.000000	407349.000000
197004.000000	424530.000000	445338.000000	230618.000000	796783.000000	396708.000000
222268.000000	420320.000000	400065.000000	232055.000000	875436.000000	398939.000000
196787.000000	393866.000000	405984.000000	211292.000000	903562.000000	397748.000000
198759.000000	421511.000000	394275.000000	735942.000000	799664.000000	392206.000000
207760.000000	392938.000000	395634.000000	206862.000000	805160.000000	400198.000000
201927.000000	399641.000000	396739.000000	236163.000000	877402.000000	413885.000000
226864.000000	409388.000000	395659.000000	209622.000000	802668.000000	394006.000000
209182.000000	412212.000000	396163.000000	210137.000000	793100.000000	409723.000000
197275.000000	396009.000000	391355.000000	208986.000000	819994.000000	404876.000000
198765.000000	400180.000000	396554.000000	219102.000000	790226.000000	398218.000000
200981.000000	404242.000000	399161.000000	460852.000000	789644.000000	393832.000000
198851.000000	399854.000000	397681.000000	208074.000000	803754.000000	395554.000000
210053.000000	403871.000000	392258.000000	209026.000000	794327.000000	394816.000000
197079.000000	398196.000000	407129.000000	208006.000000	793979.000000	395824.000000
317545.000000	408533.000000	418540.000000	209234.000000	810330.000000	405933.000000
196593.000000	400132.000000	406525.000000	211312.000000	841225.000000	418119.000000
195985.000000	401293.000000	395232.000000	208309.000000	818517.000000	405715.000000
196696.000000	418305.000000	438534.000000	206753.000000	799198.000000	392914.000000
477537.000000	395930.000000	422129.000000	208685.000000	789596.000000	392640.000000
201875.000000	395420.000000	395335.000000	207329.000000	787954.000000	407289.000000
208248.000000	404590.000000	392724.000000	208663.000000	821004.000000	393146.000000
196566.000000	403319.000000	395652.000000	230995.000000	799028.000000	437443.000000
195428.000000	422080.000000	394990.000000	208724.000000	785814.000000	394490.000000
198004.000000	393521.000000	411019.000000	217521.000000	1103251.000000	394770.000000
210409.000000	400877.000000	391995.000000	222448.000000	808306.000000	396633.000000
198306.000000	393314.000000	411294.000000	210882.000000	798444.000000	431715.000000
195692.000000	402193.000000	403638.000000	212579.000000	840483.000000	391955.000000
210429.000000	397898.000000	397369.000000	209518.000000	805219.000000	394749.000000
309997.000000	398195.000000	499914.000000	207100.000000	812946.000000	395220.000000
198184.000000	404579.000000	400516.000000	207691.000000	1204824.000000	416681.000000
205255.000000	393504.000000	393160.000000	211066.000000	803446.000000	415857.000000
196671.000000	409958.000000	409993.000000	209950.000000	785016.000000	396079.000000
197723.000000	396650.000000	408344.000000	208124.000000	801234.000000	411404.000000
198394.000000	395366.000000	398467.000000	216167.000000	810304.000000	391763.000000
197541.000000	396780.000000	397229.000000	220454.000000	787603.000000	400430.000000
199190.000000	393784.000000	427397.000000	210189.000000	816566.000000	392351.000000
198696.000000	433477.000000	395418.000000	210997.000000	803445.000000	398087.000000
197247.000000	407497.000000	411030.000000	208865.000000	822669.000000	401375.000000
198394.000000	427324.000000	393047.000000	221262.000000	789270.000000	403826.000000
209086.000000	418372.000000	393062.000000	207894.000000	788115.000000	392377.000000
199553.000000	392601.000000	405785.000000	220162.000000	785006.000000	394483.000000
198319.000000	423538.000000	410630.000000	207672.000000	808844.000000	395766.000000
196928.000000	403372.000000	393977.000000	207054.000000	796992.000000	422656.000000
196984.000000	394484.000000	396610.000000	209251.000000	824105.000000	394838.000000
196379.000000	479179.000000	393775.000000	211731.000000	788755.000000	406708.000000
196700.000000	395804.000000	396998.000000	210070.000000	794453.000000	416839.000000
207862.000000	398274.000000	394005.000000	232721.000000	811171.000000	408921.000000
196386.000000	395015.000000	400134.000000	211076.000000	799804.000000	397086.000000
196359.000000	426186.000000	394662.000000	207370.000000	799161.000000	406781.000000
197339.000000	399053.000000	393264.000000	222663.000000	787083.000000	420077.000000
197826.000000	399433.000000	418140.000000	206920.000000	789392.000000	393826.000000
196807.000000	397106.000000	399835.000000	209258.000000	795363.000000	392588.000000
205368.000000	393935.000000	402295.000000	206627.000000	788889.000000	394654.000000
198284.000000	398184.000000	395083.000000	207608.000000	801048.000000	393998.000000
196398.000000	406833.000000	395495.000000	218598.000000	848622.000000	405460.000000
207105.000000	409442.000000	406023.000000	208051.000000	820199.000000	393349.000000
197894.000000	413559.000000	419352.000000	219892.000000	796829.000000	405932.000000
198012.000000	397309.000000	395554.000000	216077.000000	802407.000000	396744.000000
225448.000000	395045.000000	399749.000000	210813.000000	813252.000000	393264.000000
208394.000000	420528.000000	403406.000000	211668.000000	795081.000000	400019.000000
198122.000000	399510.000000	394877.000000	223188.000000	818911.000000	394356.000000
197896.000000	396816.000000	396399.000000	209412.000000	829892.000000	396114.000000
205248.000000	393911.000000	398155.000000	210356.000000	803175.000000	391861.000000
196900.000000	398436.000000	394541.000000	212620.000000	801467.000000	405378.000000
222518.000000	393479.000000	408412.000000	210523.000000	785462.000000	398446.000000
196907.000000	399364.000000	417189.000000	210125.000000	820895.000000	393849.000000
197060.000000	393267.000000	406402.000000	235191.000000	880082.000000	405367.000000
197375.000000	419736.000000	431079.000000	325715.000000	791788.000000	493771.000000
374847.000000	395960.000000	458862.000000	210810.000000	788466.000000	401362.000000
200859.000000	396082.000000	402192.000000	229758.000000	817210.000000	398966.000000
211868.000000	393887.000000	394204.000000	208042.000000	828234.000000	398370.000000
208191.000000	393185.000000	417826.000000	208639.000000	788479.000000	400850.000000
199423.000000	426956.000000	396872.000000	207154.000000	791639.000000	393806.000000
211505.000000	400356.000000	416333.000000	209921.000000	836845.000000	407091.000000
197465.000000	406879.000000	408341.000000	232825.000000	786796.000000	407891.000000
195921.000000	399492.000000	396484.000000	208601.000000	802906.000000	394378.000000
199047.000000	400612.000000	433656.000000	211621.000000	788881.000000	418474.000000
201128.000000	394118.000000	397159.000000	209717.000000	801990.000000	394485.000000

Tabela 1: Tabela de tempo total

3.1.2 Tempo de processamento

[2.4] Representaremos todas as 100 medidas das 6 opções do menu[II]

Opção 1 [ms]	Opção 2 [ms]	Opção 3 [ms]	Opção 4 [ms]	Opção 5 [ms]	Opção 6 [ms]
58.000000	198042.000000	199018.000000	531.000000	599021.000000	395717.000000
33.000000	215044.000000	196152.000000	544.000000	597069.000000	394135.000000
33.000000	200049.000000	211856.000000	592.000000	596651.000000	442529.000000
33.000000	196092.000000	198068.000000	546.000000	608380.000000	400506.000000
33.000000	198288.000000	196847.000000	527.000000	645179.000000	397816.000000
33.000000	215716.000000	200255.000000	517.000000	604480.000000	399382.000000
33.000000	198679.000000	196386.000000	528.000000	596469.000000	407256.000000
33.000000	197720.000000	196773.000000	532.000000	593139.000000	391731.000000
32.000000	197664.000000	210059.000000	565.000000	622831.000000	397586.000000
32.000000	196762.000000	200068.000000	521.000000	609080.000000	404601.000000
32.000000	225444.000000	196773.000000	557.000000	600929.000000	400127.000000
33.000000	223279.000000	196086.000000	529.000000	609673.000000	399239.000000
31.000000	210538.000000	198013.000000	543.000000	603728.000000	407265.000000
33.000000	196631.000000	208305.000000	523.000000	37.000000	197058.000000
33.000000	196231.000000	197409.000000	508.000000	593579.000000	393372.000000
33.000000	201042.000000	198666.000000	524.000000	611140.000000	409742.000000
31.000000	196184.000000	196007.000000	537.000000	632171.000000	399366.000000
32.000000	199895.000000	196008.000000	551.000000	619963.000000	395139.000000
32.000000	196414.000000	198521.000000	522.000000	608667.000000	414520.000000
30.000000	198629.000000	208223.000000	551.000000	592819.000000	420302.000000
31.000000	196019.000000	199216.000000	511.000000	617614.000000	406576.000000
34.000000	198593.000000	214698.000000	525.000000	634116.000000	407349.000000
32.000000	198577.000000	200535.000000	518.000000	597235.000000	396708.000000
29.000000	210601.000000	208583.000000	535.000000	674969.000000	398939.000000
33.000000	197583.000000	196661.000000	535.000000	639392.000000	397748.000000
34.000000	225099.000000	197385.000000	525.000000	602759.000000	392206.000000
33.000000	196394.000000	199932.000000	527.000000	591312.000000	400198.000000
33.000000	200000.000000	198117.000000	606.000000	623669.000000	413885.000000
34.000000	197393.000000	195965.000000	519.000000	602347.000000	394006.000000
30.000000	197203.000000	196084.000000	545.000000	595414.000000	409723.000000
33.000000	197907.000000	197038.000000	521.000000	623271.000000	404876.000000
34.000000	201833.000000	197440.000000	543.000000	593077.000000	398218.000000
32.000000	206728.000000	197711.000000	525.000000	591382.000000	393832.000000
33.000000	198341.000000	196461.000000	533.000000	605964.000000	395554.000000
32.000000	208486.000000	197682.000000	524.000000	596095.000000	394816.000000
34.000000	198261.000000	196019.000000	520.000000	594988.000000	395824.000000
30.000000	208752.000000	209338.000000	558.000000	613790.000000	405933.000000
33.000000	197511.000000	197746.000000	526.000000	615620.000000	418119.000000
33.000000	195510.000000	213761.000000	545.000000	605788.000000	405715.000000
31.000000	221724.000000	209651.000000	521.000000	601843.000000	392914.000000
29.000000	196344.000000	197252.000000	525.000000	590716.000000	392640.000000
33.000000	196247.000000	195987.000000	576.000000	591048.000000	407289.000000
31.000000	197110.000000	200118.000000	527.000000	622313.000000	393146.000000
33.000000	200073.000000	197042.000000	525.000000	601823.000000	437443.000000
32.000000	199565.000000	211388.000000	526.000000	588613.000000	394490.000000
33.000000	196443.000000	196087.000000	531.000000	603269.000000	394770.000000
33.000000	198605.000000	196079.000000	523.000000	606256.000000	396633.000000
33.000000	196258.000000	204921.000000	538.000000	599731.000000	431715.000000
34.000000	202819.000000	198384.000000	526.000000	618113.000000	391955.000000
33.000000	196168.000000	210514.000000	524.000000	594883.000000	394749.000000
29.000000	197111.000000	202911.000000	573.000000	610388.000000	395220.000000
33.000000	208072.000000	196889.000000	528.000000	604220.000000	416681.000000
32.000000	196004.000000	197127.000000	529.000000	604178.000000	415857.000000
34.000000	200090.000000	210269.000000	541.000000	587813.000000	396079.000000
33.000000	196297.000000	197762.000000	540.000000	604600.000000	411404.000000
34.000000	197270.000000	195426.000000	529.000000	614322.000000	391763.000000
33.000000	199067.000000	228549.000000	523.000000	590497.000000	400430.000000
33.000000	198198.000000	199102.000000	521.000000	618878.000000	392351.000000
33.000000	221697.000000	196512.000000	554.000000	604097.000000	398087.000000
32.000000	196384.000000	196634.000000	537.000000	612083.000000	401375.000000
32.000000	198285.000000	197030.000000	531.000000	592142.000000	403826.000000
33.000000	222022.000000	209694.000000	522.000000	591661.000000	392377.000000
33.000000	195879.000000	198261.000000	525.000000	588680.000000	394483.000000
32.000000	224347.000000	196964.000000	518.000000	600132.000000	395766.000000
33.000000	199990.000000	196288.000000	533.000000	596438.000000	422656.000000
32.000000	197416.000000	196363.000000	522.000000	612789.000000	394838.000000
33.000000	269869.000000	198679.000000	571.000000	591667.000000	406708.000000
32.000000	198621.000000	196052.000000	522.000000	595283.000000	416839.000000
32.000000	198222.000000	196277.000000	520.000000	598718.000000	408921.000000
32.000000	198532.000000	196251.000000	521.000000	603682.000000	397086.000000
33.000000	224047.000000	195902.000000	535.000000	601264.000000	406781.000000
33.000000	195933.000000	208126.000000	544.000000	589555.000000	420077.000000
32.000000	201441.000000	197057.000000	526.000000	591825.000000	393826.000000
44.000000	196043.000000	197769.000000	542.000000	596538.000000	392588.000000
32.000000	196525.000000	196548.000000	527.000000	590730.000000	394654.000000
33.000000	197524.000000	195985.000000	524.000000	603607.000000	393998.000000
34.000000	209085.000000	209388.000000	523.000000	638848.000000	405460.000000
33.000000	209780.000000	195877.000000	597.000000	623498.000000	393349.000000
33.000000	196307.000000	197731.000000	534.000000	594363.000000	405932.000000
32.000000	198145.000000	197190.000000	520.000000	603702.000000	396744.000000
31.000000	197541.000000	196051.000000	620.000000	590847.000000	393264.000000
33.000000	221088.000000	197690.000000	545.000000	597114.000000	400019.000000
30.000000	201292.000000	196614.000000	525.000000	622549.000000	394356.000000
33.000000	198166.000000	200117.000000	531.000000	633404.000000	396114.000000
33.000000	196217.000000	197429.000000	570.000000	606300.000000	391861.000000
33.000000	198533.000000	208146.000000	526.000000	603045.000000	405378.000000
33.000000	196721.000000	195806.000000	527.000000	588940.000000	398446.000000
32.000000	199989.000000	197510.000000	530.000000	611671.000000	393849.000000
33.000000	196586.000000	232169.000000	534.000000	656005.000000	405367.000000
33.000000	198657.000000	197553.000000	635.000000	591322.000000	493771.000000
29.000000	196426.000000	198508.000000	521.000000	590573.000000	401362.000000
32.000000	198276.000000	196910.000000	581.000000	616471.000000	398966.000000
33.000000	196469.000000	197516.000000	535.000000	623102.000000	398370.000000
33.000000	195974.000000	197874.000000	549.000000	591225.000000	400850.000000
32.000000	200176.000000	200762.000000	576.000000	592113.000000	393806.000000
32.000000	199242.000000	196759.000000	524.000000	636212.000000	407091.000000
33.000000	197076.000000	196880.000000	532.000000	589738.000000	407891.000000
34.000000	200312.000000	205353.000000	525.000000	592057.000000	394378.000000
34.000000	197097.000000	198623.000000	529.000000	591334.000000	418474.000000
33.000000	196077.000000		525.000000	603083.000000	394485.000000

Tabela 2: Tabela de tempo de processamento

3.2 Médias e gráficos

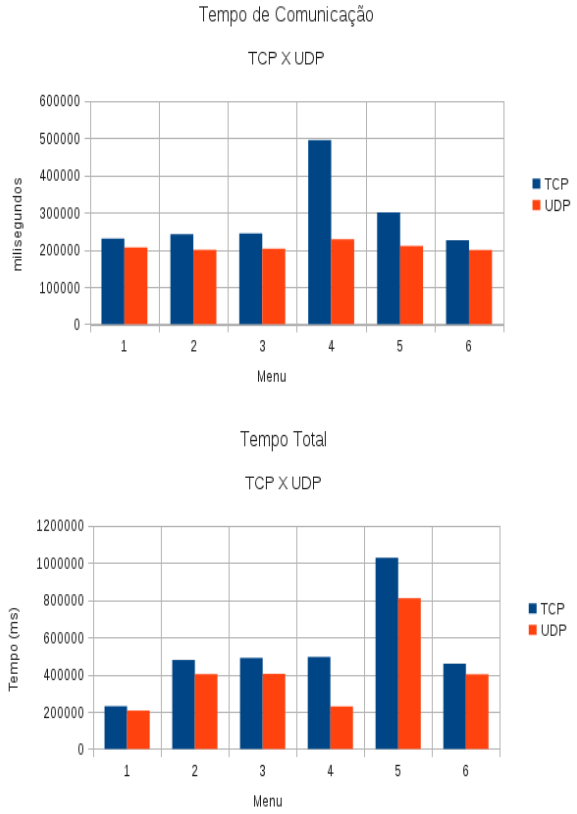
Devido ao grande número de dados decidimos por representar graficamente apenas as médias e desvios de cada opção do menu[II].

3.2.1 Cálculos

A média calculada entre os pontos foi a média simples, em outras palavras, denominamos a média como $\boxed{\mu}$ então, $\boxed{\mu = \frac{1}{N}\sum_{i=1}^{N}(x_i)}$, 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 $\boxed{\sigma = \sqrt{\frac{1}{N}\sum_{i=1}^{N}(x_i - \mu)^2}}$

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, de nominado $\boxed{\sigma_c}$.



(a) Gráficos Tempo de comunicação e total

Menu	$\boxed{\mu_t}$	$\boxed{\mu_p}$	$\boxed{\mu_c}$	$\boxed{\pm\sigma_c}$
1	206212,67	32,82	206179,85	41671.14 (20,21%)
2	401828,39	202106,08	199722,31	13750.53 (6,88%)
3	403070,83	200304,75	202766,08	19338.60 (9,54%)
4	228492,74	537,03	227955.71	74957.25 (32,88%)
5	809829,40	599657,18	210172.22	19006.89 (9,04%)
6	400696,12	201315,01	199381.11	19006.89 (8,55%)

Tabela 3: Tabela com médias e desvios para protocolo UDP

Menu	$\boxed{\mu_t}$	$\boxed{\mu_p}$	$\boxed{\mu_c}$	$\boxed{\pm\sigma_c}$
1	229928.97	59.43	229869.54	80776.07 (35,13%)
2	477881.95	236349.80	241532.15	94761.02 (39.23%)
3	489242.44	245704.79	243537.65	70828.31 (29,08%)
4	494792.56	559.17	494233.39	305558.45 (61,82%)
5	1027348.23	727262.31	300085.92	77658.26 (25,87%)
6	458069.60	232573.16	225496.44	61852.36 (30.09%)

Tabela 4: Tabela com médias e desvios para protocolo TCP

4 Conclusão

Visto a simplicidade do protocolo UDP, tal como ausência de mecanismos de controle de fluxo, confiabilidade da entrega do pacote e o uso de datagrams, ao invez de streams, utilizados no protocolo TCP, espera-se um tempo de comunicação menor. Os dados do sistema com UDP não foram comparados com os dados obtidos com o TCP no relatório anterior, e sim novos testes para esse último foram realizados. Como previsto, o procotolo UDP teve um tempo de comunicação menor, chegando a uma diferença de 46.12% no caso da opção 4 do menu (envio de apenas uma requisição por parte do cliente), 70.03% opção 5 (envio de 3 requisições, e 4 respostas entre cliente/servidor). Nota-se que os desvio padrão de ambos os protocolos mas, principalmente do TCP, são altos, isso ocorre, provavelmente, a uma grande instabilidade da rede, uma vez que utiliza-se rede sem fio (há uma perda considerável de pacotes, sendo que o TCP garante a entrega e não duplicada) e também, pelo mesmo motivo que os testes, para o TCP, foram refeitos, a rede no dia dos testes apresentava-se lenta e com falhas na comunicação para qualquer aplicação que a utiliza-se. Portanto, concluimos que o protocolo UDP, realmente, é indicado para aplicações que necessitem de velocidade e simplicidade na comunicação entre cliente e servidor. Já sistemas que requerem confiabilidade, tal como garantia da integridade e entrega da informação, o protocolo TCP seria o mais indicado.

5 Código fonte *(.c)*

5.1 Diretório *commons*

5.1.1 *error.h error.c*

```
1  /**
2   * Arquivo que manipula erros ocorridos
3   **/
4
5  #ifndef _H_ERROR
6  #define _H_ERROR
7
8  #include <stdio.h>
9  #include <stdlib.h>
10 #include <time.h>
11 #include "common.h"
12 #include "log.h"
13
14 enum ERROR {
15    ERROR_SOCKET = 1,
16    ERROR_SOCKET_CLIENT,
17    ERROR_SOCKET_CLIENT_CLOSED,
18    ERROR_SOCKET_SEND,
19    ERROR_SOCKET_RCV_MSG,
20    ERROR_SOCKET_SERVER_ERROR,
21    ERROR_SOCKET_SERVER_CLOSED,
22    ERROR_SOCKET_SERVER_BIND,
23    ERROR_SOCKET_SERVER_LISTEN,
24    ERROR_SOCKET_SERVER_ACCEPT,
25    ERROR_LOG_FILE,
26    ERROR_USAGE_MAIN,
27    ERROR_USAGE_TEST,
28    ERROR_FILE_OPEN,
29    ERROR_NUM_COLUNAS_BOOKS,
30    ERROR_UDP_MAX_RETRY
31 };
32 /**
33  * Funcao que imprime um erro ocorrido, parametro int e (codigo de
        erro)
34  * definido nesse arquivo
35  * @param int e
36  **/
37 void printError(int e);
38
39 #endif
```

```
1  #include "error.h"
2
3  void printError(int e){
4    switch(e){
5    case ERROR_SOCKET:
6      perror("socket:: naco 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_RCV_MSG:
12     perror("socket:: erro ao receber mensagem\n");
13     break;
14   case ERROR_SOCKET_SEND:
```

```
15      perror("socket:: nao foi possivel enviar mensagem para o
            servidor\n");
16      break;
17    case ERROR_SOCKET_SERVER_ERROR:
18      perror("send_msg:: ocorreu um erro na comunicacao com o
            servidor, tente novamente");
19      break;
20    case ERROR_SOCKET_CLIENT_CLOSED:
21      perror("receive_msg:: cliente fechou a conexao");
22      break;
23    case ERROR_SOCKET_SERVER_CLOSED:
24      perror("send_msg:: servidor fechou a conexao");
25      break;
26    case ERROR_SOCKET_SERVER_BIND:
27      perror("start_server:: ocorreu um erro com o bind");
28      break;
29    case ERROR_SOCKET_SERVER_LISTEN:
30      perror("start_server:: ocorreu um erro com o listen");
31      break;
32    case ERROR_SOCKET_SERVER_ACCEPT:
33      perror("start_server:: ocorreu um erro com o accept");
34      break;
35    case ERROR_USAGE_MAIN:
36      printf("erro de entrada: tente ./main <num_porta> (para iniciar
             servidor) ou ./main <endereco_ip> <porta> para iniciar o
            cliente\n");
37      break;
38    case ERROR_USAGE_TEST:
39      printf("erro de entrada: tente ./test <endereco_ip> <porta>
            para iniciar o cliente (o servidor deve estar up!)\n");
40      break;
41    case ERROR_FILE_OPEN:
42      perror("archives:: Erro ao abrir arquivo\n");
43      break;
44    case ERROR_NUM_COLUNAS_BOOKS:
45      printf("archives:: Arquivo com conteudo de livros corrompido.
            VEJA README PARA MAIS DETALHES\n");
46    case ERROR_UDP_MAX_RETRY:
47      perror("udp_sendto:: Limite maximo de tentativas de envio,
            altere UDP_MAX_RETRY \n");
48      break;
49    }
50  }
```

5.1.2 *commmon.h common.c*

```
1  #ifndef _H_COMMON
2  #define _H_COMMON
3  #include <stdio.h>
4  #include <stdlib.h>
5  #include <string.h>
6  #include <stdbool.h>
7
8  #define CHAR_NEW_LINE "\n"
9  #define DOUBLE_MILHAO 1000000.00
10 #define NUM_OPCOES_MENU 6
11 #define NUM_TESTES 100
12 #define OPT_TESTE "-t"
13
14 typedef struct timeval TimeVal;
15 typedef struct sockaddr_in SA_IN;
```

```
16 typedef struct sockaddr SA;
17 typedef struct addrinfo AI;
18
19 /**
20  * Armadilha para qualquer tipo de sinal
21  * @param int s: signal
22  **/
23 void trapAnySignal(int s);
24 /**
25  * Convert inteiro na base escolhida para string
26  * @param int val
27  * @param int base
28  * @see http://www.jb.man.ac.uk/~slowe/cpp/itoa.html
29  **/
30 char* my_itoa(int val, int base);
31
32 /**
33  *@function :   copys s[a] -> s[b] into t.
34  *@param char *t : target string (should be malloc(a + b + 1))
35  *@param char *s : source string
36  *@param int a : startpoint in source string (must be > 0 )
37  *@param int b : endpoint in source string (must be < strlen(s))
38  **/
39 void poscpy(char *t,char *s,int a,int b);
40 /**
41  * Retorna um vetor de string que foi dividida em tokens.
42  * @param char *str : string original
43  * @param char n : delimitador
44  * @param int *lenght : escreve a quantidade de tokens geradas (
        count do retorno)
45  * @return char ** array
46  * @see http://www.vivaolinux.com.br/topico/C-C++/Split-em-C
47  **/
48 char** split(char* str,char n,int *length);
49
50 #endif
```

```
1 #include "common.h"
2
3 void trapAnySignal(int s){
4   printf("Signal:: ",s);
5   exit(s);
6 }
7
8 char* my_itoa(int val, int base){
9   static char buf[32] = {0};
10  int i = 30;
11  for(; val && i ; --i, val /= base)
12    buf[i] = "0123456789abcdef"[val % base];
13  return &buf[i+1];
14 }
15 void poscpy(char *t,char *s,int a,int b){
16    while ( a < b ) *(t++) = *(s + (a++));
17    *t = 0x0;
18 }
19
20 char **split(char *str,char n,int *length){
21    register int i, j;
22    /* control */
23    int len=strlen(str);
24    int elements = 0, elpos = 0;
25    char **array;
```

```
26      /* number of new itens in array */
27      for(i=0;i<len;i++) if(str[i]==n) elements++;
28      /* get the memory */
29      array=(char **)calloc(elements,sizeof(char *));
30      if(!array){
31          printf("# Error in malloc(*).\n");
32          return NULL;
33      }
34      /* the number of elements for the caller */
35      *length=elements;
36      /* lvl1
37       *
38       * @i = will be the start point for copy
39       */
40      for(i=0;i<len;i++)
41        /* lvl2
42         *
43         * @j = will be end point for copy
44         */
45        for(j=i;j<=len;j++)
46          /* found splitChar or EoL */
47          if(str[j]==n){
48      /*
49       * @i has start point
50       * @j has end point
51       */
52      array[elpos]=(char *)malloc((j-i+1)*sizeof(char));
53      if ( !array[ elpos ] ){
54        printf("# lvl2\n");
55        printf(" # Error in malloc().\n");
56        return NULL;
57      }
58      /* copy the string into the array */
59      poscpy(array[elpos],str,i,j);
60      /* increment array position */
61      elpos++;
62      /* after the copy is done,
63       * @i must be equal to @j
64       */
65      i=j;
66      /* end loop lvl2 */
67      break;
68          }
69      /* return array */
70      return array;
71  }
```

5.1.3 *avl.h avl.c*

```
1   /**
2    * Arvore AVL - (Adelson-Velskii e Landis)
3    * Fator de Balanceamento altura_direita - altura_esquerda
4    **/
5   #ifndef _H_AVL
6   #define _H_AVL
7   #include <stdio.h>
8   #include <stdlib.h>
9   #include <string.h>
10
11  #include "books.h"
12
```

```
13
14 typedef struct avl {
15   int bal;/*balanceamento avl*/
16   char *id;/*id*/
17   struct avl *dir;/*direita*/
18   struct avl *esq;/*esquerdo*/
19   void *dado;/*dado qualquer (estrutura)*/
20 }*AVL,AVL_NO;
21 /**
22  * Inicializa uma arv avl;
23  * @param AVL *avl;
24  **/
25 void newAVLTree(AVL *avl);
26 /**
27  * Retorna altura de uma avl
28  * @param AVL avl
29  * @return int
30  **/
31 int altura(AVL arv);
32 /**
33  * Verifica que se uma arvore eh avl
34  **/
35 int verifica_AVL(AVL arv);
36 void LL(AVL *avl);
37 void RR(AVL *avl);
38 void LR(AVL *avl);
39 void RL_X(AVL *avl);
40 int aux_insereAVL(AVL *avl, Livro *l, int *cresceu);
41 int insereAVL(AVL *avl, Livro *l);
42 /**
43  * @function : Grava todos os ids separados pelo delimitador em uma
        string
44  * @param AVL avl : estrutura avl
45  * @param char **str : string q contera os ids
46  * @param char *del : delimitador entre ids
47  * @example : str tera o formato delstrdelstr....
48  **/
49 void avlIdToStr(AVL avl, char **str, char *del);
50 /**
51  * Converte todo conteudo da estrutura em uma string
52  * @param AVL avl : estrutura avl
53  * @param char **str : string q contera os ids
54  * @param char *del : delimitador entre ids
55  * @example : str tera o formato delstrdelstr....
56  **/
57 void avlToStr(AVL avl, char **str, char *del);
58 /**
59  * calcula o total de caracter presente em toda estrutura avl
60  * @param AVL avl : estrutura
61  **/
62 int totAVLchar(AVL avl);
63 /**
64  * Retorna o total de caracteres que tem em todos isbn presentes
65  * na estrutura.
66  * @param AVL avl
67  * @return int total de caracters
68  **/
69 int totISBNchar(AVL avl);
70 /**
71  * Retorna o total de elementos na arv avl
72  * na estrutura.
73  * @param AVL avl
```

```
74  * @return int total de elem.
75  **/
76 int totElemAVL(AVL avl);
77
78 /**
79  * Limpa memoria ocupada pela arv
80  * @param AVL avl
81  **/
82 void freeAVL(AVL_NO *avl);
83
84 /**
85  * Retorna o elemento encontrado ou NULL
86  * @param AVL *avl
87  * @param char *id
88  **/
89 AVL_NO *getAVLElemById(AVL_NO *avl, char *id);
90
91 #endif
```

```
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 }
```

5.1.4 *archives.h archives.c*

```
1  /**
2   * Manipula arquivos.
3   * @version: 1.1
4   * @changelog
5   * - funcao isThisExtension(a,b);
6   * - char *getFileExtension(char *nomeArq)
7   * + include avl.h
8   * + include error.h
9   * + void readFileBooks(FILE *f, AVL *avl);
10  **/
11
12 #ifndef _H_ARCHIVES
13 #define _H_ARCHIVES
14
15 #include <stdio.h>
16 #include <stdlib.h>
17 #include <stdbool.h>
18
19 #include "error.h"
20 #include "common.h"
21 #include "avl.h"
22
23 #define ARCHIVES_TAM_BUFFER 65536
24 #define NUM_COLUNAS_BOOKS 7
25
26 /**
27  * Funcao que abre o arquivo
28  * @param: char *nome
29  * @param: char *param - parametro para abertura do arquivo
30  * @param: FilaErr *pErro - ponteiro para a fila de descricao de
       erros
31  **/
32 FILE *openFile(char *nome, char *param);
33
34 /**
35  * @function : Funcao que le um arquivo separado por um delimitador
       , que contem
36  * uma lista de livro (ver sequencia de dados em  books.h
       BOOKS_FILE_ORDER)
37  * e salva na estrutura de arvore avl
38  * @param FILE *f : file resource
39  * @param AVL *avl : estrutura
40  * @see books.h , avl.h , README
41  **/
42 void readFileBooks(FILE *f, AVL *avl);
43
44 #endif
```

```
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 }
```

5.1.5 *books.h books.c*

```
1 #ifndef _H_BOOKS
2 #define _H_BOOKS
3 #include <stdio.h>
```

```
4  #include <stdlib.h>
5  #include "string.h"
6  #include "common.h"
7
8  #define FILE_BOOK "../tcp/common/livros/livros"
9  #define BOOK_NOT_FOUND "ISBN nao encontrado!\n"
10 typedef struct autor {
11   char *nome; /*nome do campo*/
12   struct autor *prox;/*apontador para o proximo*/
13 } *Autores, Autor;
14
15 typedef struct book {
16   char *isbn;/*id*/
17   char *titulo;/*title*/
18   Autores autores;/*estrutura de autor*/
19   char *desc;/*descricao*/
20   char *editora;/*editora*/
21   char *ano;/*data*/
22   int estoque;/*numero de exemplares*/
23 } Livro;
24
25 enum BOOKS_FILE_ORDER {
26   ISBN = 0,
27   AUTORES,
28   TITULO,
29   ANO,
30   DESCRICAO,
31   EDITORA,
32   ESTOQUE
33 };
34 /**
35  * Cria nova lista para autores
36  * @param Autores *a
37  **/
38 void newAutoresList(Autores *a);
39 /**
40  * Insere um novo autor na lista
41  * @param Autores *a
42  **/
43 void insertAutoresList(Autores *a, char *nome);
44 /**
45  * Limpa a lista de autores
46  * @param Autores *a
47  **/
48 void freeAutoresList(Autores *a);
49 /**
50  * Retorna o numero de autores de um livro
51  * @param Autor *a : estrutura Autor
52  * @return int
53  **/
54 int getAutoresNum(Autor *a);
55 /**
56  * Retorna o numero bytes dos autores de um livro
57  * @param Autor *a : estrutura Autor
58  * @return int
59  **/
60 int getAutoresNumBytes(Autor *a);
61 /**
62  * Retorna o numero bytes de um livro
63  * @param Livro *l : estrutura Livro
64  * @return int
65  **/
```

```
66 int getBookNumBytes(Livro *l);
67 /**
68  * Converte uma lista de autores em string com um delimitador
69  * @param Autor *n : lista de autores
70  * @param char *del : delimitador
71  * @return char *str
72  **/
73 char *autoresNodeToStr(Autor *n,char *del);
74 /**
75  * Converte um livro em string
76  * @param Autor *n : lista de autores
77  * @return char *str
78  **/
79 char *bookNodeToStr(Livro *l);
80 int getBookNodeSize();
81 #endif
```

```
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 }
```

5.1.6 *tcp.h tcp.c*

```
#ifndef _H_TCP
#define _H_TCP

#define TCP_BUF_SIZE 1024 /*tamanho do buffer de escrita*/
#define TCP_BUF_MIN_SIZE 5/*tamanho do buffer_minimo de escrita*/
#define TCP_MAX_CONN 50 /*maximo de conexoes que o server tcp aceitara*/
#define TCP_MSG_WELCOME "Welcome to Server\n"
#define TCP_MSG_HELLO "Hello\0"
#define TCP_MSG_BYE "Bye\n"
#define TCP_MSG_ACK "ACK\0"
#define TCP_MSG_COMMAND_NOT_FOUND "*ERRO*: Comando nao reconhecido\n"
#define TCP_MSG_NUM_NATURAL_REQUIRED "*ERRO*: A entrada deve ser um numero maior que zero\n"
#define TCP_MSG_SUCESS_EDIT "Campo editado com sucesso!\n"
#define TCP_MSG_LOGIN_REQUIRED "Senha : "
#define TCP_MSG_ISBN_REQUIRED "ISBN : "
#define TCP_MSG_QTD_REQUIRED "Quantidade : "
#define TCP_COMMAND_CLOSE_CONNECTION "\\q"
#define TCP_COMMAND_MENU "\\m"
#define TCP_MSG_MENU_REQUEST ":"

#define TAM_CAMPO_MSG 10
#define DEL_CAMPO_MSG "$"

#define UDP_MAX_RETRY 10
#endif
```

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

5.1.7 *tempo.h tempo.c*

```
#ifndef _H_TEMPO
#define _H_TEMPO

#include <stdlib.h>
#include <stdio.h>
#include <math.h>
#include <string.h>
#include <signal.h>
#include <sys/socket.h>
#include <netinet/in.h>
#include <arpa/inet.h>


#include "error.h"
#include "archives.h"
#define NUM_TESTES 100

typedef struct headTempo {
  char *medida;//nome da medida
  double media;//media
  double desvio;//devio padrao
  int length;//quantidade de dados
  struct tempo *dados;//tempo
}Media;
```

```
25
26 typedef struct tempo {
27   double inicio;
28   double fim;
29   double decorrido;
30   struct tempo *prox;
31 }Tempo;
32
33 /**
34  * Inicia uma nova media
35  * @param Media **m : estrutura com os dados e media
36  * @param char *mname : nome da medida (arquivo a ser salvo)
37  **/
38 void newMedia(Media **m, char *mname);
39 /**
40  * Calcula a media e desvio a partir dos dados
41  * @param Media **m : estrutura com os dados e media
42  **/
43 void calcMedia(Media **m);
44 /**
45  * Insere um tempo na estrutura
46  * @param Media **m : estrutura com os dados e media
47  * @param double i : tempo inicial
48  * @param double f : tempo final
49  **/
50 void insertTempo(Media **m,double i,double f);
51 /**
52  * Le arquivos que contem os tempos registrados pelo teste
53  * @see README
54  * @param Media **m : estrutura com os dados e media
55  * @param char *fname : caminho do arquivo
56  **/
57 void readFileMedia(Media **m, char *fname);
58 /**
59  * Escreve a media com devio em um arquivo dado pelo nome da medida
60  * @param Media **m : estrutura com os dados e media
61  **/
62 void writeFileMedia(Media **m);
63
64 #endif
```

```
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 }
```

5.2 Diretório *server*

5.2.1 *server.h server.c*

```
1  #ifndef _H_SERVER
2  #define _H_SERVER
3  #include <stdio.h>
4  #include <stdlib.h>
5  #include <sys/socket.h>
6  #include <netinet/in.h>
7  #include <arpa/inet.h>
8
9  #include "../../tcp/common/error.h"
10 #include "../../tcp/common/log.h"
11 #include "../../tcp/common/tcp.h"
12 #include "../../tcp/common/books.h"
13 #include "../../tcp/common/archives.h"
14 #include "../../tcp/common/avl.h"
15 #include "udp_server.h"
16 /**
17  * funcao que chama o bind() para o endereco e faz o listen() no
        mesmo
18  * e a funcao bindNlisten() faz chamada para receive_msg()
19  * @param char *argv[]
20  **/
21 void start_server(char *argv[]);
22
23 /**
24  * @function : Funcao que carrega os livros para a memoria
25  * @param struct avl **l : estrutura dos livros
26  * @see books.h, avl.h, archives.h
27  **/
28 void load_books(AVL *l);
29 #endif
```

```
1  #include "server.h"
2  void start_server(char *argv[]){
```

```
3     AVL l;
4     newAVLTree(&l);
5     load_books(&l);
6     init(atoi(argv[1]),&l);
7     //  freeAVL(l);
8   }
9
10  void load_books(AVL *l){
11    readFileBooks(openFile(FILE_BOOK,"r"),l);
12  }
```

5.2.2 *udpserver.h udpserver.c*

```
1  #ifndef _H_UDP_SERVER
2  #define _H_UDP_SERVER
3  #include <stdio.h>
4  #include <stdlib.h>
5  #include <stdbool.h>
6  #include <ctype.h>
7  #include <string.h>
8  #include <unistd.h>
9  #include <sys/socket.h>
10 #include <netdb.h>
11 #include <netinet/in.h>
12 #include <arpa/inet.h>
13 #include <sys/time.h>
14 #include "../../tcp/common/error.h"
15 #include "../../tcp/common/log.h"
16 #include "../../tcp/common/tcp.h"
17 #include "../../tcp/common/avl.h"
18 #include "../../tcp/common/common.h"
19 #include "login.h"
20 #define TAM_MENU 9
21
22
23 /**
24  * Inicia um servidor socker na porta especificada como parametro.
25  * Faz o bind e listen na porta e aguarda o contato de algum
        cliente.
26  * @param int sDesc : descriptor de uma conexao socket
27  * @param AVL *l : estrutura com info dos livros
28  **/
29 void init(int port, AVL *l);
30 /**
31  * Funcao que recebe uma mensagem (geralmente ACK) de um cliente
32  * @see init : com datagram e nao orientado a conexao
33  * @param struct sockaddr_in *c: endereco do cliente;
34  * @param int s : socket descriptor @see init(int port, AVL *l);
35  **/
36 void receive_ack(SA_IN *c, int s);
37 /**
38  * Funcao que envia e recebe uma mensagem do cliente. TUDO EH
        GRAVADO NO LOG
39  * @see log.h
40  * @see init : com datagram e nao orientado a conexao
41  * @param struct sockaddr_in *c: endereco do cliente;
42  * @param int s : socket descriptor @see init(int port, AVL *l);
43  * @param AVL *l : estrutura com info dos livros
44  **/
45 void receive_msg(struct sockaddr_in *c, int s, AVL *l);
46
```

```
47 /**
48  * Funcao que envia o menu para os clientes que acabaram de se
        conectar
49  * ATENCAO, funcao boa para TCP, uma vez que, espera-se que o
        protocolo
50  * envie/receba ordenado, confiavel.
51  * @see bindNlisten (accept)
52  * @param struct sockaddr_in *c: endereco do cliente;
53  * @param int s : socket descriptor @see init(int port, AVL *l);
54  * @param bool alerta : caso true concatena uma msg(
        TCP_MSG_COMMAND_NOT_FOUND) para cliente
55  * @param TimeVal *tvend : para calculo do tempo de processamento
        da requisicao
56  **/
57 void send_menu(struct sockaddr_in *c, int s, bool alert, TimeVal *
        tvend);
58
59 /**
60  * Funcao que le e executa opcao de menu digitada pelo cliente.
61  * @see bindNlisten
62  * @param struct sockaddr_in *c: endereco do cliente;
63  * @param int s : socket descriptor @see init(int port, AVL *l);
64  * @param AVL *l : estrutura com info dos livros
65  * @param char opt[] ; string digitada pelo cliente
66  * @param double ini : inicio do tempo de execucao
67  **/
68 bool read_menu(struct sockaddr_in *c, int s, AVL *l, char opt[],
        double ini);
69 /**
70  * @function : Envia uma string para o cliente contendo todos os
        ids, formatados
71  * para output no cliente.
72  * @param struct sockaddr_in *c: endereco do cliente;
73  * @param int s : socket descriptor @see init(int port, AVL *l);
74  * @param AVL *l : estrutura com info dos livros
75  * @param TimeVal *tvend : para calculo do tempo de processamento
        da requisicao
76  **/
77 void send_all_ids(struct sockaddr_in *c, int s, AVL *l, TimeVal *
        tvend);
78 /**
79  * @function : Envia uma string para o cliente contendo a descricao
        , formatados
80  * para output no cliente.
81  * @param struct sockaddr_in *c: endereco do cliente;
82  * @param int s : socket descriptor @see init(int port, AVL *l);
83  * @param AVL *l : estrutura com info dos livros
84  * @param TimeVal *tvend : para calculo do tempo de processamento
        da requisicao
85  **/
86 void send_desc_byId(struct sockaddr_in *c, int s, AVL *l, TimeVal *
        tvend);
87 /**
88  * @function : Envia uma string para o cliente contendo a
        informacao total de um livro, formatados
89  * para output no cliente.
90  * @param struct sockaddr_in *c: endereco do cliente;
91  * @param int s : socket descriptor @see init(int port, AVL *l);
92  * @param AVL *l : estrutura com info dos livros
93  * @param TimeVal *tvend : para calculo do tempo de processamento
        da requisicao
94  **/
```

```
95 void send_book_info(struct sockaddr_in *c, int s, AVL *l, TimeVal *
       tvend);
96 /**
97  * Envia todas as informacoes de todos os livros
98  * @param struct sockaddr_in *c: endereco do cliente;
99  * @param int s : socket descriptor @see init(int port, AVL *l);
100 * @param AVL *l : estrutura com info dos livros
101 * @param TimeVal *tvend : para calculo do tempo de processamento
        da requisicao
102 **/
103 void send_all_books_info(struct sockaddr_in *c, int s, AVL *l,
       TimeVal *tvend);
104 /**
105 * @function : Permite a edicao do estoque
106 * @param struct sockaddr_in *c: endereco do cliente;
107 * @param int s : socket descriptor @see init(int port, AVL *l);
108 * @param AVL *l : estrutura com info dos livros
109 * @param TimeVal *tvend : para calculo do tempo de processamento
        da requisicao
110 **/
111 void edit_estoque(struct sockaddr_in *c, int s, AVL *l, TimeVal *
       tvend);
112 /**
113 * @function : Envia a quantidade em estoque de um livro para o
        cliente
114 * @param struct sockaddr_in *c: endereco do cliente;
115 * @param int s : socket descriptor @see init(int port, AVL *l);
116 * @param AVL *l : estrutura com info dos livros
117 * @param TimeVal *tvend : para calculo do tempo de processamento
        da requisicao
118 **/
119 void send_estoque_byId(struct sockaddr_in *c, int s, AVL *l,
       TimeVal *tvend);
120 /**
121 * Envia o maximo de bytes, ou seja, o numero de bytes da maior msg
122 * (NAO ESTA INCLUIDO NO TESTE DE PERFORMANCE)
123 * @param struct sockaddr_in *a: endereco do cliente;
124 * @param int s : socket descriptor @see init(int port, AVL *l);
125 * @param AVL *l : estrutura com info dos livros;
126 **/
127 void send_max_size(struct sockaddr_in *c, int s, AVL *l);
128
129 #endif
```

```
1  #include "udp_server.h"
2  /**
3   * NAO deve aceitar conexoes concorrentes
4   * SERVIDOR UDP ITERATIVO
5   **/
6  void init(int port, AVL *l){
7    int sDesc;/*socket descriptor*/
8    int started;/*bind, listen*/
9    char *str=(char *)malloc(sizeof(char)*TCP_BUF_MIN_SIZE+1);
10   SA_IN *c = (struct sockaddr_in*)malloc(sizeof(struct sockaddr_in)
         );/*cliente*/
11   SA_IN *s = (struct sockaddr_in*)malloc(sizeof(struct sockaddr_in)
         );/*servidor*/
12   struct in_addr *addr = (struct in_addr *)malloc(sizeof(struct
         in_addr));
13   AI *ainfo=(struct addrinfo *)malloc(sizeof(struct addrinfo));/*
         addinfo*/
14   /*zerando espaco da mem*/
```

```
15    memset(ainfo,0,sizeof(struct addrinfo));
16    memset(str,'\0',sizeof(char)*TCP_BUF_MIN_SIZE+1);
17    /*atribuindo ainfo*/
18    ainfo->ai_family=AF_INET;/*IPv4*/
19    /* Usa-se datagram ao invez de byte-stream do TCP*/
20    ainfo->ai_socktype=SOCK_DGRAM;/*upd entao, datagram*/
21    ainfo->ai_protocol=IPPROTO_UDP;
22    /*abrindo socket*/
23    sDesc = socket(ainfo->ai_family,ainfo->ai_socktype,ainfo->
          ai_protocol);
24    if(sDesc<0){/*erro?*/
25      printError(ERROR_SOCKET);/*print erro*/
26      exit(1);/*saida com erro*/
27    }
28    /*zerar mem*/
29    memset(addr,0,sizeof(addr));
30    memset(s,0,sizeof(s));
31    memset(c,0,sizeof(c));
32    addr->s_addr = htonl(INADDR_ANY);/*bind all local interfaces*/
33    s->sin_family = AF_INET;/*ipv4*/
34    s->sin_port = htons(port);/*convert to network byte order*/
35    s->sin_addr = *addr;/*endereco*/
36    ainfo->ai_addr=(SA*)s;/*atribuindo servidor a info de end.*/
37    ainfo->ai_addrlen=(socklen_t)sizeof(struct sockaddr_in);/*tam do
          addr*/
38    /*atribuindo endereco ao socket_server*/
39    started = bind(sDesc,ainfo->ai_addr,ainfo->ai_addrlen);
40    if(started<0){/*erro?*/
41      printError(ERROR_SOCKET_SERVER_BIND);/*imprime erro*/
42      close(sDesc);/*fechar o socket aberto*/
43      exit(1);/*saida com erro*/
44    }
45    /*escreve log*/
46    if(LOG) writeLog("UDP_SERVER::iniciado porta",my_itoa(port,10),""
          );
47    /*conexao? NAO ha o accept encontrado no tcp -> O.C.*/
48    /*portanto vamos esperar um contato do cliente, assim gravar seu
          endereco*/
49    receive_ack(c,sDesc);
50    send_max_size(c,sDesc,1);/*envia tam da maior msg a ser enviada*/
51    receive_ack(c,sDesc);
52    while(1){
53      // send_menu(c,sDesc,false,NULL);
54      receive_msg(c,sDesc,1);/*comeca as requisicoes do sistema de
            livraria*/
55    }
56    /*fechar o socket aberto*/
57    close(sDesc);
58    /*limpar mem*/
59    free(str);
60    free(c);
61    free(addr);
62    free(ainfo);
63  }
64  void receive_ack(SA_IN *c, int s){
65    int len=strlen(TCP_MSG_ACK)*sizeof(char);
66    char *str=(char *)malloc(len);
67    unsigned int clilen=sizeof(struct sockaddr_in);
68    memset(str,'\0',len);
69    int max=recvfrom(s,str,len,0,(SA*)c,&clilen);
70    if(max<0){/*erro?*/
71      printError(ERROR_SOCKET_RCV_MSG);/*imprime erro*/
```

```
72      close(s);/*fechar o socket aberto*/
73      exit(1);/*saida com erro*/
74    }
75    if(max<len)
76      str[max]='\0';/*sem erro, limit string*/
77    if(LOG) writeLog("UDP_SERVER::rcv",inet_ntoa(c->sin_addr),str);
78  }
79  void receive_msg(SA_IN *c, int s, AVL *l){
80    char msg[TCP_BUF_SIZE];/*msg transmit*/
81    int b;/*byte received*/
82    bool transmit=false;
83    double ini;
84    unsigned int len=sizeof(struct sockaddr_in);
85    int i=0;
86    struct timeval tvini; /*apenas assim para pegar microsegundos*/
87    do{
88      if(LOG) writeLog("UDP_SERVER::wainting menu","","");
89      memset(msg,'\0',TCP_BUF_SIZE);
90      b = recvfrom(s,msg,TCP_BUF_SIZE,0,(SA*)c,&len);
91      if(LOG)
92        writeLog("UDP_SERVER::rcv",msg,"");
93      gettimeofday(&tvini, NULL);
94      msg[b] = '\0';
95      transmit=false;
96      for(i=0;i<b;i=i+3){
97        if(b-i >= 3){
98    if(msg[i]!='A' || msg[i+1]!='C' || msg[i+2] != 'K'){
99      transmit=false;
100     i=b;
101   }
102   else{
103     transmit=true;
104   }
105       }
106       else{
107   transmit=false;
108   i=b;
109       }
110     }
111     if(transmit){
112       for(i=3;i<TCP_BUF_SIZE;i++)
113   msg[i]='\0';
114     }
115     if(strcmp(msg,TCP_MSG_ACK)==0){
116       /*do nothing*/
117       memset(msg,'\0',TCP_BUF_SIZE);
118     }
119     else if(strlen(msg)>5){
120       send_menu(c,s,true,NULL);
121     }
122     else if(strcmp(msg,TCP_COMMAND_MENU)==0){
123       send_menu(c,s,false,NULL);
124     }
125     else if (strcmp(msg,TCP_COMMAND_CLOSE_CONNECTION) == 0){
126       sendto(s,TCP_MSG_BYE,strlen(TCP_MSG_BYE),0,(SA*)c,sizeof(c));
127     }
128     else{
129       ini=DOUBLE_MILHAO*(double)(tvini.tv_sec)+(double)(tvini.
              tv_usec);
130       read_menu(c,s,l,msg,ini);
131     }
132     transmit=true;
```

```
133   }while(transmit);
134 }
135
136 void send_menu(SA_IN *c, int s,bool alert, TimeVal *tvend){
137   char *msg[TAM_MENU];/*texto do menu*/
138   msg[0] = "*** MENU (tecle \\m para visualizar o menu) *********\n
          \0";
139   msg[1] = "*** (1) Listar ISBN                                  *\n
          \0";
140   msg[2] = "*** (2) Ver descricacao por ISBN (entrada: ISBN)     *\n
          \0";
141   msg[3] = "*** (3) Ver info. completa (entrada: ISBN)          *\n
          \0";
142   msg[4] = "*** (4) Ver info. completa (todos os livros)        *\n
          \0";
143   msg[5] = "*** (5) Alterar estoque (apenas adm)                 *\n
          \0";
144   msg[6] = "*** (6) Ver quantidade em estoque (entrada: ISBN)   *\n
          \0";
145   msg[7] = "*** (\\q) Fechar conexao e sair                       *\n
          \0";
146   msg[8] = "************************************************\n
          \0";
147   int i=0,j=0;
148   int last=0,tam=0;
149   unsigned int clilen=sizeof(struct sockaddr_in);
150   for(i=0;i<TAM_MENU;i++)/*contando tamanho da msg a ser criada*/
151     tam+=strlen(msg[i]);
152   if(alert)/*possui msg de alerta?*/
153     tam+=strlen(TCP_MSG_COMMAND_NOT_FOUND);/*entao soma ao tamanho
          */
154   char aux[tam];/*string auxiliar, com tamanho calculado*/
155   memset(aux,'\0',tam+1);/*zerando auxiliar*/
156   if(alert){/*tem alerta?*/
157     strcat(aux,TCP_MSG_COMMAND_NOT_FOUND);/*inicio da criacao da
          msg*/
158     last=strlen(TCP_MSG_COMMAND_NOT_FOUND);/*marcando ultimo
          caracter da msg*/
159   }
160   for(i=0;i<TAM_MENU;i++){
161     for(j=0;j<strlen(msg[i]);j++){
162       aux[last+j]=msg[i][j];/*se nao a alerta last=0*/
163     }
164     last+=j;
165   }
166   last=strlen(aux);/*tam da msg a ser enviada*/
167   if(tvend != NULL)/*marcar fim?*/
168     gettimeofday(tvend, NULL);/*marcando tempo*/
169   if(LOG)
170     writeLog("UDP_SERVER::sending menu",aux,"");
171   int max = 0;
172   do{
173     max=sendto(s,aux,last,0,(SA*)c,clilen);/*envia menu*/
174     if(max==-1){
175       printError(ERROR_SOCKET_SERVER_ERROR);/*imprime erro*/
176       close(s);/*fechar o socket aberto*/
177       exit(1);/*saida com erro*/
178     }
179   }while(max<1);/*envie pelo menos 1byte, evitar bloqueio*/
180 }
181
182 bool read_menu(SA_IN *c, int s, AVL *l, char opt[], double ini){
```

```
183   double end; /*marcar tempo*/
184   struct timeval tvend; /*marcar tempo*/
185   int i=0;
186   for(i=0;i<strlen(opt);i++){/*verifica a msg recebida*/
187     if(strcmp(opt,TCP_MSG_ACK)==0)/*eh ACK?*/
188       return false;/*do nothing*/
189     if(!isdigit(opt[i])){/*nao eh digito?*/
190       send_menu(c,s,true,NULL);/*envia menu com alerta*/
191       return false;
192     }
193   }
194   i=atoi(opt);/*ja foi verificado que eh digito*/
195   switch(i){/*entao qual eh a opcao?*/
196   case 1:
197     send_all_ids(c,s,l,&tvend);/*envie todos os ids*/
198     end=DOUBLE_MILHAO*(double)(tvend.tv_sec)+(double)(tvend.tv_usec
          );/*marcar tempo*/
199     writeDoubleToFile("./estat/tp_1","a+",end-ini);/*escreve
          estatistica*/
200     return true;/*sucesso*/
201   case 2:
202     send_desc_byId(c,s,l,&tvend);/*envie descricao, mas antes
          requisite isbn*/
203     end=DOUBLE_MILHAO*(double)(tvend.tv_sec)+(double)(tvend.tv_usec
          );/*tempo*/
204     writeDoubleToFile("./estat/tp_2","a+",end-ini);/*escreve
          estatistica*/
205     return true;/*sucesso*/
206   case 3:
207     send_book_info(c,s,l,&tvend);/*pede isbn, envia info do livro*/
208     end=DOUBLE_MILHAO*(double)(tvend.tv_sec)+(double)(tvend.tv_usec
          );/*tempo*/
209     writeDoubleToFile("./estat/tp_3","a+",end-ini);/*escreve
          estatistica*/
210     return true;/*sucesso*/
211   case 4:
212     send_all_books_info(c,s,l,&tvend);/*todas as infos, maior msg*/
213     end=DOUBLE_MILHAO*(double)(tvend.tv_sec)+(double)(tvend.tv_usec
          );/*tempo*/
214     writeDoubleToFile("./estat/tp_4","a+",end-ini);/*escreve
          estatistica*/
215     return true;/*sucesso*/
216   case 5:
217     edit_estoque(c,s,l,&tvend);/*editar, dado isbn, senha e
          quantidade*/
218     end=DOUBLE_MILHAO*(double)(tvend.tv_sec)+(double)(tvend.tv_usec
          );/*tempo*/
219     writeDoubleToFile("./estat/tp_5","a+",end-ini);/*escreve
          estatistica*/
220     return true;/*sucesso*/
221   case 6:
222     send_estoque_byId(c,s,l,&tvend);/*envia estoque dado isbn*/
223     end=DOUBLE_MILHAO*(double)(tvend.tv_sec)+(double)(tvend.tv_usec
          );/*tempo*/
224     printf("escrevendo 6");
225     writeDoubleToFile("./estat/tp_6","a+",end-ini);/*escreve
          estatistica*/
226     return true;/*sucesso*/
227   default:
228     send_menu(c,s,true,&tvend);/*reenvia menu*/
229     end=DOUBLE_MILHAO*(double)(tvend.tv_sec)+(double)(tvend.tv_usec
          );/*tempo*/
```

```
230       writeDoubleToFile("./estat/tp_menu","a+",end-ini);/*estatistica
              */
231       return false;/*sem sucesso*/
232     }
233     return true;/*sucesso*/
234 }
235
236 void send_all_ids(SA_IN *c, int s, AVL *l, TimeVal *tvend){
237     int len=totISBNchar(*l);/*tamanho da msg*/
238     int tot=totElemAVL(*l);/*tamanho da msg*/
239     char *del="\n ISBN: ";/*delimitador*/
240     char *ids=(char *)malloc(len+tot*strlen(del)+2);/*alocando mem*/
241     unsigned int clilen=sizeof(struct sockaddr_in);/*udp soh funciona
            assim*/
242     memset(ids,'\0',len+tot*strlen(del)+2);/*zerando msg*/
243     avlIdToStr(*l,&ids,del);/*convertendo avl para string*/
244     strcat(ids,"\n");/*concatenando \n*/
245     gettimeofday(tvend,NULL);/*marcar tempo (tempo de processamento)
            */
246     int max =0;
247     do{
248       max=sendto(s,ids,strlen(ids),0,(SA*)c,clilen);/*envia msg*/
249       if(max==-1){
250         printError(ERROR_SOCKET_SERVER_ERROR);/*imprime erro*/
251         close(s);/*fechar o socket aberto*/
252         exit(1);/*saida com erro*/
253       }
254     }while(max<1);/*enviar pelo menos 1 byte, evitar bloqueio*/
255     if(LOG) writeLog("UDP_SERVER::send ISBN LIST",ids,"");
256     free(ids);/*free allocated mem*/
257 }
258
259 void send_desc_byId(SA_IN *c, int s, AVL *l, TimeVal *tvend){
260     char msg[TCP_BUF_SIZE];/*msg transmited*/
261     unsigned int clilen=sizeof(struct sockaddr_in);/*cliente len*/
262     int max=0;
263     do{
264       max=sendto(s,TCP_MSG_ISBN_REQUIRED,strlen(TCP_MSG_ISBN_REQUIRED
              ),0,(SA*)c,clilen);/*requisicao do isbn*/
265       if(max==-1){
266         printError(ERROR_SOCKET_SERVER_ERROR);/*imprime erro*/
267         close(s);/*fechar o socket aberto*/
268         exit(1);/*saida com erro*/
269       }
270     }while(max<1);/*enviar pelo menos 1 byte, evitar bloqueio*/
271     int b=recvfrom(s,msg,TCP_BUF_SIZE,0,(SA*)c,&clilen);/*isbn?*/
272     msg[b] = '\0';/*marcando fim da msg*/
273     AVL_NO *livro=getAVLElemById(*l,msg);/*escolhe elem pelo id=isbn
            */
274     if(livro!=NULL){/*isbn valido?*/
275       Livro *temp=(Livro *)livro->dado;/*casting dos dados*/
276       char str[strlen(temp->desc)+4];/*msg a ser enviada*/
277       memset(str,'\0',strlen(temp->desc)+4);/*zerando mem*/
278       strcat(str,temp->desc);/*concatenando descricao a msg*/
279       strcat(str,"\n");/*concatenando \n*/
280       gettimeofday(tvend,NULL);/*marcando tempo de processamento*/
281       do{
282         max=sendto(s,str,strlen(str),0,(SA*)c,clilen);/*envia msg*/
283         if(max==-1){
284     printError(ERROR_SOCKET_SERVER_ERROR);/*imprime erro*/
285     close(s);/*fechar o socket aberto*/
286     exit(1);/*saida com erro*/
```

```
287         }
288       }while(max<1);/*enviar pelo menos 1 byte, evitar bloqueio*/
289     }
290     else{/*isbn invalido*/
291       gettimeofday(tvend,NULL);/*marcando tempo de processamento*/
292       /*envia msg de livro nao encontrado*/
293       do{
294         max=sendto(s,BOOK_NOT_FOUND,strlen(BOOK_NOT_FOUND),0,(SA*)c,
              clilen);
295         if(max==-1){
296     printError(ERROR_SOCKET_SERVER_ERROR);/*imprime erro*/
297     close(s);/*fechar o socket aberto*/
298     exit(1);/*saida com erro*/
299         }
300       }while(max<1);/*enviar pelo menos 1 byte, evitar bloqueio*/
301     }
302 }
303
304 void send_book_info(SA_IN *c, int s, AVL *l, TimeVal *tvend){
305     char msg[TCP_BUF_SIZE];/*msg transmited*/
306     unsigned int len=sizeof(struct sockaddr_in);/*cliente len*/
307     int max=0;
308     do{
309       max=sendto(s,TCP_MSG_ISBN_REQUIRED,strlen(TCP_MSG_ISBN_REQUIRED
            ),0,(SA*)c,len);/*requisicao do isbn*/
310     if(max==-1){
311         printError(ERROR_SOCKET_SERVER_ERROR);/*imprime erro*/
312         close(s);/*fechar o socket aberto*/
313         exit(1);/*saida com erro*/
314       }
315     }while(max<1);/*enviar pelo menos 1 byte, evitar bloqueio*/
316     int b=recvfrom(s,msg,TCP_BUF_SIZE,0,(SA*)c,&len);/*isbn?*/
317     msg[b] = '\0'; /*marcando fim da msg*/
318     AVL_NO *livro=getAVLElemById(*l,msg);/*escolhe elem pelo id=isbn
          */
319     if(livro!=NULL){/*isbn valido?*/
320       Livro *temp=(Livro *)livro->dado;/*casting dos dados*/
321       char *str = bookNodeToStr(temp);/*msg a ser enviada*/
322       gettimeofday(tvend,NULL);/*marcando tempo de processamento*/
323       do{
324         max=sendto(s,str,strlen(str),0,(SA*)c,len);/*envia msg*/
325         if(max==-1){
326     printError(ERROR_SOCKET_SERVER_ERROR);/*imprime erro*/
327     close(s);/*fechar o socket aberto*/
328     exit(1);/*saida com erro*/
329         }
330       }while(max<1);/*enviar pelo menos 1 byte, evitar bloqueio*/
331     }
332     else{/*isbn invalido*/
333       gettimeofday(tvend,NULL);/*marcando tempo de processamento*/
334       /*envia msg de livro nao encontrado*/
335       do{
336         max=sendto(s,BOOK_NOT_FOUND,strlen(BOOK_NOT_FOUND),0,(SA*)c,
              len);
337         if(max==-1){
338     printError(ERROR_SOCKET_SERVER_ERROR);/*imprime erro*/
339     close(s);/*fechar o socket aberto*/
340     exit(1);/*saida com erro*/
341         }
342       }while(max<1);/*enviar pelo menos 1 byte, evitar bloqueio*/
343     }
344 }
```

```
345
346 void send_all_books_info(SA_IN *c, int s, AVL *l, TimeVal *tvend){
347   int len=totAVLchar(*l);/*calculo da msg tam*/
348   int tot=totElemAVL(*l);/*calculo da msg tam*/
349   int bsize=getBookNodeSize();/*calculo da msg tam*/
350   char *del="\n";/*delimitador*/
351   int lentot=(len+tot*bsize+tot*strlen(del)*2+2);/*tamanho total*/
352   char *allin=(char *)malloc(lentot);/*alocacao de memoria para msg
        */
353   memset(allin,'\0',lentot);/*zerando*/
354   avlToStr(*l,&allin,del);/*converte toda estrutura avl para string
        */
355   strcat(allin,"\n");/*concatena \n*/
356   gettimeofday(tvend,NULL);/*tempo de processamento*/
357   unsigned int clilen=sizeof(struct sockaddr_in);/*cliente len*/
358   int max=0;
359   do{
360     max=sendto(s,allin,lentot,0,(SA*)c,clilen);/*envia msg*/
361     if(max==-1){
362       printError(ERROR_SOCKET_SERVER_ERROR);/*imprime erro*/
363       close(s);/*fechar o socket aberto*/
364       exit(1);/*saida com erro*/
365     }
366   }while(max<1);/*enviar pelo menos 1 byte, evitar bloqueio*/
367   free(allin);/*free mem*/
368 }
369
370 void edit_estoque(SA_IN *c, int s, AVL *l, TimeVal *tvend){
371   char msg[TCP_BUF_SIZE];/*msg transmited*/
372   unsigned int len=sizeof(struct sockaddr_in);/*cliente len*/
373   /*requisicao do isbn*/
374   int max=0;
375   do{
376     max=sendto(s,TCP_MSG_ISBN_REQUIRED,strlen(TCP_MSG_ISBN_REQUIRED
          ),0,(SA*)c,len);
377     if(max==-1){
378       printError(ERROR_SOCKET_SERVER_ERROR);/*imprime erro*/
379       close(s);/*fechar o socket aberto*/
380       exit(1);/*saida com erro*/
381     }
382   }while(max<1);/*enviar pelo menos 1 byte, evitar bloqueio*/
383   int b=recvfrom(s,msg,TCP_BUF_SIZE,0,(SA*)c,&len);/*isbn?*/
384   msg[b] = '\0';/*\0 ao fim da msg*/
385   AVL_NO *livro=getAVLElemById(*l,msg);/*buscando isbn na estrutura
        */
386   if(livro!=NULL){/*isbn valido?*/
387     /*entao requisicao da senha*/
388     do{
389       max=sendto(s,TCP_MSG_LOGIN_REQUIRED,strlen(
            TCP_MSG_LOGIN_REQUIRED),0,(SA*)c,len);
390       if(max==-1){
391   printError(ERROR_SOCKET_SERVER_ERROR);/*imprime erro*/
392   close(s);/*fechar o socket aberto*/
393   exit(1);/*saida com erro*/
394       }
395     }while(max<1);/*enviar pelo menos 1 byte, evitar bloqueio*/
396     b=recvfrom(s,msg,TCP_BUF_SIZE,0,(SA*)c,&len);/*senha?*/
397     msg[b] = '\0';/*limitando msg*/
398     if(doLogin(msg)){/*reliza login, senha valida?*/
399       /*entao requisicao de quantidade*/
400       do{
```

```
401     max=sendto(s,TCP_MSG_QTD_REQUIRED,strlen(TCP_MSG_QTD_REQUIRED)
            ,0,(SA*)c,len);
402     if(max==-1){
403       printError(ERROR_SOCKET_SERVER_ERROR);/*imprime erro*/
404       close(s);/*fechar o socket aberto*/
405       exit(1);/*saida com erro*/
406     }
407         }while(max<1);/*enviar pelo menos 1 byte, evitar bloqueio*/
408         b=recvfrom(s,msg,TCP_BUF_SIZE,0,(SA*)c,&len);/*quantidade*/
409         msg[b] = '\0';/*limit msg*/
410         int q=atoi(msg);/*convert str to in*/
411         if(q>=0){/*eh um numero natural?*/
412     Livro *aux = (Livro *)livro->dado;/*casting do dado*/
413     aux->estoque=q;/*atualiza memoria*/
414     livro->dado = (void *)aux;/*dado alterado*/
415     gettimeofday(tvend,NULL);/*tempo de proc.*/
416     /*editado com sucesso*/
417     do{
418       max=sendto(s,TCP_MSG_SUCESS_EDIT,strlen(TCP_MSG_SUCESS_EDIT)
              ,0,(SA*)c,len);
419       if(max==-1){
420         printError(ERROR_SOCKET_SERVER_ERROR);/*imprime erro*/
421         close(s);/*fechar o socket aberto*/
422         exit(1);/*saida com erro*/
423       }
424     }while(max<1);/*enviar pelo menos 1 byte, evitar bloqueio*/
425         }
426         else{
427     gettimeofday(tvend,NULL);/*tempo de proc.*/
428     /*erro na quantidade*/
429     do{
430       max=sendto(s,TCP_MSG_NUM_NATURAL_REQUIRED,strlen(
              TCP_MSG_NUM_NATURAL_REQUIRED),0,(SA*)c,len);
431       if(max==-1){
432         printError(ERROR_SOCKET_SERVER_ERROR);/*imprime erro*/
433         close(s);/*fechar o socket aberto*/
434         exit(1);/*saida com erro*/
435       }
436     }while(max<1);/*enviar pelo menos 1 byte, evitar bloqueio*/
437         }
438       }
439       else{
440         gettimeofday(tvend,NULL);/*tempo de processamento*/
441         /*senha invalida*/
442         do{
443     max=sendto(s,MSG_SENHA_INVALIDA,strlen(MSG_SENHA_INVALIDA),0,(SA
            *)c,len);
444     if(max==-1){
445       printError(ERROR_SOCKET_SERVER_ERROR);/*imprime erro*/
446       close(s);/*fechar o socket aberto*/
447       exit(1);/*saida com erro*/
448     }
449         }while(max<1);/*enviar pelo menos 1 byte, evitar bloqueio*/
450       }
451     }
452     else{
453       gettimeofday(tvend,NULL);/*tp*/
454       /*isbn invalido*/
455       do{
456         max=sendto(s,BOOK_NOT_FOUND,strlen(BOOK_NOT_FOUND),0,(SA*)c,
              len);
457         if(max==-1){
```

```
458     printError(ERROR_SOCKET_SERVER_ERROR);/*imprime erro*/
459     close(s);/*fechar o socket aberto*/
460     exit(1);/*saida com erro*/
461         }
462       }while(max<1);/*enviar pelo menos 1 byte, evitar bloqueio*/
463     }
464 }
465
466 void send_estoque_byId(SA_IN *c, int s, AVL *l, TimeVal *tvend){
467     char msg[TCP_BUF_SIZE];/*msg transmited*/
468     unsigned int len=sizeof(struct sockaddr_in);/*cliente len*/
469     int max=0;
470     /*requisicao pelo isbn*/
471     do{
472       max=sendto(s,TCP_MSG_ISBN_REQUIRED,strlen(TCP_MSG_ISBN_REQUIRED
              ),0,(SA*)c,len);
473       if(max==-1){
474         printError(ERROR_SOCKET_SERVER_ERROR);/*imprime erro*/
475         close(s);/*fechar o socket aberto*/
476         exit(1);/*saida com erro*/
477       }
478     }while(max<1);/*enviar pelo menos 1 byte, evitar bloqueio*/
479     int b=recvfrom(s,msg,TCP_BUF_SIZE,0,(SA*)c,&len);/*isbn?*/
480     msg[b] = '\0'; /*limit msg*/
481     AVL_NO *livro=getAVLElemById(*l,msg);/*busca por isbn*/
482     if(livro!=NULL){/*achou?*/
483       /*entao seleciona livro*/
484       Livro *aux = (Livro *)livro->dado;
485       gettimeofday(tvend,NULL);/*tp*/
486       /*envia msg*/
487       do{
488         max=sendto(s,my_itoa(aux->estoque,10),strlen(my_itoa(aux->
                estoque,10)),0,(SA*)c,len);
489         if(max==-1){
490     printError(ERROR_SOCKET_SERVER_ERROR);/*imprime erro*/
491     close(s);/*fechar o socket aberto*/
492     exit(1);/*saida com erro*/
493         }
494       }while(max<1);/*enviar pelo menos 1 byte, evitar bloqueio*/
495     }
496     else{/*isbn invalido*/
497       gettimeofday(tvend,NULL);/*tp*/
498       do{
499         max=sendto(s,BOOK_NOT_FOUND,strlen(BOOK_NOT_FOUND),0,(SA*)c,
                len);/*nao achou*/
500         if(max==-1){
501     printError(ERROR_SOCKET_SERVER_ERROR);/*imprime erro*/
502     close(s);/*fechar o socket aberto*/
503     exit(1);/*saida com erro*/
504         }
505       }while(max<1);/*enviar pelo menos 1 byte, evitar bloqueio*/
506     }
507 }
508
509 void send_max_size(struct sockaddr_in *c, int s, AVL *l){
510     int len=totAVLchar(*l);/*tam da msg*/
511     int tot=totElemAVL(*l);/*tam da msg*/
512     int bsize=getBookNodeSize();/*tam da msg*/
513     char *del="\n";/*delimitador*/
514     int lentot=(len+tot*bsize+tot*strlen(del)*2+2);/*calculo do tam
          total*/
515     char str[10];/*msg*/
```

```
516   unsigned int clilen=sizeof(struct sockaddr_in);
517   memset(str,'\0',10);/*zerando mem*/
518   /*convertendo int par string*/
519   memmove(str,my_itoa(lentot,10),strlen(my_itoa(lentot,10)));
520   /*envia msg*/
521   if(LOG)
522     writeLog("UDP_SERVER::send",str,"");
523   len = strlen(str);
524   int max=0;
525   do{
526     max=sendto(s,str,len,0,(struct sockaddr*)c,clilen);
527     if(max==-1){
528       printError(ERROR_SOCKET_SERVER_ERROR);/*imprime erro*/
529       close(s);/*fechar o socket aberto*/
530       exit(1);/*saida com erro*/
531     }
532   }while(max<1);/*enviar pelo menos 1 byte, evitar bloqueio*/
533 }
```

5.2.3 *login.h login.c*

```
1  #ifndef _H_LOGIN
2  #define _H_LOGIN
3
4  #include <stdio.h>
5  #include <stdlib.h>
6  #include <stdbool.h>
7  #include <string.h>
8
9  #define SENHA "MC823"
10 #define MSG_SENHA_INVALIDA "Senha invalida!\n"
11
12 /**
13  * Verifica se o login eh valido
14  * @param char *senha
15  * @return bool: true -> sucesso ; false -> senha invalida
16  **/
17 bool doLogin(char *senha);
18
19 #endif
```

```
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  }
```

5.3 Diretório *client*

5.3.1 *client.h client.c*

```
1  #ifndef _H_CLIENT
2  #define _H_CLIENT
3  #include <stdio.h>
4  #include <stdlib.h>
5  #include <sys/socket.h>
```

```
6 #include <netinet/in.h>
7 #include <arpa/inet.h>
8 #include "../../tcp/common/error.h"
9 #include "../../tcp/common/log.h"
10 #include "../../tcp/common/tcp.h"
11 #include "udp_client.h"
12
13 typedef struct client_ipv4 {
14   in_addr_t addr;
15   uint16_t port;
16 }cliente_ipv4;
17
18 typedef struct client_ipv6 {
19   in_addr_t addr;
20   uint32_t port;
21 }cliente_ipv6;
22 /**
23  * funcao que inicia uma conexao, e mantem aberta, com um servidor
24  * @param char *argv[]
25  **/
26 void start_client(int argc, char *argv[]);
27
28 #endif
```

```
1 #include "client.h"
2
3 void start_client(int argc, char *argv[]){
4   struct in_addr *aux = (struct in_addr *)malloc(sizeof(struct
        in_addr));
5   struct sockaddr_in *addr=(struct sockaddr_in *)malloc(sizeof(
        struct sockaddr_in));
6   struct addrinfo *ainfo=(struct addrinfo *)malloc(sizeof(struct
        addrinfo));
7   memset(ainfo,0,sizeof(struct addrinfo));
8   memset(addr,0,sizeof(struct sockaddr_in));
9   memset(aux,0,sizeof(struct in_addr));
10  addr->sin_family=AF_INET;
11  addr->sin_port=htons(atoi(argv[2]));
12  aux->s_addr=inet_addr(argv[1]);
13  addr->sin_addr=*aux;
14  ainfo->ai_addr=(SA*)addr;
15  ainfo->ai_addrlen=(socklen_t)sizeof(struct sockaddr);
16  int socket=openSocket(ainfo);
17  if(argc==3)
18    send_msg(socket,ainfo);
19  else if(argc==4 && strcmp(argv[3],OPT_TESTE)==0){
20    printf("%d %s %s",argc, argv[3],OPT_TESTE);
21    build_teste(socket,ainfo);
22  }
23  close(socket);
24  free(aux);
25  free(addr);
26  free(ainfo);
27 }
```

5.3.2 *udpclient.h udpclient.c*

```
1 #ifndef _H_UDP_CLIENT
2 #define _H_UDP_CLIENT
3 #include <stdio.h>
```

```c
#include <stdlib.h>
#include <stdbool.h>
#include <string.h>
#include <unistd.h>
#include <signal.h>
#include <sys/time.h>
#include <sys/socket.h>
#include <netinet/in.h>
#include <netdb.h>
#include <arpa/inet.h>

#include "../../tcp/common/error.h"
#include "../../tcp/common/log.h"
#include "../../tcp/common/tcp.h"
#include "../../tcp/common/archives.h"
#include "../../tcp/server/login.h"

/**
 * Funcao que abre um socket
 * Utiliza IPv4 (AF_INET) e SOCK_DGRAM, nao orientado a conexao!
 * @example
 * struct sockaddr_in {
 *    sa_family_t    sin_family; address family: AF_INET
 *    in_port_t      sin_port;   port in network byte order
 *    struct in_addr sin_addr;   internet address
 * }
 * @return int sDesc : retorna o socket descriptor
 **/
int openSocket(struct addrinfo *a);
/**
 * Funcao que envia a primeira msg ao servidor. Ha um numero maximo
      de tentativas
 * @see udp.h : UDP_MAX_RETRY
 * @see log.h
 * @param int sDesc : descriptor de um socket
 * @param struct addrinfo *a : info, tal como endereco do server
 **/
void first_msg(int s, AI *a);
/**
 * Funcao que envia ack ao servidor. Nao importa se chegou ou nao
 * @see log.h
 * @param int sDesc : descriptor de um socket
 * @param struct addrinfo *a : info, tal como endereco do server
 **/
void send_ack(int s, AI *a);
/**
 * Funcao que envia e recebe uma mensagem do servidor.
 * @see log.h
 * @param int sDesc : descriptor de um socket
 * @param struct addrinfo *a : info, tal como endereco do server
 **/
void send_msg(int sDesc,AI *a);
/**
 * Roda um teste juntamente ao server
 **/
void build_teste(int sDesc,AI *a);
/**
 * Gera um vetor com opcoes(int) para o teste.
 * @return int *msg;
 **/
int *getTesteMsg();

```

```
65 #endif
```

```
1  #include "udp_client.h"
2
3  int openSocket(struct addrinfo *a){
4    int sDesc;/*socket descriptor*/
5    a->ai_family=AF_INET;/*IPv4*/
6    a->ai_socktype=SOCK_DGRAM;/*upd entao, datagram*/
7    a->ai_protocol=IPPROTO_UDP;
8    sDesc = socket(a->ai_family,a->ai_socktype,a->ai_protocol);/*IPv4
         e upd = datagram*/
9    if(sDesc<0){/*erro?*/
10     printError(ERROR_SOCKET);/*print erro*/
11     exit(1);/*saida inesperada*/
12   }
13   /*nao eh orientado a conexao, como o tcp*/
14   return sDesc;/*retorna sockdescriptor*/
15 }
16
17 void first_msg(int s, AI *a){
18   int max=0;
19   int r=0;
20   do{
21     max = sendto(s,TCP_MSG_HELLO,strlen(TCP_MSG_HELLO),0,a->ai_addr
           ,a->ai_addrlen);
22     if(max==-1){
23       printError(ERROR_SOCKET_SERVER_ERROR);/*imprime erro*/
24       close(s);/*fechar o socket aberto*/
25       exit(1);/*saida com erro*/
26     }
27     if(LOG)
28       writeLog("UDP_CLIENT::send",my_itoa(max,10),inet_ntoa(((SA_IN
             *)a->ai_addr)->sin_addr));
29     r++;
30     if(r>UDP_MAX_RETRY){
31       printError(ERROR_UDP_MAX_RETRY);/*maximo atingido*/
32       close(s);/*fechar o socket aberto*/
33       exit(1);/*saida com erro*/
34     }
35   }while(max<strlen(TCP_MSG_HELLO));/*a msg nao foi devidamente
         entregue?*/
36 }
37
38 void send_ack(int s, AI *a){
39   int max=0;
40   int r=0;
41   do{
42     max = sendto(s,TCP_MSG_ACK,strlen(TCP_MSG_ACK),0,a->ai_addr,a->
           ai_addrlen);
43     if(max==-1){
44       printError(ERROR_SOCKET_SERVER_ERROR);/*imprime erro*/
45       close(s);/*fechar o socket aberto*/
46       exit(1);/*saida com erro*/
47     }
48     r++;
49     if(r>UDP_MAX_RETRY){
50       printError(ERROR_UDP_MAX_RETRY);/*maximo atingido*/
51       close(s);/*fechar o socket aberto*/
52       exit(1);/*saida com erro*/
53     }
54   }while(max<strlen(TCP_MSG_ACK));
55 }
```

```
56
57 void send_msg(int sDesc,AI *a){
58   char msgMaxTCPBuffSize[10];/*msg a ser enviada*/
59   memset(msgMaxTCPBuffSize,'\0',10);/*zerando mem*/
60   first_msg(sDesc,a);/*hello*/
61   int max=0;
62   /*recebendo msg*/
63   max = recvfrom(sDesc,msgMaxTCPBuffSize,10,0,a->ai_addr,&(a->
         ai_addrlen));
64   if(max<0){/*erro?*/
65     printError(ERROR_SOCKET_SERVER_ERROR);/*imprime erro*/
66     close(sDesc);/*fechar o socket aberto*/
67     exit(1);/*saida com erro*/
68   }
69   if(LOG) writeLog("UDP_CLIENT::rcv",msgMaxTCPBuffSize,"");
70   send_ack(sDesc,a);/*ack*/
71   /*-1 error, 0 closed*/
72   msgMaxTCPBuffSize[max]='\0';/*limit msg*/
73   /*msg recebida tera no maximo esse tamanho*/
74   int tcpBuffSize=atoi(msgMaxTCPBuffSize);
75   char msg[TCP_BUF_SIZE];/*msg a ser transmitida*/
76   memset(msg,'\0',TCP_BUF_SIZE);/*zerando mem*/
77   char recvMsg[tcpBuffSize];/*msg a ser recebida*/
78   bool transmit=true;/*var de saida*/
79   printf("Para sair tecle %s\n",TCP_COMMAND_CLOSE_CONNECTION);/*
         requisicao*/
80   int rc=0;/*para verificar se a msg inteira foi recebida*/
81   do{
82     if(strcmp(msg,"4")==0){/*cond. se a msg a ser recebida eh mto
           grande*/
83       do{
84   /*recebendo msg*/
85   max = recvfrom(sDesc,(void *)recvMsg,tcpBuffSize,0,a->ai_addr,&(a
         ->ai_addrlen));
86   if(max<0){/*erro?*/
87     /*qual tipo?*/
88     printError((max==-1)?ERROR_SOCKET_SERVER_ERROR:
           ERROR_SOCKET_SERVER_CLOSED);
89     close(sDesc);/*fechar socket*/
90     exit(1);/*saida inesperada*/
91   }
92   recvMsg[max]='\0';/*limit msg*/
93   printf("%s",recvMsg);/*imprimi msg ja lida do buffer*/
94   if(rc<tcpBuffSize)/*leu a msg por completo?*/
95     sendto(sDesc,TCP_MSG_ACK,strlen(TCP_MSG_ACK),0,a->ai_addr,a->
           ai_addrlen);/*quero mais msg*/
96   rc+=max;/*bom*/
97       }while(rc<tcpBuffSize);/*ja atingimos o tamanho?*/
98     }
99     else{/*msg curta*/
100      if(strlen(msg)==0)/*quero menu?*/
101  sendto(sDesc,TCP_COMMAND_MENU,strlen(TCP_COMMAND_MENU),0,a->
         ai_addr,a->ai_addrlen);
102      /*recebendo menu*/
103      max = recvfrom(sDesc,(void *)recvMsg,tcpBuffSize,0,a->ai_addr
           ,&(a->ai_addrlen));
104      recvMsg[max]='\0';/*limit msg*/
105      printf("%s",recvMsg);/*print msg recebida*/
106    }
107    rc=0;/*zerando auxiliar de msgs grandes*/
108    if(strcmp(TCP_COMMAND_CLOSE_CONNECTION,msg)==0){/*saida?*/
109      transmit=false;/*nao transmitir mais*/
```

```
110       }
111       else{/*msg curta*/
112         do{
113     fgets(msg,TCP_BUF_SIZE-1,stdin);/*lendo input*/
114     msg[strlen(msg)-strlen(CHAR_NEW_LINE)]='\0';/*remove new line*/
115     if(strlen(msg)==0)/*comando valido? ou seja, nao eh um simples \n
            */
116       printf("Digite um comando valido!\n");
117         }while(strlen(msg)==0);/*ateh um comando valido*/
118         /*temos que transmitir a msg de fechar conexao*/
119         max = sendto(sDesc,msg,strlen(msg),0,a->ai_addr,a->ai_addrlen
                );
120         if(max<0){/*msg ok?*/
121     close(sDesc);/*close socket*/
122     printError(ERROR_SOCKET_SERVER_ERROR);/*print error*/
123     exit(1);/*saida inesperada*/
124         }
125       }
126     }while(transmit);/*ateh fim da transmissao*/
127 }
128
129 void build_teste(int sDesc,AI *a){
130     char msgMaxTCPBuffSize[10];
131     memset(msgMaxTCPBuffSize,'\0',10);
132     first_msg(sDesc,a);/*hello*/
133     int max = recvfrom(sDesc,msgMaxTCPBuffSize,10,0,a->ai_addr,&(a->
            ai_addrlen));
134     if(max<0){
135       printError((max==-1)?ERROR_SOCKET_SERVER_ERROR:
              ERROR_SOCKET_SERVER_CLOSED);
136       close(sDesc);
137       exit(1);
138     }
139     msgMaxTCPBuffSize[max]='\0';
140     send_ack(sDesc,a);
141     do{
142       max=sendto(sDesc,TCP_COMMAND_MENU,strlen(TCP_COMMAND_MENU),0,a
              ->ai_addr,a->ai_addrlen);
143       if(max<0){
144         printError((max==-1)?ERROR_SOCKET_SERVER_ERROR:
                ERROR_SOCKET_SERVER_CLOSED);
145         close(sDesc);
146         exit(1);
147       }
148     }while(max<1);
149     int tcpBuffSize=atoi(msgMaxTCPBuffSize);
150     char recvMsg[tcpBuffSize];/*received msg*/
151     struct timeval tvini, tvend;
152     double ini=00.00,end=00.00;
153     int *msgT=getTesteMsg();
154     char testes[7][18];
155     strcpy(testes[0],"./estat/tt_menu\0");
156     strcpy(testes[1],"./estat/tt_1\0");
157     strcpy(testes[2],"./estat/tt_2\0");
158     strcpy(testes[3],"./estat/tt_3\0");
159     strcpy(testes[4],"./estat/tt_4\0");
160     strcpy(testes[5],"./estat/tt_5\0");
161     strcpy(testes[6],"./estat/tt_6\0");
162     char isbn[14][13];
163     strcpy(isbn[0],"978853526-0\0");
164     strcpy(isbn[1],"978052007-4\0");
165     strcpy(isbn[2],"857302773-8\0");
```

```
166     strcpy(isbn[3],"850109104-9\0");
167     strcpy(isbn[4],"843760494-X\0");
168     strcpy(isbn[5],"207036002-4\0");
169     strcpy(isbn[6],"081297215-5\0");
170     strcpy(isbn[7],"978858041-1\0");/*errado*/
171     strcpy(isbn[8],"978858041-1\0");
172     strcpy(isbn[9],"978857679-9\0");
173     strcpy(isbn[10],"978857608-5\0");
174     strcpy(isbn[11],"978853990-1\0");
175     strcpy(isbn[12],"978857521-2\0");
176     strcpy(isbn[13],"978850220-0\0");
177     int i=0,random=0;
178     int rc=0;
179     max = recvfrom(sDesc,(void *)recvMsg,tcpBuffSize,0,a->ai_addr,&(a
            ->ai_addrlen));
180     recvMsg[max]='\0';
181     printf("%s",recvMsg);
182     for(i=0;i<NUM_TESTES*NUM_OPCOES_MENU;i++){
183       rc=0;
184       gettimeofday(&tvini, NULL);/*marca envio*/
185       printf("%d",msgT[i]);
186       do{
187         max = sendto(sDesc,my_itoa(msgT[i],10),strlen(my_itoa(msgT[i
                ],10)),0,a->ai_addr,a->ai_addrlen);
188         if(max<0){
189     printError((max==-1)?ERROR_SOCKET_SERVER_ERROR:
            ERROR_SOCKET_SERVER_CLOSED);
190     close(sDesc);
191     exit(1);
192         }
193       }while(max<1);
194       if(msgT[i]==2 || msgT[i]==3 || msgT[i]==5 || msgT[i]==6){/*
              precisa de isbn*/
195         max = recvfrom(sDesc,recvMsg,tcpBuffSize,0,a->ai_addr,&(a->
                ai_addrlen));
196         recvMsg[max]='\0';
197         printf("%s",recvMsg);
198         if(strcmp(recvMsg,TCP_MSG_ISBN_REQUIRED)==0){/*caso receba
                req. isbn*/
199     random=rand()%13;
200     do{
201       max = sendto(sDesc,isbn[random],strlen(isbn[random]),0,a->
              ai_addr,a->ai_addrlen);
202       if(max<0){
203         printError((max==-1)?ERROR_SOCKET_SERVER_ERROR:
                ERROR_SOCKET_SERVER_CLOSED);
204         close(sDesc);
205         exit(1);
206       }
207     }while(max<1);
208     printf(" SEND %s+ ",isbn[random]);
209     if(msgT[i]==5){/*precisa de senha && qntidade*/
210       max = recvfrom(sDesc,recvMsg,tcpBuffSize,0,a->ai_addr,&(a->
              ai_addrlen));
211       recvMsg[max]='\0';
212       printf("%s",recvMsg);
213       if(strcmp(recvMsg,TCP_MSG_LOGIN_REQUIRED)==0){/*caso receba req
              . login*/
214         do{
215           max = sendto(sDesc,SENHA,strlen(SENHA),0,a->ai_addr,a->
                ai_addrlen);
216           if(max<0){
```

```
217       printError((max==-1)?ERROR_SOCKET_SERVER_ERROR:
              ERROR_SOCKET_SERVER_CLOSED);
218       close(sDesc);
219       exit(1);
220            }
221         }while(max<1);
222         printf(" SEND %s+ ",SENHA);
223         max = recvfrom(sDesc,recvMsg,tcpBuffSize,0,a->ai_addr,&(a->
                ai_addrlen));
224         recvMsg[max]='\0';
225         printf("%s",recvMsg);
226         if(strcmp(recvMsg,TCP_MSG_QTD_REQUIRED)==0){/*qntidade*/
227           random=rand()%1000;
228           do{
229       max = sendto(sDesc,my_itoa(random,10),strlen(my_itoa(random,10)
                ),0,a->ai_addr,a->ai_addrlen);
230       if(max<0){
231         printError((max==-1)?ERROR_SOCKET_SERVER_ERROR:
                ERROR_SOCKET_SERVER_CLOSED);
232         close(sDesc);
233         exit(1);
234       }
235            }while(max<1);
236            printf(" SEND %s+ ",my_itoa(random,10));
237            max = recvfrom(sDesc,recvMsg,tcpBuffSize,0,a->ai_addr,&(a->
                ai_addrlen));
238            recvMsg[max]='\0';
239         }/*qntd*/
240       }/*ogin*/
241     }
242     else{
243       max = recvfrom(sDesc,recvMsg,tcpBuffSize,0,a->ai_addr,&(a->
              ai_addrlen));
244       recvMsg[max]='\0';
245     }
246         }/*isbn*/
247         printf("@%s@",recvMsg);
248       }
249       else if(msgT[i]==4){
250         do{
251     max = recvfrom(sDesc,recvMsg,tcpBuffSize,0,a->ai_addr,&(a->
            ai_addrlen));
252     if(max<0){
253       printError((max==-1)?ERROR_SOCKET_SERVER_ERROR:
              ERROR_SOCKET_SERVER_CLOSED);
254         close(sDesc);
255         exit(1);
256     }
257     recvMsg[max]='\0';
258     printf("@%s@",recvMsg);
259     if(rc==0){
260       int maxaux=0;
261       do{
262         maxaux=sendto(sDesc,TCP_MSG_ACK,strlen(TCP_MSG_ACK),0,a->
                ai_addr,a->ai_addrlen);/*send message*/
263         if(maxaux<0){
264           printError((maxaux==-1)?ERROR_SOCKET_SERVER_ERROR:
                  ERROR_SOCKET_SERVER_CLOSED);
265           close(sDesc);
266           exit(1);
267         }
268       }while(maxaux<1);
```

```
269     }
270     rc+=max;
271         }while(rc<tcpBuffSize);
272         }
273       else{
274         max = recvfrom(sDesc,(void *)recvMsg,tcpBuffSize,0,a->ai_addr
                ,&(a->ai_addrlen));
275         recvMsg[max]='\0';
276         printf("@%s@",recvMsg);
277       }
278       gettimeofday(&tvend, NULL);/*marca recebimento*/
279       ini=DOUBLE_MILHAO*(double)(tvini.tv_sec)+(double)(tvini.tv_usec
              );
280       end=DOUBLE_MILHAO*(double)(tvend.tv_sec)+(double)(tvend.tv_usec
              );
281       writeDoubleToFile(testes[msgT[i]],"a+",end-ini);
282     }
283 }
284 int *getTesteMsg(){
285     int *msg=(int *)malloc(sizeof(int)*NUM_OPCOES_MENU*NUM_TESTES);
286     int i=0,j=0,c=0;
287     for(i=0;i<NUM_TESTES;i++){
288       for(j=0;j<NUM_OPCOES_MENU;j++){
289         *(msg+c)=j+1;
290         c++;
291       }
292     }
293     return msg;
294 }
```