

MC823 - Relatório - Servidor Interativo sobre UDP

André Nakagaki Fillettaz RA: 104595

Guilherme Alcarde Gallo RA: 105008

9 de Maio de 2013

Conteúdo

1	UDP x TCP	2
2	Servidor	3
2.1	Operações	3
2.1.1	Listagem Geral	3
2.1.2	Listagem Específica	4
2.1.3	Autenticação do Cliente Livraria e Mudança no estoque	5
2.2	Contando o tempo	5
2.3	Código Completo	6
3	Cliente	13
3.1	Explicações das Funções Principais	18
3.2	A Estrutura addrinfo	18
4	Análise de Dados	19
4.1	Servidor e Cliente UDP	19
4.2	Comparando UDP e TCP	21
5	Conclusão	23
6	Referências Bibliográficas	24

Capítulo 1

UDP x TCP

Antes de entrar em explicações e códigos do projeto client-server UDP implementado, convém uma análise prévia de algumas diferenças entre os dois protocolos, e as decisões de projeto tomadas em função dessas diferenças.

O UDP é um protocolo de transporte livre de conexão, o que já difere do TCP. Como não há nenhum tipo de orientação a conexão (apesar da função `connect()` ainda estar disponível para este tipo de protocolo), o servidor UDP não utiliza processos filhos para tratar do envio de dados. Existe apenas um processo que é responsável tanto pelo tratamento da requisição quanto do envio de dados.

Outra diferença decorrente disso é o envio, baseado em datagramas ao invés de stream. Isso altera em grande parte a composição do socket, e da chamada da função `socket()`, conforme á visto nos códigos.

Por fim, já que o client e o servidor nunca estão de fato conectados, as funções de envio e recepção de dados devem ser mudadas para *sendto* e *recvfrom*. A diferença fundamental destas é que como não existe conexão, elas exigem por parâmetro os dados do endereço (tipo e tamanho). Isso é razoável e na verdade torna muito difícil de garantir que haverá comunicação entre os processos client-server, sendo possível executar o client por exemplo sem o server estar aberto (ocasionando obviamente erros e bugs).

O UDP não é confiável, ou seja, não há garantia de entrega de datagramas, o que exige que a aplicação implemente essa “confiabilidade”.

Esse comportamento não confiável e não orientado à conexão torna o protocolo UDP no geral mais rápido do que o TCP, que checa a ordem de envio e procura por erros, conforme é visto na análise de dados.

Capítulo 2

Servidor

Processo responsável por tratar do banco de dados, tratar requisições e enviar dados ao client.

Para tratar do banco de dados, novamente foi utilizada a biblioteca do C, SQLite3. Conforme explicado no relatório do projeto 01. Não houveram grandes mudanças nessa parte do projeto, e as funções que realizam a chamada ao banco continuam as mesmas.

2.1 Operações

Cada operação é representada por um inteiro no servidor:

1	Listar ISBN e Título de todos os livros
2	Dado o ISBN de um livro, retornar sua descrição
3	Dado o ISBN de um livro, retornar todas as suas informações
4	Listar todas as informações de todos os livros
5	Atualiza estoque, caso seja cliente livraria
6	Dado o ISBN de um livro, retorna seu estoque
7	Dado a senha correta, é iniciada a seção do cliente livraria

2.1.1 Listagem Geral

Aquitratam-se as operações de listagem geral. Dentre elas destacam-se entao as operações (1) e (4).

A ideia aqui utiliza a biblioteca do SQLite3 e através de queries consulta-se o banco. Convém notar que nesse caso, a única real computação que é deixada para a função main é realizar a computação de tempo, conforme explicado a parte.

A própria resposta da Query é então enviada ao client. Como exemplo, a operação (1) que lista todos os ISBNs e seus respectivos livros na biblioteca:

```
1 case 1: // Lista de ISBN e titulo dos livros
    elapsed = 0;
3 // Enviando tuplas de ISBN e titulo de todos os livros
    rc = sqlite3_exec(db, "select ISBN10,titulo from livro;", callback ,
5     0, &zErrMsg);
    // Tempo percorrido ate agora
7 gettimeofday(&t1, 0);
    if (rc != SQLITE_OK) {
9         sqlite3_free(zErrMsg);
    }

11
12 // Finalizando a mensagem
13 // Calculo do tempo de operacao
    elapsed = (t1.tv_sec - t0.tv_sec) * 1000000 + t1.tv_usec
15     - t0.tv_usec;
    // Transformando em string com caracteres de "seguranca" para postumo atoi
17 sprintf(query, "%6li#", elapsed); // Caractere # e um identificador de fim da mensagem
    // DEBUG
19 printf("\nOperation Time: %s\n\n", query);
    // Calculando o tamanho
21 length = strlen(query);
    // Finalmente envia para o cliente
23 // sendall(client_sock, query, &length);
    if ((numbytes = sendto(sockfd, query, strlen(query), 0, (struct sockaddr *) &their_addr, addr_len)
25     ) == -1) {
        perror("server: sendto");
        exit(1);
27 }
    break;
```

2.1.2 Listagem Específica

Aqui explica-se o funcionamento do segundo tipo de operação, as listagem específicas.

Ao contrário da geral, nesse caso, o client deve enviar ao servidor mais do que simplesmente o número da operação solicitada, já que, nesses tipos de operações, também é necessário o número ISBN. Desse forma, são necessários duas operações de envio por parte do client. Além disso, é necessário mais do que simplesmente realizar a query, verificar se o ISBN realmente consta no Banco de Dados.

Para exemplificar esse processo de checagem e envio, o código da operação (3), listagem de informações:

```
case 3:
    elapsed = 0;
    // Esperando o cliente mandar o ISBN desejado
    // Tempo percorrido ate agora
    gettimeofday(&t1, 0);
    // ReCalculo do tempo de operacao
    elapsed += (t1.tv_sec - t0.tv_sec) * 1000000 + t1.tv_usec
               - t0.tv_usec;
    // Esperando o cliente mandar o ISBN desejado
    if ((read_size = recvfrom(sockfd, client_message, 2000, 0,
                              (struct sockaddr *) &their_addr, &addr_len)) == -1) {
    }
    gettimeofday(&t0, 0);
    // Montando a query

    strcpy(query,
           "select l.ISBN10, l.titulo, a.autor, a.autor2, a.autor3, a.autor4, l.descricao, l.editora, l.
           ano, l.estoque from livro l, autor a where l.autores=a.a_id and ISBN10 = ");
    strcpy(query2, "select count(*) from livro where ISBN10 = ");
    // Concatenando o ISBN
    strcat(query, client_message);
    strcat(query2, client_message);
    // Fim do comando SQLite
    strcat(query, ";");

    // Verificando se ha livros
    // callbackSilent nao envia dados ao cliente
    // existe recebe o resultado de contagem de livros (0 ou 1)
    rc = sqlite3_exec(db, query2, callbackSilent, existe, &zErrMsg);
    if (*((int *) existe) == 0) {
        length = 41;
        // sendall(client_sock, "\nEste ISBN nao consta na nossa livraria!\n",&length);
        gettimeofday(&t1, 0);
        if ((numbytes = sendto(sockfd,
                              "\nEste ISBN nao consta na nossa livraria!\n", 42,
                              0, (struct sockaddr *) &their_addr, addr_len))
            == -1) {
            perror("server: sendto");
            exit(1);
        }
    }
    // Tempo percorrido ate agora
} else {
    // Executando query - Callback ja faz os sends
    rc = sqlite3_exec(db, query, callbackFmt, 0, &zErrMsg);
    // Tempo percorrido ate agora
    gettimeofday(&t1, 0);

    // Fim da mensagem
    // ReCalculo do tempo de operacao
    elapsed += (t1.tv_sec - t0.tv_sec) * 1000000 + t1.tv_usec
               - t0.tv_usec;
    // Transformando em string com chars de "seguranca" para postumo atoi
    sprintf(query, "          %6li#", elapsed);
    // DEBUG
    printf("\nOperation Time: %s\n\n", query);
    // Calculando o tamanho
    length = strlen(query);
    // Finalmente envia para o cliente
    // sendall(client_sock, query, &length);
    if ((numbytes = sendto(sockfd, query, strlen(query), 0,
                          (struct sockaddr *) &their_addr, addr_len)) == -1) {
        perror("server: sendto");
        exit(1);
    }
}
break;
```

2.1.3 Autenticação do Cliente Livraria e Mudança no estoque

Considera-se aqui que o client normal não tem acesso a operação de mudança de estoque, apesar de poder verificar a quantidade de livros em estoque (dado um determinado ISBN).

Caso o cliente queira mudar para o modo Livraria (Superuser) o que ele deve fazer é requisitar essa mudança e então enviar o password, que é definido no processo servidor. Aqui poderíamos expandir ainda mais o conceito, e desenvolver livrarias com bancos de dados específicos e para tanto, colocar a senha do user livraria no próprio banco de dados.

2.2 Contando o tempo

O tempo de comunicação é contado a partir do *sendall* dos callbacks, pela correção no tempo gasto pelas operações feitas no próprio servidor, para que o cliente tenha informações corretas do tempo gasto somente na comunicação, e pelo respectivo *receive* acionado pelo cliente.

Para contar o tempo, utilizou-se a função *gettimeofday* e duas variáveis do tipo *struct timeval*, ambos da biblioteca *time.h*.

A contagem de tempo funciona como uma sequencia de cronometragens. No servidor, estas ignoram o tempo gasto com *receives* e contam apenas o tempo das *operações*, já – no cliente – a sequencia de cronometragens é cautelosa para apenas medir o tempo dos *receives*.

A ideia de cronometragem vem de uma subtração de tempo *final – inicial*, calculado em milisegundos, pela fórmula dada pela equação:

$$elapsed+ = (t_{final}.tv_sec - t_{inicial}.tv_sec) * 1000000 + t_{final}.tv_usec - t_{inicial}.tv_usec; \quad (2.1)$$

Para exemplificar, o código da operação 2:

```
1 case 2: // Descricao de um livro
2     elapsed = 0;
3     // Esperando o cliente mandar o ISBN desejado
4     // Tempo percorrido ate agora
5     gettimeofday(&t1, 0);
6     // ReCalculo do tempo de operacao
7     elapsed += (t1.tv_sec - t0.tv_sec) * 1000000 + t1.tv_usec
8         - t0.tv_usec;
9
10    printf("\nISBN1:%s\n", client_message);
11    if ((numbytes = recvfrom(sockfd, client_message, MAXBUFLen - 1, 0,
12        (struct sockaddr *) &their_addr, &addr_len)) == -1) {
13        perror("recvfrom");
14        exit(1);
15    }
16    gettimeofday(&t0, 0);
17    // Montando a query
18    strcpy(query, "select descricao from livro where ISBN10 = ");
19    strcpy(query2,
20        "select count(descricao) from livro where ISBN10 = ");
21    printf("\nISBN2:%s\n", client_message);
22    // Concatenando o ISBN
23    strcat(query, client_message);
24    strcat(query2, client_message);
25    // Fim do comando SQLite
26    strcat(query, ";");
27
28    // Verificando se ha livros
29    // callbackSilent nao envia dados ao cliente
30    // existe recebe o resultado de contagem de livros (0 ou 1)
31    rc = sqlite3_exec(db, query2, callbackSilent, existe, &zErrMsg);
32    // Tempo percorrido ate agora
33    gettimeofday(&t1, 0);
34    if (*((int *) existe) == 0) {
35        // sendall(client_sock, "\nEste ISBN nao consta na nossa livraria!\n",&length);
36        if ((numbytes = sendto(sockfd,
37            "\nEste ISBN nao consta na nossa livraria!\n", 42,
38            0, (struct sockaddr *) &their_addr, addr_len))
39            == -1) {
40            perror("server: sendto");
41            exit(1);
42        }
43    } else
44        // Executando query - Callback ja faz os sends
45        rc = sqlite3_exec(db, query, callback, 0, &zErrMsg);
46    gettimeofday(&t1, 0);
47    // Fim da mensagem
48    // ReCalculo do tempo de operacao
49    elapsed += (t1.tv_sec - t0.tv_sec) * 1000000 + t1.tv_usec
50        - t0.tv_usec;
51    // Transformando em string com chars de "seguranca" para postumo atoi
52    sprintf(query, "%6li#", elapsed);
53    // DEBUG
54    printf("\nOperation Time: %s\n\n", query);
```

```

55     // Calculando o tamanho
length = strlen(query);
57     // Finalmente envia para o cliente
//sendall(client_sock, query, &length);
59     if ((numbytes = sendto(sockfd, query, strlen(query), 0,
        (struct sockaddr *) &their_addr, addr_len)) == -1) {
61         perror("server: sendto");
        exit(1);
63     }
    break;

```

2.3 Código Completo

Assim como no TCP, o servidor UDP foi fortemente baseado no Beej's Guide to Network Programming.

```

/*
2  ** listener.c — a datagram sockets "server" demo
*/

4
#include <stdio.h>
6 #include <stdlib.h>
#include <unistd.h>
8 #include <errno.h>
#include <string.h>
10 #include <sys/types.h>
#include <sys/socket.h>
12 #include <netinet/in.h>
#include <arpa/inet.h>
14 #include <netdb.h>

16 #define MYPORT "4950" // the port users will be connecting to
#define MAXBUFLEN 100

18
// SQLite3
20 #include <sqlite3.h>

22 #define PASSWORD "numaPistacheCottapie"
int sockfd;
24 struct sockaddr_storage their_addr;
socklen_t addr_len;
26 int numbytes;

28 typedef struct livro {
    char i[20]; // ISBN
30     char q[4]; // Stock Quantity
} Livro;

32
int socket_desc, client_sock, c, read_size, connfd;
34
// Tempo
36 long elapsed = 0; // Conta o tempo percorrido
struct timeval t0, t1;
38
// get sockaddr, IPv4 or IPv6:
40 void *get_in_addr(struct sockaddr *sa) {
    if (sa->sa_family == AF_INET) {
42         return &(((struct sockaddr_in*) sa)->sin_addr);
    }

44     return &(((struct sockaddr_in6*) sa)->sin6_addr);
46 }

48 // Callback do SQLite3. Versao silenciosa, nao envia nada para o cliente
static int callbackSilent(void *NotUsed, int argc, char **ans,
    char **azColName) {
50     int i, num;
52     int *v;
    v = (int *) alloca(sizeof(int));
54     // Aaaaaaah Moleque!!!!
    v = NotUsed;
56     *v = atoi(ans[0]);
    // printf("\n-----\n%d\n-----\n", *v);
58     return 0;
}

60
// Callback do SQLite3. Versao formatada para varios detalhes
62 static int callbackFmt(void *NotUsed, int argc, char **ans, char **azColName) {
    int i, num;
64     char aux[20000];

```

```

66 // Nome da coluna
strcpy(aux, azColName[0]);
68 strcat(aux, ": ");
// Concatena valor
70 strcat(aux, ans[0]);
strcat(aux, "\n");
72 // Formata
for (i = 1; i < argc; i++) { // Fazendo isso para o resto da tupla
74     if (ans[i]) {
        strcat(aux, azColName[i]);
        strcat(aux, ": ");
        // Melhorando a visualizacao da descricao
76         if (i == 3) {
            strcat(aux, "\n\t");
80         }
        strcat(aux, ans[i]);
82         strcat(aux, "\n");
    }
84 }
strcat(aux, "\n");
86 // DEBUG
printf("%s", aux);
88 // Enviando para o cliente
num = strlen(aux);
gettimeofday(&t1, 0);
elapsed += (t1.tv_sec-t0.tv_sec)*1000000 + t1.tv_usec-t0.tv_usec;
92 if ((numbytes = sendto(sockfd, aux, strlen(aux), 0,
    (struct sockaddr *) &their_addr, addr_len)) == -1) {
94     perror("server: sendto");
    exit(1);
96 }
gettimeofday(&t0, 0);
98 return 0;
}

100 // Callback formatado em tupla simples
102 static int callback(void *NotUsed, int argc, char **ans, char **azColName) {
    int i, num;
104     char aux[20000];

    strcpy(aux, ans[0]);
    for (i = 0; i < argc; i++) {
106         if (i && ans[i]) {
            strcat(aux, " | ");
108             strcat(aux, ans[i]);
110         }
    }
112     strcat(aux, "\n");
    printf("Tamanho= %d\n%s", (int)strlen(aux), aux);
    num = strlen(aux);
    gettimeofday(&t1, 0);
    elapsed += (t1.tv_sec-t0.tv_sec)*1000000 + t1.tv_usec-t0.tv_usec;
118     if ((numbytes = sendto(sockfd, aux, strlen(aux), 0,
        (struct sockaddr *) &their_addr, addr_len)) == -1) {
120         perror("server: sendto");
        exit(1);
122     }
    gettimeofday(&t0, 0);
124     return 0;
}

126 int main(void) {
128     struct addrinfo hints, *servinfo, *p;
    int rv;
130     char buf[MAXBUFLen];
    char s[INET6_ADDRSTRLEN];

132

134     char client_message[2000], query[2500], query2[2500];

136     int opcao;

138     // SQLite3
    sqlite3 *db;
140     char *zErrMsg = 0, *msg;
    int rc;
142     void *existe; // Para requisicoes invalidas
    int superuser = 0; // Cliente Livraria
144     existe = (void *) alloca(sizeof(int)); // Usado para saber se o ISBN existe
    Livro cm;

146

148     // Timeout setado para imprevistos ...
    struct timeval tv;

```



```

150 tv.tv_sec = 5; /* 30 Secs Timeout */
151 tv.tv_usec = 0; // Not init'ing this can cause strange errors
152
153 setsockopt(connfd, SOL_SOCKET, SO_RCVTIMEO, (char *) &tv,
154           sizeof(struct timeval));
155 // ... Timeout setado.
156
157 // Abrindo Banco de Dados do SQLite3
158 rc = sqlite3_open("livraria2.db", &db);
159
160 memset(&hints, 0, sizeof hints);
161 hints.ai_family = AF_UNSPEC; // set to AF_INET to force IPv4
162 hints.ai_socktype = SOCK_DGRAM;
163 hints.ai_flags = AI_PASSIVE; // use my IP
164
165 // Servidor interativo
166 while (1) {
167     if ((rv = getaddrinfo(NULL, MYPORT, &hints, &servinfo)) != 0) {
168         fprintf(stderr, "getaddrinfo: %s\n", gai_strerror(rv));
169         return 1;
170     }
171
172     // loop through all the results and bind to the first we can
173     for (p = servinfo; p != NULL; p = p->ai_next) {
174         if ((sockfd = socket(p->ai_family, p->ai_socktype, p->ai_protocol))
175             == -1) {
176             perror("listener: socket");
177             continue;
178         }
179
180         if (bind(sockfd, p->ai_addr, p->ai_addrlen) == -1) {
181             close(sockfd);
182             perror("listener: bind");
183             continue;
184         }
185
186         break;
187     }
188
189     if (p == NULL) {
190         fprintf(stderr, "listener: failed to bind socket\n");
191         return 2;
192     }
193
194     freeaddrinfo(servinfo);
195
196     printf("listener: waiting to recvfrom...\n");
197
198     addr_len = sizeof their_addr;
199     if ((numbytes = recvfrom(sockfd, buf, MAXBUFLEN - 1, 0,
200                             (struct sockaddr *) &their_addr, &addr_len)) == -1) {
201         perror("recvfrom");
202         exit(1);
203     }
204     // Comecando a contar o tempo de operacao
205     gettimeofday(&t0, 0);
206
207     printf("listener: got packet from %s\n",
208           inet_ntop(their_addr.ss_family,
209                     get_in_addr((struct sockaddr *) &their_addr), s,
210                     sizeof s));
211
212     printf("listener: packet is %d bytes long\n", numbytes);
213     buf[numbytes] = '\0';
214     printf("listener: packet contains \"%s\"\n", buf);
215
216     opcao = atoi(buf);
217
218     switch (opcao) {
219         case 1: // Lista de ISBN e titulo dos livros
220             elapsed = 0;
221             // Enviando tuplas de ISBN e titulo de todos os livros
222             rc = sqlite3_exec(db, "select ISBN10,titulo from livro;", callback,
223                               0, &zErrMsg);
224             // Tempo percorrido ate agora
225             gettimeofday(&t1, 0);
226             if (rc != SQLITE_OK) {
227                 sqlite3_free(zErrMsg);
228             }
229
230             // Finalizando a mensagem
231             // Calculo do tempo de operacao
232             elapsed = (t1.tv_sec - t0.tv_sec) * 1000000 + t1.tv_usec
233                     - t0.tv_usec;

```

```

234 // Transformando em string com caracteres de "seguranca" para postumo atoi
sprintf(query, "                %6li^D", elapsed); // Caractere ^D e um identificador de fim da
        mensagem
236 // DEBUG
printf("\nOperation Time: %s\n\n", query);
238 // Calculando o tamanho
// Finalmente envia para o cliente
240 if ((numbytes = sendto(sockfd, query, strlen(query), 0, (struct sockaddr *) &their_addr, addr_len)
        ) == -1) {
242     perror("server: sendto");
        exit(1);
244 }
break;

246 /*****/

248 case 2: // Descricao de um livro
        elapsed = 0;
250 // Esperando o cliente mandar o ISBN desejado
// Tempo percorrido ate agora
252 gettimeofday(&t1, 0);
// ReCalculo do tempo de operacao
254 elapsed += (t1.tv_sec - t0.tv_sec) * 1000000 + t1.tv_usec
        - t0.tv_usec;

256 printf("\nISBN1:%s\n", client_message);
258 if ((numbytes = recvfrom(sockfd, client_message, MAXBUFLen - 1, 0,
        (struct sockaddr *) &their_addr, &addr_len)) == -1) {
260     perror("recvfrom");
        exit(1);
262 }
gettimeofday(&t0, 0);
264 // Montando a query
strcpy(query, "select descricao from livro where ISBN10 = ");
266 strcpy(query2,
        "select count(descricao) from livro where ISBN10 = ");
268 printf("\nISBN2:%s\n", client_message);
// Concatenando o ISBN
270 strcat(query, client_message);
strcat(query2, client_message);
272 // Fim do comando SQLite
strcat(query, ";");

274 // Verificando se ha livros
// callbackSilent nao envia dados ao cliente
// existe recebe o resultado de contagem de livros (0 ou 1)
278 rc = sqlite3_exec(db, query2, callbackSilent, existe, &zErrMsg);
// Tempo percorrido ate agora
// gettimeofday(&t1, 0);
280 if (*((int *) existe) == 0) {
282     if ((numbytes = sendto(sockfd,
        "\nEste ISBN nao consta na nossa livraria!\n", 42,
284         0, (struct sockaddr *) &their_addr, addr_len))
        == -1) {
286         perror("server: sendto");
            exit(1);
288     }
} else
290 // Executando query - Callback ja faz os sends
rc = sqlite3_exec(db, query, callback, 0, &zErrMsg);
gettimeofday(&t1, 0);
292 // Fim da mensagem
// ReCalculo do tempo de operacao
294 elapsed += (t1.tv_sec - t0.tv_sec) * 1000000 + t1.tv_usec
        - t0.tv_usec;
// Transformando em string com chars de "seguranca" para postumo atoi
298 sprintf(query, "                %6li^D", elapsed);
// DEBUG
printf("\nOperation Time: %s\n\n", query);
// Calculando o tamanho
302 // Finalmente envia para o cliente
if ((numbytes = sendto(sockfd, query, strlen(query), 0,
304     (struct sockaddr *) &their_addr, addr_len)) == -1) {
        perror("server: sendto");
        exit(1);
306 }
308 break;

310 /*****/

312 case 3: // Todas as informacoes de um livro
        elapsed = 0;
314 // Esperando o cliente mandar o ISBN desejado

```

```

316 // Tempo percorrido ate agora
317 gettimeofday(&t1, 0);
318 // ReCalculo do tempo de operacao
319 elapsed += (t1.tv_sec - t0.tv_sec) * 1000000 + t1.tv_usec
320           - t0.tv_usec;
321 // Esperando o cliente mandar o ISBN desejado
322 if ((read_size = recvfrom(sockfd, client_message, 2000, 0,
323                          (struct sockaddr *) &their_addr, &addr_len)) == -1) {
324 }
325 gettimeofday(&t0, 0);
326 // Montando a query
327
328 strcpy(query,
329        "select l.ISBN10, l.titulo, a.autor, a.autor2, a.autor3, a.autor4, l.descricao, l.editora, l.
330           ano, l.estoque from livro l, autor a where l.autores=a.a_id and ISBN10 = ");
331 strcpy(query2, "select count(*) from livro where ISBN10 = ");
332 // Concatenando o ISBN
333 strcat(query, client_message);
334 strcat(query2, client_message);
335 // Fim do comando SQLite
336 strcat(query, ";");
337
338 // Verificando se ha livros
339 // callbackSilent nao envia dados ao cliente
340 // existe recebe o resultado de contagem de livros (0 ou 1)
341 rc = sqlite3_exec(db, query2, callbackSilent, existe, &zErrMsg);
342 if (*(int *) existe) == 0 {
343     gettimeofday(&t1, 0);
344     if ((numbytes = sendto(sockfd,
345                          "\nEste ISBN nao consta na nossa livraria!\n", 42,
346                          0, (struct sockaddr *) &their_addr, addr_len))
347         == -1) {
348         perror("server: sendto");
349         exit(1);
350     }
351     // Tempo percorrido ate agora
352 } else {
353     // Executando query - Callback ja faz os sends
354     rc = sqlite3_exec(db, query, callbackFmt, 0, &zErrMsg);
355     // Tempo percorrido ate agora
356     gettimeofday(&t1, 0);
357 }
358
359 // Fim da mensagem
360 // ReCalculo do tempo de operacao
361 elapsed += (t1.tv_sec - t0.tv_sec) * 1000000 + t1.tv_usec
362           - t0.tv_usec;
363 // Transformando em string com chars de "seguranca" para postumo atoi
364 sprintf(query, "          %6li^D", elapsed);
365 // DEBUG
366 printf("\nOperation Time: %s\n\n", query);
367 // Calculando o tamanho
368 // Finalmente envia para o cliente
369 if ((numbytes = sendto(sockfd, query, strlen(query), 0,
370                      (struct sockaddr *) &their_addr, addr_len)) == -1) {
371     perror("server: sendto");
372     exit(1);
373 }
374 break;
375
376 case 4: // Todas as informacoes de todos os livros
377     elapsed = 0;
378     // Executando query
379     rc =
380         sqlite3_exec(db,
381                     "select l.ISBN10, l.titulo, a.autor, a.autor2, a.autor3, a.autor4, l.descricao, l.editora, l.
382                        .ano, l.estoque from livro l, autor a where l.autores=a.a_id;",
383                     callbackFmt, 0, &zErrMsg);
384
385     // Fim da mensagem
386     // Tempo percorrido ate agora
387     gettimeofday(&t1, 0);
388     // Calculo do tempo de operacao
389     elapsed = (t1.tv_sec - t0.tv_sec) * 1000000 + t1.tv_usec
390             - t0.tv_usec;
391     // Transformando em string com chars de "seguranca" para postumo atoi
392     sprintf(query, "          %6li^D", elapsed);
393     // DEBUG
394     printf("\nOperation Time: %s\n\n", query);
395     // Calculando o tamanho
396     // Finalmente envia para o cliente
397     if ((numbytes = sendto(sockfd, query, strlen(query), 0,
398                          (struct sockaddr *) &their_addr, addr_len)) == -1) {
399         perror("server: sendto");

```

```

398     exit(1);
399 }
400
401 break;
402
403 /*****
404
405 case 5:    // Atualizar estoque
406     elapsed = 0;
407     // Tempo percorrido ate agora
408     gettimeofday(&t1, 0);
409     // ReCalculo do tempo de operacao
410     elapsed += (t1.tv_sec - t0.tv_sec) * 1000000 + t1.tv_usec
411               - t0.tv_usec;
412     if ((read_size = recvfrom(sockfd, &cm, 2000, 0,
413                               (struct sockaddr *) &their_addr, &addr_len)) == -1) {
414         perror("server: sendto");
415         exit(1);
416     }
417     gettimeofday(&t0, 0);
418     if (superuser) {
419         // Montando a query
420         strcpy(query, "update livro set estoque = ");
421         // Concatenando o nova quantidade do estoque
422         strcat(query, cm.q);
423         // Concatenando o ISBN
424         strcat(query, " where ISBN10 = ");
425         strcat(query, cm.i);
426         // Fim do comando SQLite
427         strcat(query, ";");
428         printf("%s\n", query);
429         // Executando query - Callback ja faz os writes
430         rc = sqlite3_exec(db, query, callback, 0, &zErrMsg);
431     } else {
432         if ((numbytes = sendto(sockfd, "Sem permissoes para modificar estoque!\n", 42, 0,
433                               (struct sockaddr *) &their_addr, addr_len)) == -1) {
434             perror("server: sendto");
435             exit(1);
436         }
437     }
438 }
439
440 // Fim da mensagem
441 // Tempo percorrido ate agora
442 gettimeofday(&t1, 0);
443 // ReCalculo do tempo de operacao
444 elapsed += (t1.tv_sec - t0.tv_sec) * 1000000 + t1.tv_usec
445           - t0.tv_usec;
446 // Transformando em string com chars de "seguranca" para postumo atoi
447 sprintf(query, "          %6li^D", elapsed);
448 // DEBUG
449 printf("\nTime: %s\n\n", query);
450 // Calculando o tamanho
451 // Finalmente envia para o cliente
452 if ((numbytes = sendto(sockfd, query, strlen(query), 0,
453                       (struct sockaddr *) &their_addr, addr_len)) == -1) {
454     perror("server: sendto");
455     exit(1);
456 }
457
458 break;
459
460 case 6:    // Mostra estoque de um livro
461     elapsed = 0;
462     // Tempo percorrido ate agora
463     gettimeofday(&t1, 0);
464     // ReCalculo do tempo de operacao
465     elapsed += (t1.tv_sec - t0.tv_sec) * 1000000 + t1.tv_usec - t0.tv_usec;
466     // Esperando o cliente mandar o ISBN desejado
467     if ((numbytes = recvfrom(sockfd, client_message, MAXBUFLen - 1, 0,
468                             (struct sockaddr *) &their_addr, &addr_len)) == -1) {
469         perror("recvfrom");
470         exit(1);
471     }
472     gettimeofday(&t0, 0);
473     // Montando a query
474     strcpy(query, "select estoque from livro where ISBN10 = ");
475     // Concatenando o ISBN
476     strcat(query, client_message);
477     // Fim do comando SQLite
478     strcat(query, ";");
479     printf("\n%s\n", query);
480     // Executando query - Callback ja faz os writes
481     rc = sqlite3_exec(db, query, callbackFmt, 0, &zErrMsg);

```

```

482 // Fim da mensagem
483 // Tempo percorrido ate agora
484 gettimeofday(&t1, 0);
485 // ReCalculo do tempo de operacao
486 elapsed += (t1.tv_sec-t0.tv_sec)*1000000 + t1.tv_usec-t0.tv_usec;
487 // Transformando em string com chars de "seguranca" para postumo atoi
488 sprintf(query,"                %6li^D",elapsed);
489 // DEBUG
490 printf("\nTime: %s\n\n",query);
491 // Calculando o tamanho
492 // Finalmente envia para o cliente
493 if ((numbytes = sendto(sockfd, query, strlen(query), 0,
494         (struct sockaddr *) &their_addr, addr_len)) == -1) {
495     perror("server: sendto");
496     exit(1);
497 }
498 break;
499 case 7: // Autentica o cliente livraria
500 // Recebe a senha
501 if ((numbytes = recvfrom(sockfd, client_message, 50, 0,
502         (struct sockaddr *) &their_addr, &addr_len)) == -1) {
503     perror("recvfrom");
504     exit(1);
505 }
506 // Compara as senhas
507 if( (strcmp(client_message,PASSWORD) ) == 0) {
508     superuser = 1; // Sessao de superusuario
509     if ((numbytes = sendto(sockfd, "Bem-vindo, Chuck Norris!\n\n", 31, 0,
510         (struct sockaddr *) &their_addr, addr_len)) == -1) {
511         perror("server: sendto");
512         exit(1);
513     }
514 }
515 else {
516     superuser = 0; // Usuario invalido
517     if ((numbytes = sendto(sockfd, "Senha Invalida!\n\n", 18, 0,
518         (struct sockaddr *) &their_addr, addr_len)) == -1) {
519         perror("server: sendto");
520         exit(1);
521     }
522 }
523 memset(client_message,0,strlen(client_message));
524 break;
525 }
526 close(sockfd);
527 }
528 return 0;
529 }

```

Capítulo 3

Cliente

O processo client tem por responsabilidade receber e lidar com as requisições por parte do user do sistema e fazer essas requisições para um servidor que opere em protocolo UDP.

Inicia apresentando as opções ao usuario, e depois disso espera por uma opção de entrada. Ao receber essa opção, novamente através da função dataFetch (agora modificada para operar em protocolo UDP). Essa função trata de quase todos os tipos de operação do servidor, sendo que apenas para as funções que mexem no estoque ela não é utilizada.

Código Completo

Arquivo: Client.h

```
1  /* Client.c - UDP Socket -----
2  * Andre Nakagaki Filliettaz   - RA104595 -----
3  * Guilherme Alcarde Gallo     - RA105008 -----
4  -----*/
5
6  /* This programs deals with the interface with the humans and requests to
7  * to the server. Uses the standarts UDP sockets and SQLite3 librarys */
8
9  /* Include all the stuff need to execute the program */
10 #include "Client.h"
11 #define MYPOR  "4950"
12 #define SERVIP "143.106.16.244"    // Ribeiro - IC301
13
14 /* Main function */
15 int main (int argc, char *argv[]) {
16     /* Control Variables */
17     char op, isbn[11], pwd[50], qtt[4], sIP[20];
18
19     /* With the connection done, read to send requests to the server */
20     int sockfd;
21     struct addrinfo hints, *servinfo, *tmp;
22     char message[1000], server_reply[2000];
23     struct timeval tv;
24
25     tv.tv_sec = 5; /* 30 Secs Timeout */
26     tv.tv_usec = 0; // Not init'ing this can cause strange errors
27
28     /* Defining the IP */
29     if (argv[1] == NULL) strcpy(sIP, SERVIP); /* Default */
30     else if (argv[1] == "0") strcpy(sIP, "127.0.0.1"); /* Host */
31     else strcpy(sIP, argv[1]); /* Passed */
32
33     /* Setting hints to get the list of addrinfo struct */
34     memset(&hints, 0, sizeof hints);
35     hints.ai_family = AF_UNSPEC; /* Any kind of IP */
36     hints.ai_socktype = SOCK_DGRAM; /* UDP */
37
38     /* Create the list of structs */
39     if (getaddrinfo (sIP, MYPOR, &hints, &servinfo) != 0) {
40         printf("Erro na alocação de Enderecos!\n");
41         return -1;
42     }
43
44     /* Take the first of the addrinfo inside the list */
45     for (tmp = servinfo; tmp != NULL; tmp = tmp->ai_next) {
46         /* Get the Socket Descriptor */
47         if ((sockfd = socket(tmp->ai_family, tmp->ai_socktype, tmp->ai_protocol)) == -1 ) {
48             printf("Error on socket creation! Trying next address struct!\n");
49             continue;
50         }
51     }
52 }
```

```

54     break;
55 }
56 if (tmp == NULL) {
57     printf("Falha ao criar sockets! Abortando!\n");
58     return -2;
59 }
60
61 /* Start the loop of requests to the server! */
62 while(1) {
63     showOptions(); // Explains the options to the User
64
65     scanf(" %c", &op); // Take the option from user
66
67     switch(op) {
68         case 'h': // A little help
69             break;
70
71         case 'l': // Looking at the Store
72             if( dataFetch(sockfd, NULL, "1", tmp) < 0)
73                 printf("PROBLEMS!!!!!!\n");
74             break;
75
76         case 'd': // Searching for Description
77             printf("Waiting for ISBN of the Book!\n");
78             scanf(" %s", isbn); // Getting ISBN
79
80             /* Calling the fetching result function */
81             dataFetch(sockfd, isbn, "2", tmp);
82             break;
83
84         case 'i': // Searching for Info
85             printf("Waiting for ISBN of the Book!\n");
86             scanf(" %s", isbn); // Getting ISBN
87
88             /* Calling the fetching result function */
89             printf("%d",dataFetch(sockfd, isbn, "3", tmp));
90             break;
91
92         case 'a': // All Infos
93             /* Calling the fetching result function */
94             if( dataFetch(sockfd, NULL, "4", tmp) < 0)
95                 printf("PROBLEMS!!!!!!\n");
96             break;
97
98         case 'c': // Changing the stores numbers
99             printf("Waiting for the new stock amount!\n");
100             scanf(" %s", qtt); // Getting Quantity
101             printf("Waiting for ISBN of the Book!\n");
102             scanf(" %s", isbn); // Getting ISBN
103             alterStock(sockfd, isbn, qtt, tmp);
104             break;
105
106         case 'n': // Numbers on stock
107             printf("Waiting for ISBN of the Book!\n");
108             scanf("%s", isbn); // Getting ISBN
109
110             /* Calling the stocks numbers */
111             dataFetch(sockfd, isbn, "6", tmp);
112             break;
113
114         case 'p':
115             printf("Digite a senha para cliente livraria...\n");
116             scanf(" %s", pwd);
117             pass(sockfd, pwd, tmp);
118             break;
119
120         case 'q': // Quitting the program!
121             printf("Quitting now!\n");
122             break;
123
124         default: // Unknow command
125             printf("Bad instruction, try again!\n");
126

```

```

        break;
138 } /* End Switch */

140 if (op == 'q')
    break;

142 }

144 close(sockfd);
146 return 0; // Terminating program
148 }

```

Arquivo: CliFunctions.c

```

/* CliFunctions.c - UDP Socket _____
2  * Andre Nakagaki Filliettaz   - RA104595 _____
  * Guilherme Alcarde Gallo     - RA105008 _____
4  * _____ */

6  /* Implementation of all the functions used on Client.c */

8  /* Include all the stuff need to execute the program */
#include "Client.h"

10 int check_str(char str[], char alpha) {
12     int it=0, count=0;

14     /* Looping on the string */
    for(it=0; it < strlen(str); it++) {
16         if(str[it] == alpha)    count++;

18     }

20     /* char didn't find */
    return count;
22 }

24 void logger(char option[], int time, int countR, int tOp) {
    FILE *logfile;
26     char text[50], name[10];

28     sprintf(name, "LOG%s", option);
    // Gravando opcao e tempo percorrido em foramto CSV
30     sprintf(text, "%s,%d,%d,%d\n", option, time, countR, tOp);
    logfile = fopen(name, "a");
32     fputs(text, logfile);
    fclose(logfile);
34 }

36 /* _____ */

38 void showOptions() {
    printf("Welcome to the Library! Enter the option following the notation:\n");
40     printf("[h]: Help      - Show this message again!\n");
    printf("[l]: List      - List all the ISBN and his respects Titles\n");
42     printf("[d]: Description - Show the description of a given ISBN\n");
    printf("[i]: Information - Displays the infos from a given ISBN\n");
44     printf("[a]: All Infos  - Show all the infos from all the books\n");
    printf("[p]: Password   - Authenticate the livraria account\n");
46     printf("[c]: Changing   - Change the numbers of the Stock **\n");
    printf("[n]: On Stock   - Numbers on Stock\n");
48     printf("[q]: Quit      - Bye Bye!\n\n");

50     printf("_____\n");
    printf("** Administrator Only!\n\n");
52     printf("Make your choice: ");
54 }

56 /* _____ */

58 void alterStock(int sockfd, char isbn[], char qtd[], struct addrinfo *saddr) {
    char asw[5000]; // Response from server
60     char *time; // Operation Time from the Server
    int read_bytes, sig=0;

62     Livro tt;
64     strcpy(tt.i, isbn);
    strcpy(tt.q, qtd);

```



```

66 long elapsed = 0; // Guarda intervalo de tempo
67 int nRevc = 0; // Guarda numero de receives
68 struct timeval t0, t1; // Guarda tempo percorrido
69
70 // Sending request for password to server
71 if ( sendto(sockfd, "5", 2, 0, saddr->ai_addr, saddr->ai_addrlen) < 0 ) {
72     printf("SEND FAILURE!\n"); // DEBUG
73     return;
74 }
75
76 // Sending the password string to server
77 if ( sendto(sockfd, &tt, 30, 0, saddr->ai_addr, saddr->ai_addrlen) < 0 ) {
78     printf("SEND FAILURE!\n"); // DEBUG
79     return;
80 }
81
82 // Receiving the answer of server authentication
83 while(1) {
84
85     /* Cleans the string */
86     memset( asw, 0, 5000 );
87
88     gettimeofday(&t0, 0); // Capturando tempo de inicio
89     /* Read a maximum of 500 bytes from buffer */
90     if ( read_bytes = recvfrom(sockfd, asw, 5000, 0, saddr->ai_addr, &saddr->ai_addrlen) < 0 ) {
91         printf("Erro no receive!!\n\n");
92         return;
93     } else {
94         gettimeofday(&t1, 0); // Capturando tempo de termino
95         nRevc++; // Atualizando contagem de receive
96         // Calculando intervalo de tempo em microsegundos
97         elapsed += (t1.tv_sec-t0.tv_sec)*1000000 + t1.tv_usec-t0.tv_usec;
98         /* Tests if received string contains the char
99          * '^D', which means TRANSMISSION OVER */
100         sig=check_str(asw, '^D');
101
102         /* Testing here what is what... */
103         if ( sig > 0 ) {
104             /* End Reading */
105             printf("%s", asw);
106             break;
107         } else /* Continue Reading! */
108             printf("%s", asw);
109     }
110 }
111
112 }
113
114 printf("%s",asw);
115 time = asw+strlen(asw)-8;
116 printf("\n");
117 printf("Tempo gasto: %lius || %li || %li", elapsed - (long)atoi(time), elapsed, (long)atoi(time));
118 printf("\n");
119 logger("5",elapsed - atoi(time),nRevc,atoi(time));
120 elapsed = 0;
121 }
122
123 /* ----- */
124
125 void pass(int sockfd, char pwd[], struct addrinfo *saddr) {
126     char asw[5000]; // Response from server
127
128     // Sending request for password to server
129     if ( sendto(sockfd, "7", 2, 0, saddr->ai_addr, saddr->ai_addrlen) < 0 ) {
130         printf("SEND FAILURE!\n"); // DEBUG
131         return;
132     }
133
134     // Sending the password string to server
135     if ( sendto(sockfd, pwd, 50, 0, saddr->ai_addr, saddr->ai_addrlen) < 0 ) {
136         printf("SEND FAILURE!\n"); // DEBUG
137         return;
138     }
139
140     // Receiving the answer of server authentication
141     if ( recvfrom(sockfd, asw, 200, 0, saddr->ai_addr, &saddr->ai_addrlen) < 0 ) {
142         printf("[1] RECEIVE FAILURE\nasw: %s", asw ); // DEBUG
143         return;
144     }
145
146     printf("%s",asw);
147 }
148

```

```

150 /* ----- */
152 int dataFetch(int sockfd, char *ISBN, char op[], struct addrinfo *saddr) {
153     char asw[5000]; // Answer from the Server
154     char *time; // Operation Time from the Server
155     int read_bytes, sig=0, errc=0;
156
157     socklen_t addr_len;
158
159     long elapsed = 0; // Guarda intervalo de tempo
160     int nRevs = 0; // Guarda numero de receives
161     struct timeval t0, t1; // Guarda tempo percorrido
162
163     /* Formating output */
164     printf("\n");
165
166     /* Sends the Request to the Server and check errors */
167     if ( sendto(sockfd, op, strlen(op), 0, saddr->ai_addr, saddr->ai_addrlen) < 0 ) {
168         printf("Sending Error! Aborting!\n");
169         return -1;
170     }
171
172     if ( op[0] != '1' && op[0] != '4' ) {
173         /* Send the ISBN required to the operation in case
174          * of operations 2 and 3 */
175         if ( sendto(sockfd, ISBN, strlen(ISBN), 0, saddr->ai_addr, saddr->ai_addrlen) < 0 ) {
176             printf("Sending Error! Aborting!\n");
177             return -1;
178         }
179     }
180 }
181
182 /* Reading L00P! Read from the buffer as long there is data at the buffer,
183  * if receives the signal to stop (char '^D') then stop reading */
184 elapsed = 0;
185
186 while(1) {
187
188     /* Cleans the string */
189     memset(asw, 0, 5000);
190
191     gettimeofday(&t0, 0); // Capturando tempo de inicio
192
193     /* Read a maximum of 5000 bytes from buffer */
194     if ( read_bytes = recvfrom(sockfd, asw, 5000, 0, saddr->ai_addr, &saddr->ai_addrlen) < 0 )
195         return -1;
196     else {
197         gettimeofday(&t1, 0); // Capturando tempo de termino
198         nRevs++; // Atualizando contagem de receive
199
200         // Calculando intervalo de tempo em microsegundos
201         elapsed += (t1.tv_sec-t0.tv_sec)*1000000 + t1.tv_usec-t0.tv_usec;
202
203         /* Tests if received string contains the char
204          * '^D', which means TRANSMISSION OVER */
205         sig=check_str(asw, '^D');
206
207         /* Testing here what is what... */
208         if ( sig > 0 ) {
209             /* End Reading */
210             printf("%s", asw);
211             break;
212         } else /* Continue Reading! */
213             printf("%s", asw);
214
215     }
216 }
217
218 /* Formating the output! */
219 time = asw+strlen(asw)-8;
220 printf("\n");
221 printf("Tempo gasto: %lius || %li || %li — %d", elapsed - (long)atoi(time), elapsed, (long)atoi(time),
222        nRevs);
223 printf("\n");
224 printf("%s", time);
225 printf("\n");
226 // Guardando num log CSV
227 logger(op, elapsed-atoi(time), nRevs, atoi(time));
228 elapsed = 0;
229 return 0;
230 }
231
232

```

```

/* _____ */
/* _____ */

```

3.1 Explicações das Funções Principais

Essas quatro primeiras rotinas foram implementadas para fazer o trabalho principal de enviar requisições ao sistema e receber dados do usuário do mesmo:

- **int main(int argc, char *argv[]):** é a função responsável por gerenciar as chamadas de outras funções que realizarão as requisições de dados. Uma coisa importante implementada nesta função é a criação do socket. Além disso, ela pode ou não receber parâmetros em sua chamada. Se receber “0”, utiliza como IP o do próprio host, ou seja, busca se conectar a um servidor em 127.0.0.1. Caso o primeiro argumento de sua chamada for qualquer tipo de string, essa string será interpretada como um endereço IP. Por fim, se não houverem argumentos na chamada, o endereço utilizado para fazer o envio de dados será um default, especificado no código;
- **int dataFetch(int, char, char [], addrinfo *):** a função mais importante do processo Client! Recebe como parâmetros o descritor de socket, uma string, que pode conter o ISBN do livro dependendo da opção escolhida pelo usuário, ou NULL, a tarefa que deve ser executada pelo servidor e um pointer para uma estrutura *addrinfo* (cujo uso será explicado mais abaixo). É chamada múltiplas vezes pela função main, por ser a rotina responsável por enviar as requisições ao servidor, bem como imprimir os resultados na tela. A impressão é feita diretamente da leitura do buffer, sendo que a função sabe que o servidor terminou o envio de dados quando percebe um caracter ‘^D’ no final do buffer;
- **int check_str(char *, char):** esta função é auxiliar e é chamada sempre pela função dataFetch. Na realidade ela busca pelo char recebido no segundo parâmetro na string recebida no primeiro;

Essas são as principais funções utilizadas pelo programa para fazer a leitura de dados do servidor. Entretanto, como devem existir dois tipos de modos de operação (user comum e user livraria), foram implementadas mais tres rotinas, que são utilizadas para gerenciar se a opção que altera as quantidades em estoque deve ser ativa:

- **void pass(int, char, addrinfo *):** recebe o socket da conexão e o password. Ela envia a requisição e o password para o servidor. É utilizada para entrar no modo livraria;
- **void alterStock(int, char, char, addrinfo *):** recebe o socket, o ISBN e a quantidade a ser modificada desse ISBN no servidor. Essa rotina só é executável em modo Livraria! Utiliza uma estrutura simples de livro para auxiliar (única rotina a utilizar essa estrutura);

Por fim, foi implementada uma última rotina que funciona como Logger, que deixa um arquivo no formato *.csv* com os tempos de cada uma das operações realizadas pelo servidor e pelo client. Esse arquivo é o que foi postumamente utilizado para fazermos a análise dos dados e plotarmos os gráficos.

- **void logger(char, int, int, int):** abre (cria caso não exista) um arquivo em formato *.csv* e despeja no mesmo a operação realizada, o tempo de transmissão entre o cliente e o servidor, o número de recv necessários para transferir todos os dados e o tempo de computação por parte do servidor.

3.2 A Estrutura addrinfo

Visto que esta estrutura (nativa do C) é amplamente usada em programação em redes (em especial nessa implementação do client-server UDP), aqui analisa-se um pouco mais a fundo o seu uso nesta aplicação.

A estrutura foi incorporada como parâmetro na função *dataFetch* e em todas as outras que fazem comunicação com o server. Como o próprio nome sugere, a estrutura é usada para conseguirmos informações a respeito dos endereços utilizados pelo socket.

Nas linhas 35 a 44 do código do *Client.c*, uma estrutura auxiliar *addrinfo* é utilizada para fazer as definições do tipo e da família do endereço, sendo que da linha 41 em frente, essa estrutura com o auxílio da função *getaddrinfo* é utilizada para conseguir uma lista ligada de estruturas de *addrinfo*.

Essa lista ligada é utilizada logo em seguida para criarmos o descritor de socket. Para tal, utilizasse a primeira estrutura disponível na lista.

Além disso tudo, a estrutura retirada da lista utilizada é depois passada por parâmetro para as funções auxiliares que fazem a comunicação com servidor. Isso porque as funções *sendto* e *recvfrom* utilizam como dois ultimos parâmetros o tipo de endereço bem como o tamanho dele.

Capítulo 4

Análise de Dados

Novamente com o auxílio dos arquivos .csv produzidos pela função logger, foi possível montar tabelas com os tempos de transmissão e de operação para cada uma das funcionalidades oferecidas pelo client-server. Para tal, coletou-se um mínimo 100 medições de tempo para cada funcionalidade. Mediu-se o desvio padrão (aqui considerado como erro) e com isso foram feitos gráficos com barras de erros, que levaram a análise e a conclusão do projeto.

4.1 Servidor e Cliente UDP

Cabe observar que os testes foram realizados em dois computadores diferentes que estavam conectados a mesma rede no IC. O computador ribeiro – 143.106.16.244 foi o computador aonde executou-se o processo servidor. Já o computador rocha – 143.106.16.245 foi o computador responsável pelo processo cliente. Ambos os computadores se encontram na sala 301 do IC03.

Existem dois fatores que podem ser destacados quando fazemos a análise do tempo de envio de cada uma das operações: I) O tamanho do dado enviado pela operação; II) A quantidade de vezes que foram enviados dados para a execução da operação (quantidade de *recvfrom* que o cliente precisou executar para concluir a operação);

Feitos os testes, montou-se a seguinte tabela de dados:

Tabela 4.1: Tabela Final UDP

Operação	Transmissão (us)	Erro (us)	Receives (us)	Erros (us)	Tempo de Opeação (us)	Erro (us)
List	4900	500	33	0	400	100
Description	400	70	2	0	7900	500
Information	350	70	2	0	8000	4000
All Informations	9000	7000	31	0	800	200
Change	660	40	1	0	50000	1000
Numbers	650	50	2	0	4000	600

Tabela 4.2: Dados da Transmissão entre dois computadores por protocolo UDP

Figura 4.1: Dados da Transmissão em todas as operações numa mesma Máquina

Figura 4.2: Gráfico Operação x Tempo Médio de Comunicação

Da tabela, percebe-se que os tempos de envio de um datagrama são basicamente o mesmo. Entretanto, também é possível perceber uma diferença muito grande nos tempos de envio para as operações de List e de All Information, sendo que o tempo desta última é quase o dobro do tempo da primeira. Elas são de longe as operações mais custosas realizadas quando analisamos os tempos de transmissão, apesar de não necessariamente serem as mais custosas quando o enfoque é o tempo de operação.

Como previsto, a operação mais custosa é também a que envia a maior quantidade de dados. De fato, All Information envia todos os dados que estão no banco de dados, então é de se esperar que ela envie uma quantidade de dados sempre muito maior do que a enviada por qualquer outra operação. Isso fez com que seu tempo de envio fosse maior do que a soma de todos os outros tempos de envio!

Sob esse mesmo raciocínio, seria estranho, então, a operação List ser a segunda mais custosa, já que na realidade, ela pode não enviar uma quantidade tão grande assim de dados. Na realidade, no caso do banco de dados utilizados para esse projeto, a quantidade de dados enviados pela operação List não passou de 1200 bytes.

Considere como exemplo o livro Dom Casmurro – ISBN 9878765654, cuja operação de envio de descrição enviaria 2010 bytes, um pouco menos do que o dobro do enviado pela List. Entretanto, o tempo médio gasto com essa é mais do que 10 vezes o tempo médio do envio de uma descrição.

Esse aparente erro é explicado quando considera-se o segundo fator apontado! Da tabela vemos que a implementação do servidor faz com que a média de chamadas de *recvfrom* por parte do cliente seja de 33 (2 a mais do que na operação All Information). Essa quantidade é enorme quando comparada com as das outras operações. O servidor acaba por quebrar os

dados em vários datagramas, e envia-os através de várias chamadas. Isso explica então o tempo de envio absurdo da operação List comprovando que quando se deseja determinar o tempo de transmissão de um dado, tão importante quanto determinar o tamanho da mensagem é determinar como essa mensagem será enviada!

Por fim, montou-se um gráfico com essa tabela e seus desvios padrão:

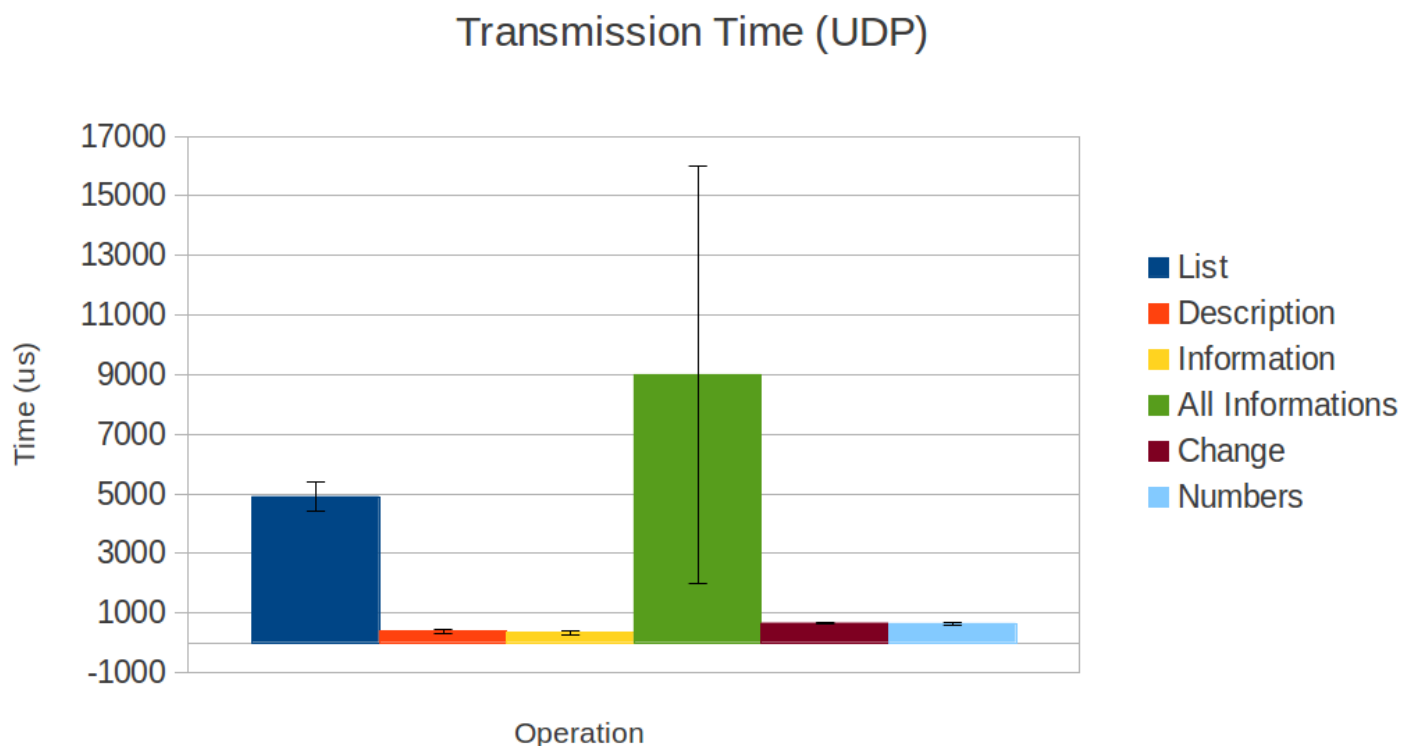


Figura 4.3: Gráfico Operação x Tempo Médio de Comunicação

Note que o desvio padrão envolvido com cada medição é maior conforme o tempo da medição é maior. Isso também é natural do comportamento UDP, já que os tempos de operação podem variar muito devido a ser um protocolo não orientado a conexão.

4.2 Comparando UDP e TCP

Para fazer uma comparação boa entre os dois protocolos, da mesma forma que no UDP, foram feitas uma série de medidas. Para o caso do TCP foram coletadas no mínimo 100 medidas, todas realizadas nas mesmas máquinas que as utilizadas no UDP. Com os dados, foi montada a seguinte tabela, que segue o formato da Tabela 01:

Tabela 4.3: Tabela Final TCP

Operação	Transmissão (us)	Erro (us)	Receives (us)	Erros (us)	Tempo de Opeação (us)	Erro (us)
List	5300	300	3	0	400	30
Description	600	100	2	0.06	48100	700
Information	500	100	2	0.07	49000	6000
All Informations	4200	500	6	0.9	700	100
Change	630	40	1	0	49000	9000
Numbers	700	200	2	0.07	45000	2000

Tabela 4.4: Dados da Transmissão entre dois computadores por protocolo TCP

Para uma maior facilidade na comparação, a Tabela 01 é mostrada abaixo:

Tabela 4.5: Tabela Final UDP

Operação	Transmissão (us)	Erro (us)	Receives (us)	Erros (us)	Tempo de Opeação (us)	Erro (us)
List	4900	500	33	0	400	100
Description	400	70	2	0	7900	500
Information	350	70	2	0	8000	4000
All Informations	9000	7000	31	0	800	200
Change	660	40	1	0	50000	1000
Numbers	650	50	2	0	4000	600

Tabela 4.6: Dados da Transmissão entre dois computadores por protocolo UDP

Observa-se que para os tempos de envio, o protocolo UDP mostrou-se mais rápido em quase todas as operações.

Entretanto, os tempos do UDP não se mostraram tão satisfatórios assim quando comparam-se 3 operações (List, All Informations e Change) sendo que para as duas últimas os tempos foram na realidade maiores, apesar de que se considerarmos os erros envolvidos, na realidade todas estão aceitáveis, pois são cobertas pelos erros.

Como considerou-se o mesmo banco de dados, os dados a serem enviados tinham o mesmo tamanho, logo esse não pode ser um fator tão importante para o envio.

Novamente, a resposta está no segundo fator de atraso: a quantidade de mensagens enviadas! Podemos ver que para o caso de List, o UDP enviou cerca de onze vezes mais mensagens enquanto que para All Informations, esse numero foi de aproximadamente 5,2 vezes. Apesar disso, essa explicação não pode ser usada para justificar a operação Change, que no final das contas teve um desempenho pior no UDP.

Uma observação final deve ser feita ainda. No caso especial da operação List, é notável que o UDP seja muito mais rápido, mesmo enviando uma quantidade tão grande de mensagens quando comparadas com número de mensagens enviadas pelo protocolo TCP. Atribuímos esse grande atraso do TCP as operações que tornam o protocolo confiável, ou seja, a checagem de erros e ordem que o protocolo executa sempre que envia mensagens.

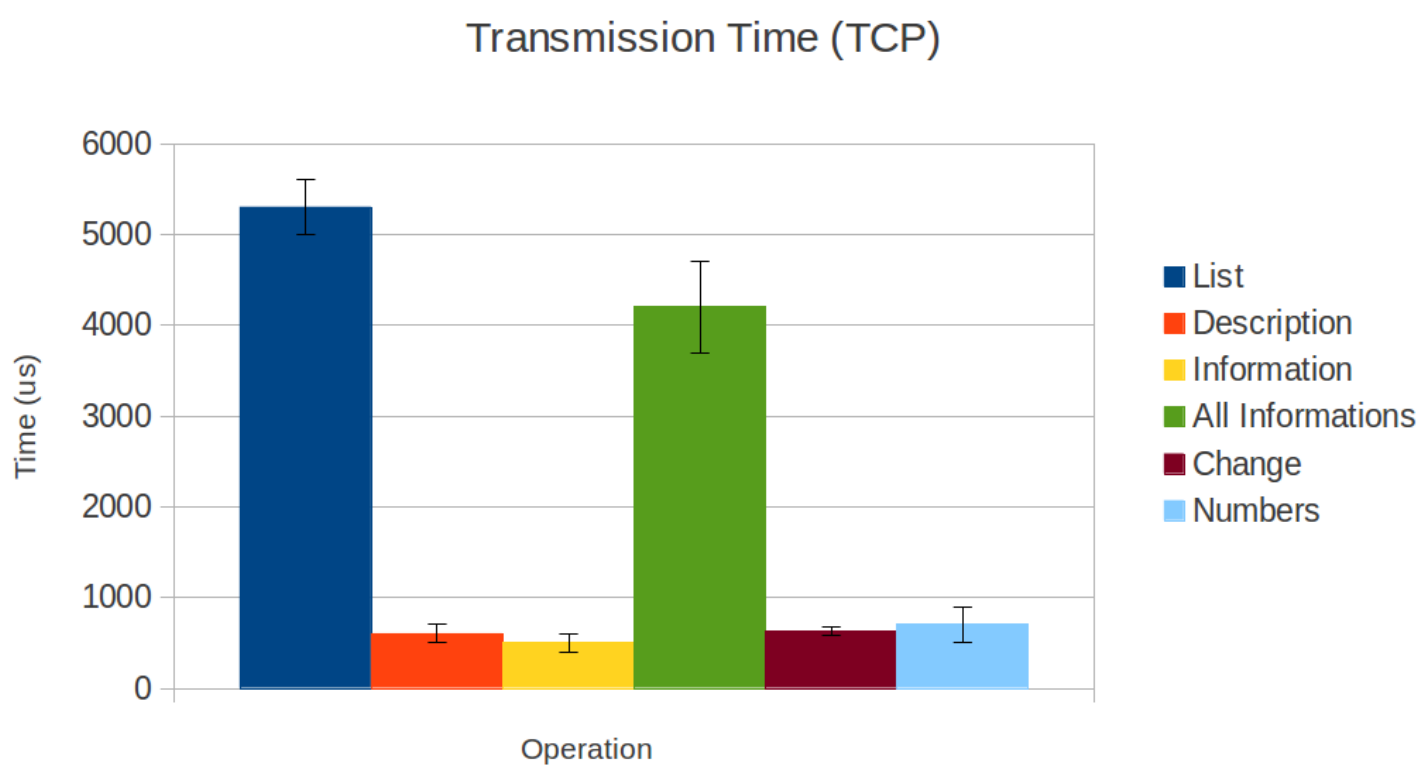


Figura 4.4: Gráfico Operação x Tempo Médio de Comunicação

Capítulo 5

Conclusão

Assim como observado para o protocolo TCP, pudemos determinar aqui os fatores que importam quando consideramos o protocolo UDP.

Assim como no TCP, o UDP tem dois fatores principais. O primeiro e mais óbvio, é o tamanho e quantidade de dados que será enviado pelo protocolo. Vimos que no caso do UDP, esse foi o fator determinante para as medições de tempo, ao contrário do que aconteceu no caso do TCP. Isso era o esperado, pois o UDP não é um protocolo confiável, logo ele simplesmente envia o que foi pedido para ser enviado, sem se preocupar com qualquer tipo de erro no envio.

Entretanto, nunca poderemos desconsiderar a quantidade de mensagens enviadas para termos a transmissão do dado. Mesmo não sendo confiável, pela Tabela 01, pode-se observar que o tempo de envio de uma operação é fortemente influenciado pela quantidade de mensagens envolvidas nessa transmissão.

Ao final do projeto, percebe-se aqui as grandes diferenças que um protocolo confiável e orientado a conexão (TCP) possui de um protocolo que prioriza a velocidade (UDP). De fato, o UDP provou-se mais rápido em todas as situações, se considerarmos a quantidade de mensagens enviadas. Isso explica porque ele é usado por aplicações que necessitam de grandes velocidades.

De fato, essa velocidade deve ser sempre considerada, mesmo em casos onde faz-se necessário a confiabilidade. Para tais casos, o UDP pode ser utilizado, valendo-se o esforço em nível de aplicação para garantir-se a confiabilidade! Exemplo de aplicação assim é o DNS.

Capítulo 6

Referências Bibliográficas

"Beej's Guide to Network Programming". <<http://beej.us/guide/bgnet/>>. (Acesso em: 04 abril 2013).
"SQLite Documents". <<http://www.sqlite.org/docs.html>>. (Acesso em: 03 abril 2013).