# MC823 - Relatório - Servidor Concorrente sobre TCP

André Nakagaki Fillettaz RA: 104595
Guilherme Alcarde Gallo RA: 105008

18 de Abril de 2013

# Conteúdo

# Capítulo 1

# Servidor

## 1.1 Callback e o envio da mensagem

Callback é um apontador de função que é chamado a cada processamento de tupla feito pelo wrapper nativo da biblioteca SQLite3 (Ver Banco de Dados): a função **sqlite3_exec**.

Foram usados 3 tipos de Callback, cada um com sua utilidade:

- **callbackSilent**: Retorna valor num ponteiro para apenas o servidor enxergar
- **callbackFmt**: Envia dados formatados para tuplas grandes
- **callback**: Envia dados formatados para tuplas pequenas

Vou mostrar o funcionamento do callbackFmt, o qual é o mais complicado.

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

  // Nome da coluna
  strcpy(aux,azColName[0]);
  strcat(aux,": ");
  // Concatena valor
  strcat(aux,ans[0]);
  strcat(aux,"\n");
  // Formata
  for(i=1; i<argc; i++) {    // Fazendo isso para o resto da tupla
    if (ans[i]) {
      strcat(aux,azColName[i]);
      strcat(aux,": ");
      // Melhorando a visualizacao da descricao
      if (i==3) {
        strcat(aux,"\n\t");
      }
      strcat(aux,ans[i]);
      strcat(aux,"\n");
    }
  }
  strcat(aux,"\n");
  // DEBUG
  printf("%s", aux);
  // Enviando para o cliente
  num = strlen(aux);
  sendall(client_sock, aux, &num);
  return 0;
}
```

Lembrando que o callback é chamado a cada processamento de tupla, a função callbackFmt recebe em **ans** a resposta e em **azColName** os nomes da coluna. Primeiro, a função copia o nome da primeira coluna depois concatena seu valor.

E faz isso até o fim da tupla, quebrando linhas a cada coluna da tupla, para facilitar a visualização de descrições gigantes.

Essa formatação é armazenada num produto final que é uma string: **aux**. Depois de tudo, **aux** é enviada de forma persistente ao cliente.

Nota: a função callbackSilent faz uso do *void * NotUsed* para armazenar um valor para uma leitura posterior do próprio servidor. É usado nos casos que é necessário verificar a existência de um livro no banco.

## 1.2 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 |

### 1.2.1 Listar dados

Essa seção abrange o funcionamento de listar:

- (1) todos os livros retornando seu ISBN e título;
- (4) todas as suas informações

A ideia básica aqui é mandar a query para o SQLite3 e retornar seu resultado. A vantagem dessas operações é que a query não depende de nenhuma informação adicional, tornando-se uma query constante. Usando como exemplo o caso de listar todas as informações de todos os livros:

```
case 4:         // Todas as informacoes de todos os livros
        // Zerando timer
        elapsed = 0;
        // Executando query
        rc = sqlite3_exec(db, "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;",
                        callbackFmt, 0, &zErrMsg);

        // Fim da mensagem
        // Tempo percorrido ate agora
        gettimeofday(&t1, 0);
        // Calculo 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^D",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);
        break;
```

### 1.2.2 Listar dados específicos

Agora tem-se que receber dados do cliente com o valor do ISBN, isso muda o código, porque é necessário saber se tal ISBN consta no banco de dados e retorna um valor coerente para o cliente.

Esse caso se aplica às operações:

- (2) pedir descrição de 1 livro;
- (3) todas as suas informações de 1 livro;
- (6) o estoque de 1 livro;

Utilizando como exemplo o trecho de código da operação 2:

```
case 2:         // Descricao de um livro
        // Zerando timer
        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;
        if ( (read_size = recv(client_sock , client_message , 2000 , 0)) > 0 ) {
                //                                                  gettimeofday(&t0, 0);
                // Montando a query
                strcpy(query, "select descricao from livro where ISBN10 = ");
                strcpy(query2, "select count(descricao) 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
```

```
21                  // callbackSilent nao envia dados ao cliente
                    // existe recebe o resultado de contagem de livros (0 ou 1)
23                  rc = sqlite3_exec(db, query2, callbackSilent, existe, &zErrMsg);
                    // Tempo percorrido ate agora
25                  //      gettimeofday(&t1, 0);
                    if ( *((int *)existe) == 0) {
27                          length = 41;
                            sendall(client_sock, "\nEste ISBN nao consta na nossa livraria!\n",&length);
29                  }
                    else
31                          // Executando query - Callback ja faz os sends
                            rc = sqlite3_exec(db, query, callback, 0, &zErrMsg);
33          }
            length = 1;
35
            gettimeofday(&t1, 0);
37          // Fim da mensagem
            // ReCalculo do tempo de operacao
39          elapsed += (t1.tv_sec-t0.tv_sec)*1000000 + t1.tv_usec-t0.tv_usec;
            // Transformando em string com chars de "seguranca" para postumo atoi
41          sprintf(query,"          %6li^D",elapsed);
            // DEBUG
43          printf("\nOperation Time: %s\n\n",query);
            // Calculando o tamanho
45          length = strlen(query);
            // Finalmente envia para o cliente
47          sendall(client_sock, query, &length);
            break;
```

Note que o método usado para verificar se um livro existe ou não é verificar se o retorno do **callbackSilent** no ponteiro para *int* **existe** é ou não é 0 para a operação SQL de COUNT nos livros com o ISBN dado pelo cliente.

### 1.2.3 Autenticação do Cliente Livraria e Mudança no estoque

Neste sistema, é considerado um usuário normal qualquer usuário que não se logar como *livraria*. A ideia de se logar como cliente livraria é bem simples: basta pedir a senha para o cliente enviar e comparar com a senha armazenada no seu código. Caso a mesma seja válida, habilitar o modo super usuário, através de um *int*: **superuser**. A mudança no estoque só ocorre se o super usuário estiver setado.

## 1.3 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:

$$elapsed+ = (t_{final}.tv\_sec - t_{inicial}.tv\_sec) * 1000000 + t_{final}.tv\_usec - t_{inicial}.tv\_usec; \quad (1.1)$$

### 1.3.1 Cliente contando tempo

Como pode ser necessário mais de um receive para o cliente receber toda a mensagem, é tomado o cuidado de somar o tempo a cada receive feito: Linhas **8** e **13**.

```
/* Reading L00P! Read from the buffer as long there is data at the buffer,
2  * if receives the signal to stop (char '^D') then stop reading */
elapsed = 0;
4 while(1) {
        /* Cleans the string */
6       memset(asw,0,5000);

8       gettimeofday(&t0, 0);           // Capturando tempo de inicio
        /* Read a maximum of 500 bytes from buffer */
10      if ( read_bytes = recv(sockfd, asw, 5000, 0) < 0 )
                return -1;
12      else {
                gettimeofday(&t1, 0);   // Capturando tempo de termino
14              nRevcs++;               // Atualizando contagem de receive
                // Calculando intervalo de tempo em microsegundos
16              elapsed += (t1.tv_sec-t0.tv_sec)*1000000 + t1.tv_usec-t0.tv_usec;
```

```
                /* Tests if received string contains the char
                 * '^D', which means TRANSMISSION OVER */
                sig=check_str(asw, '^D');

                /* Testing here what is what... */
                if (sig > 0) {
                        /* End Reading */
                        printf("%s", asw);
                        break;
                } else  /* Continue Reading! */
                        printf("%s", asw);

        }
}
```

### 1.3.2 Servidor contando tempo

O Servidor conta o tempo que ele gasta fazer operações internas para corrigir o tempo que o cliente contou, isso é feito no **sendall** e em todas as operações do sistema.

Função *sendall*

```
// Funcao que persiste no send ate toda a mensagem ser enviada
int sendall(int s, char *buf, int *len)
{
  int total = 0;
  // how many bytes we've sent
  int bytesleft = *len; // how many we have left to send
  int n;
  while(total < *len) {
    gettimeofday(&t1, 0);
    elapsed += (t1.tv_sec-t0.tv_sec)*1000000 + t1.tv_usec-t0.tv_usec;
    // Ignorar tempo de send
    n = send(s, buf+total, bytesleft, 0);
    // Voltando a calcular o tempo de operacao
    gettimeofday(&t0, 0);
    if (n == -1) { break; }
    total += n;
    bytesleft -= n;
  }
  gettimeofday(&t1, 0);
  elapsed += (t1.tv_sec-t0.tv_sec)*1000000 + t1.tv_usec-t0.tv_usec;
  gettimeofday(&t0, 0);
  *len = total; // return number actually sent here
  return n==-1?-1:0; // return -1 on failure, 0 on success
}
```

As linhas 9, 14, 19 e 21 fazem parte do cronômetro feito para evitar contar o tempo de send.

Exemplificando a contagem nas operações: Função da operação 3

```
case 3:         // Todas as informacoes de um livro
        // Zerando timer
        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 = recv(client_sock , client_message , 2000 , 0)) > 0 ) {
                // Montando a query
                //strcpy(query, "select * from livro where ISBN10 = ");

                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);
26              if ( *((int *)existe) == 0) {
                        length = 41;
28                      sendall(client_sock, "\nEste ISBN nao consta na nossa livraria!\n",&length);
                        // Tempo percorrido ate agora
30                      gettimeofday(&t1, 0);
                }
32              else {
                        // Executando query - Callback ja faz os sends
34                      rc = sqlite3_exec(db, query, callbackFmt, 0, &zErrMsg);
                        // Tempo percorrido ate agora
36                      gettimeofday(&t1, 0);
                }
38      }

40      // Fim da mensagem
        // ReCalculo do tempo de operacao
42      elapsed += (t1.tv_sec-t0.tv_sec)*1000000 + t1.tv_usec-t0.tv_usec;
        // Transformando em string com chars de "seguranca" para postumo atoi
44      // espacos nao sao considerados no atoi
        sprintf(query,"          %6li^D",elapsed);
46      // DEBUG
        printf("\nOperation Time: %s\n\n",query);
48      // Calculando o tamanho
        length = strlen(query);
50      // Finalmente envia para o cliente
        sendall(client_sock, query, &length);
52      break;
```

As linhas 6, 30 e 36 compõem os tempos cronometrados. Note que o receive da linha 10 é uma exceção: ele é contado para não interferir no padrão do cálculo de receive pelo cliente.

## 1.4 Código Completo

A estrutura de conexão TCP foi fortemente baseada no *Beej's Guide to Network Programming*.
Os comentários das partes mantidas dos trechos de código foram mantidos.

```
#include<stdio.h>
2 #include<stdlib.h>
#include<string.h>  //strlen, strcpy, strcat
4 #include<sys/socket.h>  //hton, bind, accept
#include<arpa/inet.h>   //inet_addr
6 #include<unistd.h>      //write
#include<errno.h> // perror
8 #include<sys/wait.h>  // waitpid
#include<signal.h>  // sigaction
10 #include<unistd.h>  // fork

12 // SQLite3
#include <sqlite3.h>

14
#define PASSWORD "numaPistacheCottapie"

16
typedef struct livro {
18   char i[20]; // ISBN
  char q[4];  // Stock Quantity
20 } Livro;

22 int socket_desc , client_sock , c , read_size, connfd;

24 // Tempo
long elapsed=0;     // Conta o tempo percorrido
26 struct timeval t0, t1;

28 // Funcao que persiste no send ate toda a mensagem ser enviada
int sendall(int s, char *buf, int *len)
30 {
//  gettimeofday(&t0, 0);
32   int total = 0;
  // how many bytes we've sent
34   int bytesleft = *len; // how many we have left to send
  int n;
36   while(total < *len) {
    gettimeofday(&t1, 0);
38    elapsed += (t1.tv_sec-t0.tv_sec)*1000000 + t1.tv_usec-t0.tv_usec;
    // Ignorar tempo de send
40    n = send(s, buf+total, bytesleft, 0);
    // Voltando a calcular o tempo de operacao
42    gettimeofday(&t0, 0);
    if (n == -1) { break; }
```

```
    total += n;
    bytesleft -= n;
  }
  gettimeofday(&t1, 0);
  elapsed += (t1.tv_sec-t0.tv_sec)*1000000 + t1.tv_usec-t0.tv_usec;
  gettimeofday(&t0, 0);
  *len = total; // return number actually sent here
  return n==-1?-1:0; // return -1 on failure, 0 on success
}

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

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

  // Nome da coluna
  strcpy(aux,azColName[0]);
  strcat(aux,": ");
  // Concatena valor
  strcat(aux,ans[0]);
  strcat(aux,"\n");
  // Formata
  for(i=1; i<argc; i++) {   // Fazendo isso para o resto da tupla
    if (ans[i]) {
      strcat(aux,azColName[i]);
      strcat(aux,": ");
      // Melhorando a visualizacao da descricao
      if (i==3) {
        strcat(aux,"\n\t");
      }
      strcat(aux,ans[i]);
      strcat(aux,"\n");
    }
  }
  strcat(aux,"\n");
  // DEBUG
  printf("%s", aux);
  // Enviando para o cliente
  num = strlen(aux);
  sendall(client_sock, aux, &num);
  return 0;
}

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

  strcpy(aux,ans[0]);
  for(i=0; i<argc; i++){
    if (i && ans[i]) {
      strcat(aux," | ");
      strcat(aux,ans[i]);
    }
  }
  strcat(aux,"\n");
  printf("%s", aux);
  num = strlen(aux);
  sendall(client_sock, aux, &num);
  return 0;
}

// Funcao que espera chamado dos filhos
void sigchld_handler(int s)
{
  // -1   : chamadas de processos-filhos, apenas
  // NULL : nao precisa ler estado
  while(waitpid(-1, NULL, WNOHANG) > 0);
}

int main(int argc , char *argv[])
```

```
{
  struct sockaddr_in server, client;  // Descritores de socket
  struct sigaction sa;      // Sigaction
  pid_t childpid;       // ID do processo-filho
  socklen_t clilen;     // Tamanho do socket
  char client_message[2000], query[2500], query2[2500]; // Strings utilizadas

  int opcao, length=1;

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

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

  tv.tv_sec = 5;  /* 30 Secs Timeout */
  tv.tv_usec = 0;  // Not init'ing this can cause strange errors

  setsockopt(connfd, SOL_SOCKET, SO_RCVTIMEO, (char *)&tv,sizeof(struct timeval));
  // ... Timeout setado.

  // Abrindo Banco de Dados do SQLite3
  rc = sqlite3_open("livraria2.db", &db);
  //Create socket
  socket_desc = socket(AF_INET, SOCK_STREAM, 0);
  if (socket_desc == -1)
  {
    printf("Could not create socket");
  }
  puts("Socket created");

  //Prepare the sockaddr_in structure
  server.sin_family = AF_INET;
  server.sin_addr.s_addr = INADDR_ANY;
  server.sin_port = htons( 8888 );

  //Bind
  if( bind(socket_desc,(struct sockaddr *)&server , sizeof(server)) < 0)
  {
    //print the error message
    perror("bind failed. Error");
    return 1;
  }
  puts("bind done");

  //Listen
  listen(socket_desc , 3);

  // Matando todos os processos zumbis
  sa.sa_handler = sigchld_handler; // reap all dead processes
  sigemptyset(&sa.sa_mask);
  sa.sa_flags = SA_RESTART;
  if (sigaction(SIGCHLD, &sa, NULL) == -1) {
    perror("sigaction");
    exit(1);
  }


  while (1) {
    //Accept and incoming connection
    puts("Waiting for incoming connections...");
    c = sizeof(struct sockaddr_in);

    //accept connection from an incoming client
    client_sock = accept(socket_desc, (struct sockaddr *)&client , (socklen_t*)&c);
    if (client_sock < 0)
    {
      perror("accept failed");
      if(errno == EINTR) {
        continue;
      }
      else {
        //  return 1;
        perror("Erro no accept!");
      }
    }
    puts("Connection accepted");
```

```
    // Criando um processo-filho para tratar a requisicao
    if ( (childpid = fork()) == 0 ) {
      // Processo-filho nao trata requisicoes para novas coneccoes
      close(socket_desc);

      // Servidor em espera da requisicao do cliente
      while( (read_size = recv(client_sock , client_message , 2000 , 0)) > 0 )
      {
        // Comecando a contar o tempo de operacao
        gettimeofday(&t0, 0);

        // Transformando em inteiro
        opcao = atoi(client_message);

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

            // Finalizando a mensagem
            // Calculo do tempo de operacao
            elapsed = (t1.tv_sec-t0.tv_sec)*1000000 + t1.tv_usec-t0.tv_usec;
            // Transformando em string com caracteres de "seguranca" para postumo atoi
            sprintf(query,"          %6liEOT",elapsed); // Caractere EOT (End of Transmission) e um
                identificador de fim da mensagem
            // DEBUG
            printf("\nOperation Time: %s\n\n",query);
            // Calculando o tamanho
            length = strlen(query);
            // Finalmente envia para o cliente
            sendall(client_sock, query, &length);
            break;

          case 2:   // Descricao de um livro
            // Zerando timer
            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;
            if ( (read_size = recv(client_sock , client_message , 2000 , 0)) > 0 ) {
              // Montando a query
              strcpy(query, "select descricao from livro where ISBN10 = ");
              strcpy(query2, "select count(descricao) 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);
              // Tempo percorrido ate agora
            //  gettimeofday(&t1, 0);
              if ( *((int *)existe) == 0) {
                length = 41;
                sendall(client_sock, "\nEste ISBN nao consta na nossa livraria!\n",&length);
              }
              else
              // Executando query - Callback ja faz os sends
                rc = sqlite3_exec(db, query, callback, 0, &zErrMsg);
            }
            length = 1;

            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,"          %6liEOT",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);
            break;

          case 3:   // Todas as informacoes de um livro
            // Zerando timer
            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 = recv(client_sock , client_message , 2000 , 0)) > 0 ) {
              // Montando a query
              //strcpy(query, "select * from livro where ISBN10 = ");

              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);
                // Tempo percorrido ate agora
                gettimeofday(&t1, 0);
              }
              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
            // espacos nao sao considerados no atoi
            sprintf(query,"          %6liEOT",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);
            break;

          case 4:   // Todas as informacoes de todos os livros
            // Zerando timer
            elapsed = 0;
            // Executando query
            rc = sqlite3_exec(db, "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;",
            callbackFmt, 0, &zErrMsg);

            // Fim da mensagem
            // Tempo percorrido ate agora
            gettimeofday(&t1, 0);
            // Calculo 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,"          %6liEOT",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);
            break;

          case 5:   // Atualizar estoque
            // Zerando timer
```

```
            elapsed = 0;
            // 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 novo estoque do livro
            if ( (read_size = recv(client_sock , &cm , 2000 , 0)) > 0 ) {
              if (superuser) {
                // Montando a query
                strcpy(query, "update livro set estoque = ");
                // Concatenando o nova quantidade do estoque
                strcat(query, cm.q);
                // Concatenando o ISBN
                strcat(query, " where ISBN10 = ");
                strcat(query, cm.i);
                // Fim do comando SQLite
                strcat(query, ";");
                printf("%s\n",query);
                // Executando query - Callback ja faz os writes
                rc = sqlite3_exec(db, query, callback, 0, &zErrMsg);
              }
              else {
                length = 41;
                sendall(client_sock, "Sem permissoes para modificar estoque!\n", &length);
              }
            }

            // Fim da mensagem
            // 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;
            // Transformando em string com chars de "seguranca" para postumo atoi
            sprintf(query,"          %6liEOT",elapsed);
            // DEBUG
            printf("\nTime: %s\n\n",query);
            // Calculando o tamanho
            length = strlen(query);
            // Finalmente envia para o cliente
            sendall(client_sock, query, &length);
            break;

          case 6:   // Mostra estoque de um livro
            // Zerando timer
            elapsed = 0;
            // 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 = recv(client_sock , client_message , 2000 , 0)) > 0 ) {
              // Montando a query
              strcpy(query, "select estoque from livro where ISBN10 = ");
              // Concatenando o ISBN
              strcat(query, client_message);
              // Fim do comando SQLite
              strcat(query, ";");
              // Executando query - Callback ja faz os writes
              rc = sqlite3_exec(db, query, callbackFmt, 0, &zErrMsg);
            }
            // Fim da mensagem
            // 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;
            // Transformando em string com chars de "seguranca" para postumo atoi
            sprintf(query,"          %6liEOT",elapsed);
            // DEBUG
            printf("\nTime: %s\n\n",query);
            // Calculando o tamanho
            length = strlen(query);
            // Finalmente envia para o cliente
            sendall(client_sock, query, &length);
            break;
          case 7:   // Autentica o cliente livraria
            // Recebe a senha
            if ( (read_size = recv(client_sock , client_message , 50 , 0)) > 0 ) {
              length = 31;
              // Compara as senhas
              if( (strcmp(client_message,PASSWORD) ) == 0) {
                superuser = 1;    // Sessao de superusuario
                sendall(client_sock, "Bem-vindo, Chuck Norris!\n\n",&length);
              }
              else {
```

```
460                 superuser = 0;    // Usuario invalido
                    sendall(client_sock, "Senha Invalida!\n\n",&length);
462               }
                  length = 1;
464             }
                break;
466         }

468       }

470       if(read_size == 0)
          {
472         puts("Client disconnected");
            fflush(stdout);
474         exit(0);
          }
476     }
        close(connfd);
478   }

480   return 0;
}
```

# Capítulo 2

# Cliente

O programa client é responsável por fazer as leituras de dados do usuário do sistema das livrarias e, após isso, interpretar esses dados e enviar requisições para o servidor que manipula a base de dados. Em resumo, ele simplesmente apresenta o menu de opções ao usuário do client e espera por uma entrada de dados na entrada padrão. Quando detecta essa entrada de dados, o programa então faz a interpretação desses dados da entrada e após isso envia requisições de dados ao servidor. É importante observar que o processo client não trata os dados que são enviados do servidor. Esses dados vem como uma grande string que é impressa na saída padrão do programa, ou seja, o client é responsável por tratar os dados enviados pelo client e o servidor trata dos dados que estão no banco de dados!

## Código Completo

Arquivo: Client.h

```c
/* Client.h ------------------------------------------------------------------
 * Andre Nakagaki Filliettaz  - RA104595 ------------------------------------
 * Guilherme Alcarde Gallo  - RA105008 -------------------------------------
 * ------------------------------------------------------------------------- */

/* HEADER - Support the program with the includes, typedefs and all king of
 * declaration and preprocessor problems*/

/* Standarts input and output librarys */
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <time.h>

/* Library to deal with the networks syscalls */
#include <arpa/inet.h>
//#include <sys/types.h>
#include <sys/socket.h>

/* Type Definitions */
typedef struct livro {
  char i[20]; // ISBN
  char q[4];  // Stock Quantity
} Livro;

/* Functions Declarations */
void showOptions();     // Explanation Function
int dataFetch(int, char *, char []);  // Fetch the Description or Infos
void alterStock(int, char, char);
void pass(int, char);

/* Auxiliar Function */
void logger(char [], int, int);
int check_str(char *, char);
```

Arquivo: Client.c

```c
/* Client.c ------------------------------------------------------------------
 * Andre Nakagaki Filliettaz  - RA104595 ------------------------------------
 * Guilerme Alcarde Gallo   - RA105008 -------------------------------------
 ------------------------------------------------------------------------- */

/* This programs deals with the interface with the humans and requests to
 * to the server. Uses the standarts TCP sockets and SQLite3 librarys */

/* Include all the stuff need to execute the program */
```

```c
#include "Client.h"

/* Main function */
int main (int argc, char *argv[]) {
  /* Control Variables */
  char op, isbn[11], pwd[50], qtt[4];

  /* With the connection done, read to send requests to the server */
  int sockfd;
  struct sockaddr_in server;
  char message[1000] , server_reply[2000];
  struct timeval tv;

  tv.tv_sec = 5;  /* 30 Secs Timeout */
  tv.tv_usec = 0;  // Not init'ing this can cause strange errors

  sockfd = socket(AF_INET , SOCK_STREAM , 0);

  //Create socket
  if (sockfd == -1)
  {
    printf("Could not create socket");
  }
  puts("Socket created");
  setsockopt(sockfd, SOL_SOCKET, SO_RCVTIMEO, (char *)&tv,sizeof(struct timeval));


  server.sin_addr.s_addr = inet_addr("127.0.0.1");

  server.sin_family = AF_INET;
  server.sin_port = htons( 8888 );

  //Connect to remote server
  if (connect(sockfd , (struct sockaddr *)&server , sizeof(server)) < 0)
  {
    perror("connect failed. Error");
    return 1;
  }
  while(1) {
    showOptions();  // Explains the options to the User

    scanf(" %c", &op);  // Take the option from user

    switch(op) {
      case 'h': // A little help
        break;
      case 'l': // Looking at the Store
        if( dataFetch(sockfd, NULL, "1") < 0)
          printf("PROBLEMS!!!!!!!\n");
        break;
      case 'd': // Searching for Description
        printf("Waiting for ISBN of the Book!\n");
        scanf(" %s", isbn); // Getting ISBN

        /* Calling the fetching result function */
        dataFetch(sockfd, isbn, "2");
        break;
      case 'i': // Searching for Info
        printf("Waiting for ISBN of the Book!\n");
        scanf(" %s", isbn); // Getting ISBN

        /* Calling the fetching result function  */
        printf("%d",dataFetch(sockfd, isbn, "3"));
        break;
      case 'a': // All Infos
        /* Calling the fetching result function  */
        if( dataFetch(sockfd, NULL, "4") < 0)
          printf("PROBLEMS!!!!!!!\n");
        break;
      case 'c': // Changing the stores numbers
        printf("Waiting for the new stock amount!\n");
        scanf(" %s", qtt);  // Getting Quantity
        printf("Waiting for ISBN of the Book!\n");
        scanf(" %s", isbn); // Getting ISBN
        alterStock(sockfd, isbn, qtt);
        break;
      case 'n': // Numbers on stock
        printf("Waiting for ISBN of the Book!\n");
        scanf("%s", isbn);  // Getting ISBN

        /* Calling the stocks numbers */
        dataFetch(sockfd, isbn, "6");
        break;
      case 'p':
```

```
        printf("Digite a senha para cliente livraria...\n");
        scanf(" %s", pwd);
        pass(sockfd, pwd);
        break;
      case 'q': // Quiting the program!
        printf("Quiting now!\n");
        break;
      default:  // Unknow command
        printf("Bad instruction, try again!\n");
        break;
    } /* End Switch */

    if (op == 'q')
      break;

  }

  close(sockfd);
  return 0; // Terminating program

}
```

Arquivo: CliFunctions.c

```
/* CliFunctions.c ------------------------------------------------------------
 * Andre Nakagaki Filliettaz  - RA104595 ------------------------------------
 * Guilherme Alcarde Gallo  - RA105008 -------------------------------------
 * ------------------------------------------------------------------------- */

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

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

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

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

  }

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

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

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

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

  printf("---------\n");
  printf("** Administrator Only!\n\n");

  printf("Make your choice: ");
}

/* ------------------------------------------------------------------------- */

```

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

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

  long elapsed = 0;   // Guarda intervalo de tempo
  int nRevcs = 0;   // Guarda numero de receives
  struct timeval t0, t1;  // Guarda tempo percorrido

  // Sending request for password to server
  if ( send(sockfd, "5", 2, 0) < 0) {
    printf("SEND FAILURE!\n");  // DEBUG
    return;
  }

  // Sending the password string to server
  if ( send(sockfd, &tt, 30, 0) < 0) {
    printf("SEND FAILURE!\n");  // DEBUG
    return;
  }

  // Receiving the answer of server authentication
  while(1) {

    /* Cleans the string */
    memset(asw,0,5000);

    gettimeofday(&t0, 0);   // Capturando tempo de inicio
    /* Read a maximum of 500 bytes from buffer */
    if ( read_bytes = recv(sockfd, asw, 5000, 0) < 0 ) {
      printf("Erro no receive!!\n\n");
      return;
    } else {
      gettimeofday(&t1, 0); // Capturando tempo de termino
      nRevcs++;   // Atualizando contagem de receive
      // Calculando intervalo de tempo em microsegundos
      elapsed += (t1.tv_sec-t0.tv_sec)*1000000 + t1.tv_usec-t0.tv_usec;
       /* Tests if received string contains the char
        * '^D', which means TRANSMISSION OVER */
       sig=check_str(asw, '#');

       /* Testing here what is what... */
       if (sig > 0) {
         /* End Reading */
         printf("%s", asw);
         break;
       } else  /* Continue Reading! */
         printf("%s", asw);

    }
  }

  time = asw+strlen(asw)-8; // Cauda da ultima mensagem
  printf("\n");
  // Guardando num log CSV
  logger("5",elapsed - atoi(time),nRevcs,atoi(time));
  elapsed = 0;
}

void pass(int sockfd, char pwd[]) {
  char asw[5000]; // Response from server

  // Sending request for password to server
  if ( send(sockfd, "7", 2, 0) < 0) {
    printf("SEND FAILURE!\n");  // DEBUG
    return;
  }

  // Sending the password string to server
  if ( send(sockfd, pwd, 50, 0) < 0) {
    printf("SEND FAILURE!\n");  // DEBUG
    return;
  }

  // Receiving the answer of server authentication
  if ( recv(sockfd, asw, 200, 0) < 0 ) {
    printf("[1] RECEIVE FAILURE\nasw: %s", asw ); // DEBUG
    return;
  }

```

```
  printf("%s",asw);
}


/* ------------------------------------------------------------------------ */

int dataFetch(int sockfd, char *ISBN, char op[]) {
  char asw[5000]; // Answer from the Server
  char *time; // Operation Time from the Server
  int read_bytes, sig=0, errc=0;

  long elapsed = 0;   // Guarda intervalo de tempo
  int nRevcs = 0;   // Guarda numero de receives
  struct timeval t0, t1;  // Guarda tempo percorrido

  /* Formating output */
  printf("\n");

  /* Sends the Request to the Server and check errors*/
  if ( send(sockfd, op, 2, 0) < 0 ) {
    printf("Sending Error! Aborting!\n");
    return -1;
  }

  if ( op[0] != '1' && op[0] != '4' ) {
    /* Send the ISBN required to the operation in case
     * of operations 2 and 3 */
    if ( send(sockfd, ISBN, strlen(ISBN), 0) < 0 ) {
      printf("Sending Error! Aborting!\n");
      return -1;
    }

  }


  /* Reading L00P! Read from the buffer as long there is data at the buffer,
   * if receives the signal to stop (char '^D') then stop reading */

  while(1) {

    /* Cleans the string */
    memset(asw,0,5000);

    gettimeofday(&t0, 0);   // Capturando tempo de inicio
    /* Read a maximum of 500 bytes from buffer */
    if ( read_bytes = recv(sockfd, asw, 5000, 0) < 0 )
       return -1;
    else {
      gettimeofday(&t1, 0); // Capturando tempo de termino
      nRevcs++;   // Atualizando contagem de receive
      // Calculando intervalo de tempo em microsegundos
      elapsed += (t1.tv_sec-t0.tv_sec)*1000000 + t1.tv_usec-t0.tv_usec;
       /* Tests if received string contains the char
        * '^D', which means TRANSMISSION OVER */
       sig=check_str(asw, '#');

       /* Testing here what is what... */
       if (sig > 0) {
         /* End Reading */
         printf("%s", asw);
         break;
       } else  /* Continue Reading! */
         printf("%s", asw);

    }
  }

  /* Formating the output! */
  time = asw+strlen(asw)-8; // Cauda da ultima mensagem
  printf("\n");
  // Guardando num log CSV
  logger(op,elapsed-atoi(time),nRevcs,atoi(time));
  elapsed = 0;
  return 0;

}

/* ------------------------------------------------------------------------ */
```

## 2.1 Explicações das Funções

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 e a conexão com o servidor da livraria;
- **void showOptions()**: simplesmente mostra ao usuário as opções de entrada para se comunicar com o servidor;
- **int dataFetch(int, char, char [])**: 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, e por fim a tarefa que deve ser executada pelo servidor. É chamada multiplas 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 'D̂' 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)**: 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)**: 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 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.

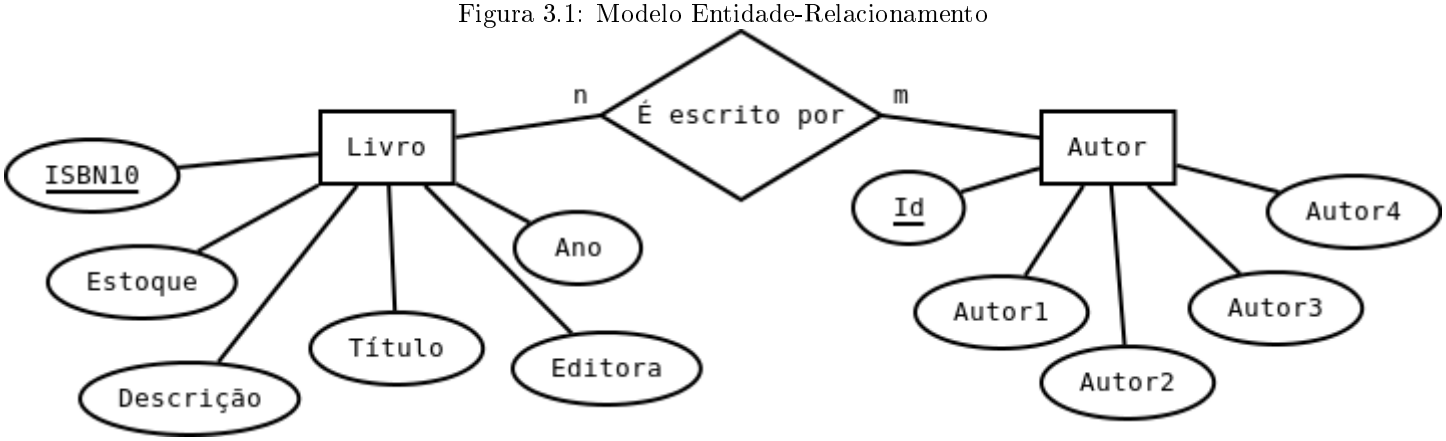
# Capítulo 3

# Banco de Dados

Para armazenar e buscar os dados, o servidor utiliza biblioteca SQLite3. O SGBD por si só é extremamente simples, não implementando planos com transações mistas por exemplo, mas para os fins do projeto ele foi suficientemente útil.

O diagrama abaixo explica de forma sucinta o Modelo Entidade-Relacionamento (MER) utilizado para implementar o banco de dados:

Figura 3.1: Modelo Entidade-Relacionamento




Como foi dito anteriormente, o servidor é o processo responsável pelo tratamento dos dados, logo (como já era de se esperar) é o processo aonde está o banco de dados e o processo que manipula os acessos ao BD.

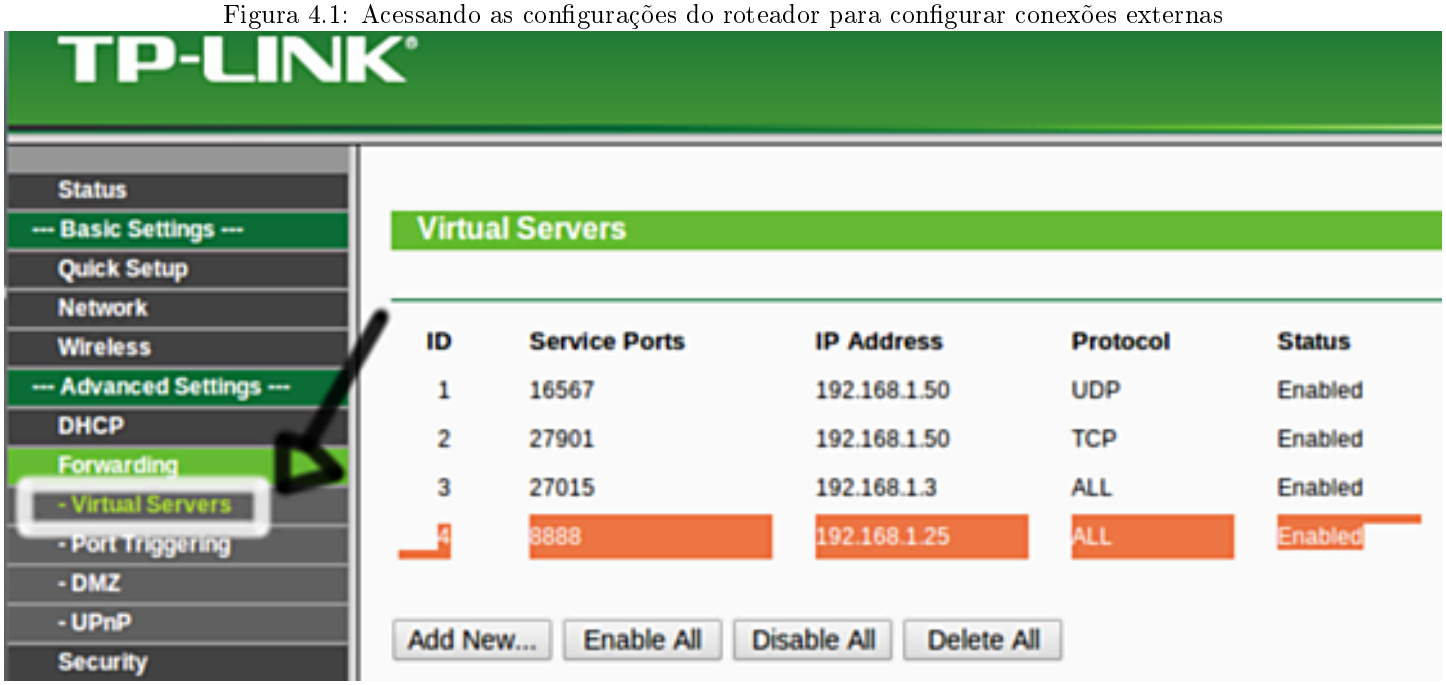
Vale observar que esses tempos foram descontados na hora de fazer a análise do tempo, já que na realidade estávamos interessados em analisar o tempo de transmissão e não o tempo total de execução do processo.

# Capítulo 4

# Rodando o Servidor em casa

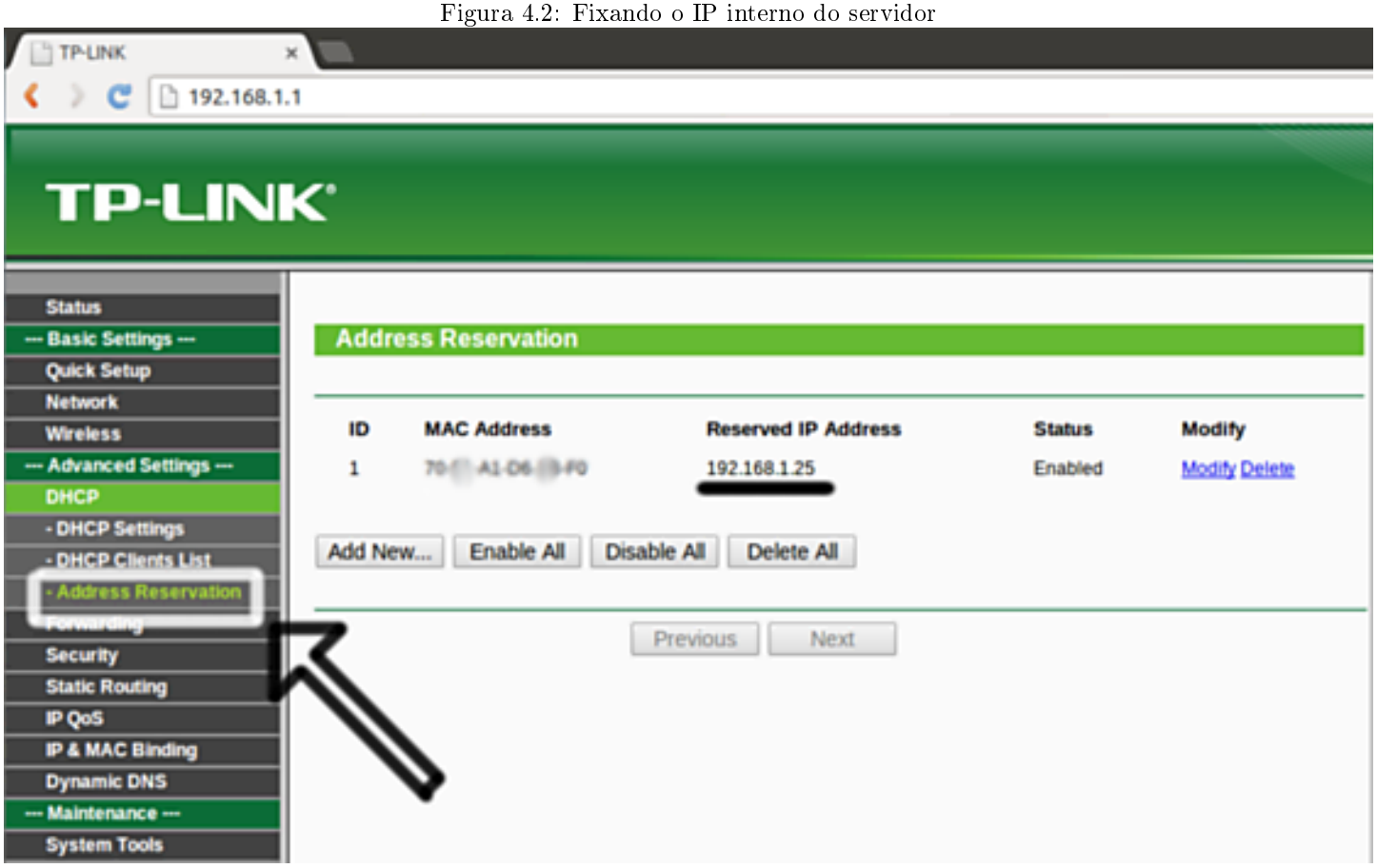
Como o roteador usa NAT para distribuir o mesmo IP para varios computadores, foi utilizado o recurso de *Virtual Address* para atribuir uma conexão externa a uma porta específica, no caso **8888**, para um computador com IP específico na rede interna. Como visto na Figura 1.

Figura 4.1: Acessando as configurações do roteador para configurar conexões externas




Após configurar o NAT, foi fixado o IP laptop que sera o servidor, via seu MAC Address, ja que o DHCP gera dinamicamente IP internos à rede. Como visto na Figura 2.

Figura 4.2: Fixando o IP interno do servidor




Como o IP provido pela NET Virtua é dinâmico, fez-se um programa em Python apenas para mandar e-mails para a dupla a fim de saber o atual IP da máquina.
Código em Python:

```
import smtplib
import time
import subprocess

while 1:

  # Encontrando o IP externo atraves de uma resposta do icanhazip.com
        ip = subprocess.check_output(['curl', '-s', 'icanhazip.com'])

        # Formatando saida do site
        ip = ip.strip()
        ip = ip.decode("utf-8")
        print(ip)

        fromaddr = 'guilhermeag@bol.com.br'
        toaddrs  = 'gagallo7@gmail.com'
        addr2 = 'andrentaz@gmail.com'
        msg = 'IP do laptop: '+ip

        print(msg)

# Credentials (if needed)
        username = 'guilhermeag'
        password = 'segredo'

# The actual mail send
    # Servidor do e-mail e o BOL
        server = smtplib.SMTP('smtp.bol.com.br:587')
        server.starttls()
        server.login(username,password)
        server.sendmail(fromaddr, toaddrs, msg)
        server.sendmail(fromaddr, addr2, msg)
        server.quit()

        time.sleep(120) #espera 2 minutos
```

# Capítulo 5

# Análise de Dados

Utilizando os arquivos **.csv** criados pela rotina *logger*, foi possível montarmos tabelas que continham os dados da transmissão.

Foi calculada a média de tempos de transferência de 119 operações para cada tipo de operação, bem como seu desvio padrão, obtendo assim o respectivo erro.

## 5.1 Servidor e Cliente na mesma máquina

Primeiramente, calculamos o tempo de transmissão entre processos rodandos na mesma máquina, ou seja, o processo cliente rodava no host. Isso servirá para comparar o atraso que a rede está dando para a comunicação. A tabela com os tempos de transmissão, computação no servidor e números de recv() estão representadas abaixo:

Como se pode perceber, a operação que mais demora em tempo de *comunicação* na média é a 4, a qual envia mais bytes. Sendo seguida da 1, que tem mesmo princípio, só que envia menos bytes que a primeira. Os tempos de *operação* de 2 e 3 são maiores, porque o SQLite3 tem que fazer a busca no banco de dados, o qual está armazenado no HD do servidor. Na verdade, o fator mais importante para a lentidão é que o servidor espera para receber o valor do ISBN nestes dois casos.

Como explicado anteriormente, esses dados foram retirados da situação aonde ambos os processos estão rodando na mesma máquina host. Pelo gráfico, podemos observar que como esperado, a operação mais custosa a nível de tempo é a de enviar todas as informações a respeito de todos os livros para o cliente (barra amarela).

Entretanto não podemos dizer que o tamanho da mensagem na hora do envio é o único fator de peso na hora de se determinar o tempo necessário para o mesmo. Observando por exemplo a operação de Listar todos os ISBNs e seus respectivos Títulos (barra azul escura), vemos um grande aumento no tempo de transmissão. No caso, o total de dados enviado pelo servidor nesse tipo de operação nem se compara ao tamanho da mensagem em casos como o da operação que envia a descrição de uma dado ISBN (barra laranja). Por exemplo, para o livro "Dom Casmurro", a descrição do livro supera em muito o tamanho da mensagem enviada para o cliente quando os ISBNs estão sendo listados.

Contudo, a resposta para essa grande quantidade de tempo gasta também pode ser facilmente encontrada se analisarmos outro dado nas tabelas. Podemos ver que para as duas operações mais custosas, a quantidade de procedimentos recv() chamados foi esmagadoramente maior, ou seja, uma grande quantidade de mensagens foi enviada! Isso com certeza é um fator determinante no tempo.

Apesar dessa análise (com processos rodando na mesma máquina) ser de boa ajuda para compreender como funciona o envio de dados, devemos fazer uma análise mais complexa, para um caso real, onde os processos client e server rodam em máquinas diferentes.

Figura 5.1: Dados da Transmissão em todas as operações numa mesma Máquina

| Operação | Média de tempo de Comunicação (us) | Desvio Padrão (us) | Nº médio Recv() | Tempo de Operação (us) |
|---|---|---|---|---|
| 1 | 320 | 40 | 20 | 20 |
| 2 | 60 | 10 | 2 | 38500 |
| 3 | 60 | 20 | 2 | 39300 |
| 4 | 440 | 90 | 4 | 39300 |
| 5 | 90 | 20 | 1 | 40000 |
| 6 | 70 | 20 | 2 | 39800 |

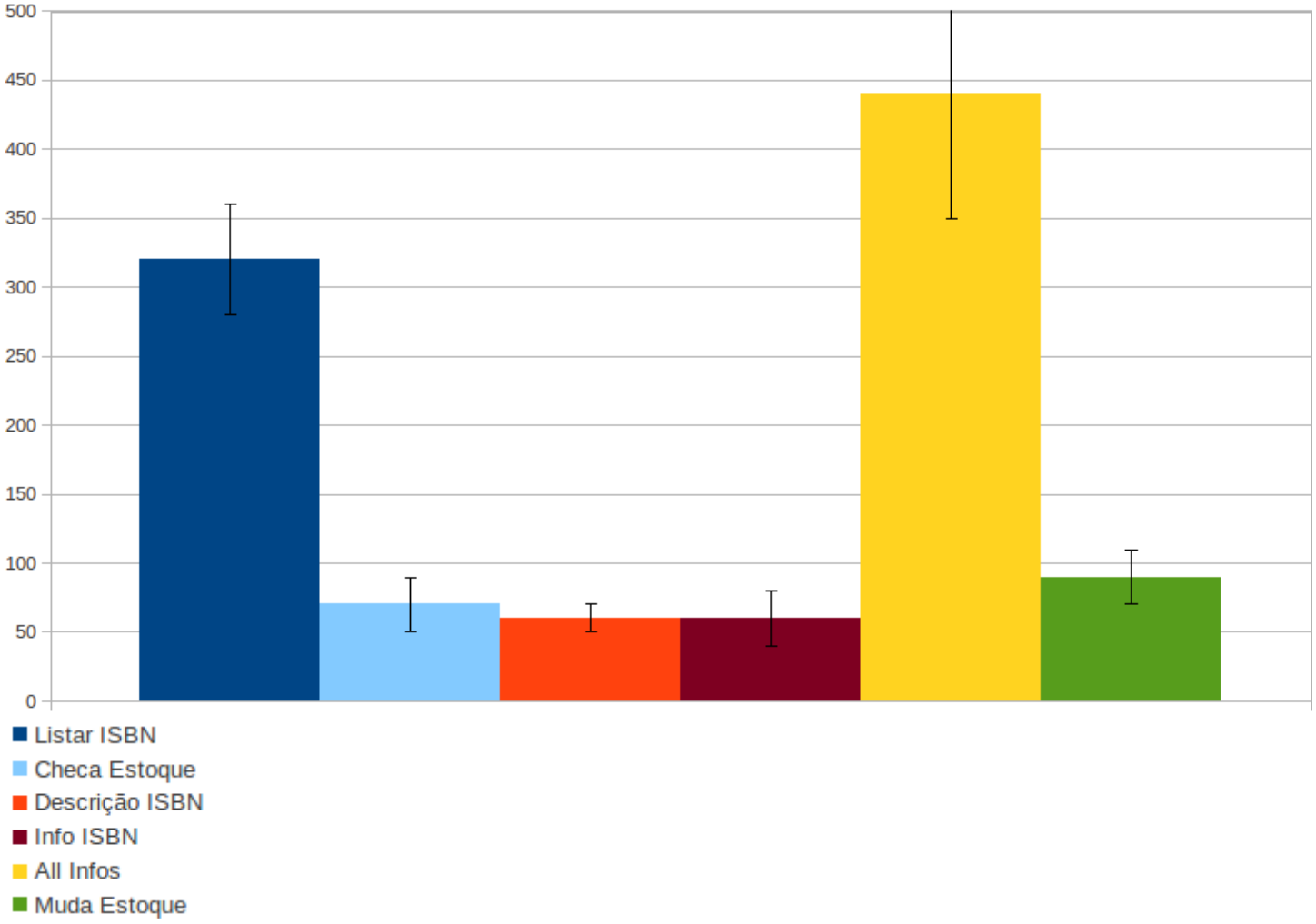



Figura 5.2: Gráfico Operação x Tempo Médio de Comunicação

## 5.2 Servidor numa rede e Cliente na outra

Para isso, o processo server foi executado no notebook do aluno Guilherme Alcarde Gallo, enquanto que o processo client teve sua execução na máquina de nome 'costa', da sala 301, IC03. Vale notar que neste caso, os além de rodar em máquinas diferentes, os processos rodavam em máquinas que estavam em redes completamente distintas. Isso contribuiu de forma significativa para os tempos de transmissão medidos, conforme podemos ver nas tabelas e gráficos abaixo, montados com os programas Calc e Writer do Libre Office:

Os resultados a seguir foram obtidos quando o servidor estava localizado na rede Eduroam e o Cliente na rede do Instituto de Computação:

Neste caso, podemos observar que os tempos para quase todas as operações (apesar de suas barras de erros) são muito semelhantes. De fato, se considerarmos os erros, podemos dizer que são quase iguais.

Novamente, se observarmos a tabela correspondente ao gráfico (Figura 5.3), podemos ver que diferentemente da tabela anterior, as duas operações mais discrepantes (Listar ISBN e Todas as Informações) enviam um número de mensagens semelhante ao enviado pelas outras mensagens.

Nesse caso, a operação mais custosa se manteve como a que Lista todas as Informações (o que era de se esperar, já que está operação é a que envia a maior quantidade de dados indiscutivelmente), entretanto, a diferença para a segunda operação mais custosa (nesse caso, a que devolve as informações de um dado ISBN) é de apenas 1000 us.

Figura 5.3: Dados da Transmissão em todas as operações em redes diferentes

| Operação | Média de tempo de Comunicação (us) | Desvio Padrão (us) | Nº médio Recv() | Tempo de Operação (us) |
|---|---|---|---|---|
| 1 | 6000 | 2000 | 2 | 16 |
| 2 | 6000 | 2000 | 2 | 40000 |
| 3 | 7000 | 3000 | 2 | 40000 |
| 4 | 8000 | 2000 | 7 | 30 |
| 5 | 4000 | 2000 | 1 | 160000 |
| 6 | 3000 | 2000 | 1 | 40000 |

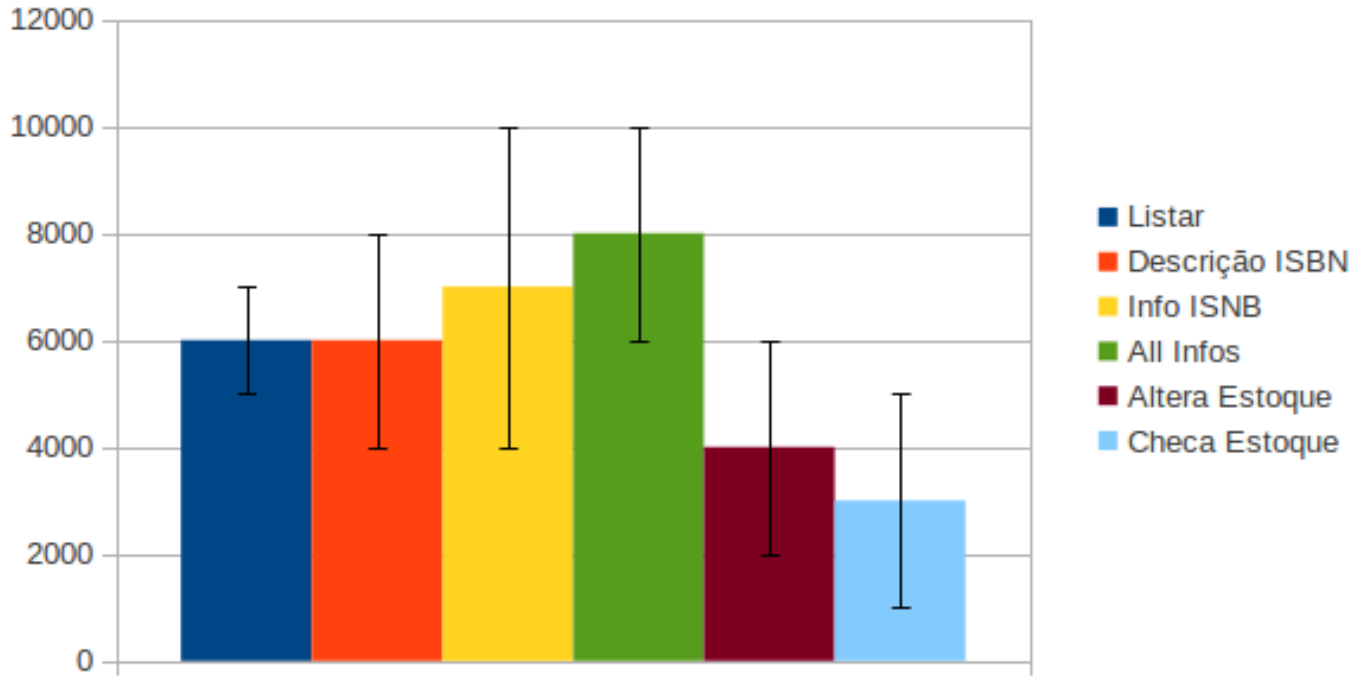



Figura 5.4: Gráfico Operação x Tempo Médio de Comunicação

Cabe notar algo interessante aqui: quando os processos rodaram no mesmo host, o tempo para listar as informações de um determinado ISBN era aproximadamente 18,75% do tempo para listar todos os ISBNs e seus Títulos, mesmo que na grande maioria das vezes a mensagem enviada para listar a informação seja muito maior do que a da outra operação.

Agora que as máquinas rodam em dois hosts diferentes, podemos notar que não só a situação se inverteu como a operação mais barata (enviar ISBNs e seus Títulos) consome um tempo equivalente a 85,71% do tempo de Listar as Informações de um dado ISBN. Ou seja, ambos os tempos estão muito próximos.

# Capítulo 6

# Conclusão

Como observado na parte da análise, existem muitos fatores que determinam os tempos de envio das mensagens através de um protocólo TCP.

Primeira e obviamente, a rede pela qual a mensagem trafega. Quando executamos os processos em duas máquinas diferentes, os tempos subiram de 440 us para 8000 us (para o processo mais lento), ou seja, quase 20 vezes mais lento! Isso era de se esperar, já que os processos não só estavam em máquinas diferentes, mas em redes diferentes.

Em segundo lugar, o tamanho da mensagem! Tanto para processos rodando num mesmo host, quanto para processos executados em hosts separados, foi possível observar que a operação que mais consumiu tempo foi a que enviava a maior mensagem.

Por fim, tão determinante quanto, a quantidade de mensagens enviada. No caso desse projeto, podemos dizer que esse foi o fator determinante para se determinar qual seria a operação mais custosa.

Apesar dos tamanhos das mensagens serem importantes, como estamos enviando os dados através do protocolo TCP (um protocolo confiável) cada mensagem enviada recebia um tratamento de checagem de erro e de ordem de envio. Ou seja, como no caso dos hosts serem os mesmos, a mensagem foi quebrada em 20 diferentes chamadas de recv(), isso significou que esse protocólo de checagem foi feito muito mais vezes do que no caso de outras mensagens que foram enviadas com apenas 1 único recv().

Talvez, quando realizarmos o projeto utilizando um protocólo de transporte como o UDP, esse fator não seja mais o determinante, já que nesse caso, as mensagens não recebem essa checagem a nível de transporte, sendo que ela deve ser implementada a nível de aplicação.

Na prática, percebe-se que o protocolo TCP faz o que promete: mantém uma conexão entre 2 computadores e não deixa nenhuma mensagem se perder. Tanto que a função de read/write ou receive/send só se preocupa em cortar o buffer e saber se enviou/leu tudo o que lhe foi requisitado.

A alta compatibilidade do Unix com este tipo de protocolo também foi muito importante, visto que não houveram problemas com a parte da conexão TCP.

A facilidade de uso do SQLite3 também deve ser citada. Pode salvar muito tempo de trabalho se comparado a fazer um banco de dados em arquivo para um projeto desse porte.

Um fato curioso foi que podemos usar o sistema servidor-cliente e até protocolos de rede para fazer comunicação inter-processos. Apesar de não ser a proposta principal, o sistema na mesma máquina funcionou bem.

# Capítulo 7

# 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).