MC823 - Relatório - Servidor Interativo sobre UDP

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

9 de Maio de 2013

Conteúdo

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 vardade 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 protocólo 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 querys 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:

```
case 1:   // Lista de ISBN e titulo dos livros
        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, "          %6li#", elapsed); // Caractere # 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);
        if ((numbytes = sendto(sockfd, query, strlen(query), 0, (struct sockaddr *) &
            their_addr, addr_len)) == -1) {
          perror("server: sendto");
          exit(1);
        }
        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 necessario o número ISBN. Desse forma, são necessarios duas operações de envio por parte do client. Além disso, é necessario mais do que simplesmente realizar a query, verificar se o ISBN realmente consta no Banco de Dados.

Para exemplicar 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);
        }
```

```
48          // Fim da mensagem
            // ReCalculo do tempo de operacao
50          elapsed += (t1.tv_sec - t0.tv_sec) * 1000000 + t1.tv_usec
              - t0.tv_usec;
52          // Transformando em string com chars de "seguranca" para postumo atoi
            sprintf(query, "          %6li#", elapsed);
54          // DEBUG
            printf("\nOperation Time: %s\n\n", query);
56          // Calculando o tamanho
            length = strlen(query);
58          // Finalmente envia para o cliente
            //sendall(client_sock, query, &length);
60          if ((numbytes = sendto(sockfd, query, strlen(query), 0,
                    (struct sockaddr *) &their_addr, addr_len)) == -1) {
62            perror("server: sendto");
              exit(1);
64          }
            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 poderiamos 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:

```
case 2:   // Descricao de um livro
        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;

        printf("\nISBN1:%s\n", client_message);
        if ((numbytes = recvfrom(sockfd, client_message, MAXBUFLEN - 1, 0,
                (struct sockaddr *) &their_addr, &addr_len)) == -1) {
          perror("recvfrom");
          exit(1);
        }
        gettimeofday(&t0, 0);
        // Montando a query
        strcpy(query, "select descricao from livro where ISBN10 = ");
        strcpy(query2,
            "select count(descricao) from livro where ISBN10 = ");
        printf("\nISBN2:%s\n", client_message);
        // 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) {
          //        sendall(client_sock, "\nEste ISBN nao consta na nossa livraria!\n",&length
              );
          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);
          }
        } else
          // Executando query - Callback ja faz os sends
          rc = sqlite3_exec(db, query, callback, 0, &zErrMsg);
        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;
```

2.3 Código Completo

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

```
/*
 ** listener.c -- a datagram sockets "server" demo
 */

#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <errno.h>
#include <string.h>
#include <sys/types.h>
#include <sys/socket.h>
#include <netinet/in.h>
#include <arpa/inet.h>
#include <netdb.h>

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

// SQLite3
#include <sqlite3.h>

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

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

int socket_desc, client_sock, c, read_size, connfd;

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

```

```
// get sockaddr, IPv4 or IPv6:
void *get_in_addr(struct sockaddr *sa) {
  if (sa->sa_family == AF_INET) {
    return &(((struct sockaddr_in*) sa)->sin_addr);
  }

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

// 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);
  return 0;
}

// Callback do SQLite3. Versao 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);
  gettimeofday(&t1, 0);
  elapsed += (t1.tv_sec-t0.tv_sec)*1000000 + t1.tv_usec-t0.tv_usec;
  if ((numbytes = sendto(sockfd, aux, strlen(aux), 0,
          (struct sockaddr *) &their_addr, addr_len)) == -1) {
    perror("server: sendto");
    exit(1);
  }
  gettimeofday(&t0, 0);
  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("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;
  if ((numbytes = sendto(sockfd, aux, strlen(aux), 0,
          (struct sockaddr *) &their_addr, addr_len)) == -1) {
    perror("server: sendto");
    exit(1);
  }
  gettimeofday(&t0, 0);
  return 0;
}

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


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

  int opcao;

  // 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);

  memset(&hints, 0, sizeof hints);
  hints.ai_family = AF_UNSPEC; // set to AF_INET to force IPv4
  hints.ai_socktype = SOCK_DGRAM;
  hints.ai_flags = AI_PASSIVE; // use my IP

  // Servidor interativo
  while (1) {
    if ((rv = getaddrinfo(NULL, MYPORT, &hints, &servinfo)) != 0) {
      fprintf(stderr, "getaddrinfo: %s\n", gai_strerror(rv));
      return 1;
    }

    // loop through all the results and bind to the first we can
    for (p = servinfo; p != NULL; p = p->ai_next) {
      if ((sockfd = socket(p->ai_family, p->ai_socktype, p->ai_protocol))
          == -1) {
        perror("listener: socket");
        continue;
      }

      if (bind(sockfd, p->ai_addr, p->ai_addrlen) == -1) {
        close(sockfd);
        perror("listener: bind");
```

```
        continue;
      }

      break;
    }

    if (p == NULL) {
      fprintf(stderr, "listener: failed to bind socket\n");
      return 2;
    }

    freeaddrinfo(servinfo);

    printf("listener: waiting to recvfrom...\n");

    addr_len = sizeof their_addr;
    if ((numbytes = recvfrom(sockfd, buf, MAXBUFLEN - 1, 0,
            (struct sockaddr *) &their_addr, &addr_len)) == -1) {
      perror("recvfrom");
      exit(1);
    }
    // Comecando a contar o tempo de operacao
    gettimeofday(&t0, 0);

    printf("listener: got packet from %s\n",
        inet_ntop(their_addr.ss_family,
          get_in_addr((struct sockaddr *) &their_addr), s,
          sizeof s));

    printf("listener: packet is %d bytes long\n", numbytes);
    buf[numbytes] = '\0';
    printf("listener: packet contains \"%s\"\n", buf);

    opcao = atoi(buf);

    switch (opcao) {
      case 1:   // Lista de ISBN e titulo dos livros
        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, "          %6li^D", elapsed);  // Caractere ^D e um identificador de
            fim da mensagem
        // DEBUG
        printf("\nOperation Time: %s\n\n", query);
        // Calculando o tamanho
        // Finalmente envia para o cliente
        if ((numbytes = sendto(sockfd, query, strlen(query), 0, (struct sockaddr *) &
            their_addr, addr_len)) == -1) {
          perror("server: sendto");
          exit(1);
        }
        break;

        /*********************************/

      case 2:   // Descricao de um livro
        elapsed = 0;
        // Esperando o cliente mandar o ISBN desejado
        // Tempo percorrido ate agora
        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
276     // 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
280     //  gettimeofday(&t1, 0);
        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);
292     gettimeofday(&t1, 0);
        // Fim da mensagem
294     // ReCalculo do tempo de operacao
        elapsed += (t1.tv_sec - t0.tv_sec) * 1000000 + t1.tv_usec
296       - t0.tv_usec;
        // Transformando em string com chars de "seguranca" para postumo atoi
298     sprintf(query, "          %6li^D", elapsed);
        // DEBUG
300     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");
306       exit(1);
        }
308     break;

310     /*****************************************/

312
      case 3:   // Todas as informacoes de um livro
314     elapsed = 0;
        // Esperando o cliente mandar o ISBN desejado
316     // Tempo percorrido ate agora
        gettimeofday(&t1, 0);
318     // ReCalculo do tempo de operacao
        elapsed += (t1.tv_sec - t0.tv_sec) * 1000000 + t1.tv_usec
320       - t0.tv_usec;
        // Esperando o cliente mandar o ISBN desejado
322     if ((read_size = recvfrom(sockfd, client_message, 2000, 0,
                (struct sockaddr *) &their_addr, &addr_len)) == -1) {
324     }
```

```
        gettimeofday(&t0, 0);
326     // Montando a query

328     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 = ");
330     strcpy(query2, "select count(*) from livro where ISBN10 = ");
        // Concatenando o ISBN
332     strcat(query, client_message);
        strcat(query2, client_message);
334     // Fim do comando SQLite
        strcat(query, ";");
336
        // Verificando se ha livros
338     // callbackSilent nao envia dados ao cliente
        // existe recebe o resultado de contagem de livros (0 ou 1)
340     rc = sqlite3_exec(db, query2, callbackSilent, existe, &zErrMsg);
        if (*((int *) existe) == 0) {
342       gettimeofday(&t1, 0);
          if ((numbytes = sendto(sockfd,
344                 "\nEste ISBN nao consta na nossa livraria!\n", 42,
                   0, (struct sockaddr *) &their_addr, addr_len))
346           == -1) {
            perror("server: sendto");
348         exit(1);
          }
350       // Tempo percorrido ate agora
        } else {
352       // Executando query - Callback ja faz os sends
          rc = sqlite3_exec(db, query, callbackFmt, 0, &zErrMsg);
354       // Tempo percorrido ate agora
          gettimeofday(&t1, 0);
356     }

358     // Fim da mensagem
        // ReCalculo do tempo de operacao
360     elapsed += (t1.tv_sec - t0.tv_sec) * 1000000 + t1.tv_usec
          - t0.tv_usec;
362     // Transformando em string com chars de "seguranca" para postumo atoi
        sprintf(query, "          %6li^D", elapsed);
364     // DEBUG
        printf("\nOperation Time: %s\n\n", query);
366     // Calculando o tamanho
        // Finalmente envia para o cliente
368     if ((numbytes = sendto(sockfd, query, strlen(query), 0,
                (struct sockaddr *) &their_addr, addr_len)) == -1) {
370       perror("server: sendto");
          exit(1);
372     }
        break;
374
      case 4:   // Todas as informacoes de todos os livros
376     elapsed = 0;
        // Executando query
378     rc =
          sqlite3_exec(db,
380           "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);
382
        // Fim da mensagem
384     // Tempo percorrido ate agora
        gettimeofday(&t1, 0);
386     // Calculo do tempo de operacao
        elapsed = (t1.tv_sec - t0.tv_sec) * 1000000 + t1.tv_usec
388       - t0.tv_usec;
        // Transformando em string com chars de "seguranca" para postumo atoi
390     sprintf(query, "          %6li^D", elapsed);
        // DEBUG
392     printf("\nOperation Time: %s\n\n", query);
        // Calculando o tamanho
```

```
394         // Finalmente envia para o cliente
            if ((numbytes = sendto(sockfd, query, strlen(query), 0,
396                 (struct sockaddr *) &their_addr, addr_len)) == -1) {
              perror("server: sendto");
398           exit(1);
            }
400
            break;
402
            /****************************************/
404
        case 5:   // Atualizar estoque
406       elapsed = 0;
          // Tempo percorrido ate agora
408       gettimeofday(&t1, 0);
          // ReCalculo do tempo de operacao
410       elapsed += (t1.tv_sec - t0.tv_sec) * 1000000 + t1.tv_usec
            - t0.tv_usec;
412       if ((read_size = recvfrom(sockfd, &cm, 2000, 0,
                (struct sockaddr *) &their_addr, &addr_len)) == -1) {
414         perror("server: sendto");
            exit(1);
416       }
          gettimeofday(&t0, 0);
418       if (superuser) {
            // Montando a query
420         strcpy(query, "update livro set estoque = ");
            // Concatenando o nova quantidade do estoque
422         strcat(query, cm.q);
            // Concatenando o ISBN
424         strcat(query, " where ISBN10 = ");
            strcat(query, cm.i);
426         // Fim do comando SQLite
            strcat(query, ";");
428         printf("%s\n", query);
            // Executando query - Callback ja faz os writes
430         rc = sqlite3_exec(db, query, callback, 0, &zErrMsg);
          } else {
432         if ((numbytes = sendto(sockfd, "Sem permissoes para modificar estoque!\n", 42, 0,
                  (struct sockaddr *) &their_addr, addr_len)) == -1) {
434           perror("server: sendto");
              exit(1);
436         }

438       }

440       // Fim da mensagem
          // Tempo percorrido ate agora
442       gettimeofday(&t1, 0);
          // ReCalculo do tempo de operacao
444       elapsed += (t1.tv_sec - t0.tv_sec) * 1000000 + t1.tv_usec
            - t0.tv_usec;
446       // Transformando em string com chars de "seguranca" para postumo atoi
          sprintf(query, "          %6li^D", elapsed);
448       // DEBUG
          printf("\nTime: %s\n\n", query);
450       // Calculando o tamanho
          // Finalmente envia para o cliente
452       if ((numbytes = sendto(sockfd, query, strlen(query), 0,
                (struct sockaddr *) &their_addr, addr_len)) == -1) {
454         perror("server: sendto");
            exit(1);
456       }

458       break;

460   case 6:   // Mostra estoque de um livro
          elapsed = 0;
462       // Tempo percorrido ate agora
          gettimeofday(&t1, 0);
464       // ReCalculo do tempo de operacao
          elapsed += (t1.tv_sec-t0.tv_sec)*1000000 + t1.tv_usec-t0.tv_usec;
```

```
466         // Esperando o cliente mandar o ISBN desejado
            if ((numbytes = recvfrom(sockfd, client_message, MAXBUFLEN - 1, 0,
468                 (struct sockaddr *) &their_addr, &addr_len)) == -1) {
              perror("recvfrom");
470           exit(1);
            }
472         gettimeofday(&t0, 0);
            // Montando a query
474         strcpy(query, "select estoque from livro where ISBN10 = ");
            // Concatenando o ISBN
476         strcat(query, client_message);
            // Fim do comando SQLite
478         strcat(query, ";");
            printf("\n%s\n", query);
480         // Executando query - Callback ja faz os writes
            rc = sqlite3_exec(db, query, callbackFmt, 0, &zErrMsg);
482         // Fim da mensagem
            // Tempo percorrido ate agora
484         gettimeofday(&t1, 0);
            // ReCalculo do tempo de operacao
486         elapsed += (t1.tv_sec-t0.tv_sec)*1000000 + t1.tv_usec-t0.tv_usec;
            // Transformando em string com chars de "seguranca" para postumo atoi
488         sprintf(query,"          %6li^D",elapsed);
            // DEBUG
490         printf("\nTime: %s\n\n",query);
            // Calculando o tamanho
492         // Finalmente envia para o cliente
            if ((numbytes = sendto(sockfd, query, strlen(query), 0,
494                 (struct sockaddr *) &their_addr, addr_len)) == -1) {
              perror("server: sendto");
496           exit(1);
            }
498         break;
          case 7:   // Autentica o cliente livraria
            // Recebe a senha
            if ((numbytes = recvfrom(sockfd, client_message, 50, 0,
502                 (struct sockaddr *) &their_addr, &addr_len)) == -1) {
              perror("recvfrom");
504           exit(1);
            }
506           // Compara as senhas
              if( (strcmp(client_message,PASSWORD) ) == 0) {
508             superuser = 1;     // Sessao de superusuario
                if ((numbytes = sendto(sockfd, "Bem-vindo, Chuck Norris!\n\n", 31, 0,
510                     (struct sockaddr *) &their_addr, addr_len)) == -1) {
                  perror("server: sendto");
512               exit(1);
                }
514           }
              else {
516             superuser = 0;     // Usuario invalido
                if ((numbytes = sendto(sockfd, "Senha Invalida!\n\n", 18, 0,
518                     (struct sockaddr *) &their_addr, addr_len)) == -1) {
                  perror("server: sendto");
520               exit(1);
                }
522           }
              memset(client_message,0,strlen(client_message));
524         break;

526     }
        close(sockfd);
528   }
      return 0;
530 }
```

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 protocólo 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

```
/* Client.c - UDP Socket ------------------------------------------------------
 * 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 UDP sockets and SQLite3 librarys */

/* Include all the stuff need to execute the program */
#include "Client.h"
#define MYPORT "4950"
#define SERVIP "143.106.16.244"   // Ribeiro - IC301

/* Main function */
int main (int argc, char *argv[]) {
  /* Control Variables */
  char op, isbn[11], pwd[50], qtt[4], sIP[20];

  /* With the connection done, read to send requests to the server */
  int sockfd;
  struct addrinfo hints, *servinfo, *tmp;
  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


  /* Defining the IP */
  if (argv[1] == NULL)  strcpy(sIP, SERVIP);     /* Default */
  else if (argv[1]=="0")  strcpy(sIP, "127.0.0.1"); /* Host */
  else       strcpy(sIP, argv[1]);    /* Passed */

  /* Setting hints to get the list of addrinfo struct */
  memset(&hints, 0, sizeof hints);
  hints.ai_family = AF_UNSPEC;  /* Any kind of IP */
  hints.ai_socktype = SOCK_DGRAM; /* UDP */

```

```
40  /* Create the list of structs */
    if (getaddrinfo (sIP,MYPORT, &hints, &servinfo) != 0) {
42    printf("Erro na alocacao de Enderecos!\n");
      return -1;
44  }

46  /* Take the first of the addrinfo inside the list */
    for (tmp = servinfo; tmp != NULL; tmp = tmp->ai_next) {
48    /* Get the Socket Descriptor */
      if ((sockfd = socket(tmp->ai_family, tmp->ai_socktype, tmp->ai_protocol)) == -1 ) {
50      printf("Error on socket creation! Trying next address struct!\n");
        continue;
52    }
      break;
54  }

56  if (tmp == NULL) {
      printf("Falha ao criar sockets! Abortando!\n");
58    return -2;
    }
60


62  /* Start the loop of requests to the server! */
    while(1) {
64    showOptions();  // Explains the options to the User

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


68
      switch(op) {
70      case 'h': // A little help
          break;
72


74      case 'l': // Looking at the Store
          if( dataFetch(sockfd, NULL, "1", tmp) < 0)
76          printf("PROBLEMS!!!!!!!\n");
          break;
78


80      case 'd': // Searching for Description
          printf("Waiting for ISBN of the Book!\n");
82        scanf(" %s", isbn); // Getting ISBN

84        /* Calling the fetching result function */
          dataFetch(sockfd, isbn, "2", tmp);
86        break;


88
        case 'i': // Searching for Info
90        printf("Waiting for ISBN of the Book!\n");
          scanf(" %s", isbn); // Getting ISBN
92
          /* Calling the fetching result function  */
94        printf("%d",dataFetch(sockfd, isbn, "3", tmp));
          break;
96


98      case 'a': // All Infos
            /* Calling the fetching result function  */
100       if( dataFetch(sockfd, NULL, "4", tmp) < 0)
            printf("PROBLEMS!!!!!!!\n");
102       break;


104
        case 'c': // Changing the stores numbers
106       printf("Waiting for the new stock amount!\n");
          scanf(" %s", qtt);  // Getting Quantity
108       printf("Waiting for ISBN of the Book!\n");
          scanf(" %s", isbn); // Getting ISBN
110       alterStock(sockfd, isbn, qtt, tmp);
          break;
```

```
112
114       case 'n': // Numbers on stock
            printf("Waiting for ISBN of the Book!\n");
116         scanf("%s", isbn);  // Getting ISBN
118         /* Calling the stocks numbers */
            dataFetch(sockfd, isbn, "6", tmp);
120         break;
122
       case 'p':
124         printf("Digite a senha para cliente livraria...\n");
            scanf(" %s", pwd);
126         pass(sockfd, pwd, tmp);
            break;
128
130       case 'q': // Quiting the program!
            printf("Quiting now!\n");
132         break;
134
       default:  // Unknow command
136         printf("Bad instruction, try again!\n");
            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 Fillietaz  - 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
  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[], struct addrinfo *saddr) {
  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 ( sendto(sockfd, "5", 2, 0, saddr->ai_addr, saddr->ai_addrlen) < 0) {
    printf("SEND FAILURE!\n");  // DEBUG
    return;
  }

  // Sending the password string to server
  if ( sendto(sockfd, &tt, 30, 0, saddr->ai_addr, saddr->ai_addrlen) < 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 = recvfrom(sockfd, asw, 5000, 0, saddr->ai_addr, &saddr->ai_addrlen) < 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
```

```
100         * '^D', which means TRANSMISSION OVER */
            sig=check_str(asw, '^D');
102
            /* Testing here what is what... */
104         if (sig > 0) {
              /* End Reading */
106           printf("%s", asw);
              break;
108         } else  /* Continue Reading! */
              printf("%s", asw);
110
        }
112   }

114   printf("%s",asw);
      time = asw+strlen(asw)-8;
116   printf("\n");
      printf("Tempo gasto: %lius || %li || %li", elapsed - (long)atoi(time), elapsed, (long)atoi(
          time));
118   printf("\n");
      logger("5",elapsed - atoi(time),nRevcs,atoi(time));
120   elapsed = 0;
}
122
/* ------------------------------------------------------------------------ */
124
void pass(int sockfd, char pwd[], struct addrinfo *saddr) {
126   char asw[5000]; // Response from server

128   // Sending request for password to server
      if ( sendto(sockfd, "7", 2, 0, saddr->ai_addr, saddr->ai_addrlen) < 0) {
130     printf("SEND FAILURE!\n");  // DEBUG
        return;
132   }

134   // Sending the password string to server
      if ( sendto(sockfd, pwd, 50, 0, saddr->ai_addr, saddr->ai_addrlen) < 0) {
136     printf("SEND FAILURE!\n");  // DEBUG
        return;
138   }

140   // Receiving the answer of server authentication
      if ( recvfrom(sockfd, asw, 200, 0, saddr->ai_addr, &saddr->ai_addrlen) < 0 ) {
142     printf("[1] RECEIVE FAILURE\nasw: %s", asw ); // DEBUG
        return;
144   }

146   printf("%s",asw);
}
148

150 /* ------------------------------------------------------------------------ */

152 int dataFetch(int sockfd, char *ISBN, char op[], struct addrinfo *saddr) {
      char asw[5000]; // Answer from the Server
154   char *time; // Operation Time from the Server
      int read_bytes, sig=0, errc=0;
156
      socklen_t addr_len;
158
      long elapsed = 0;    // Guarda intervalo de tempo
160   int nRevcs = 0;    // Guarda numero de receives
      struct timeval t0, t1;  // Guarda tempo percorrido
162
      /* Formating output */
164   printf("\n");

166   /* Sends the Request to the Server and check errors */
      if ( sendto(sockfd, op, strlen(op), 0, saddr->ai_addr, saddr->ai_addrlen) < 0 ) {
168     printf("Sending Error! Aborting!\n");
        return -1;
170   }
```

```
172   if ( op[0] != '1' && op[0] != '4' ) {
        /* Send the ISBN required to the operation in case
174      * of operations 2 and 3 */
        if ( sendto(sockfd, ISBN, strlen(ISBN), 0, saddr->ai_addr, saddr->ai_addrlen) < 0 ) {
176       printf("Sending Error! Aborting!\n");
          return -1;
178     }

180   }


182
   /* Reading L00P! Read from the buffer as long there is data at the buffer,
184    * if receives the signal to stop (char '^D') then stop reading */
   elapsed = 0;
186
   while(1) {
188
     /* Cleans the string */
190  memset(asw,0,5000);

192  gettimeofday(&t0, 0);   // Capturando tempo de inicio

194  /* Read a maximum of 5000 bytes from buffer */
     if ( read_bytes = recvfrom(sockfd, asw, 5000, 0, saddr->ai_addr, &saddr->ai_addrlen) < 0 )
196     return -1;
     else {
198    gettimeofday(&t1, 0); // Capturando tempo de termino
       nRevcs++;   // Atualizando contagem de receive
200
       // Calculando intervalo de tempo em microsegundos
202    elapsed += (t1.tv_sec-t0.tv_sec)*1000000 + t1.tv_usec-t0.tv_usec;

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

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

216  }
   }
218
   /* Formating the output! */
220 time = asw+strlen(asw)-8;
   printf("\n");
222 printf("Tempo gasto: %lius || %li || %li -- %d", elapsed - (long)atoi(time), elapsed, (long)
       atoi(time), nRevcs);
   printf("\n");
224 printf("%s",time);
   printf("\n");
226 // Guardando num log CSV
   logger(op,elapsed-atoi(time),nRevcs,atoi(time));
228 elapsed = 0;
   return 0;
230

}
232
/* ------------------------------------------------------------------------------ */
234
/* ------------------------------------------------------------------------------ */
```

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 proprio 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 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 '^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 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 auxilio 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 parâmetreo 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 auxilio 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 analise 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. Entretantos, 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:

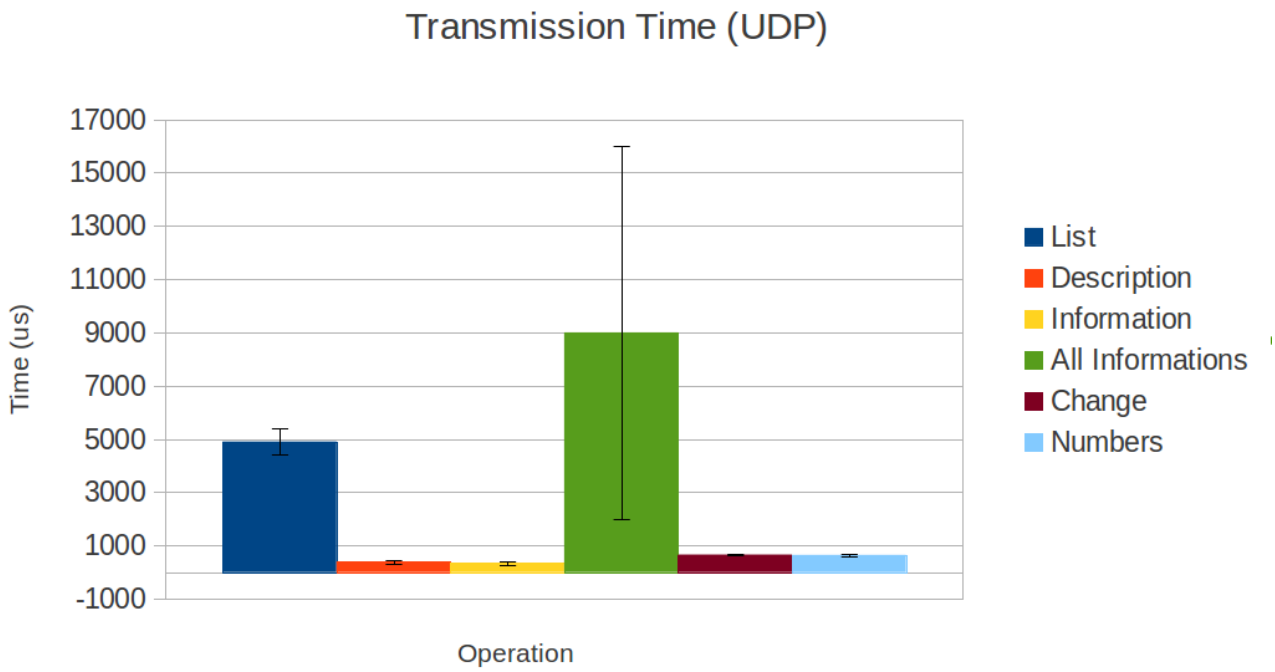

Figura 4.3: Gráfico Operação x Tempo Médio de Comunicação

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 aceitaveis, 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. Atribuimos esse grande atraso do TCP as operações que tornam o protocolo confiavel, ou seja, a checagem de erros e ordem que o protocolo executa sempre que envia mensagens.

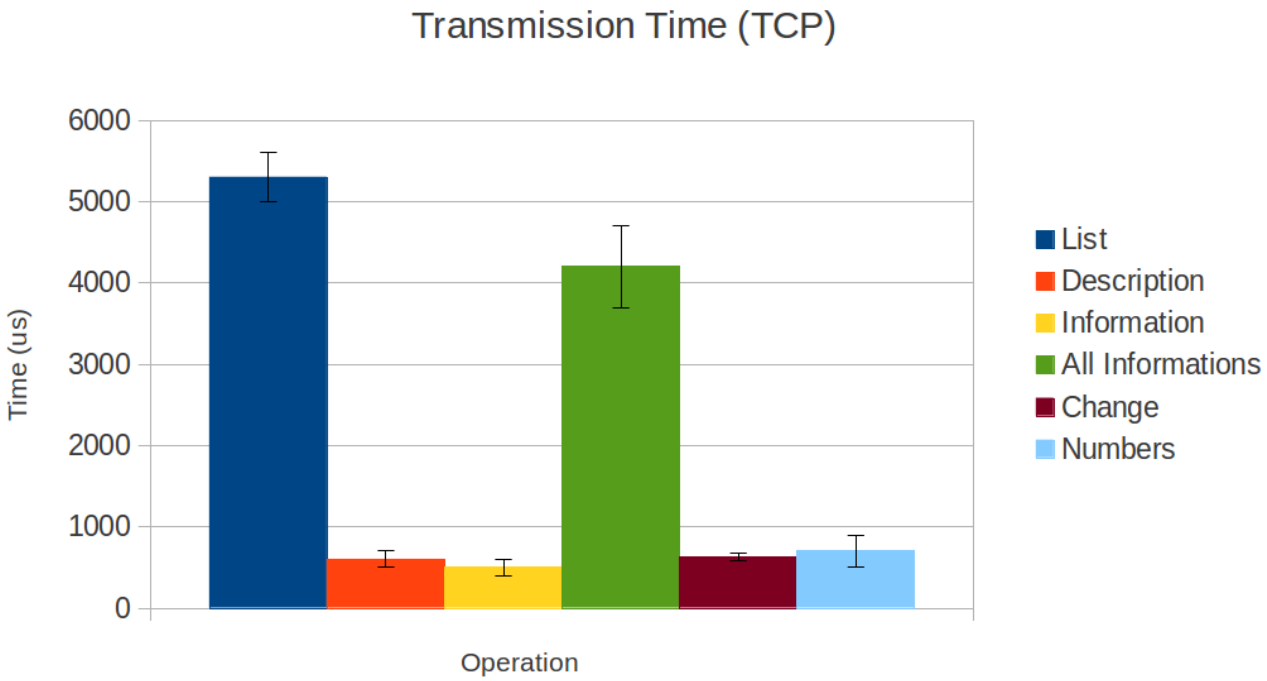

Figura 4.4: Gráfico Operação x Tempo Médio de Comunicação

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 confiavel, 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 confiavel, pela Tabela 01, pode-se observar que o tempo de envio de uma operação é fortemente influenciado pela quantidade de mensagens envolvidas nessa transmissao.

Ao final do projeto, percebe-se aqui as grandes diferenças que um protocolo confiavel 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 necessario 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).