MC823 – 1º Relatório

Marco Antônio Lasmar Almada – RA 092208
Patricia Fernanda Hongo – RA 103715

1 Introdução

O presente relatório tem como objetivo descrever a implementação e os resultados do segundo projeto de MC823 no primeiro semestre de 2013, comparando-os com o resultados obtidos no primeiro projeto. Tal como o Projeto 1, este projeto consistia na implementação de um sistema para consulta e controle de estoque em uma biblioteca, utilizando uma arquitetura cliente-servidor. Entretanto, a implementação o sistema desta vez utilizou o protocolo de camada de transporte UDP, ao invés de TCP.

O servidor armazena as seguintes informações sobre livros:

- ISBN;
- Título;
- Descrição;
- Estoque;
- Autor;
- Editora;
- Ano;

O cliente pode efetuar três tipos de operações: listar todos os livros (o cliente pode escolher entre receber todos os dados de cada livro ou somente ISBN e título), obter dados de um livro cujo ISBN é conhecido (pode pedir a descrição, o número de livros em estoque ou todos os dados do livro) ou alterar o número de unidades em estoque de um livro cujo ISBN seja conhecido.

2 Implementação

A implementação do sistema de livraria neste projeto foi bastate semelhante à implementação do Projeto 1. Cliente e servidor foram implementados em linguagem C, e seus códigos se encontram nos anexos 1 e 2. O banco de dados da livraria, novamente, foi implementado como um arquivo `.txt`. O pré-processamento utilizado foi o mesmo: o arquivo com os dados dos livros é lido pelo servidor, que armazena os dados de cada livro em um vetor de `struct livros`. Dessa forma, não é necessária uma nova consulta ao banco de dados a cada operação requisitada pelo cliente.

Após o pré-processamento, o servidor fica na escuta em um *socket* UDP, aguardando mensagens enviadas por clientes. As mensagens são tratadas iterativamente: aqui, ao contrário da implementação com TCP,

o servidor não faz chamadas a `fork` para que processos filhos cuidem das requisições. Todas são tratadas por um mesmo processo, na ordem em que são recebidas.

A fim de responder as requisições, o servidor varre o vetor de livros em busca dos dados desejados. Estes são guardados em um *buffer* de mensagens que, ao final do processamento da operação, é enviado ao cliente pela requisição via UDP.

O cliente, ao ser inicializado, fornece ao usuário um menu com as funcionalidades disponíveis ao programa, elencadas na Introdução e na especificação do projeto. O usuário seleciona uma opção e fornece os parâmetros requeridos pela operação. Com base na opção selecionada e em eventuais parâmetros, o cliente constrói uma mensagem em *buffer* específico para este fim. A solicitação é então enviada para o servidor via UDP, e a mensagem de retornada é processada e impressa na tela.

A fim de implementar a comunicação entre cliente e servidor, criamos um protocolo simples especificando como cada requisição e resposta deveria ser construída. As requisições mandam em uma única mensagem o número da operação desejada juntamente com ISBN e estoque, caso necessário. O servidor devolve como resposta uma única mensagem contendo a resposta da requisição. Tal protocolo facilita a contagem dos tempos de comunicação e consulta: caso cada requisição precisasse mandar e processar mais mensagens, servidor e cliente necessitariam armazenar e processar os tempos de comunicação e consulta de várias mensagem a fim de calcular o tempo tomado por uma única requisição.

3 Vantagens e Desvantagens da Solução

O pré-processamento pareceu uma opção viável, uma vez que não seria necessário lidar com um número grande de livros. Assim, o consumo de memória é relativamente pequeno e há um ganho em desempenho. Ao final de sua execução, o servidor salva seus dados em disco, efetivamente atualizando a versão permanente do banco de dados. Como a chance de panes no servidor é relativamente baixa, os riscos associados a essa escolha são poucos, mas um sistema mais robusto salvaria os dados de uma maneira mais eficiente. O uso de um sistema gerenciador de banco de dados facilitaria o tratamento desse risco.

A comunicação com o protocolo UDP faz com que não haja o estabelecimento de uma conexão entre servidor e cliente, ao contrário do que ocorria na implementação em TCP, o que diminui o *overhead* devido ao uso do protocolo, mas significa que não há proteção contra perda de datagramas, recebimento fora de ordem ou outros problemas na entrega dos datagramas (ou falta desta). Para lidar com tais problemas, seria necessário implementar mecanismos de controle no programa, de forma a replicar as facilidades oferecidas pelo protocolo TCP (ou um subconjunto conveniente destas).

O tamanho adotado para os *buffers* foi deliberadamente elevado, para que a preocupação em partir a mensagem em blocos adequados para transmissão não estivesse no nível da aplicação, mas sim fosse jogado para camadas mais baixas. Isso tornou prática a composição dos *buffers*, que certamente terão tamanho o bastante para lidar com a pequena escala do banco de dados em questão.

A implementação iterativa para o servidor UDP elimina o problema de concorrência que existia no servidor TCP: como não são realizados mais *forks*, não ocorre mais o problema de dois clientes tentarem alterar o estoque de um livro ao mesmo tempo. Como as requisições são processadas de forma sequencial, não há condição de corrida.

Não foram adotadas medidas de segurança para o acesso com senha à base de dados.

4 Resultados

Os resultados estão presentes na tabelas anexas a este documento. Para todas as 6 operações, as medições foram obtidas a partir da realização de 50 requisições sequenciais originadas por um único cliente.

O tempo de consulta é o tempo decorrido entre imediatamente após o servidor receber a requisição e imediatamente antes de ele enviar a mensagem com a resposta. O tempo total é medido desde antes do envio da mensagem pelo cliente até depois do recebimento da resposta por ele, e o tempo de comunicação é metade da diferença entre esses tempos.

Como no projeto 1, o tempo de comunicação representou a maior parte do tempo total de atividade do cliente; há alguma diferença no tempo de consulta, mas isso se deve ao fato de os servidores terem sido executados em máquinas diferentes, o que resulta em diferenças de desempenho nas operações locais.

Os tempos médios de comunicação verificados no projeto 2 foram entre duas e dez vezes menores do que os tempos verificados para as mesmas operações realizadas no projeto 1, dependendo da operação realizada.

5 Conclusão

Os *sockets* UDP funcionam bem para a comunicação entre processos, assim como os *sockets* TCP empregados no projeto anterior. Em testes com mais de um cliente (feitos para verificar consistência; os resultados não foram avaliados da mesma maneira que o teste para um cliente descrito na seção anterior), o servidor iterativo não apresentou problemas para processar as mensagens recebidas e enviá-las aos destinos corretos, uma vez que o UDP especifica o "endereço de retorno"para cada datagrama.

Na rede do IC, onde foram efetuados os testes cujos resultados foram apresentados aqui, não houve uma perda significativa de datagramas. Como o protocolo UDP não oferece garantias quanto à entrega dos datagramas, ao contrário do TCP, seria necessária a implementação de salvaguardas para garantir que o cliente conseguisse solicitar de novo as informações que fossem perdidas no percurso, além de um *buffer* no servidor para armazenar as requisições recebidas e processá-las na ordem de recebimento. Porém, os alunos optaram por não realizar essas implementações neste projeto, tanto pelo sistema estar sendo usado em um contexto em que as perdas são bem pequenas (de fato, uma inspeção das saídas de teste revelou poucos erros em uma inspeção visual), mas também para evidenciar as diferenças entre as duas abordagens para comunicação.

Como o *overhead* do protocolo UDP é menor que o do TCP, uma vez que ele não estabelece conexão e não oferece várias das garantias do TCP, é esperado que o tempo de comunicação usando datagramas UDP seja menor que o tempo de comunicação com pacotes TCP, resultado verificado acima. Uma vez que os testes foram realizados na mesma rede, no mesmo dia da semana e aproximadamente no mesmo horário, podemos concluir que a maior parte dessa diferença se deve às diferenças entre os protocolos, já que os niveis de uso da rede são aproximadamente os mesmos.

6 Referências

HALL, Brian. *Beej's Guide to Network Programming* – Using Internet Sockets. Versão 3.0.15, 3 de julho de 2012. Disponível em <http://beej.us/guide/bgnet/>.

Anexo 1: Código do Servidor

```
#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

#define MAXDATASIZE 200123 // max number of bytes we can get at once

#define MAXBIBSIZE 10

struct livro{
  char ISBN[14];
  char titulo[1000];
  char descricao[100123];
  int estoque;
  char autor[100];
  char editora[100];
  int ano;
};

// 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);
}

int main(void)
{
  int sockfd;
  struct addrinfo hints, *servinfo, *p;
```

```
int rv;
int numbytes;
struct sockaddr_storage their_addr;
char buf[MAXDATASIZE];
socklen_t addr_len;
char s[INET6_ADDRSTRLEN];

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

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);

//preprocessamento
struct livro biblioteca[MAXBIBSIZE];
char aux[MAXDATASIZE];
FILE *dados = fopen("dados.txt", "r");
```

```
int i = 0, j;
while(!feof(dados) ){
  fgets(biblioteca[i].ISBN, 11, dados);
  fgetc(dados);

  fgets(biblioteca[i].titulo, 1000, dados);
  fgets(biblioteca[i].descricao, MAXDATASIZE-1, dados);

  fgets(aux, MAXDATASIZE, dados);
  biblioteca[i].estoque = atoi(aux);

  fgets(biblioteca[i].autor, MAXDATASIZE-1, dados);

  fgets(biblioteca[i].editora, MAXDATASIZE-1, dados);

  fgets(aux, MAXDATASIZE-1, dados);
  biblioteca[i].ano = atoi(aux);
  i++;

}
int total_livros = i;
printf("--> %d\n", total_livros);

addr_len = sizeof their_addr;
while (1) {
  if ((numbytes = recvfrom(sockfd, buf, MAXDATASIZE-1 , 0,
                           (struct sockaddr *)&their_addr, &addr_len)) == -1) {
    perror("erro no recvfrom");
    exit(1);
  }

  int opt, cont, qte;
  char ISBN[20];
  char bufs[MAXDATASIZE]; // exit buffer
  int mudou = 0; // mudanca no estoque

  //pega tempo
  struct timeval tempo_in, tempo_fim;
  gettimeofday(&tempo_in, NULL);

  sscanf(buf, "%d", &opt);

  if(opt == 1){
    bufs[0] = '\0';
    for(i = 0; i < total_livros; ++i){
```

```
    strcat(bufs, biblioteca[i].ISBN);
    strcat(bufs, " ");
    strcat(bufs, biblioteca[i].titulo);
  }
}

else if(opt == 2){
  bufs[0] = '\0';
  sscanf(buf, "%d %s", &opt, ISBN);
  for(i = 0; i < total_livros; ++i){
    if(strcmp(biblioteca[i].ISBN, ISBN) == 0){
      strcat(bufs, biblioteca[i].descricao);
      break;
    }
  }
}

else if(opt == 3){
  bufs[0] = '\0';
  sscanf(buf, "%d %s", &opt, ISBN);
  for(i = 0; i < total_livros; ++i){
    if(strcmp(biblioteca[i].ISBN, ISBN) == 0){
      strcat(bufs, biblioteca[i].ISBN);
      strcat(bufs, "\n");
      strcat(bufs, biblioteca[i].titulo);
      strcat(bufs, biblioteca[i].descricao);
      sprintf(aux, "%d\n", biblioteca[i].estoque);
      strcat(bufs, aux);
      strcat(bufs, biblioteca[i].autor);
      strcat(bufs, biblioteca[i].editora);
      sprintf(aux, "%d\n", biblioteca[i].ano);
      strcat(bufs, aux);
      break;
    }
  }
}

else if(opt == 4){
  bufs[0] = '\0';
  for(i = 0; i < total_livros; ++i){
    strcat(bufs, biblioteca[i].ISBN);
    strcat(bufs, "\n");
    strcat(bufs, biblioteca[i].titulo);
    strcat(bufs, biblioteca[i].descricao);
    sprintf(aux, "%d\n", biblioteca[i].estoque);
```

```
    strcat(bufs, aux);
    strcat(bufs, biblioteca[i].autor);
    strcat(bufs, biblioteca[i].editora);
    sprintf(aux, "%d\n", biblioteca[i].ano);
    strcat(bufs, aux);
  }
}

else if(opt == 6){
  bufs[0] = '\0';
  //strcat(bufs, "0 ");
  sscanf(buf, "%d %s", &opt, ISBN);
  for(i = 0; i < total_livros; ++i){
    if(strcmp(biblioteca[i].ISBN, ISBN) == 0){
      sprintf(aux, "%d\n", biblioteca[i].estoque);
      strcat(bufs, aux);
      break;
    }
  }
}

else if(opt == 5){
  bufs[0] = '\0';
  //strcat(bufs, "0 ");
  sscanf(buf, "%d %s %d", &opt, ISBN, &qte);
  mudou = 1;
  for(i = 0; i < total_livros; ++i){
    if(strcmp(biblioteca[i].ISBN, ISBN) == 0){
      biblioteca[i].estoque = qte;
      break;
    }
  }
}

//pega tempo final
gettimeofday(&tempo_fim, NULL);
double tempo1, tempo2;
tempo1 = tempo_in.tv_sec + 0.000001*tempo_in.tv_usec;
tempo2 = tempo_fim.tv_sec + 0.000001*tempo_fim.tv_usec;
printf("\nTempo total: %lf\n", tempo2-tempo1);

if ((numbytes = sendto(sockfd, bufs, strlen(bufs)+1, 0,
         (struct sockaddr *)&their_addr, addr_len)) == -1) {
  perror("listener: sendto");
  exit(1);
```

```
    }

    if(mudou){
      bufs[0] = '\0';
      for(i = 0; i < total_livros-1; ++i){
        strcat(bufs, biblioteca[i].ISBN);
        strcat(bufs, "\n");
        strcat(bufs, biblioteca[i].titulo);
        strcat(bufs, biblioteca[i].descricao);
        sprintf(aux, "%d\n", biblioteca[i].estoque);
        strcat(bufs, aux);
        strcat(bufs, biblioteca[i].autor);
        strcat(bufs, biblioteca[i].editora);
        sprintf(aux, "%d\n", biblioteca[i].ano);
        strcat(bufs, aux);
      }
      FILE *fout = fopen("dados.txt", "w");
      fprintf(fout, "%s", bufs);
      fclose(fout);
    }
  }
  close(sockfd);
  return 0;
}
```

Anexo 2: Código do Cliente

```
#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>
#include <sys/time.h>

#define SERVERPORT "4950"    // the port users will be connecting to
#define MAXDATASIZE 200123

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

```
char buf[MAXDATASIZE];
struct addrinfo hints, *servinfo, *p;
struct sockaddr_storage their_addr;
socklen_t addr_len;
int rv;
int numbytes;
int estoque;

if (argc != 2) {
    fprintf(stderr,"usage: talker hostname\n");
    exit(1);
}

memset(&hints, 0, sizeof hints);
hints.ai_family = AF_UNSPEC;
hints.ai_socktype = SOCK_DGRAM;

if ((rv = getaddrinfo(argv[1], SERVERPORT, &hints, &servinfo)) != 0) {
    fprintf(stderr, "getaddrinfo: %s\n", gai_strerror(rv));
    return 1;
}

// loop through all the results and make a socket
for(p = servinfo; p != NULL; p = p->ai_next) {
    if ((sockfd = socket(p->ai_family, p->ai_socktype,
            p->ai_protocol)) == -1) {
        perror("talker: socket");
        continue;
    }

    break;
}

if (p == NULL) {
    fprintf(stderr, "talker: failed to bind socket\n");
    return 2;
}
int opt, cont;
char ISBN[20];
char msg[MAXDATASIZE];
char pass[20];
int bytes_sent, len, bytes_rcv;
while(1){
  ISBN[0] = '\0';
  //pseudo user interface
```

```
printf("Escolha uma opcao:\n");
printf("1- listar todos os ISBN e seus respectivos titulos\n");
printf("2- dado o ISBN de um livro, retornar descricao\n");
printf("3- dado o ISBN de um livro, retornar todas as informacoes do livro\n");
printf("4- listar todas as informacoes de todos os livros\n");
printf("5- alterar numero de exemplares em estoque\n");
printf("6- dado o ISBN de um livro, retornar numero de exemplares em estoque\n");
printf("7- fechar cliente\n");
scanf("%d", &opt);
if(opt == 7) break;
if(opt == 5){
  printf("Digite a senha\n");
  scanf(" %s", pass);
  if(strcmp(pass, "senhalivraria") != 0)
    continue;
  else{
    printf("Digite o ISBN\n");
    scanf(" %s", ISBN);
    printf("Digite o novo valor do estoque\n");
    scanf("%d", &estoque);
  }
}
if(opt == 2 || opt == 3 || opt == 6){
  printf("Digite o ISBN\n");
  scanf(" %s", ISBN);
}

//escreve requisicao
if (opt == 5) sprintf(msg, "%d %s %d", opt, ISBN, estoque);
else sprintf(msg, "%d %s", opt, ISBN);

 //pega tempo inicial
 struct timeval tempo_in, tempo_fim;
 gettimeofday(&tempo_in, NULL);

  len = strlen(msg);
  if ((numbytes = sendto(sockfd, msg, len, 0,
  p->ai_addr, p->ai_addrlen)) == -1) {
    perror("talker: sendto");
    exit(1);
  }
  //le resultado
  addr_len = sizeof their_addr;
  if ((numbytes = recvfrom(sockfd, buf, MAXDATASIZE-1 , 0,
  (struct sockaddr *)&their_addr, &addr_len)) == -1) {
```

```
        perror("erro no recvfrom");
        exit(1);
      }
      sscanf(buf, "%d", &cont);
      printf("%s", buf);
      gettimeofday(&tempo_fim, NULL);
      double tempo1, tempo2;
      tempo1 = tempo_in.tv_sec + 0.000001*tempo_in.tv_usec;
      tempo2 = tempo_fim.tv_sec + 0.000001*tempo_fim.tv_usec;
      printf("\nTempo total: %lf\n", tempo2-tempo1);
    }
    freeaddrinfo(servinfo);
    close(sockfd);
    return 0;
}
```