

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 primeiro projeto de MC823 no primeiro semestre de 2013. Tal projeto consistia na implementação de um sistema para consulta e controle de estoque em uma biblioteca, utilizando uma arquitetura cliente-servidor. No primeiro projeto, a comunicação entre servidor e clientes será realizada através do protocolo TCP, com uma implementação de servidor concorrente.

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.

O mesmo sistema será posteriormente reimplementado utilizando o protocolo UDP, com a finalidade de identificar as semelhanças e as diferenças no funcionamento das duas implementações. Assim, os dados obtidos durante o primeiro projeto servirão de referência para o projeto seguinte.

2 Implementação

Tanto o servidor quanto o cliente foram implementados em linguagem C, e os códigos serão apresentados como anexos deste relatório. O banco de dados foi implementado como um arquivo `.txt`, que passa por uma etapa de pré-processamento no servidor: o arquivo com os dados dos livros é lido pelo servidor, que armazena os dados de cada livro em um vetor de `struct livros`, para que não seja necessária uma nova consulta ao banco de dados a cada operação.

Após o pré-processamento, o servidor fica na escuta em um *socket* TCP, aguardando algum cliente se conectar. Ao receber uma mensagem de um cliente, o servidor faz um *fork* para processá-la, recebendo o código da operação selecionada pelo cliente e os eventuais parâmetros. O vetor de livros é então varrido em busca dos dados desejados, que são adicionados a um *buffer* de mensagens. Ao final do processamento da operação, o *buffer* é enviado ao cliente responsável pela requisição, por TCP.

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, se for o caso, fornece os parâmetros necessários. O ISBN é o tipo de parâmetro mais comum das operações que o pedem, mas a alteração de estoque também pede uma senha de acesso e o novo valor do estoque. Com base na opção selecionada e em eventuais parâmetros, o cliente constrói uma mensagem no *buffer* específico. A solicitação é então enviada para o servidor por TCP, e a mensagem de volta é processada e impressa em 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 inicialmente o número da operação desejada, juntamente com ISBN e estoque, caso necessário. O servidor devolve como resposta uma string contendo um inteiro e a resposta da requisição. Especificamos que o inteiro seria 1 caso o servidor ainda tivesse mais mensagens a mandar como resposta da requisição, e 0 caso contrário. Entretanto, visto que o número de livros do banco de dados era pequeno, observamos que seria suficiente utilizar um buffer de aproximadamente 200kB no lugar do referido mecanismo.

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, e também de outro problema decorrente de nossa escolha: caso dois processos tentem editar o estoque em tempos parecidos, não há exclusão mútua implementada, de forma que se verifica uma corrida. Porém, uma vez utilizamos a suposição de que há apenas um cliente livraria, esta passa a ser uma preocupação menor; um sistema real com múltiplos usuários com privilégio de escrita necessitaria incluir um mecanismo de tratamento de concorrência.

A comunicação com o protocolo TCP garante que não haverá perda de mensagens e que elas chegarão em ordem. Todavia, ela adiciona um *overhead* na transmissão de mensagens, que poderá ser melhor verificado a partir da comparação com o desempenho da implementação realizada em UDP. De antemão, as garantias oferecidas pelo TCP na entrega de pacotes já nos pouparam de tratar problemas de perda e ordenação de pacotes, tornando o código mais enxuto.

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.

4 Resultados

Os resultados estão presentes nas tabelas anexas a este documento. As medições realizadas para a operação 4, por um pequeno *bug*, foram feitas com cada cliente se conectando ao servidor uma vez, efetuando a solicitação e encerrando a conexão após efetuar a resposta. Todos os outros testes foram efetuados a partir da realização de 50 requisições sequenciais.

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 `write` até depois do `read`, e o tempo de comunicação é metade da diferença entre esses tempos.

5 Conclusão

Os *sockets* TCP funcionam bem para a comunicação entre processos. Embora os testes tenham sido realizados apenas em máquinas em uma mesma rede local, é possível supor que se manterá a funcionalidade quando as conexões forem realizadas entre máquinas em redes diferentes, graças às garantias oferecidas pelo protocolo TCP. Assim, a versão do sistema empregando o protocolo UDP provavelmente exigirá a implementação de salvaguardas adicionais para garantir o mesmo nível de confiabilidade atualmente obtido somente com o uso de instruções básicas para a criação dos *sockets* TCP.

A implementação realizada pode ter alguns problemas com consistência dos dados e robustez. O uso de um sistema gerenciador de banco de dados, assim como o emprego de diretivas de semáforo ou *mutexes*, eliminaria a chance de dois processos-filhos do servidor possuírem valores diferentes para o estoque, assim como facilitaria a criação de uma operação de adição de livros ao sistema. O problema também poderia ser resolvido utilizando *memory maps*, mas a pequena escala planejada para o projeto, assim como o intuito pedagógico, contribuíram para que tal solução fosse vista como algo de baixa prioridade. Também não foram adotadas medidas de segurança para o acesso com senha à base de dados.

Em geral, os tempos de consulta respondem por uma fração muito pequena do tempo total. Ou seja, mesmo em computadores dentro de uma mesma rede, há predominância do tempo de conexão, como as tabelas anexas evidenciam. Os tempos medidos para as operações 1 e 4 apresentam menor desvio padrão, que chega a ser pequeno em relação aos tempos médios, mas as demais operações apresentam desvios consideráveis em seus tempos durante a execução.

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 <netdb.h>
#include <arpa/inet.h>
#include <sys/wait.h>
#include <signal.h>
#include <sys/time.h>

#define PORT "3490" // the port users will be connecting to

#define BACKLOG 10 // how many pending connections queue will hold

#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;
};

void sigchld_handler(int s) {
    while(waitpid(-1, NULL, WNOHANG) > 0);
}

// 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, new_fd; // listen on sock_fd, new connection on new_fd
    struct addrinfo hints, *servinfo, *p;
    struct sockaddr_storage their_addr; // connector's address information
    socklen_t sin_size;
    struct sigaction sa;

```

```

int yes=1;
char s[INET6_ADDRSTRLEN];
int rv;

memset(&hints, 0, sizeof hints);
hints.ai_family = AF_UNSPEC;
hints.ai_socktype = SOCK_STREAM;
hints.ai_flags = AI_PASSIVE; // use my IP

if ((rv = getaddrinfo(NULL, PORT, &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("server: socket");
        continue;
    }

    if (setsockopt(sockfd, SOL_SOCKET, SO_REUSEADDR, &yes,
sizeof(int)) == -1) {
        perror("setsockopt");
        exit(1);
    }

    if (bind(sockfd, p->ai_addr, p->ai_addrlen) == -1) {
        close(sockfd);
        perror("server: bind");
        continue;
    }

    break;
}

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

freeaddrinfo(servinfo); // all done with this structure

```

```

if (listen(sockfd, BACKLOG) == -1) {
    perror("listen");
    exit(1);
}

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);
}

printf("server: waiting for connections...\n");

//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);
    //printf("%s \n", biblioteca[i].titulo);
    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);

//memory map
//char *addr;

```

```

//addr = mmap(NULL, sizeof(biblioteca), PROT_WRITE | PROT_READ, MAP_SHARED,
//      int fd, off_t offset);

int bytes_rcv;
char buf[MAXDATASIZE];
int opt, cont, qte;
char ISBN[20];
FILE *db = fopen("dados.txt", "rw");
while(1) { // main accept() loop
    sin_size = sizeof their_addr;
    new_fd = accept(sockfd, (struct sockaddr *)&their_addr, &sin_size);
    if (new_fd == -1) {
        perror("accept");
        continue;
    }

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

    if (!fork()) { // this is the child process
        close(sockfd); // child doesn't need the listener

        //LOOP QUE PROCESSA REQUISICOES
        char bufs[MAXDATASIZE];
        opt = -1;
        /**/while(1){
if ((bytes_rcv = recv(new_fd, buf, MAXDATASIZE-1, 0)) == -1) {
    perror("erro no recv");
    break;
}

int mudou = 0; //mudou estoque

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

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

if(opt == 1){
    bufs[0] = '\0';

```

```

    strcat(bufs, "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';
    strcat(bufs, "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';
    strcat(bufs, "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';
    strcat(bufs, "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\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 (send(new_fd, bufs, strlen(bufs)+1, 0) == -1)
    perror("send");

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(new_fd);
    exit(0);
}
close(new_fd); // parent doesn't need this
}
return 0;
}

```

Anexo 2: Código do Cliente

```

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

```

```

#define PORT "3490" // the port client will be connecting to

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

// 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(int argc, char *argv[])
{
    int sockfd, numbytes;
    char buf[MAXDATASIZE];
    struct addrinfo hints, *servinfo, *p;
    int rv;
    char s[INET6_ADDRSTRLEN];
    int estoque;

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

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

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

    // loop through all the results and connect 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("client: socket");
            continue;
        }

```

```

    if (connect(sockfd, p->ai_addr, p->ai_addrlen) == -1) {
        close(sockfd);
        perror("client: connect");
        continue;
    }

    break;
}

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

inet_ntop(p->ai_family, get_in_addr((struct sockaddr *)p->ai_addr),
          s, sizeof s);
printf("client: connecting to %s\n", s);

freeaddrinfo(servinfo); // all done with this structure

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((bytes_sent = send(sockfd, msg, len, 0)) != len){
        printf("erro no send\n");
        continue;
    }

    //le resultado
    while(1){
        if ((bytes_rcv = recv(sockfd, buf, MAXDATASIZE-1, 0)) == -1) {
            perror("erro no recv");
            break;
        }
        sscanf(buf, "%d", &cont);
        printf("%s", buf+2);
        if(cont == 0) 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);
}

```

```
    close(sockfd);  
    return 0;  
}
```