Projeto 1 - Servidor concorrente TCP

MC833 Programação em Redes de Computadores | Projeto 1
1o semestre de 2015

Nicholas Matuzita Mizoguchi - RA 103659
Fábio Yudi Murakami - RA 138311

Introdução

O intuito deste relatório é descrever o projeto, implementação e resultados do primeiro projeto da disciplina MC833 do primeiro semestre de 2015. O projeto consiste na implementação de um sistema para consulta e controle de uma locadora de filmes, no qual é possível consultar informações sobre os filmes disponíveis na loja, assim como controlar os exemplares a serem alugados pelos clientes. Nesse primeiro projeto será utilizada uma arquitetura cliente-servidor utilizando o protocolo TCP para a implementação de um servidor concorrente.

O servidor armazena as informações de cada filme seguindo a seguinte estrutura de dados:

- Identificador único;
- Título;
- Data de lançamento;
- Gênero;
- Sinopse;
- Número de exemplares disponível;

O cliente por sua vez pode consultar essas informações seguindo um protocolo de mensagens definido, podendo obter todas as informações relevantes, e há ainda um acesso restrito apenas para o administrador para que o mesmo tenha um maior controle do número de exemplares.

No projeto posterior será implementado o mesmo sistema utilizando o protocolo UDP, permitindo a comparação entre os dois protocolos e implementações.

Implementação

Todo o código cliente-servidor foi desenvolvido na linguagem C, e está em anexo ao final deste relatório. O banco de dados do servidor foi alimentado utilizando informações obtidas através da API do website rottentomatoes.com. A API fornece as informações em formato JSON, e foi utilizado um script em python para obter e tratar os dados iniciais para o servidor. O servidor mantêm uma cópia do banco de dados em um arquivo, mas mantém as informações em memória durante sua execução para diminuir o acesso à disco, melhorando a performance das consultas.

Inicialmente, o servidor passa por uma fase de pré-processamento, na qual recupera seu estado a partir de um arquivo que foi salvo em seu último funcionamento. Os filmes são carregados em um *array* de *struct movie*.

Após essa fase o servidor passa a escutar conexões de clientes em um *socket TCP*. A implementação inicial do projeto funcionava com o fork do processo, permitindo que outro

processo cuidasse do cliente, e o processo principal continuasse escutando novos clientes. Isso se tornou um problema no compartilhamento de informações entre os clientes, uma vez que o fork apenas faz uma cópia do processo, e nenhuma informação é compartilhada. Por motivos de performance, foi descartada a idéia de escrever em arquivo e ler do mesmo a cada operação, e foi levada em consideração a implementação por *pthreads*. A cada nova conexão, a thread principal do servidor delega a tarefa de cuidar do cliente para uma nova thread, que por sua vez responde devidamente aos pedidos do cliente. Foi utilizado também um mutex para garantir que não haja concorrência na modificação do número de exemplares.

O servidor oferece uma interface de comunicação simples, sendo possível se comunicar com ele somente via telnet, se desejado. Foi feita a implementação de um cliente para possibilitar a coleta de informações e facilitar a análise dos tempos de execução.

Protocolo do servidor

O servidor responde às seguintes mensagens:

- **list <optional_projection>+**
  Retorna uma lista de filmes e todas as suas informações. Pode ser usada apenas como *list*, recebendo as informações de todos os filmes disponíveis, ou pode-se passar argumentos de projeção, podendo pegar informações como título, gênero, etc. (ex.: list title genre synopsis).
- **genre <genre_name>**
  Retorna a lista com todas as informações dos filmes com o gênero passado como parâmetro. (ex.: genre Comedy)
- **synopsis <id>**
  Retorna a sinopse do filme com o *id* passado como parâmetro. (ex.: synopsis 14)
- **get <id>**
  Retorna todas as informações do filme com o id passado como parâmetro.
- **admin <password>**
  Com o password correto, permite o acesso ao sistema como administrador. Em nosso projeto, a senha padrão está definida como "qwerty".
- **reserve <id> <num>**
  Reserva *num* exemplares do filme com id passado como parâmetro. Necessita que o usuário esteja logado como administrador. (ex.: reserve 14 2)
- **return <id> <num>**
  Retorna *num* exemplares do filme com id passado como parâmetro, pode ser usado para adicionar novos exemplares. Necessita que o usuário esteja logado como administrador.
- **copies <id>**
  Retorna o número de exemplares do filme com id passado como parâmetro.

Vantagens e Desvantagens

A decisão de manter as informações em memória é possível se considerarmos o baixo número de filmes disponíveis em uma locadora. Como citado anteriormente, essa decisão acarreta em um ganho em performance considerável. Ainda há a possibilidade de melhorar o projeto acrescentando algum tipo de gerenciador de banco de dados, como por exemplo o sqlite, que também foi considerada a utilização, mas por fugir um pouco do escopo do projeto, foi deixado de lado como uma possibilidade de melhora do sistema, tornando o servidor mais

robusto e eficiente. A decisão de utilização da biblioteca *pthread* com a implementação correta para garantir que as operações são *thread-safe* foi crucial para permitir que o servidor aceitasse conexões simultâneas e funcionasse de forma concorrente.

O protocolo TCP garante que não haverá perda de pacotes, assim como a ordem de chegada dos pacotes, o que facilitou na implementação. Em contrapartida, há um overhead no tempo de transmissão, o que será observado possivelmente na comparação com o servidor UDP.

Resultados

Todos os resultados estão anexos à esse relatório. Todos os testes foram feitos para 30 repetições das operações:

1. listar todos os títulos dos filmes e o ano de lançamento;
2. listar todos os títulos dos filmes e o ano de lançamento de um gênero determinado;
3. dado o identificador de um filme, retornar a sinopse do filme;
4. dado o identificador de um filme, retornar todas as informações deste filme;
5. listar todas as informações de todos os filmes;
6. alterar o número de exemplares em estoque (apenas cliente locadora);
7. dado o identificador de um filme, retornar o número de exemplares em estoque.

O tempo total de execução é o tempo imediatamente após a conexão ter sido estabelecida e logo antes da mensagem ser enviada ao servidor, até o momento em que a informação chega completamente ao cliente. O tempo de consulta é o tempo entre o recebimento do pedido do cliente e o envio da resposta ao cliente, e finalmente, o tempo de comunicação é a metade da diferença do tempo de execução e o tempo de consulta.

A simulação foi feita em dois computadores em redes diferentes (rede de longa distância: doméstica e rede da UNICAMP).

Conclusão

Os sockets TCP são uma ótima ferramenta para comunicação entre aplicações. Todas as garantias oferecidas pelo protocolo trazem a alta confiabilidade na entrega, e consequentemente na execução das aplicações, além de tornar o código mais enxuto. Em um projeto pequeno, a preocupação com a concorrência entre processos e threads não é tão grande, pois não há tantas restrições de exclusão mútua, mas em grandes projetos é necessário levar em conta algum tipo de gerenciador de banco de dados, para garantir consistência dos dados.

Pode-se observar também que algumas medidas tiveram tempos discrepantes, relacionados ao atraso de conexão em redes diferentes, e que os mesmos são outliers em nossos dados. Em geral, os tempos de comunicação foram de certa forma constantes, tendo apenas uma variação dependendo da operação sendo realizada no servidor, modificando o tempo total e tempo de execução de acordo com a complexidade da operação.

Bibliografia

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

```c
#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 <stdarg.h>
#include <pthread.h>
#include <sys/time.h>

#define MAXDATASIZE 1000 // max number of bytes we can get at once
#define PORT "3490"  // the port users will be connecting to
#define BACKLOG 10     // how many pending connections queue will hold


typedef struct movie {
    int id;
    char *title;
    char *launchDate;
    char *genre;
    char *synopsis;
    int copies;
} Movie;


void *handle_client(void *sock);
void send_response(int sockfd, char *response);
char *concat(int count, ...);

char *movie_to_string(Movie movie);
char *movie_projection(Movie movie, char **projection, int num_args);

Movie *movie;
int movie_count;
int is_root = 0;
pthread_mutex_t lock;

void initialize_data() {

    movie = (Movie*) malloc(sizeof(Movie)*100);
    movie_count = 0;
    char buffer[10000];

    FILE *F = fopen("data.xls", "r");
    while(fgets(buffer, sizeof buffer, F)) {
        char *data = strtok(buffer, ";");
```

```
        movie[movie_count].id = strtol(data, NULL, 10);
        movie[movie_count].title = strdup(strtok(NULL, ";"));
        movie[movie_count].launchDate = strdup(strtok(NULL, ";"));
        movie[movie_count].genre = strdup(strtok(NULL, ";"));
        movie[movie_count].synopsis = strdup(strtok(NULL, ";"));
        movie[movie_count].copies = strtol(strtok(NULL, ";"), NULL, 10);

        movie_count++;
    }
}

void save_data() {
    FILE *F = fopen("data.xls", "w");
    for(int i = 0; i < movie_count; i++) {
        fprintf(F, "%d;%s;%s;%s;%s;%d;\n",
            movie[i].id,
            movie[i].title,
            movie[i].launchDate,
            movie[i].genre,
            movie[i].synopsis,
            movie[i].copies);
    }

    fclose(F);
}

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;

    // Initialize mutex
    if (pthread_mutex_init(&lock, NULL) != 0)
```

```
{
    printf("\n mutex init failed\n");
    return 1;
}

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);
    }
    // Server ready
    // Initialize data
    initialize_data();

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

    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);

        pthread_t thread;
        int rc = pthread_create(&thread, NULL,
                          handle_client, (void *)&new_fd);
    }

    pthread_mutex_destroy(&lock);

    return 0;
}

int handle_command(int sockfd, char *command) {
    char *token, *saveptr_tok, *info;
    char *args[10];

    struct timeval time_in, time_out;
    gettimeofday(&time_in,NULL);

    info = "";

    token = strtok_r(command, " ", &saveptr_tok);
    if(token != NULL) {
        // Exit
        if(strcmp(token, "exit") == 0) {
            return 0;

        /* Get all information of a specific movie
            _id:  id of the desired movie */
        } else if(strcmp(token, "admin") == 0) {
            token = strtok_r(NULL, " ", &saveptr_tok);
            if(token == NULL) return -1;

            if(strcmp(token, "qwerty") == 0) {
```

```
            is_root = 1;
            info = "Logged in as admin.\n\n";
        } else {
            info = "Wrong password.\n\n";
        }

    /* Get all information of a specific movie
        _id:  id of the desired movie */
    } else if(strcmp(token, "get") == 0) {
        token = strtok_r(NULL, " ", &saveptr_tok);
        if(token == NULL) return -1;
        int id = strtol(token, NULL, 10);

        for(int i = 0; i < movie_count; i++) {
            if(movie[i].id == id) {
                info = movie_to_string(movie[i]);
                break;
            }
        }

    /* Get the synopsis of a specific movie
        _id:  id of the desired movie's synopsis */
    } else if(strcmp(token, "synopsis") == 0) {
        token = strtok_r(NULL, " ", &saveptr_tok);
        int id = strtol(token, NULL, 10);
        if(token == NULL) return -1;

        for(int i = 0; i < movie_count; i++) {
            if(movie[i].id == id) {
                info = movie[i].synopsis;
                break;
            }
        }

    /* Reserve a number of copies of a specific movie.
        Gives an error when not possible to reserve (lack of copies)
        _id:            id of the desired movie to be reserved
        _num_copies:    number of copies to be reserved */
    } else if(strcmp(token, "reserve") == 0) {
        if(is_root) {
            pthread_mutex_lock(&lock);
            token = strtok_r(NULL, " ", &saveptr_tok);
            if(token == NULL) return -1;

            int id = strtol(token, NULL, 10);

            token = strtok_r(NULL, " ", &saveptr_tok);
            if(token == NULL) return -1;

            int num = strtol(token, NULL, 10);

            for(int i = 0; i < movie_count; i++) {
                if(movie[i].id == id) {
                    if(movie[i].copies >= num) {
```

```
                            movie[i].copies -= num;
                            info = concat(4, info, "Reserved ", token, " copies
successfully.\n");

                        } else {
                            info = "Operation cancelled. Not enough copies.\n";
                        }
                        break;
                    }
                }
                pthread_mutex_unlock(&lock);
            } else return -2;

        /* return a number of copies of a specific movie. Can be
        used to add new copies of a movie to the store.
            _id:            id of the desired movie to be reserved
            _num_copies:    number of copies to be freed */
        } else if(strcmp(token, "return") == 0) {
            if(is_root) {
                pthread_mutex_lock(&lock);
                token = strtok_r(NULL, " ", &saveptr_tok);
                if(token == NULL) return -1;
                int id = strtol(token, NULL, 10);
                token = strtok_r(NULL, " ", &saveptr_tok);
                if(token == NULL) return -1;
                int num = strtol(token, NULL, 10);

                for(int i = 0; i < movie_count; i++) {
                    if(movie[i].id == id) {
                        movie[i].copies += num;
                        info = concat(4, info, "Returned ", token, " copies successfully.\n");
                        break;
                    }
                }
                pthread_mutex_unlock(&lock);
            } else return -2;

        /* Get the number of copies of a specific movie.
            _id:  id of the desired movie's number of copies */
        } else if(strcmp(token, "copies") == 0) {
            token = strtok_r(NULL, " ", &saveptr_tok);
            int id = strtol(token, NULL, 10);

            for(int i = 0; i < movie_count; i++) {
                if(movie[i].id == id) {
                    char buffer[20];
                    sprintf(buffer, "%d", movie[i].copies);
                    info = concat(3, "There are ", buffer, " copies available.\n");
                    break;
                }
            }

        /* Get specific information of all movies.
            _id:    id of the desired movie's synopsis
```

```
            args*:  the projection to be returned */
        } else if(strcmp(token, "genre") == 0) {

            int count = 0;
            token = strtok_r(NULL, " ", &saveptr_tok);
            if(token == NULL) return -1;

            for(int i = 0; i < movie_count; i++) {
                if(strcmp(movie[i].genre,token) == 0) {
                    info = concat(2, info, movie_to_string(movie[i]));
                }
            }

        /* Get specific information of all movies.
            _id:    id of the desired movie's synopsis
            args*:  the projection to be returned */
        } else if(strcmp(token, "list") == 0) {

            int count = 0;
            token = strtok_r(NULL, " ", &saveptr_tok);

            if(token == NULL) {
                for(int i = 0; i < movie_count; i++)
                    info = concat(2, info, movie_to_string(movie[i]));
            } else {
                while(token != NULL) {
                    args[count] = strdup(token);
                    printf("%s ", args[count]);
                    count++;
                    token = strtok_r(NULL, " ", &saveptr_tok);
                }

                for(int i = 0; i < movie_count; i++) {
                    info = concat(2, info, movie_projection(movie[i], args, count));
                }
            }
        }  else {
            return -404;
        }
    }

    gettimeofday(&time_out,NULL);

    double time1, time2;

    // usec are microseconds!
    time1 = time_in.tv_sec + 0.000001*time_in.tv_usec;
    time2 = time_out.tv_sec + 0.000001*time_out.tv_usec;
    printf("execution time: %lf\n", time2-time1);
    send_response(sockfd,info);
    return 1;
}

void *handle_client(void *sock) {
```

```
    int sockfd = *((int *)sock);
    char buf[MAXDATASIZE];
    char *command, *token, *saveptr_cmd, *saveptr_tok;
    int numbytes, response;

    while(1) {
        if ((numbytes = recv(sockfd, buf, MAXDATASIZE-1, 0)) == -1) {
            perror("recv");
            pthread_exit(0);

        } else {
            buf[numbytes] = '\0';
            // Parse commands separated by linefeeds
            command = strtok_r(buf, "\n",&saveptr_cmd);
            while(command != NULL) {
                if(command[strlen(command)-1] == '\r')
                    command[strlen(command)-1] = '\0';
                printf("server: received command '%s'\n",command);
                response = handle_command(sockfd, command);
                if(response == 0) {
                    close(sockfd);
                    pthread_exit(0);

                } else if(response == -1) {
                    send_response(sockfd, "*** missing arguments\n\n");
                } else if(response == -2) {
                    send_response(sockfd, "*** Access denied. Login as admin to perform this
operation.\n\n");
                } else if(response == -404) {
                    // invalid command
                    send_response(sockfd, concat(3, "*** invalid command: ", command, "\n\n"));
                }
                command = strtok_r(NULL, "\n", &saveptr_cmd);
            }
        } // else case that there are things yet to be sent
    }
    pthread_exit(0);
}

char *movie_to_string(Movie movie) {
    char *movie_info;
    char buffer[20];

    sprintf(buffer, "Movie id: %d", movie.id);
    movie_info = strdup(buffer);
    movie_info = concat(3, movie_info, "\n\tTitle: ", movie.title);
    movie_info = concat(3, movie_info, "\n\tLaunched at: ", movie.launchDate);
    movie_info = concat(3, movie_info, "\n\tGenre: ", movie.genre);
    movie_info = concat(3, movie_info, "\n\tSynopsis: ", movie.synopsis);
    sprintf(buffer, "%d", movie.copies);
    movie_info = concat(3, movie_info, "\n\tAvailable copies: ", buffer);

    movie_info = concat(2, movie_info, "\n\n");
    return movie_info;
```

```
}

char *movie_projection(Movie movie, char **projection, int num_args) {
    char *movie_info;
    char buffer[20];

    sprintf(buffer, "Movie id: %d", movie.id);
    movie_info = strdup(buffer);

    for(int i = 0; i < num_args; i++) {
        if(strcmp(projection[i], "title") == 0) movie_info = concat(3, movie_info, "\n\tTitle:
", movie.title);
        if(strcmp(projection[i], "launchdate") == 0) movie_info = concat(3, movie_info,
"\n\tLaunched at: ", movie.launchDate);
        if(strcmp(projection[i], "genre") == 0) movie_info = concat(3, movie_info, "\n\tGenre:
", movie.genre);
        if(strcmp(projection[i], "synopsis") == 0) movie_info = concat(3, movie_info,
"\n\tSynopsis: ", movie.synopsis);
        if(strcmp(projection[i], "copies") == 0) {
            sprintf(buffer, "%d", movie.copies);
            movie_info = concat(3, movie_info, "\n\tAvailable copies: : ", buffer);
        }
    }
    movie_info = concat(2, movie_info, "\n\n");
    return movie_info;
}

void send_response(int sockfd, char *response) {
    if (send(sockfd, response, strlen(response), 0) == -1)
        perror("send");
}

char* concat(int count, ...)
{
    va_list ap;
    int len = 1, i;

    va_start(ap, count);
    for(i=0 ; i<count ; i++)
        len += strlen(va_arg(ap, char*));
    va_end(ap);

    char *result = (char*) calloc(sizeof(char),len);
    int pos = 0;

    // Actually concatenate strings
    va_start(ap, count);
    for(i=0 ; i<count ; i++)
    {
        char *s = va_arg(ap, char*);
        strcpy(result+pos, s);
        pos += strlen(s);
    }
    va_end(ap);
```

```
    return result;
}
```

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 100000 // 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];

    if (argc != 3) {
        fprintf(stderr,"usage: client hostname action\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

send(sockfd, "admin qwerty\n", 13, 0);

    struct timeval time_in, time_out;
    gettimeofday(&time_in,NULL);
    switch(atoi(argv[2])) {
        case 0:
            if (send(sockfd, "list title launchdate\n", 22, 0) == -1)
                perror("send");
            break;
        case 1:
            if (send(sockfd, "genre Comedy\n", 13, 0) == -1)
                perror("send");
            break;
        case 2:
            if (send(sockfd, "synopsis 10\n", 12, 0) == -1)
                perror("send");
            break;
        case 3:
            if (send(sockfd, "get 11\n", 7, 0) == -1)
                perror("send");
            break;
        case 4:
            if (send(sockfd, "list\n", 5, 0) == -1)
                perror("send");
            break;
        case 5:
```

```
                if (send(sockfd, "reserve 12 1\n", 13, 0) == -1)
                    perror("send");
                break;
            case 6:
                if (send(sockfd, "copies 13\n", 10, 0) == -1)
                    perror("send");
                break;
        }
        if ((numbytes = recv(sockfd, buf, MAXDATASIZE-1, 0)) == -1) {
            perror("recv");
            exit(1);
        }

        gettimeofday(&time_out,NULL);

        double time1, time2;

        // usec are microseconds!
        time1 = time_in.tv_sec + 0.000001*time_in.tv_usec;
        time2 = time_out.tv_sec + 0.000001*time_out.tv_usec;
        printf("%lf\n", time2-time1);

    send(sockfd, "exit\n", 5, 0);

    close(sockfd);

    return 0;
}
```

Tabelas de resultados: tempos medidos em segundos

	Operação 1			Operação 2			Operação 3		
n	T consulta	T total	T communic.	T consulta	T total	T communic.	T consulta	T total	T communic.
1	0.00025	0.015926	0.007838	0.000145	0.013727	0.006791	0.000003	0.015762	0.0078795
2	0.000195	0.015604	0.0077045	0.000131	0.017877	0.008873	0.000003	0.017766	0.0088815
3	0.00018	0.017673	0.0087465	0.000122	0.019569	0.0097235	0.000003	0.019901	0.009949
4	0.000196	0.015694	0.007749	0.000094	0.016031	0.0079685	0.000003	0.015794	0.0078955
5	0.000207	0.014234	0.0070135	0.000206	0.015878	0.007836	0.000003	0.01598	0.0079885
6	0.000212	0.01654	0.008164	0.000144	0.015945	0.0079005	0.000003	0.013943	0.00697
7	0.000235	0.016313	0.008039	0.000351	0.019772	0.0097105	0.000003	0.014777	0.007387
8	0.000231	0.018264	0.0090165	0.000132	0.017885	0.0088765	0.000003	0.018034	0.0090155
9	0.000265	0.016022	0.0078785	0.000101	0.017968	0.0089335	0.000003	0.021399	0.010698
10	0.000227	0.014593	0.007183	0.0002	0.015838	0.007819	0.000003	0.01809	0.0090435
11	0.000161	0.017886	0.0088625	0.000123	0.015566	0.0077215	0.000003	0.019572	0.0097845
12	0.000197	0.019112	0.0094575	0.000133	0.012644	0.0062555	0.000003	0.017668	0.0088325
13	0.000195	0.014975	0.00739	0.00012	0.01583	0.007855	0.000002	0.017624	0.008811
14	0.000227	0.014229	0.007001	0.00012	0.018635	0.0092575	0.000004	0.016194	0.008095
15	0.000201	0.015745	0.007772	0.000114	0.014384	0.007135	0.000004	0.017769	0.0088825
16	0.000178	0.013784	0.006803	0.000211	0.012693	0.006241	0.000001	0.013689	0.006844
17	0.000169	0.014297	0.007064	0.000119	0.01793	0.0089055	0.000003	0.015609	0.007803
18	0.000183	0.016426	0.0081215	0.000189	0.018733	0.009272	0.000003	0.016943	0.00847
19	0.000156	0.018587	0.0092155	0.000121	0.018224	0.0090515	0.000003	0.015829	0.007913
20	0.000203	0.013783	0.00679	0.000122	0.014093	0.0069855	0.000003	0.015792	0.0078945
21	0.000241	0.014201	0.00698	0.000123	0.016429	0.008153	0.000004	0.014893	0.0074445
22	0.000216	0.014482	0.007133	0.000147	0.017289	0.008571	0.000002	0.017542	0.00877
23	0.00022	0.020583	0.0101815	0.00012	0.017723	0.0088015	0.000002	0.013981	0.0069895
24	0.000239	0.014031	0.006896	0.000115	0.018681	0.009283	0.000003	0.015643	0.00782
25	0.000208	0.016585	0.0081885	0.000136	0.015731	0.0077975	0.000003	0.021703	0.01085
26	0.000277	0.014726	0.0072245	0.000173	0.015977	0.007902	0.000005	0.015769	0.007882
27	0.000194	0.014697	0.0072515	0.000114	0.016577	0.0082315	0.000003	0.015777	0.007887
28	0.000197	0.014232	0.0070175	0.000131	0.018029	0.008949	0.000006	0.017821	0.0089075
29	0.000167	0.014392	0.0071125	0.000116	0.014427	0.0071555	0.000003	0.015526	0.0077615
30	0.000204	0.018171	0.0089835	0.000127	0.017418	0.0086455	0.000002	0.021514	0.010756
SUM	0.006231	0.475787	0.234778	0.0043	0.497503	0.2466015	0.000092	0.508304	0.254106
DESVIO	2.96301E-05	0.00180689	0.00090563	4.9355E-05	0.0019089	0.0009521	9.0719E-07	0.00217584	0.0010879

	Operação 4			Operação 5			Operação 6			Operação 7		
n	T consulta	T total	T communic.	T consulta	T total	T communic.	T consulta	T total	T communic.	T consulta	T total	T communic.
1	0.000037	0.018069	0.009016	0.000991	0.015778	0.0073935	0.000023	0.020166	0.0100715	0.000034	0.019308	0.009637
2	0.000015	0.014842	0.0074135	0.000852	0.018684	0.008916	0.000008	0.017787	0.0088895	0.000011	0.015874	0.0079315
3	0.000038	0.017975	0.0089685	0.001246	0.013932	0.006343	0.000009	0.013733	0.006862	0.000016	0.014143	0.0070635
4	0.000015	0.01779	0.0088875	0.001332	0.017489	0.0080785	0.000007	0.015357	0.007675	0.000014	0.019514	0.00975
5	0.000018	0.018921	0.0094515	0.001165	0.019817	0.009326	0.000022	0.0196	0.009789	0.000017	0.013748	0.0068655
6	0.000031	0.018034	0.0090015	0.000778	0.017789	0.0085055	0.000003	0.019681	0.009839	0.000012	0.015575	0.0077815
7	0.000019	0.019219	0.0096	0.00074	0.019812	0.009536	0.000003	0.019971	0.009984	0.000014	0.017884	0.008935
8	0.000027	0.017382	0.0086775	0.000814	0.015474	0.00733	0.000004	0.013674	0.006835	0.000022	0.019924	0.009951
9	0.000029	0.01398	0.0069755	0.000995	0.014582	0.0067935	0.000004	0.017897	0.0089465	0.000013	0.017684	0.0088355
10	0.000021	0.013947	0.006963	0.000811	0.018633	0.008911	0.000003	0.021514	0.0107555	0.000016	0.020365	0.0101745
11	0.000019	0.016029	0.008005	0.001181	0.016824	0.0078215	0.000004	0.019733	0.0098645	0.00002	0.014243	0.0071115
12	0.000032	0.017983	0.0089755	0.000711	0.017816	0.0085525	0.000004	0.013931	0.0069635	0.000015	0.014978	0.0074815
13	0.000016	0.015986	0.007985	0.000985	0.01857	0.0087925	0.000005	0.017583	0.008789	0.000015	0.025825	0.012905
14	0.000027	0.015544	0.0077585	0.000903	0.019762	0.0094295	0.000005	0.023861	0.011928	0.000023	0.014128	0.0070525
15	0.000031	0.015774	0.0078715	0.000972	0.013984	0.006506	0.00001	0.016328	0.008159	0.00004	0.013941	0.0069505
16	0.000077	0.019983	0.009953	0.000801	0.016643	0.007921	0.000004	0.015654	0.007825	0.00002	0.017775	0.0088775
17	0.000036	0.015762	0.007863	0.000752	0.018722	0.008985	0.000005	0.01503	0.0075125	0.000026	0.017341	0.0086575
18	0.000019	0.015835	0.007908	0.000839	0.017784	0.0084725	0.000006	0.021915	0.0109545	0.000016	0.021369	0.0106765
19	0.000018	0.015785	0.0078835	0.000766	0.017825	0.0085295	0.000005	0.01759	0.0087925	0.000011	0.01804	0.0090145
20	0.000027	0.013992	0.0069825	0.001097	0.017934	0.0084185	0.000004	0.015932	0.007964	0.000015	0.017874	0.0089295
21	0.00002	0.017823	0.0089015	0.001048	0.015725	0.0073385	0.000003	0.015791	0.007894	0.000013	0.013797	0.006892
22	0.000019	0.01787	0.0089255	0.000686	0.021171	0.0102425	0.000046	0.01997	0.009962	0.000013	0.014035	0.007011
23	0.000027	0.017436	0.0087045	0.001169	0.018574	0.0087025	0.000003	0.019127	0.009562	0.000014	0.019936	0.009961
24	0.000039	0.015279	0.00762	0.000906	0.017839	0.0084665	0.000003	0.015529	0.007763	0.00003	0.013936	0.006953
25	0.000031	0.017564	0.0087665	0.001438	0.015228	0.006895	0.000003	0.01604	0.0080185	0.00001	0.014097	0.0070435
26	0.00003	0.016879	0.0084245	0.001021	0.016694	0.0078365	0.000004	0.020768	0.010382	0.00003	0.017872	0.008921
27	0.000019	0.019814	0.0098975	0.000807	0.017177	0.008185	0.000003	0.01594	0.0079685	0.000013	0.013579	0.006783
28	0.000014	0.017822	0.008904	0.001193	0.019831	0.009319	0.000003	0.014035	0.007016	0.000014	0.017523	0.0087545
29	0.000015	0.013962	0.0069735	0.000847	0.015972	0.0075625	0.000003	0.01579	0.0078935	0.000012	0.015931	0.0079595
30	0.000016	0.013524	0.006754	0.000904	0.01983	0.009463	0.000004	0.015782	0.007889	0.000021	0.017855	0.008917
SUM	0.000782	0.500805	0.2500115	0.02875	0.525895	0.2485725	0.000213	0.525709	0.262748	0.00054	0.508094	0.253777
DESVIO	1.23398E-05	0.0018244	0.00091035	0.0001957	0.00186217	0.00096568	8.837E-06	0.00270282	0.00135031	7.3999E-06	0.00289832	0.00144936