

MC833 - 2s2017

Bruno Orsi Berton
Fábio Takahashi Tanniguchi

RA 150573 - Turma B
RA 145980 - Turma A

Makefile

```
CC      = gcc
FLAGS   = -Wall
APIS     = my_socket_api.o

all: $(APIS) servidor cliente

servidor: servidor.c
    $(CC) $(FLAGS) -o servidor $(APIS) servidor.c

cliente: cliente.c
    $(CC) $(FLAGS) -o cliente $(APIS) cliente.c

my_socket_api.o: my_socket_api.c
    $(CC) $(FLAGS) -c my_socket_api.c

clean:
    rm $(APIS) servidor cliente log_server.txt
```

my_socket_api.h

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

#define MAXLINE 4096
#define LISTENQ 5

/*
    Define um tipo booleano
*/
typedef enum {
    FALSE = 0,
    TRUE = 1
} bool;

/*
```

```

    Funções que abstraem a interface de sockets
*/
int Socket(int family, int type, int flags);
void Connect(int socket, const struct sockaddr *sockaddr, socklen_t sockaddr_len);
void Bind(int socket, const struct sockaddr *sockaddr, socklen_t sockaddr_len);
void Listen(int socket, int queue_size);
int Accept(int socket, struct sockaddr *sockaddr, socklen_t *sockaddr_len);

/*
    Funções auxiliares
*/
bool isExit(const char *message);
pid_t Fork();
void PrintClientData(const struct sockaddr_in *sockaddr, char *clientName, int
clientName_len);
void FPrintClientData(const struct sockaddr_in *sockaddr, char *clientName, int
clientName_len, FILE *f);
void PrintClientDataClose(const struct sockaddr_in *sockaddr, char *clientName, int
clientName_len);
void FPrintClientDataClose(const struct sockaddr_in *sockaddr, char *clientName, int
clientName_len, FILE *f);
void PrintData(int socket, const struct sockaddr_in *sockaddr, char *localHost, char
*localPort);

```

my_socket_api.c

```

#include "my_socket_api.h"

/*
    Função para criação de sockets
*/
int Socket(int family, int type, int flags) {
    int sockfd;
    if ((sockfd = socket(family, type, flags)) < 0) {
        perror("Error creating socket!");
        exit(1);
    }

    return(sockfd);
}

/*
    Função para abrir uma conexão
*/
void Connect(int socket, const struct sockaddr *sockaddr, socklen_t sockaddr_len) {
    if (connect(socket, sockaddr, sockaddr_len) < 0) {
        perror("Connect error");
        exit(1);
    }
}

/*

```

```

    Função para fazer o bind do socket
*/
void Bind(int socket, const struct sockaddr *sockaddr, socklen_t sockaddr_len) {
    if (bind(socket, sockaddr, sockaddr_len) == -1) {
        perror("Bind error");
        exit(1);
    }
}

/*
    Função para deixar o socket ouvindo conexões com um certo buffer
*/
void Listen(int socket, int queue_size) {
    if (listen(socket, queue_size) == -1) {
        perror("Listen error");
        exit(1);
    }
}

/*
    Função para aceitar conexões em um socket
*/
int Accept(int socket, struct sockaddr *sockaddr, socklen_t *sockaddr_len) {
    int sockfd;
    if ((sockfd = accept(socket, sockaddr, sockaddr_len)) == -1) {
        perror("Accept error");
        exit(1);
    }

    return(sockfd);
}

/*
    Função auxiliar para sair da conexão
*/
bool isExit(const char *message) {
    if (strncmp(message, "exit\n", strlen(message)) == 0) {
        return TRUE;
    }

    return FALSE;
}

/*
    Função auxiliar que abre outro processo
*/
pid_t Fork() {
    pid_t pid;
    if ((pid = fork()) < 0) {
        perror("Fork error");
        exit(1);
    }

    return pid;
}

```

```

}

/*
Função auxiliar que imprime os dados do socket cliente
*/
void PrintClientData(const struct sockaddr_in *sockaddr, char *clientName, int
clientName_len) {
    if (inet_ntop(AF_INET, &sockaddr->sin_addr.s_addr, clientName, clientName_len) != NULL) {
        time_t timer;
        char buffer[26];
        struct tm* tm_info;

        time(&timer);
        tm_info = localtime(&timer);

        strftime(buffer, 26, "%Y-%m-%d %H:%M:%S", tm_info);

        printf("%s - Connection opened\n", buffer);
        printf("Endereco IP do cliente: %s\n", clientName);
        printf("Porta do cliente: %d\n", ntohs(sockaddr->sin_port));
    } else {
        printf("Erro ao imprimir dados do cliente!\n");
    }
}

/*
Função auxiliar que imprime os dados do socket cliente em arquivo
*/
void FPrintClientData(const struct sockaddr_in *sockaddr, char *clientName, int
clientName_len, FILE *f) {
    if (inet_ntop(AF_INET, &sockaddr->sin_addr.s_addr, clientName, clientName_len) != NULL) {
        time_t timer;
        char buffer[26];
        struct tm* tm_info;

        time(&timer);
        tm_info = localtime(&timer);

        strftime(buffer, 26, "%Y-%m-%d %H:%M:%S", tm_info);

        fprintf(f, "%s - Connection opened\n", buffer);
        fprintf(f, "Endereco IP do cliente: %s\n", clientName);
        fprintf(f, "Porta do cliente: %d\n", ntohs(sockaddr->sin_port));
    } else {
        fprintf(f, "Erro ao imprimir dados do cliente!\n");
    }
}

/*
Função auxiliar que imprime os dados do socket cliente ao fechar socket
*/
void PrintClientDataClose(const struct sockaddr_in *sockaddr, char *clientName, int
clientName_len) {
    if (inet_ntop(AF_INET, &sockaddr->sin_addr.s_addr, clientName, clientName_len) != NULL) {

```

```

    time_t timer;
    char buffer[26];
    struct tm* tm_info;

    time(&timer);
    tm_info = localtime(&timer);

    strftime(buffer, 26, "%Y-%m-%d %H:%M:%S", tm_info);

    printf("%s - closing connection with %s/%d\n", buffer, clientName,
ntohs(sockaddr->sin_port));
    } else {
        printf("Erro ao imprimir dados do cliente!\n");
    }
}

/*
Função auxiliar que imprime os dados do socket cliente em arquivo ao fechar socket
*/
void FPrintClientDataClose(const struct sockaddr_in *sockaddr, char *clientName, int
clientName_len, FILE *f) {
    if (inet_ntop(AF_INET, &sockaddr->sin_addr.s_addr, clientName, clientName_len) != NULL) {
        time_t timer;
        char buffer[26];
        struct tm* tm_info;

        time(&timer);
        tm_info = localtime(&timer);

        strftime(buffer, 26, "%Y-%m-%d %H:%M:%S", tm_info);

        fprintf(f, "%s - closing connection with %s/%d\n", buffer, clientName,
ntohs(sockaddr->sin_port));
    } else {
        fprintf(f, "Erro ao imprimir dados do cliente!\n");
    }
}

/*
Função auxiliar que imprime os dados do socket do lado do cliente
*/
void PrintData(int socket, const struct sockaddr_in *sockaddr, char *localHost, char
*localPort) {
    unsigned int sockaddr_len = sizeof(struct sockaddr);
    if (getsockname(socket, (struct sockaddr *) sockaddr, &sockaddr_len) == -1) {
        perror("getsockname() failed");
        exit(1);
    }
    printf("Endereco IP remoto do socket: %s\n", inet_ntoa(sockaddr->sin_addr));
    printf("Porta remota do socket: %d\n", (int) ntohs(sockaddr->sin_port));
    printf("Endereco IP local do socket: %s\n", localHost);
    printf("Porta local do socket: %s\n", localPort);
}

```

servidor.c

```
#include "my_socket_api.h"

int main (int argc, char **argv) {
    int listenfd, connfd, n;
    unsigned int clientaddr_len;
    struct sockaddr_in servaddr, clientaddr;
    char buf[MAXLINE], systembuf[MAXLINE], output[MAXLINE];
    char clientName[INET_ADDRSTRLEN];
    pid_t pid;
    bool isExiting = FALSE;

    // verifica se a porta foi passado por parametro
    if (argc != 2) {
        perror("Porta nao informada!");
        exit(1);
    }

    // cria um socket TCP
    listenfd = Socket(AF_INET, SOCK_STREAM, 0);

    // configura os parâmetros da conexão
    bzero(&servaddr, sizeof(servaddr));
    servaddr.sin_family = AF_INET;
    servaddr.sin_addr.s_addr = INADDR_ANY;
    servaddr.sin_port = htons(atoi(argv[1]));

    // faz o bind do socket TCP com o host:porta escolhidos
    Bind(listenfd, (struct sockaddr *)&servaddr, sizeof(servaddr));

    // ativa a socket para começar a receber conexões
    Listen(listenfd, LISTENQ);

    // abre arquivo de log
    FILE *f = fopen("log_server.txt", "a+");
    if (f == NULL) {
        printf("Error opening file!\n");
        exit(1);
    }

    // espera por conexões de clientes indefinidamente
    for ( ; ; ) {

        // aceita as conexões
        clientaddr_len = sizeof(clientaddr);
        connfd = Accept(listenfd, (struct sockaddr *) &clientaddr, &clientaddr_len);

        // cria um processo filho
        pid = Fork();

        // caso seja o processo filho
        if (pid == 0) {
```

```

// fecha a conexão de escuta para esse processo filho
close(listenfd);

// imprime no log os dados da conexão aberta
PrintClientData((struct sockaddr_in *) &clientaddr, clientName,
sizeof(clientName));
FPrintClientData((struct sockaddr_in *) &clientaddr, clientName,
sizeof(clientName), f);

// le dados do cliente indefinidamente
while ((n = read(connfd, buf, MAXLINE)) > 0) {

    printf("Executando comando (%s%c%d): %s", clientName, '/',
ntohs(clientaddr.sin_port), buf);
    fprintf(f, "Executando comando (%s%c%d): %s", clientName, '/',
ntohs(clientaddr.sin_port), buf);

    // Encerra a conexão
    if (isExit(buf)) {
        isExiting = TRUE;
        PrintClientDataClose((struct sockaddr_in *) &clientaddr, clientName,
sizeof(clientName));
        FPrintClientDataClose((struct sockaddr_in *) &clientaddr, clientName,
sizeof(clientName), f);
        break;
    }

    // abre um terminal com o comando lido do cliente
    FILE *fp = popen(buf, "r");

    // le o resultado do comando executado
    while (fgets(systembuf, sizeof(systembuf) - 1, fp) != NULL) {
        strcat(output, systembuf);
    }

    // coloca o resultado no arquivo de log
    fprintf(f, "Retorno:\n%s\n", output);

    // fecha pseudoarquivo "terminal"
    pclose(fp);

    // envia ao cliente o resultado do comando
    write(connfd, output, strlen(output));
    memset(buf, 0, sizeof(buf));
    memset(output, 0, sizeof(output));
}

close(connfd);
} else {
    // caso seja o processo pai
    close(connfd);
}

if (isExiting) {

```

```

        break;
    }

}

// fecha arquivo de log
fclose(f);

return(0);
}

```

cliente.c

```

#include "my_socket_api.h"

int main(int argc, char **argv) {
    int sockfd, n;
    char recvline[MAXLINE], input[MAXLINE];
    struct sockaddr_in servaddr;

    // verifica se o host e a porta foram passados
    if (argc != 3) {
        perror("Host/Porta nao informados!");
        exit(1);
    }

    // cria um socket TCP
    sockfd = Socket(AF_INET, SOCK_STREAM, 0);

    // configura os parâmetros da conexão
    bzero(&servaddr, sizeof(servaddr));
    servaddr.sin_family = AF_INET;
    servaddr.sin_addr.s_addr = inet_addr(argv[1]);
    servaddr.sin_port = htons(atoi(argv[2]));

    if (inet_pton(AF_INET, argv[1], &servaddr.sin_addr) <= 0) {
        perror("inet_pton error");
        exit(1);
    }

    // abre a conexão com o servidor
    Connect(sockfd, (struct sockaddr *) &servaddr, sizeof(servaddr));

    // imprime dados do socket
    PrintData(sockfd, (struct sockaddr_in *) &servaddr, argv[1], argv[2]);

    while (fgets(input, MAXLINE, stdin) != NULL) {
        write(sockfd, input, strlen(input));

        // termina execução do cliente caso o cliente queira
        if (isExit(input)) {
            break;
        }
    }
}

```



```
// le o que foi recebido através do socket e imprime o conteúdo
if ((n = read(sockfd, recvline, MAXLINE)) < 0) {
    perror("read error");
    exit(1);
}
recvline[n++] = 0;
printf("%s\n", recvline);
}

exit(0);
}
```