MC833 - 2s2017

Bruno Orsi Berton **RA 150573 - Turma B**
Fábio Takahashi Tanniguchi **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
```

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>
#include <sys/select.h>

#define MAXLINE 4096
#define LISTENQ 5

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

```c
/*
  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);
int Select(int nfds, fd_set *restrict readfds, fd_set *restrict writefds, fd_set *restrict
errorfds, struct timeval *restrict timeout);
```

my_socket_api.c

```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
*/
```

```c
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);
}

/*
  Função auxiliar para realizar o select e tratar o erro caso necessario
*/
int Select(int nfds, fd_set *restrict readfds, fd_set *restrict writefds, fd_set *restrict
errorfds, struct timeval *restrict timeout) {
  int n;

  if ((n = select(nfds, readfds, writefds, errorfds, timeout)) < 0) {
    perror("Select error");
    return 1;
  }

  return n;
}
```

servidor.c

```
#include "my_socket_api.h"

int main (int argc, char **argv) {
   int listenfd, connfd, activity, nread;
   unsigned int clientaddr_len;
   struct sockaddr_in servaddr, clientaddr;
   char buf[MAXLINE];
   fd_set selectfd;
        fd_set selectfd_aux;

   // verifica se o nome do arquivo foi passado por parametro
   if (argc != 2) {
      perror("Porta nao informado!");
      exit(0);
   }

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

   // cria o socket TCP
    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);

   FD_ZERO(&selectfd);
   FD_SET(listenfd, &selectfd);
   // espera por conexões de clientes indefinidamente
```

```
    for ( ; ; ) {
      selectfd_aux = selectfd;
      activity = Select(FD_SETSIZE, &selectfd_aux, NULL, NULL, NULL);

      // se houve uma conexao a flag é setada e a conexao eh aceita
      if (FD_ISSET(listenfd, &selectfd_aux)) {
        activity--;
        clientaddr_len = sizeof(clientaddr);
        connfd = Accept(listenfd, (struct sockaddr *) &clientaddr, &clientaddr_len);
        FD_SET(connfd, &selectfd);
      }

      // se a flag estiver setada
      if (FD_ISSET(connfd, &selectfd_aux)) {
        activity--;

        // le a linha e manda de volta para o cliente
        nread = read(connfd, buf, MAXLINE);
        write(connfd, buf, nread);

        // caso não tenha nada para ler, limpa
        if (nread == 0) {
          close(connfd);
          FD_CLR(connfd, &selectfd);
        }
      }
    }

    // fecha o socket de listen
    close(listenfd);

    return(0);
}
```

cliente.c

```
#include "my_socket_api.h"

int main(int argc, char **argv) {
    int    sockfd, n, data_sent = 0, data_received = 0;
    char   recvline[MAXLINE], input[MAXLINE];
    struct sockaddr_in servaddr;
    fd_set selectfd;

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

    FD_ZERO(&selectfd);

    // le entrada do cliente
    for ( ; ; ) {
      // faz o set no stdin e no socket
      FD_SET(fileno(stdin), &selectfd);
      FD_SET(sockfd, &selectfd);

      // faz o select em selectfd
      Select(FD_SETSIZE, &selectfd, NULL, NULL, NULL);

      // verifica se o stdin foi setado
      if (FD_ISSET(fileno(stdin), &selectfd)) {
        // le do servidor
        n = read(fileno(stdin), recvline, MAXLINE);

        // caso algo tenha sido enviado
        if (n > 0) {
          data_sent =+ n;
          write(sockfd, recvline, n);
        }
      }

      // verifica se o socket foi setado
      if (FD_ISSET(sockfd, &selectfd)) {
        n = read(sockfd, input, MAXLINE);
        data_received =+ n;
        write(fileno(stdout), input, n);
      }

      // caso todos os bytes forem enviados, encerra o cliente
      if (data_received == data_sent) {
        break;
      }
    }

    return(0);
}
```