MC833 - 2s2017

Bruno Orsi Berton **RA 150573 - Turma B**
Fábio Takahashi Tanniguchi **RA 145980 - Turma A**

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 typo 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 PrintData(int socket, const struct sockaddr_in *sockaddr, char *localHost, char
*localPort);
```

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) {
     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 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);
}
```

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));

     // 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", recvline);
   }

   exit(0);
```

```
}
```

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];
   char clientName[INET_ADDRSTRLEN];
   pid_t pid;

   // 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);

   // 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);

        PrintClientData((struct sockaddr_in *) &clientaddr, clientName,
sizeof(clientName));
```

```
        // 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);
          system(buf);

          // Encerra a conexao PARTE 2 DO TRABALHO
          //if (isExit(buf)) {
          //  break;
          //}

          // retorna o que foi enviado pelo cliente para o cliente
          write(connfd, buf, strlen(buf));
          memset(buf, 0, sizeof(buf));
        }

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

   }

   return(0);
}
```

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
```