```c
/* Cliente TCP */
#include "socket_utils.h"

void ClearStr(char* buffer) {
   int i;
   for(i = 0; i < MAXDATASIZE; i++) {
      buffer[i] = '\0';
   }
}

int main(int argc, char **argv) {
   // Declaracao de variaveis
   int sockfd, reading_input = TRUE, reading_socket = TRUE;
   char buf[MAXDATASIZE + 1], server[MAXDATASIZE];
   struct sockaddr_in servaddr;
   fd_set rset;

   // Checa a presenca do parametro de IP e Porta
   // caso ausente, fecha o programa
   if (argc != 3) {
      strcpy(buf, "uso: ");
      strcat(buf, argv[0]);
      strcat(buf, " <IPaddress> <Port>");
      perror(buf);
      exit(1);
   }

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

   // Limpa o que estiver no ponteiro do socket que representa o servidor
   // Seta o socket do servidor como IPv4 e seta a porta de conexao para a porta
 da aplicacao.
   bzero(&servaddr, sizeof(servaddr));
   servaddr.sin_family = AF_INET;
   servaddr.sin_port   = htons(atoi(argv[2]));

   // Converte o IP recebido na entrada para a forma binÃ¡ria da struct
   InetPton(AF_INET, argv[1], servaddr);

   // Conecta o socket local com o socket servidor
   Connect(sockfd, servaddr);

   while(reading_input || reading_socket) {
      // Resetar rset
      FD_ZERO(&rset);
      if (reading_input)
         FD_SET(STDIN_FILENO, &rset);
      if (reading_socket)
         FD_SET(sockfd, &rset);
      // Como o STDIN_FILENO = 0, podemos usar sempre sockfd como valor MAX
      Select(sockfd + 1, &rset, NULL, NULL, NULL);
      // se tem atividade no socket
      if (FD_ISSET(sockfd, &rset)) {
         ClearStr(server);
         // le os dados enviados pelo servidor
         if (Read(sockfd, server) == 0) {
            reading_socket = FALSE;
         } else {
            // Imprime o texto devolvida pelo servidor
            printf("%s", server);
         }
      }
      // se atividade na entrada padrao
      if (FD_ISSET(STDIN_FILENO, &rset)) {
         ClearStr(buf);
         // le uma cadeia de caracteres da entrada padrao
```

```
         if (fgets(buf, MAXDATASIZE, stdin) == NULL) {
            shutdown(sockfd, SHUT_WR);
            reading_input = FALSE;
         } else {
            // envia os dados lidos ao servidor
            Write(sockfd, buf);
         }
      }
   }
   return 0;
}
```

```
/* Servidor TCP */
#include "socket_utils.h"


// Limpa uma string
void ClearStr(char* buffer) {
   int i;
   for(i = 0; i < MAXDATASIZE; i++) {
      buffer[i] = '\0';
   }
}

int main (int argc, char **argv) {
   // Declaracao de variaveis
   int listenfd, connfd;
   pid_t pid;
   struct sockaddr_in servaddr;
   struct sockaddr_in clientaddr;
   char buf[MAXDATASIZE], client[MAXDATASIZE];

   // Checa a presenca do parametro Porta
   // caso ausente, fecha o programa
   if (argc != 2) {
      strcpy(buf, "uso: ");
      strcat(buf, argv[0]);
      strcat(buf, " <Port>");
      perror(buf);
      exit(1);
   }

   // Tenta criar um socket local TCP IPv4
   listenfd = Socket(AF_INET, SOCK_STREAM, 0);

   // Limpa o que estiver no ponteiro do socket que representa o servidor
   // Seta o socket do servidor como IPv4 e seta a porta de conexao passada por
parametro.
   // Seta uma mascara para aceitar conexoes de qualquer IP
   bzero(&servaddr, sizeof(servaddr));
   servaddr.sin_family      = AF_INET;
   servaddr.sin_addr.s_addr = htonl(INADDR_ANY);
   servaddr.sin_port        = htons(atoi(argv[1]));

   // Tentar fazer o bind do socket de servidor na porta escolhida
   Bind(listenfd, servaddr);

   // Setar socket como passivo (aceita conexoes)
   // Em caso de falha, fechar o programa
   Listen(listenfd, LISTENQ);

   // Loop infinito
   for ( ; ; ) {
      // Se chegou uma conexao
      // Em caso de falha fechar o programa
      connfd = Accept(listenfd, &clientaddr);

      // cria um processo filho (copia identica do pai)
      if( (pid = fork()) == 0) {
         // fecha a conexÃ£o com o processo pai
         Close(listenfd);
         // Converter informacao do IP de binario para string
         // armazenar o resultado no buffer
         InetNtop(AF_INET, buf, clientaddr);
         // Escrever IP, porta e string do cliente na saida padrao
         printf("OPEN -> Client - IP: %s - Port: %d\n", buf, htons(clientaddr.sin_port));
         int reading_client = TRUE;
         while(reading_client) {
            // limpa o buffer
```

```
            ClearStr(client);
            // Recebe o comando do cliente
            if (Read(connfd, client) == FALSE) {
               reading_client = FALSE;
            } else {
               printf("%s", client);
               // Envia a mensagem de volta para o cliente
               Write(connfd, client);
            }
         }
         // fecha a conexÃ£o do processo filho
         Close(connfd);
         // Escrever IP, porta e string do cliente que se desconectou
         printf("\nCLOSE-> Client - IP: %s - Port: %d\n", buf, htons(clientaddr.sin_port)
);
         // Limpa o que estiver no ponteiro do socket do client
         bzero(&clientaddr, sizeof(clientaddr));
         exit(0);
      }
      // Finalizar a conexao
      Close(connfd);
   }

   return 0;
}
```

```
#include "socket_utils.h"

// Aceita a conexao do cliente
// Em caso de falha fechar o programa
int Accept(int listenfd, struct sockaddr_in *clientaddr) {
  int connfd, clientsize = sizeof(clientaddr);
  if ((connfd = accept(listenfd, (struct sockaddr *)clientaddr, (socklen_t*)&cli
entsize)) == -1 ) {
    perror("accept");
    exit(1);
  }
  return connfd;
}

// Fazer um bind do socket com os parametros escolhidos
// Fechar o programa em caso de erro
void Bind(int listenfd, struct sockaddr_in servaddr) {
  if (bind(listenfd, (struct sockaddr *)&servaddr, sizeof(servaddr)) == -1) {
    perror("bind");
    exit(1);
  }
}

// Fecha a conexao
void Close(int connection) {
  close(connection);
}

// Tenta conectar um socket local a um outro socket, que pode ser remoto
// Fecha o programa em caso de erro
void Connect(int sockfd, struct sockaddr_in sockaddress) {
  if (connect(sockfd, (struct sockaddr *)&sockaddress, sizeof(sockaddress)) < 0)
 {
    perror("connect error");
    exit(1);
  }
}

// Coleta informacoes locais sobre um socket, retorna o socket com as informacoe
s preenchidas
// Fecha o programa em caso de erro
struct sockaddr_in Getsockname(int sockfd, struct sockaddr_in sockaddress) {
  socklen_t socksize = sizeof(sockaddress);
  bzero(&sockaddress, sizeof(sockaddress));
  if (getsockname(sockfd, (struct sockaddr *) &sockaddress, &socksize) < 0) {
    perror("getsockname error");
    exit(1);
  }
  return sockaddress;
}

// Converte o IP da forma binaria da struct sockaddr_in para uma string
// e armazena em buffer
// Fecha o programa em caso de erro
void InetNtop(int family, char* buffer, struct sockaddr_in sockaddress) {
  if (inet_ntop(family, &sockaddress.sin_addr, buffer, sizeof(char)*MAXDATASIZE)
 <= 0) {
    perror("inet_ntop error");
    exit(1);
  }
}

// Converte um IP string para a forma binaria da struct sockaddr_in
// Fecha o programa em caso de erro
void InetPton(int family, char *ipaddress, struct sockaddr_in sockaddress) {
  if (inet_pton(family, ipaddress, &sockaddress.sin_addr) <= 0) {
    perror("inet_pton error");
```

```
    exit(1);
  }
}

// Setar socket como passivo (aceita conexoes)
// Fechar o programa em caso de erro
void Listen(int listenfd, int listenq) {
  if (listen(listenfd, listenq) == -1) {
    perror("listen");
    exit(1);
  }
}

// Recebe dados do cliente e escreve em um buffer
// Se retornar algo > 0, ainda ha dados a serem escritos (ultrapassaram o tamanho do buffer)
int Read(int sockfd, char* buffer) {
  int read_size = recv(sockfd, buffer, MAXDATASIZE, 0);
  if (read_size < 0) {
    perror("read error");
    exit(1);
  }
  return read_size;
}

// Criar um socket com as opcoes especificadas
// Fecha o programa em caso de erro
int Socket(int family, int type, int flags) {
  int sockfd;
  if ( (sockfd = socket(family, type, flags)) < 0) {
    perror("socket error");
    exit(1);
  }
  return sockfd;
}

// Envia dados do cliente e escreve em um buffer
// Se retornar algo > 0, ainda ha dados a serem escritos (ultrapassaram o tamanho do buffer)
void Write(int sockfd, char* buffer) {
  int write_size = write(sockfd, buffer, strlen(buffer));
  if (write_size < 0) {
    perror("write error");
    exit(1);
  }
}

void Select(int maxfdp1, fd_set *readset, fd_set *writeset, fd_set *exceptset, struct timeval *timeout) {
   if ( !select(maxfdp1, readset, writeset, exceptset, timeout) ) {
      perror("select error");
      exit(1);
   }
}
```

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

#define LISTENQ 10
#define MAXDATASIZE 40000

#define TRUE 1
#define FALSE 0

int Accept(int listenfd, struct sockaddr_in *clientaddr);
void Bind(int listenfd, struct sockaddr_in servaddr);
void Close(int connection);
void Connect(int sockfd, struct sockaddr_in sockaddress);
struct sockaddr_in Getsockname(int sockfd, struct sockaddr_in sockaddress);
void InetNtop(int family, char* buffer, struct sockaddr_in sockaddress);
void InetPton(int family, char *ipaddress, struct sockaddr_in sockaddress);
void Listen(int listenfd, int listenq);
int Read(int sockfd, char* buffer);
int Socket(int family, int type, int flags);
void Write(int sockfd, char* buffer);
void Select(int maxfdp1, fd_set *readset, fd_set *writeset, fd_set *exceptset, s
truct timeval *timeout);
```