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

#define MAXDATASIZE 4096

// 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);
        } else {
                return sockfd;
        }
}

// 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);
        }
}

// 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);
        }
}

// 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)*MAXDAT
ASIZE) <= 0) {
                perror("inet_ntop 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;
}

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

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

/*
   Cliente
   Aplicacao simples de cliente tcp que se conecta num
   IP e PORTA passados por parametro, envia um comando ao
   servidor e escreve na saida padrao o retorno
*/
int main(int argc, char **argv) {
   // Declaracao de variaveis
   int sockfd;
   char buf[MAXDATASIZE + 1], error[MAXDATASIZE + 1];
   char server[MAXDATASIZE + 1], server_reply[MAXDATASIZE + 1];
   struct sockaddr_in servaddr;

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

```
        // Escrever IP e porta do servidor na saida padrao
    printf("Server: IP %s − Port %d\n", argv[1], atoi(argv[2]));

    // Coletar informacoes sobre o socket com o servidor
    servaddr = Getsockname(sockfd, servaddr);

    // Converter informacao do IP de binario para string
    // armazenar o resultado no buffer
    InetNtop(AF_INET, server, servaddr);

        // Escrever IP e porta do cliente no socket na saida padrao
        printf("Client: IP %s − Port %d\n", server, ntohs(servaddr.sin_port));

    // lÃª uma cadeia de caracteres do teclado
    printf("Digite um comando:\n");
    fgets(buf, MAXDATASIZE, stdin);

    // Imprime a linha de comando digitada pelo usuario
    printf("Linha de comando digitada: %s", buf);

    // envia os dados lidos ao servidor
    Write(sockfd , buf);

    // le os dados enviados pelo servidor
    Read(sockfd, server_reply);

    // Imprime a linha de comando devolvida pelo servidor
    printf("Linha de comando recebida: %s\n", server_reply);

    exit(0);
}
```

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

#define LISTENQ 10
#define MAXDATASIZE 4096

// 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);
        } else {
          return sockfd;
        }
}

// 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);
        }
}

// 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);
   }
}

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

// Recebe dados do cliente e escreve em um buffer
// Se retornar algo > 0, ainda ha dados a serem escritos (ultrapassaram o tamanh
o do buffer)
void Read(int sockfd, char* buffer) {
        int read_size;
        read_size = recv(sockfd, buffer, MAXDATASIZE, 0);
        if (read_size < 0) {
```

```
                perror("read error");
                exit(1);
        }
}

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

// 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);
 }
}

/*
   Servidor
   Aplicacao simples de servidor tcp que recebe varias
   conexoes na porta passada por parametro e executa
   o comando enviado pelo cliente
*/
int main (int argc, char **argv) {
   // Declaracao de variaveis
   int    listenfd, connfd;
   struct sockaddr_in servaddr;
   struct sockaddr_in clientaddr;
   char   buf[MAXDATASIZE + 1], error[MAXDATASIZE + 1], client[MAXDATASIZE + 1];

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

      // Converter informacao do IP de binario para string
      // armazenar o resultado no buffer
      InetNtop(AF_INET, buf, clientaddr);

      // Escrever IP e porta do cliente na saida padrao
                printf("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));

      // Recebe o comando do cliente
      Read(connfd, client);

                // Imprime a linha de comando recebida do usuario
                printf("Linha de comando recebida cliente: %s", client);

                // Enviar a mensagem de volta para o cliente
                write(connfd, client, strlen(client));

                // Executar o comando
                printf("Execucao do Comando:\n");
                system(client);
                printf("\n");

      // Finalizar a conexao
      close(connfd);
   }
   return(0);
}
```