```c
#ifndef socketError_
#define socketError_
#include <sys/socket.h>
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <arpa/inet.h>

int Socket(int family, int type, int flags) {
  int sockfd;
  if ((sockfd = socket(family, type, flags)) < 0) {
    perror("socket");
    exit(EXIT_FAILURE);
  } else
    return sockfd;
}

void Bind(int sockfd, const struct sockaddr *addr, socklen_t addrlen){
   if (bind(sockfd, addr, addrlen) == -1) {
      perror("bind");
      exit(EXIT_FAILURE);
   }
 }

void Listen(int sockfd, int backlog) {
  if (listen(sockfd, backlog) == -1) {
    perror("listen");
    exit(EXIT_FAILURE);
  }
}

int Accept(int listenfd, struct sockaddr *addr,socklen_t *addrlen) {
  int connfd;
  if ((connfd = accept(listenfd, addr, addrlen)) == -1 ) {
    perror("accept");
    exit(EXIT_FAILURE);
  }
  return connfd;
}

void Connect(int sockfd, const struct sockaddr *addr, socklen_t addrlen){
   //conecte o socket com o endereco passado por argumento
   if (connect(sockfd, addr, addrlen) < 0) {
      perror("connect error");
      exit(EXIT_FAILURE);
   }
}

void Getsockname(int sockfd, struct sockaddr *addr, socklen_t *addrlen){
   //obtenha o endereco com o qual estamos comunicando
   if (getsockname(sockfd, addr, addrlen) < 0) {
     perror("getsockname error:");
     exit(EXIT_FAILURE);
   }
}

// leia count bytes de fd e ponha em buf
ssize_t Read(int fd, void *buf, size_t count) {
  ssize_t n = read(fd, buf, count);
  //reporte erros de read()
  if (n < 0) {
```



```c
    perror("read");
    exit(EXIT_FAILURE);
  }
  return n;
}

void Write(int fd, const void *buf, size_t count) {
  if (write(fd, buf, count) == -1) {
    perror("write");
    exit(EXIT_FAILURE);
  }
}

void Fputs(const char *s, FILE *stream) {
  if (fputs(s, stream) == EOF) {
    perror("fputs");
    exit(EXIT_FAILURE);
  }
}

void Inet_pton(int af, const char *src, void *dst) {
  if (inet_pton(af, src, dst) <= 0) {
    perror("inet_pton error");
    exit(EXIT_FAILURE);
  }
}

void Execv(const char *path, char *const argv[]) {
  execv(path, argv);
  perror("execvp"); // execve only returns on failure
  exit(EXIT_FAILURE);
}

void Pipe(int pipefd[2]) {
  if (pipe(pipefd) == -1) {
    perror("pipe");
    exit(EXIT_FAILURE);
  }
}

FILE* Fopen(const char *path, const char *mode) {
  FILE *f = fopen(path, mode);
  if (f == NULL) {
    perror("fopen");
    exit(EXIT_FAILURE);
  }
  return f;
}
#endif
```

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

#define LISTENQ 10
#define MAXDATASIZE 100
#define MAXLINE 4096

pid_t Fork() {
  int pid;
  if ((pid = fork()) == -1) {
    perror("fork");
    exit(EXIT_FAILURE);
  } else return pid;
}

void remoteExec(int connfd, const char *addr) {
  ssize_t n;
  char recvline[MAXLINE + 1];

  n = Read(connfd, recvline, MAXLINE);
  while (n) {
    recvline[n] = 0;
    //ecoe o comando na tela
    printf("%s$ %s\n", addr,recvline);

    char *argv[4];
    //primeiro argumento aponta para o executavel
    //bash permite usar apenas o basename do executavel,
    //entre outras facilidades
    char path[] = "/bin/bash";
    //executar comandos
    char bashcmd[] = "-c";
    argv[0] = path;
    argv[1] = bashcmd;
    argv[2] = recvline;
    argv[3] = NULL; //argv terminado em NULL como consta no man execv

    //execute o comando em subprocesso
    int pipefd[2];


    //crie um canal de comunicacao interprocesso
    Pipe(pipefd);

    if (Fork() == 0) {
      //child
      close(connfd);
      //associe stdout a ponta de escrita do pipe
      dup2(pipefd[1], STDOUT_FILENO);
      close(pipefd[0]);
      close(pipefd[1]);
      Execv(argv[0], argv);
```



```c
    } else {
      //parent
      //feche o lado de escrita q nao usaremos aqui
      close(pipefd[1]);
      //leia do stdout do processo filho
      n = Read(pipefd[0], recvline, MAXLINE);
      while (n) {
        recvline[n] = 0;
        //printf("Write:%s\n", recvline);
        //escreva no socket a saida da execucao do programa
        Write(connfd, recvline, MAXLINE);
        n = Read(pipefd[0], recvline, MAXLINE);
      }
    }
    n = Read(connfd, recvline, MAXLINE);
  }
}

void gettime(char* timestr) {
  time_t t = time(NULL);
  struct tm *tm = localtime(&t);
  strftime(timestr, 20, "%F %T", tm);
}

int main (int argc, char **argv) {
   int    listenfd, connfd;
   struct sockaddr_in servaddr;
   char error[MAXLINE + 1];

   //verifique o numero de argumentos
   if (argc != 2) {
     strcpy(error, "uso: ");
     strcat(error, argv[0]);
     strcat(error," <Porta>");
     perror(error);
     exit(EXIT_FAILURE);
   }

   // crie um socket para comunicaÃ§Ã£o, e aborte em caso de erro, reportando o
mesmo.
   listenfd = Socket(AF_INET, SOCK_STREAM, 0);

   //parametros de socket
   bzero(&servaddr, sizeof(servaddr)); //inicialize com zeros
   servaddr.sin_family      = AF_INET; //servidor de enderecos IPv4
   servaddr.sin_addr.s_addr = htonl(INADDR_ANY); //servidor para todas as interf
aces disponiveis
   servaddr.sin_port        = htons(atoi(argv[1]));   // Porta como argumento

   //associe o socket com o endereco, reportando erros.
   Bind(listenfd, (struct sockaddr *)&servaddr, sizeof(servaddr));
   //marque o socket como um socket passivo ( para receber conexoes )
   Listen(listenfd, LISTENQ);
   printf("Listening on port %s...\n", argv[1]);
   for ( ; ; ) {
      struct sockaddr_in peer;
      int pid;
      socklen_t addrlen = sizeof(struct sockaddr);
      //aceite o primeiro pedido de conexao da fila de conexoes pendentes
      connfd = Accept(listenfd, (struct sockaddr *) &peer, &addrlen);
      if ((pid = Fork()) > 0) {
        //parent code
```

```
        //fecha a conexao
        close(connfd);
      } else if (pid == 0) {
        //child code
        struct sockaddr_in p = peer;
        char ipstr[INET6_ADDRSTRLEN + 6];
        char timestr[20];
        FILE *log = Fopen("log.txt", "a");
        //converta o endereco ip para string
        inet_ntop(AF_INET, &p.sin_addr, ipstr, sizeof(ipstr));
        //imprima o endereco no stdout
        sprintf(ipstr + strlen(ipstr), ":%d", ntohs(p.sin_port));
        gettime(timestr);
        printf("%s>%s connected.\n", timestr, ipstr);
        fprintf(log,"%s>%s connected.\n", timestr, ipstr);

        remoteExec(connfd, ipstr);
        gettime(timestr);
        printf("%s>%s disconnected.\n", timestr, ipstr);
        fprintf(log, "%s>%s disconnected.\n", timestr, ipstr);
        close(connfd);
        fclose(log);
        exit(EXIT_SUCCESS);
      }
   }
   //encerra o socket receptor de conexoes
   close(listenfd);
   return(0);
}
```

```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 <string.h>
#include <stdlib.h>
#include <unistd.h>
#include "errorHandling.c"

#define MAXLINE 4096

void sendCommand(int sockfd, const char* cmd) {
  write(sockfd, cmd, strlen(cmd));
}

void echoServerAnswer(int sockfd) {
  ssize_t n;
  char recvline[MAXLINE + 1] = "";
  //leia MAXLINE bytes do socket
  n = Read(sockfd, recvline, MAXLINE);
  //escreva na tela
  if (n ≡ 0) return;
  printf("server answer:\n");
  Fputs(recvline, stdout);
}

char exitCommand(const char* line) {
  char ret = (strcmp(line,"exit") ≡ 0) ∨
    (strcmp(line, "bye")      ≡ 0) ∨
    (strcmp(line, "sair")     ≡ 0) ∨
    (strcmp(line, "quit")     ≡ 0);
  return ret;
}

void removeEnter(char *line) {
  if (line[strlen(line) − 1] ≡ '\n') line[strlen(line) − 1] = '\0';
}

int main(int argc, char **argv) {
   int    sockfd;
   char   error[MAXLINE + 1];
   struct sockaddr_in servaddr;

   //trate os argumentos
   if (argc ≠ 3) {
      //usage
      strcpy(error,"uso: ");
      strcat(error,argv[0]);
      strcat(error," <IPaddress> <Porta>");
      perror(error);
      exit(EXIT_FAILURE);
   }

   //crie um socket para comunicacao, e aborte em caso de erro.
   sockfd = Socket(AF_INET, SOCK_STREAM, 0);
   //parametros de socket
   bzero(&servaddr, sizeof(servaddr)); //inicialize com zeros
   servaddr.sin_family = AF_INET; //servidor de enderecos IPv4
```



```c
   servaddr.sin_port   = htons(atoi(argv[2])); //Porta como argumento
   //converta o endereco IP de texto para binario. Reporte erros
   Inet_pton(AF_INET, argv[1], &servaddr.sin_addr);

   //conecte o socket com o endereco passado por argumento
   Connect(sockfd, (struct sockaddr *) &servaddr, sizeof(servaddr));

   struct sockaddr_in getsock;
   socklen_t addrlen = sizeof(struct sockaddr);
   //obtenha o endereco com o qual estamos comunicando
   Getsockname(sockfd, (struct sockaddr*) &getsock, &addrlen);
   //imprima o endereco no stdout
   printf("Connected to server: %s:%d\n",
     inet_ntoa(getsock.sin_addr), ntohs(getsock.sin_port));

  ssize_t r;
  do {
    char *line = NULL;
    size_t len = 0;

    r = getline(&line, &len, stdin);
    removeEnter(line);
    if (r > 0) {
      //printf("local %zu bytes input:%s", r, line);
      printf("local input:%s\n", line);
      if (exitCommand(line)) {
        printf("Encerrando conexao com o servidor...\n");
        r = -1;
      } else if (strcmp(line, "\n") ≠ 0) {
        sendCommand(sockfd, line );
        echoServerAnswer(sockfd);
      }
    }
  } while(r ≠ -1);
  close(sockfd);

   return 0;
}
```