```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);
}
```