1 cliente.c

```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>
#include <sys/select.h>
#include <dirent.h>
#include <stdbool.h>

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

//##############################################################
//wrapper functions
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;
}

void Inet_pton(int af, const char *src, void *dst){
   if (inet_pton(af, src, dst) <= 0) {
      perror("inet_pton error");
      exit(1);
   }
}

void Fputs(char *msg){
   if (fputs(msg, stdout) == EOF) {
      perror("error fputs");
      exit(1);
   }
}

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

```
    }
}

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

int Accept(int sockfd, struct sockaddr *addr, socklen_t *len){
    int connfd;

    if ((connfd = accept(sockfd, (struct sockaddr *) addr, len)) == -1 )
        {
        perror("accept");
        exit(1);
    }

    return connfd;
}

void Connect(int sockfd, struct sockaddr *serv_addr, socklen_t len){
   if (connect(sockfd, serv_addr, len) < 0) {
      perror("connect error");
      exit(1);
   }
}


void Close(int sockfd){
    if (close(sockfd) == -1 ) {
        perror("close");
        exit(1);
    }
}

ssize_t Recvfrom(int socket, void *restrict buffer, size_t length,
       int flags, struct sockaddr *restrict address,
       socklen_t *restrict address_len){

   ssize_t r = recvfrom(socket, buffer, length, flags, address,
        address_len);

    if (r == -1) {
        perror("listen");
        exit(1);
    }

    else
      return r;
}

ssize_t Sendto(int socket, const void *message, size_t length,
```

```
       int flags, const struct sockaddr *dest_addr,
       socklen_t dest_len){

   ssize_t r = sendto(socket, message, length, flags, dest_addr,
       dest_len);

    if (r == -1) {
        perror("listen");
        exit(1);
    }

    else
      return r;
}

int Select(int nfds, fd_set *restrict readfds, fd_set *restrict
    writefds, fd_set *restrict errorfds, struct timeval *restrict
    timeout){
   int r = select(nfds, readfds, writefds, errorfds, timeout);

   if(r > 0)
      return r;
   else{
      perror("error select");
      exit(1);
   }
}

//#################################################################
//helper functions
int max (int a, int b){
   if(a > b)
      return a;
   else
      return b;
}

//check if the process with a given name is running
bool findProcess(char *name){
   const char* directory = "/proc";
   size_t     taskNameSize = 1024;
   char*      taskName = calloc(1, taskNameSize);

   DIR* dir = opendir(directory);

   if (dir)
   {
      struct dirent* de = 0;

      while ((de = readdir(dir)) != 0)
      {
         if (strcmp(de->d_name, ".") == 0 || strcmp(de->d_name, "..") ==
             0)
            continue;
```

```
        int pid = -1;
        int res = sscanf(de->d_name, "%d", &pid);

        if (res == 1)
        {
           // we have a valid pid

           // open the cmdline file to determine what's the name of the
               process running
           char cmdline_file[1024] = {0};
           sprintf(cmdline_file, "%s/%d/cmdline", directory, pid);

           FILE* cmdline = fopen(cmdline_file, "r");

           if (getline(&taskName, &taskNameSize, cmdline) > 0)
           {
              // is it the process we care about?
              if (strstr(taskName, name) != 0)
              {
                 return true;
              }
           }

        fclose(cmdline);
        }
     }

     closedir(dir);
  }

  free(taskName);
  return false;
}

// Get ip from domain name
bool hostname_to_ip(char * hostname , char* ip)
{
    struct hostent *he;
    struct in_addr **addr_list;
    int i;

    if ( (he = gethostbyname( hostname ) ) == NULL)
    {
      return false;
    }

    addr_list = (struct in_addr **) he->h_addr_list;

    for(i = 0; addr_list[i] != NULL; i++)
    {
        //Return the first one;
        strcpy(ip , inet_ntoa(*addr_list[i]) );
        return true;
```

```
    }

    return false;
}

//checa se o nome eh um ip (de maneira simplificada)
bool isIp(char *name){
   int n = strlen(name);
   int i;

   for(i = 0; i < n; i++){
      if(!((name[i] - 48 >= 0 && name[i] - 48 <= 9) || name[i] == '.'))
         return false;
   }

   return true;
}


//main function
int main(int argc, char **argv) {
   int    sockfd, n;
   char   recvline[MAXLINE + 1];
   struct sockaddr_in servaddr;
   char   buf[MAXDATASIZE];
   socklen_t len = sizeof(struct sockaddr);
   char file_name[MAXFILENAME];
   pid_t pid;

   //numero errado de argumentos passados pela linha de comando
   if (argc != 3) {
      printf("modo de utilizacao:\n");
      printf("./cliente [IP_servidor | nome_servidor]
          [porta_servidor]\n");
      exit(1);
   }

   //Exibe dados do servidor ao qual cliente esta se conectando
   //printf("Conecting to server...\n");
   //printf("IP: %s\n", argv[1]);
   //printf("Porta: %s\n", argv[2]);

   //inicializa um socket e coloca sua referencia (file descriptor) em
       sockfd
   sockfd = Socket(AF_INET, SOCK_DGRAM, 0);

   //preenche com zeros a estrutura servaddr
   bzero(&servaddr, sizeof(servaddr));

   //informa que as mensagens trocadas no socket utilizam o IPV4
   servaddr.sin_family = AF_INET;

   //informa a porta utilizada pelo socket, convertendo para big endian
       se necessario
```

```
servaddr.sin_port = htons(atoi(argv[2]));

//checa se recebeu um ip ou o nome
char ip[100];
strcpy(ip, argv[1]);

if(!isIp(argv[1])){
   //checa se tem um processo local com o nome
   if(findProcess(argv[1]))
      strcpy(ip, "127.0.0.1");

   //checa se existe esse nome usando o servico dns
   else if(!hostname_to_ip(argv[1], ip)){
      printf("Nao encontrei o servidor especificado.\n");
      return 0;
   }
}

//coloca o ip recebido por linha de comando na estrutura servaddr
printf("%s\n", ip);
Inet_pton(AF_INET, ip, &servaddr.sin_addr);

//variaveis que serao parametros para o select
int maxfdp1;
 fd_set rset;
 int eof_stdin = 0;

 //envia uma mensagem vazia para o servidor para receber a porta
     correta
 bzero(buf, MAXDATASIZE);
 Sendto(sockfd, buf, 1, 0, (struct sockaddr *) &servaddr, len);

 n = Recvfrom(sockfd, recvline, MAXLINE, 0, (struct sockaddr *)
     &servaddr, &len);
unsigned short new_port;

recvline[n] = 0;
sscanf(recvline, "%hu", &new_port);

 printf("\nNEW PORT: %hu\n", ntohs(new_port));

 //atualiza a porta utilizada pelo socket, convertendo para big
     endian se necessario
servaddr.sin_port = htons(new_port);

 //envia uma nova mensagem vazia para o servidor para receber o bem
     vindo de volta
 bzero(buf, MAXDATASIZE);
 Sendto(sockfd, buf, 1, 0, (struct sockaddr *) &servaddr, len);

 //envia para o servidor
 while(1){
   //seta os descritores do stdin e do socket
   if(!eof_stdin)
```

```
   FD_SET(fileno(stdin), &rset);
FD_SET(sockfd, &rset);

  //seta um limitante para o numero dos descritores (isso ajuda no
      desempenho do select)
  maxfdp1 = max(fileno(stdin), sockfd) + 1;

  //chama select, quando ele voltar, somente os descritores que
      receberam algo terao 1 no fd_set
  Select(maxfdp1, &rset, NULL, NULL, NULL);

  //stdin recebeu algo para ler
  if (FD_ISSET(fileno(stdin), &rset)){
   bzero(buf, MAXDATASIZE);
   read(fileno(stdin), buf, MAXDATASIZE);

   char subbuf[6];
   strncpy(subbuf, buf, 6);

   //ativa transferencia de arquivo
   if(strcmp(subbuf, "\\file ") == 0){
      bzero(file_name, MAXFILENAME);
      strncpy(file_name, buf + 6, strlen(buf) - 7);

      //envia mensagem para o servidor para abrir conexao tcp no
          amigo
      sprintf(buf, "\\file1\n");
      Sendto(sockfd, buf, strlen(buf), 0, (struct sockaddr *)
          &servaddr, len);
   }

   else{
      Sendto(sockfd, buf, strlen(buf), 0, (struct sockaddr *)
          &servaddr, len);

      //saiu do chat
      if(strcmp(buf, "\\exit\n") == 0)
         break;
   }
}

//socket recebeu algo para ler
  if (FD_ISSET(sockfd, &rset)){
   bzero(recvline, MAXLINE);

   n = Recvfrom(sockfd, recvline, MAXLINE, 0, (struct sockaddr *)
       &servaddr, &len);

   if(n > 0){
      char cmd[100];
      unsigned long ip;
      unsigned short port;

      recvline[n] = 0;
```

```
sscanf(recvline, "%s %lu %hu", cmd, &ip, &port);

//transferencia de arquivo: precisa abrir conexao para receber
if(strcmp(cmd, "\\file_receiver") == 0){
   int listenfd, connfd;
   struct sockaddr_in myaddr;

   //coloca endereco local passado em myaddr
   myaddr.sin_family = AF_INET;
   myaddr.sin_port = port;
   myaddr.sin_addr.s_addr = ip;

   //cria o socket e associa ao endereco cadastrado no
       servidor
   listenfd = Socket(AF_INET, SOCK_STREAM, 0);
   Bind(listenfd, (struct sockaddr *)&myaddr, len);

   //coloca o socket para escutar novas requisicoes
   Listen(listenfd, LISTENQ);

   //avisa o servidor para ele avisar o cliente
   sprintf(buf, "\\file2\n");
   Sendto(sockfd, buf, strlen(buf), 0, (struct sockaddr *)
       &servaddr, len);

   //aceita uma conexao nova e cria um novo socket para ela
   connfd = Accept(listenfd, (struct sockaddr *) &myaddr,
       &len);

   //esse bloco eh executado somente no filho
   if((pid = fork()) == 0) {
      FILE *fp;
      fp = fopen ("output.txt", "w");
      Close(listenfd); //filho deve fechar o listening socket

      //agora processamos a requisicao
      while(1){
         //recebe as mensagens do servidor
         while ( (n = read(connfd, buf, MAXDATASIZE)) > 0) {
            buf[n] = 0;
            fprintf(fp, buf);
            break;
         }

         //problema de leitura
         if (n < 0) {
            perror("error reading socket");
            exit(1);
         }

         if(n == 0)
            break;
      }
```

```
            Close(connfd); //filho terminou de processar,
                connection socket pode ser fechado
            exit(0); //termina
        }

        printf("FECHEI SOCKET1!\n");
        Close(connfd); //pai deve fechar o connection socket
        Close(listenfd);
    }

    //transferencia de arquivo: precisa abrir conexao para enviar
    else if(strcmp(cmd, "\\file_sender") == 0){
        int writefd;
        struct sockaddr_in cliaddr;

        //coloca endereco local passado para em myaddr
        cliaddr.sin_family = AF_INET;
        cliaddr.sin_port = port;
        cliaddr.sin_addr.s_addr = ip;

        //esse bloco eh executado somente no filho
        if((pid = fork()) == 0) {
            //inicializa socket
            writefd = Socket(AF_INET, SOCK_STREAM, 0);

            //tenta realizar a conexao
            Connect(writefd, (struct sockaddr *) &cliaddr, len);

            FILE *fp;
            fp = fopen (file_name, "r");

            //envia para o servidor
            while(fgets(buf, MAXDATASIZE, fp) != NULL){
                n = write(writefd, buf, strlen(buf));

                if (n < 0)
                    printf("error writing to socket\n");

                bzero(buf, MAXDATASIZE);
            }

            fclose(fp);
            Close(writefd);
            printf("FECHEI SOCKET2!\n");
            exit(0); //termina
        }
    }

    else{
        printf("[RECEBIDO:] ");
        Fputs(recvline);
    }
  }
}
```

```
   }

   exit(0);
}
```

2 servidor.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 <time.h>
#include <ifaddrs.h>
#include <linux/if_link.h>
#include <pthread.h>

#define LISTENQ 10
#define MAXDATASIZE 4096
#define MAXCLIENTS 100
#define NAMESIZE 100

//########################################################################
//GET IP

//retorna um endereco de ip para ser usado pelo servidor
void getServerIP(char ip[]){
    struct ifaddrs *ifaddr, *ifa;
    int n;
    struct sockaddr_in *pAddr;
    int family;

    if (getifaddrs(&ifaddr) == -1) {
       perror("getifaddrs");
       exit(EXIT_FAILURE);
    }

    //caminha pela lista ligada retornada por getifaddrs
    for (ifa = ifaddr, n = 0; ifa != NULL; ifa = ifa->ifa_next, n++) {
        if (ifa->ifa_addr == NULL)
           continue;

        family = ifa->ifa_addr->sa_family;

        if(family == AF_INET){
```

```
            //verifica se o nome eh eno1
            pAddr = (struct sockaddr_in *)ifa->ifa_addr;
            if(strcmp(ifa -> ifa_name, "eno1") == 0){
                //copia para o ip
                strcpy(ip, inet_ntoa(pAddr->sin_addr));
                break;
            }
        }
    }

    freeifaddrs(ifaddr);
}

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

ssize_t Recvfrom(int socket, void *restrict buffer, size_t length,
       int flags, struct sockaddr *restrict address,
       socklen_t *restrict address_len){

   ssize_t r = recvfrom(socket, buffer, length, flags, address,
       address_len);

    if (r == -1) {
        perror("listen");
        exit(1);
    }

    else
      return r;
}

ssize_t Sendto(int socket, const void *message, size_t length,
       int flags, const struct sockaddr *dest_addr,
       socklen_t dest_len){

   ssize_t r = sendto(socket, message, length, flags, dest_addr,
       dest_len);

    if (r == -1) {
        perror("listen");
        exit(1);
    }

    else
      return r;
}
```

```
void Close(int sockfd){
    if (close(sockfd) == -1 ) {
        perror("close");
        exit(1);
    }
}

void Fputs(char *msg){
   if (fputs(msg, stdout) == EOF) {
      perror("error fputs");
      exit(1);
   }
}

//#################################################################
//LOG

void insertLog(char *msg)
{
   time_t start;

   //captura tempo em que evento ocorreu
   time(&start);

   //insere entrada no log
   FILE *logfile = fopen("log.txt","ab+");
   fprintf(logfile, "%s%s\n------------\n",ctime(&start),msg);
   fclose(logfile);

   //limpa array temporario
   memset(msg, 0, 1000);
}

//#################################################################
//RELACIONADO AO CHAT
typedef struct Client_aux{
   char name[NAMESIZE];
   struct sockaddr_in addr;
   struct Client_aux *friend;
   struct Client_aux *next;
   struct Client_aux *prev;
} Client;

Client *clients_head, *clients_tail;
int n_clients;

//cria uma lista ligada de clientes
void newClientList(){
   clients_head = malloc(sizeof(Client));
   clients_head -> next = NULL;
   clients_head -> prev = NULL;
   clients_tail = clients_head;
}
```

```
//insere um cliente novo na lista do chat
void insertClient(char *name, struct sockaddr_in *addr){
   if(n_clients + 1 > MAXCLIENTS)
      printf("CHAT TA CHEIO\n");

   else{
      Client *c = malloc(sizeof(Client));

      strcpy(c -> name, name);
      memcpy(&(c -> addr), addr, sizeof(struct sockaddr_in));
      c -> friend = NULL;
      c -> next = NULL;
      c -> prev = clients_tail;

      clients_tail -> next = c;
      clients_tail = c;

      printf("addr: %p, next: %p, prev: %p \n", (void *)c, (void *)c ->
          next, (void *)c -> prev);
      n_clients++;
   }

   Client *c = clients_head -> next;

   printf("Clientes conectados:\n");

   while(c != NULL){
      printf("addr: %p name: %s next: %p prev: %p \n", (void *)c, c ->
          name, (void *)c -> next, (void *)c -> prev);
      c = c -> next;
   }
}

//encontra cliente pelo nome
Client* findClientByName(char *name){
   Client *c = clients_head -> next;

   while(c != NULL){
      if(strcmp(c -> name, name) == 0)
         break;

      c = c -> next;
   }

   return c;
}

//encontra cliente pelo endereco
Client* findClientByAddr(struct sockaddr_in addr){
   Client *c = clients_head -> next;

   while(c != NULL){
      if(c -> addr.sin_port == addr.sin_port &&
         c -> addr.sin_addr.s_addr == addr.sin_addr.s_addr)
```

```
            break;

        c = c -> next;
    }

    return c;
}

//concatena em buf mensagem nomes dos clientes na list
void concatClientsList(char* buf){
    Client *c = clients_head -> next;

    strcat(buf, "Clientes conectados:\n");

    while(c != NULL){
        strcat(buf, "\n");
        strcat(buf, c -> name);

        c = c -> next;
    }

    strcat(buf, "\n\n");
}

//remove cliente da lista
void removeClient(Client *c){
    c -> prev -> next = c -> next;

    if(c -> next != NULL)
        c -> next -> prev = c -> prev;
    else
        clients_tail = c -> prev;

    free(c);
}

//################################################################
//RELACIONADO A THREADS

//estrutura para os argumentos de processClient
typedef struct args_aux{
    char buf[MAXDATASIZE];
    struct sockaddr_in cliaddr;
    int sockfd;
    int n;
    unsigned short servport;
} Args;

pthread_mutex_t lock;

//faz o que precisa ser feito com o cliente
void *processClient(void *args2){
    Client *c1, *c2;
    char buf[MAXDATASIZE];
```

```
char tempMsg[1000];
int n, sockfd;
struct sockaddr_in cliaddr;
Args *args = (Args *)args2;

//copia os argumentos de processClient para args
strcpy(buf, args -> buf);
sockfd = args -> sockfd;
cliaddr.sin_family = args -> cliaddr.sin_family;
cliaddr.sin_port = args -> cliaddr.sin_port;
cliaddr.sin_addr.s_addr = args -> cliaddr.sin_addr.s_addr;
n = args -> n;

//lista os clientes conectados
if(strcmp(buf, "\\list\n") == 0){
   bzero(buf, MAXDATASIZE);
   concatClientsList(buf);
   Sendto(sockfd, buf, strlen(buf), 0, (struct sockaddr *) &cliaddr,
       sizeof(cliaddr));
   bzero(buf, MAXDATASIZE);
   return NULL;
}

//Cliente desconectando
if(strcmp(buf, "\\exit\n") == 0){
   c1 = findClientByAddr(cliaddr);

   //remove da lista caso ja cadastrado
   if(c1 != NULL){

      c2 = c1 -> friend;

      //avisa o amigo caso exista
      if(c2 != NULL){
         sprintf(buf, "Seu amigo %s saiu do chat.\n", c1 -> name);
         strcat(buf, "Por favor, use \\list para ver os usuarios
             disponiveis e envie outro comando no formato \\friend
             [nome_do_amigo] para continuar utilizando o chat.\n");
         Sendto(sockfd, buf, strlen(buf), 0, (struct sockaddr *) &(c2
             -> addr), sizeof(c2 -> addr));
         c2 -> friend = NULL;
      }

      //mensagem de desconexao para o log
      sprintf(tempMsg, "%s@%s:%d se desconectou", c1->name,
          inet_ntoa(c1->addr.sin_addr), ntohs(c1->addr.sin_port));

      pthread_mutex_lock(&lock);
      removeClient(c1);
      pthread_mutex_unlock(&lock);
   }
   else {
      //mensagem de desconexao para o log (sem nome de usuario)
```

```
      sprintf(tempMsg, "%s:%d se desconectou",
          inet_ntoa(cliaddr.sin_addr), ntohs(cliaddr.sin_port));
   }

   //registra desconexao no log
   pthread_mutex_lock(&lock);
   insertLog(tempMsg);
   pthread_mutex_unlock(&lock);

   bzero(buf, MAXDATASIZE);
   return NULL;
}

//se n == 0 entao o datagrama so tem o header
if(n > 0){
   //imprime informacoes do cliente
   printf("received: %s\n", buf);
   printf("Client IP : %s\n", inet_ntoa(cliaddr.sin_addr));
   printf("Client Port : %d\n", ntohs(cliaddr.sin_port));

   c1 = findClientByAddr(cliaddr);

   //cliente nao esta na lista
   if(c1 == NULL){

      char subbuf[6];
      strncpy(subbuf, buf, 6);

      //insere nova conexao no log
      sprintf(tempMsg, "%s:%d se conectou ao servidor",
          inet_ntoa(cliaddr.sin_addr), ntohs(cliaddr.sin_port));
      pthread_mutex_lock(&lock);
      insertLog(tempMsg);
      pthread_mutex_unlock(&lock);

      //cliente esta enviando o nome dele
      if(strcmp(subbuf, "\\name ") == 0){
         char name[NAMESIZE];

         bzero(name, NAMESIZE);
         strncpy(name, buf + 6, strlen(buf) - 7);

         //printf("Name: %s\n", name);

         //checa se esse nome ja esta sendo usado
         if(findClientByName(name) != NULL){
            sprintf(buf, "Nome ja utilizado. Por favor, envie um novo
                nome.\n");
         }

         else{
            pthread_mutex_lock(&lock);
            insertClient(name, &cliaddr);
            pthread_mutex_unlock(&lock);
```

```
            //insere novo usuario no log
            sprintf(tempMsg, "%s:%d atualizou seu nome para %s",
                inet_ntoa(cliaddr.sin_addr), ntohs(cliaddr.sin_port),
                name);
            pthread_mutex_lock(&lock);
            insertLog(tempMsg);
            pthread_mutex_unlock(&lock);

            sprintf(buf, "Usuario cadastrado com sucesso! Agora envie
                \\friend [nome_do_amigo] para se comunicar com
                alguem.\n");
        }
    }

    //cliente nao esta enviando o nome
    else{
        bzero(buf, MAXDATASIZE);
        strcat(buf, "Bem vindo ao chat!\n");
        strcat(buf, "Digite \\list para exibir os clientes conectados
            e \\exit para sair.\n");
        concatClientsList(buf);
        strcat(buf, "Por favor, envie o seu nome no formato \\name
            [seu_nome] para poder utilizar o chat.\n");
    }

    Sendto(sockfd, buf, strlen(buf), 0, (struct sockaddr *)
        &cliaddr, sizeof(cliaddr));
    bzero(buf, MAXDATASIZE);
    return NULL;
}

//cliente ja esta na lista
else{
    c2 = c1 -> friend;

    //cliente ainda nao possui amigo
    if(c2 == NULL){
        char subbuf[8];
        strncpy(subbuf, buf, 8);

        //cliente esta enviando o amigo
        if(strcmp(subbuf, "\\friend ") == 0){
            char fname[NAMESIZE];
            bzero(fname, NAMESIZE);

            strncpy(fname, buf + 8, strlen(buf) - 9);

            if(strcmp(fname, c1 -> name) == 0) {
                sprintf(buf, "Assim vc me deixa triste...\n");

                //insere tentativa de falar consigo mesmo no log
                sprintf(tempMsg, "%s@%s:%d tentou falar consigo
                    mesmo...", c1->name, inet_ntoa(c1->addr.sin_addr),
```

```
            ntohs(c1->addr.sin_port));
        pthread_mutex_lock(&lock);
        insertLog(tempMsg);
        pthread_mutex_unlock(&lock);
    }

    else{
        //encontra o amigo
        c2 = findClientByName(fname);

        if(c2 == NULL) {
            sprintf(buf, "Seu amigo %s nao esta no chat.\n",
                fname);

            //insere tentativa de falar consigo mesmo no log
            sprintf(tempMsg, "%s@%s:%d tentou falar com %s (nao
                existente)", c1->name,
                inet_ntoa(c1->addr.sin_addr),
                ntohs(c1->addr.sin_port), fname);
            pthread_mutex_lock(&lock);
            insertLog(tempMsg);
            pthread_mutex_unlock(&lock);
        }

        else{
            //conecta os amigos na lista
            c1 -> friend = c2;
            c2 -> friend = c1;

            //envia mensagem para o amigo que alguem se conectou
                a ele
            sprintf(buf, "Cliente %s se conectou a voce.\n", c1
                -> name);
            Sendto(sockfd, buf, strlen(buf), 0, (struct sockaddr
                *) &(c2 -> addr), sizeof(cliaddr));

            //insere no log
            sprintf(tempMsg, "%s@%s:%d se conectou a %s@%s:%d",
                c1->name, inet_ntoa(c1->addr.sin_addr),
                ntohs(c1->addr.sin_port),
                                                c2->name,
                                                    inet_ntoa(c2->addr.sin_addr),
                                                    ntohs(c2->addr.sin_port));
            pthread_mutex_lock(&lock);
            insertLog(tempMsg);
            pthread_mutex_unlock(&lock);

            //mensagem de retorno para o cliente
            sprintf(buf, "Pronto! Voce esta pronto para se
                comunicar com %s. Digite \\exit para sair do
                chat.\n", fname);
        }
    }
}
```

```
   //cliente nao esta enviando o amigo
   else{
      bzero(buf, MAXDATASIZE);
      strcat(buf, "Por favor, envie o seu amigo no formato
          \\friend [nome_do_amigo] para poder utilizar o
          chat.\n");
      concatClientsList(buf);
   }

   Sendto(sockfd, buf, strlen(buf), 0, (struct sockaddr *)
       &cliaddr, sizeof(cliaddr));
   return NULL;
}

//cliente ja tem um amigo e esta se comunicando normalmente
else{
   //ativa transferencia de arquivos no amigo que recebe
   if(strcmp(buf, "\\file1\n") == 0) {
      sprintf(buf, "\\file_receiver %lu %hu", (unsigned long)(c2
          -> addr).sin_addr.s_addr, c2 -> addr.sin_port);

      //registra transferencia no log
      sprintf(tempMsg, "[%s@%s:%d -> %s@%s:%d] ~ transferencia
          de arquivo", c1->name, inet_ntoa(c1->addr.sin_addr),
          ntohs(c1->addr.sin_port),
                                                    c2->name,
                                                        inet_ntoa(c2->addr.sin_addr),
                                                        ntohs(c2->addr.sin_port));
      pthread_mutex_lock(&lock);
      insertLog(tempMsg);
      pthread_mutex_unlock(&lock);
   }

   //ativa transferencia de arquivo no amigo que envia
   else if(strcmp(buf, "\\file2\n") == 0)
      sprintf(buf, "\\file_sender %lu %hu", (unsigned long)(c1
          -> addr).sin_addr.s_addr, c1 -> addr.sin_port);

   else {
      printf("Enviando de %s para %s: %s", c1 -> name, c2 ->
          name, buf);

      //registra mensagem no log
      sprintf(tempMsg, "[%s@%s:%d -> %s@%s:%d]: %s", c1->name,
          inet_ntoa(c1->addr.sin_addr), ntohs(c1->addr.sin_port),
                                      c2->name,
                                          inet_ntoa(c2->addr.sin_addr),
                                          ntohs(c2->addr.sin_port),
                                          buf);
      pthread_mutex_lock(&lock);
      insertLog(tempMsg);
      pthread_mutex_unlock(&lock);
   }
```

```
            Sendto(sockfd, buf, strlen(buf), 0, (struct sockaddr *) &(c2
                -> addr), sizeof(c2 -> addr));
            bzero(buf, MAXDATASIZE);
            return NULL;
          }
        }
    }

    bzero(buf, MAXDATASIZE);
    return NULL;
}

// dedicado a processar um unico cliente numa porta separada
// cada thread ficara num loop dentro desta procedure
void *parallelClient(void *args2){

    Args *args = (Args *) args2;
    int    sockfd, n;
     struct sockaddr_in servaddr, cliaddr;
     char   buf[MAXDATASIZE];
     char   tempMsg[1000];
     char ip[100];
     socklen_t len = sizeof(struct sockaddr);

    cliaddr.sin_family = args -> cliaddr.sin_family;
    cliaddr.sin_port = args -> cliaddr.sin_port;
    cliaddr.sin_addr.s_addr = args -> cliaddr.sin_addr.s_addr;
    n = args -> n;

     //inicializa um socket novo e coloca sua referencia (file
         descriptor) em listenfd
     if ((sockfd = socket(AF_INET, SOCK_DGRAM, 0)) == -1) {
       perror("socket");
       exit(1);
     }

     //coloca o ip da interface eno1 do servidor
     getServerIP(ip);
     //strcpy(ip, "127.0.0.1");
     inet_pton(AF_INET, ip, &servaddr.sin_addr);

     //preenche com zeros a estrutura servaddr
     bzero(&servaddr, sizeof(servaddr));

     //informa que as mensagens trocadas no socket utilizam o IPV4
     servaddr.sin_family    = AF_INET;

     //informa a porta utilizada pelo socket, convertendo para big endian
         se necessario
     servaddr.sin_port      = htons(args -> servport);

     //associa o fluxo de dados associado ao file descriptor listenfd com
         as especificacoes em servaddr
```

```
    Bind(sockfd, (struct sockaddr *)&servaddr, len);

    //envia handshake de volta para o cliente
    bzero(buf, MAXDATASIZE);
    sprintf(buf, "%hu", args -> servport);
   Sendto(sockfd, buf, strlen(buf), 0, (struct sockaddr *) &cliaddr,
       sizeof(cliaddr));

   //registra porta do cliente no log
   bzero(tempMsg, 1000);
   sprintf(tempMsg, "%s:%d atrelado a porta %hu",
       inet_ntoa(cliaddr.sin_addr),
                                    ntohs(cliaddr.sin_port),
                                    ntohs(args -> servport));
   pthread_mutex_lock(&lock);
   insertLog(tempMsg);
   pthread_mutex_unlock(&lock);

    //loop principal da thread (dedicado a este cliente)
    for ( ; ; ) {
      bzero(buf, MAXDATASIZE);
        n = Recvfrom(sockfd, buf, MAXDATASIZE, 0, (struct sockaddr *)
            &cliaddr, &len);

        Args args_loop;

        //copia os argumentos de processClient para args
        strcpy(args_loop.buf, buf);
        args_loop.sockfd = sockfd;
        args_loop.cliaddr.sin_family = cliaddr.sin_family;
        args_loop.cliaddr.sin_port = cliaddr.sin_port;
        args_loop.cliaddr.sin_addr.s_addr = cliaddr.sin_addr.s_addr;
        args_loop.n = n;

        //processa cliente
        processClient(&args_loop);
    }
}

int main (int argc, char **argv) {
    int    sockfd, n, thread_id;
    struct sockaddr_in servaddr, cliaddr;
    char   buf[MAXDATASIZE];
    char ip[100];
    socklen_t len = sizeof(struct sockaddr);

   //numero errado de argumentos passados pela linha de comando
   if (argc != 2) {
      printf("modo de utilizacao:\n");
      printf("./servidor [porta_servidor]\n");
      exit(1);
   }
```

```
//inicializa um socket e coloca sua referencia (file descriptor) em
    listenfd
if ((sockfd = socket(AF_INET, SOCK_DGRAM, 0)) == -1) {
  perror("socket");
  exit(1);
}

//coloca o ip da interface eno1 do servidor
getServerIP(ip);
//strcpy(ip, "127.0.0.1");
printf("using ip: %s\n", ip);
inet_pton(AF_INET, ip, &servaddr.sin_addr);

//preenche com zeros a estrutura servaddr
bzero(&servaddr, sizeof(servaddr));

//informa que as mensagens trocadas no socket utilizam o IPV4
servaddr.sin_family    = AF_INET;

//informa a porta utilizada pelo socket, convertendo para big endian
    se necessario
servaddr.sin_port      = htons(atoi(argv[1]));

//associa o fluxo de dados associado ao file descriptor listenfd com
    as especificacoes em servaddr
Bind(sockfd, (struct sockaddr *)&servaddr, len);

newClientList();

// inicializa thread id
thread_id = 1;

//loop principal do servidor
for ( ; ; ) {
  bzero(buf, MAXDATASIZE);
    n = Recvfrom(sockfd, buf, MAXDATASIZE, 0, (struct sockaddr *)
        &cliaddr, &len);

    pthread_t t;

    Args args;

    //copia os argumentos de processClient para args
    strcpy(args.buf, buf);
    args.sockfd = sockfd;
    args.servport = htons(atoi(argv[1]) + thread_id);
    args.cliaddr.sin_family = cliaddr.sin_family;
    args.cliaddr.sin_port = cliaddr.sin_port;
    args.cliaddr.sin_addr.s_addr = cliaddr.sin_addr.s_addr;
    args.n = n;

    //cria a thread para processar o cliente
    //if(pthread_create(&t, NULL, processClient, &args)){
    if(pthread_create(&t, NULL, parallelClient, &args)){
```

```
            printf("Erro criando a thread.\n");
            exit(1);
        }

        thread_id++;
    }

    return(0);
}
```