client–multi–chat–securise.c

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

#define LOCALPORT   11111                   /* Le port UDP d'ecoute local */
#define DISTANTPORT 12345                   /* Le port TCP du serveur multi-chat */
#define MAXMSGLENGTH 256                    /* Taille maximale d'un message */

int main(int argc, char** argv)
{
  int sock_tcp;                             /* Socket TCP pour l'echange des messages cryptes */
  struct sockaddr_in localTCPaddr;          /* Adresse source de la socket TCP */
  struct sockaddr_in distantTCPaddr;        /* Adresse destination de la socket TCP */
  int sock_udp;                             /* Socket UDP pour l'echange de la clef de cryptage */
  struct sockaddr_in localUDPaddr;          /* Adresse source de la socket UDP */
  struct sockaddr_in distantUDPaddr;        /* Adresse destination de la socket UDP */
  fd_set surveil_fds;                       /* Ensemble des descripteurs qu'on surveille en lecture */
  fd_set read_fds;                          /* Ensemble des descripteurs qu'on va utiliser dans SELECT */
  int fdmax;                                /* Memorise le plus grand descripteur */
  unsigned char msg_crypte[MAXMSGLENGTH];   /* Le message crypte */
  unsigned char msg_clair[MAXMSGLENGTH];    /* Le message clair */
  int taille_msg;                           /* La taille du message */
  int addrlen = sizeof(struct sockaddr_in); /* Taille de la structure sockaddr_in */
  unsigned char clef;                       /* La clef de cryptage */
  int i;                                    /* variable pour boucles */

  /* Verifier que l'utilisateur a bien donne un argument au programme */
  if (argc != 2) {
     fprintf(stderr,"usage: %s adresse_ip_serveur\n", argv[0]);
     exit(1);
  }

  /* Ouverture de la socket TCP */
  if ((sock_tcp = socket( ??? )) == -1) {
     perror("Erreur creation socket");
     exit(1);
  }
  bzero(&distantTCPaddr, addrlen);
  distantTCPaddr.sin_family = ??? ;
  distantTCPaddr.sin_port = ??? ;
  if (inet_aton(argv[1],(struct in_addr *)&distantTCPaddr.sin_addr) <= 0) {
     perror("Erreur conversion adresse IP");
     close(sock_tcp);
     exit(1);
  }

  /* On ouvre la socket UDP en ecoute pour recevoir la clef de cryptage */
  if ((sock_udp = socket( ??? )) == -1) {
     perror("Erreur de creation de la socket UDP");
     close(sock_tcp);
     exit(1);
  }
  bzero(&localUDPaddr, addrlen);
  localUDPaddr.sin_family = ??? ;
  localUDPaddr.sin_port = ??? ;
  localUDPaddr.sin_addr.s_addr = ??? ;

  /* Association de la socket UDP avec l'adresse localUDPaddr */
  if (bind( ??? ) < 0) {
     perror("Erreur de bind de la socket UDP");
     exit(1);
  }

  /* Le client se connect par TCP au serveur multi chat*/
  if (connect( ??? ) < 0)  {
     perror("Erreur de connection TCP");
     close(sock_tcp);
     exit(1);
  }
```

```
75
76    /* On recoit sur la socket UDP la clef de cryptage */
77    bzero(&distantUDPaddr, addrlen);
78    if (recvfrom( ??? , (char *)&clef, 1, 0, ??? , ??? ) < 1)
79    {
80       perror("Erreur de reception de la clef de cryptage");
81       close(sock_tcp);
82       exit(1);
83    }
84
85    /* On initialize surveil_fds */
86    FD_ZERO(&surveil_fds);
87    /* On positionne sock_tcp dans l'ensemble des descripteurs a surveiller */
88    FD_SET(sock_tcp, &surveil_fds);
89    /* On positionne le descripteur de l'entree standard dans l'ensemble des descripteurs a surveiller */
90    FD_SET(fileno(stdin), &surveil_fds);
91    /* On memorise le plus grand descripteur a surveiller */
92    fdmax = (sock_tcp > fileno(stdin)) ? sock_tcp : fileno(stdin);
93
94    while (1)
95    {
96      read_fds = surveil_fds;
97      if (select(fdmax+1, &read_fds, NULL, NULL, NULL) == -1)
98      {
99         perror("Erreur select");
100        exit(1);
101     }
102     /* Si ce qui a debloque SELECT est une entree au clavier de l'utilisateur alors on va lire la */
103     /* ligne entree par l'utilisateur avant de la crypter puis de l'envoyer au serveur multi-chat */
104     if (FD_ISSET(fileno(stdin), &read_fds))
105     {
106        bzero(msg_clair, MAXMSGLENGTH);
107        fgets(msg_clair, MAXMSGLENGTH, stdin);
108        taille_msg = strlen(msg_clair);
109
110        /* Nous enlevons le caractere retour chariot du message */
111        if (msg_clair[taille_msg-1] == '\n') msg_clair[taille_msg-1] = 0;
112
113        /* On crypte le message */
114        bzero(msg_crypte, MAXMSGLENGTH);
115
116        for (i = 0; i < taille_msg; i++)
117          msg_crypte[i] = msg_clair[i] + clef + (clef % 2);
118
119        /* On envoie le message crypte */
120        if (write( ??? ) < taille_msg)
121        {
122           perror("Erreur d'envoi du message crypte");
123           close(sock_tcp);
124           exit(1);
125        }
126     }
127     /* Sinon, si ce qui a debloque SELECT est l'arrivee d'un message sur la socket TCP, alors on va */
128     /* lire le message crypte, puis on le decode avant d el'afficher a l'ecran */
129     else if (FD_ISSET(sock_tcp, &read_fds))
130          {
131             /* Lecture depuis la socket du message crypte */
132             bzero(msg_crypte, MAXMSGLENGTH);
133             if ((taille_msg = read( ??? )) <= 0)
134             {
135                if (taille_msg) perror("Erreur de reception d'un message crypte");
136                else printf("\nDeconnexion par le serveur .. bye !\n");
137                close(sock_tcp);
138                exit(1);
139             }
140
141             /* On decrypte le message */
142             bzero(msg_clair, MAXMSGLENGTH);
143             for (i = 0; i < taille_msg; i++)
144               msg_clair[i] = msg_crypte[i] - clef - (clef % 2);
145
146             /* On affiche le message en clair */
147             printf("\n%s\n",msg_clair);
148          }
149   }
150   exit(0);
151 }
```