serveur–multi–chat–securise.c

```c
/* Serveur-multi-chat-securise.c */

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

#define LOCALPORT   12345  /* Le port TCP d'ecoute */
#define DISTANTPORT 11111  /* Le port UDP destinataire de la clef de cryptage */
#define MAXMSGLENGTH 256   /* Taille maximale d'un message */
#define NBMAXCLIENTS 8     /* Le nombre maximal de clients */

/* Une structure permettant de memoriser a la fois la socket TCP et la clef de cryptage pour chaque client */
typedef struct {
  int sock;            /* La socket TCP du client */
  unsigned char clef;  /* La clef de cryptage du client */
}  Client;


int main(int argc, char **argv)
{
  int sock_tcp;                                    /* Socket TCP pour ouverture de connexion */
  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 envoyer la clef de cryptage aux clients */
  struct sockaddr_in localUDPaddr;                 /* Adresse source de la socket UDP */
  struct sockaddr_in distantUDPaddr;               /* Adresse destination de la socket UDP */
  socklen_t addrlen = sizeof(struct sockaddr_in); /* Taille de la structure sockaddr_in */
  int newfd;                                       /* Descripteur de la socket d'une nouvelle connexion */
  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 d'un client */
  unsigned char msg_clair[MAXMSGLENGTH];           /* Le message en clair d'un client */
  int taille_msg;                                  /* La taille du message */
  Client client[NBMAXCLIENTS];                     /* Liste des clients */
  int nb_clients = 0;                              /* Nombre de clients */
  int i, j, k;                                     /* Variables pour des boucles */

  /* Creation de la socket UDP pour envoyer la clef de cryptage */
  if ((sock_udp = socket( ??? )) == -1)
  {
     perror("Erreur socket UDP");
     exit(1);
  }

  /* Ouverture de la socket TCP */
  if ((sock_tcp = socket( ??? )) == -1)
  {
     perror("Erreur creation socket");
     exit(1);
  }

  bzero(&localTCPaddr, addrlen);
  localTCPaddr.sin_family = ??? ;
  localTCPaddr.sin_port = ??? ;
  localTCPaddr.sin_addr.s_addr = ??? ;

  /* Association de la socket TCP avec l'adresse localTCPaddr */
  if (bind( ??? ) == -1)
  {
     perror("Erreur bind");
     exit(1);
  }

  /* Ecoute sur la socket tcp, on autorise jusqu'a 10 requetes en attente */
  if (listen( ??? ) == -1)
  {
     perror("Erreur Listen");
```

```
75      exit(1);
76    }
77
78    /* On initialize surveil_fds */
79    FD_ZERO(&surveil_fds);
80
81    /* On positionne sock_tcp dans l'ensemble des descripteurs a surveiller */
82    FD_SET(sock_tcp, &surveil_fds);
83
84    /* On memorise le plus grand descripteur a surveiller */
85    fdmax = sock_tcp;
86    while (1)
87    {
88      /* On reactive les drapeaux des descripteurs qu'on veut surveiller */
89      read_fds = surveil_fds;
90
91      /* select attend que des donnees soient pretes a etre lues sur un des */
92      /* descripteurs de l'ensemble read_fs */
93      if (select(fdmax+1, &read_fds, NULL, NULL, NULL) == -1)
94      {
95         perror("Erreur select");
96         exit(1);
97      }
98
99      /* Nous allons parcourir l'ensemble des descripteurs pour determiner celui */
100     /* qui a debloque select */
101     for (i = 0; i <= fdmax; i++) if (FD_ISSET(i, &read_fds))
102     {
103       /* si le descripteur qui a debloque select est sock_tcp alors on */
104       /* accepte la demande de connexion TCP du nouveau client */
105       if (i == sock_tcp)
106       {
107         if ((newfd = accept( ??? )) == -1)
108           perror("Erreur accept");
109
110         /* Le nouveau client est accepte si nb_clients < NBMAXCLIENTS */
111         else if (nb_clients < NBMAXCLIENTS)
112              {
113                client[nb_clients].sock = newfd;
114
115                /* On va generer une clef de cryptage aleatoire */
116                client[nb_clients].clef = (unsigned char) (rand() % 256);
117
118                /* Initalisation de l'adresse destination de la socket UDP */
119                bzero(&localUDPaddr, addrlen);
120                distantUDPaddr.sin_family = ??? ;
121                distantUDPaddr.sin_port = ??? ;
122
123                /* l'adresse IP destination pour la socket UDP */
124                /* est celle de la nouvelle connexion TCP entrante */
125                distantUDPaddr.sin_addr = ??? ;
126
127                /* Envoi de la clef au nouveau client via la socket UDP */
128                if (sendto( ??? , (char *)&client[nb_clients].clef, 1, 0, ??? , ???) < 1)
129                {
130                  perror("Erreur sendto");
131                  close(newfd);
132                }
133
134                FD_SET(newfd, &surveil_fds);
135                if (newfd > fdmax) fdmax = newfd;
136                nb_clients++;
137              }
138              else  close(newfd);
139       }
140       /* Sinon c'est une ecriture d'un client sur la socket associee au */
141       /* descripteur i. Auquel cas, on va lire ce message et l'envoyer */
142       /* sur les sockets TCP de tous les autres clients  */
143       else
144       {
145         /* On va lire le message crypte d'un client */
146         if ((taille_msg = recv( ??? )) <= 0)
147         {
148           if (taille_msg) perror("Erreur de reception d'un message");
149           close(i);
150
151           /* Il faut enlever le client de la liste des clients */
```

```
152           for (j = 0; j < nb_clients - 1; j++)
153             if (i == client[j].sock) {
154               for (k = j; k < nb_clients - 1; k++)
155                 client[k] = client[k+1];
156               break;
157             }
158           nb_clients--;
159 
160           /* On enleve la socket du client de l'ensemble des descripteurs a surveiller */
161           FD_CLR(i, &surveil_fds);
162         }
163         else
164         {
165           /* On decrypte le message */
166           bzero(msg_clair, MAXMSGLENGTH);
167           for (j = 0; j < nb_clients; j++)
168             if (client[j].sock == i)
169             {
170               for (k = 0; k < taille_msg; k++)
171                 msg_clair[k] = msg_crypte[k] - client[j].clef - (client[j].clef % 2);
172               break;
173             }
174 
175           /* On envoie le message a tous les clients, excepte celui qui a genere le message */
176           for (j = 0; j < nb_clients; j++)
177             if (client[j].sock != i)
178             {
179               /* On crypte le message avec la clef du client destinataire */
180               bzero(msg_crypte, MAXMSGLENGTH);
181               for (k = 0; k < taille_msg; k++)
182                 msg_crypte[k] = msg_clair[k] + client[j].clef + (client[j].clef % 2);
183 
184               /* On envoie le message crypte au destinataire */
185               if (send( ??? ) < taille_msg)
186               {
187                 perror("Erreur send");
188                 close(j);
189 
190                 /* Il faut enlever le client de la liste des clients */
191                 if (j < nb_clients - 1)
192                   for (k = j; k < nb_clients - 1; k++)
193                     client[k] = client[k+1];
194                 nb_clients--;
195 
196                 /* On enleve la socket du client de l'ensemble des descripteurs a surveiller */
197                 FD_CLR(j, &surveil_fds);
198               }
199             }
200         }
201       }
202     }
203   }
204   return 0;
205 }
```