```
#include <stdio.h>#include <errno.h>#include <string.h>#include <time.h>
#include <stdlib.h>#include <unistd.h>#include <sys/types.h>#include
<signal.h>#include <sys/socket.h>#include <netdb.h>#include
<arpa/inet.h>#include "qrac.h"
void parse_cmdline(int*argc, char**argv, const char*qbase);
// table des questions/réponses
// A changer pour identfier votre serveur si vous n'avez pas d'idée:
//  carré de Q, puissance de 2 de Q, Q+1, ...
static struct Tq_r { char*q; char*r; } q_r_t[] = {
   { "Gibson"   , "Mel"     }, { "Jolie"  , "Angelina" } , { "De Niro", "Robert" },
   { "Di Caprio", "Leonardo" }, { "Roberts", "Julia"    } , { "Pitt"   , "Brad"   },
   { "Willams"  , "Robin"    }, { "Ford"   , "Harrison" } , { "Cruise" , "Tom"    },
   { "Murphy"   , "Eddie"    }, { "Depp"   , "Johnny"   } , { "Willis" , "Bruce"  }
};
static int q_r_n = sizeof(q_r_t)/sizeof(q_r_t[0]);
void gest_sigpipe(int sig)
{
   fprintf(stderr,"%s: client déconnecté\n",prgname);
}
int main(int argc, char** argv)
{
 srand(time(NULL));
 prgname = argv[0];
 int statut;

 /* vérification des arguments */
 char * qbase = "Prénom de";
 parse_cmdline(&argc,argv, qbase);
 if ( argc!=2 ) {
    fprintf(stderr,"%s:usage: %s [OPTIONS] port\n",prgname,prgname);
    fprintf(stderr,"%s:usage: %s -h   (pour obtenir les
options)\n",prgname,prgname);
    exit(1);
 }
 char* service = argv[1];
 printf("serveur: %d questions \"%s\" ; clé=0x%04x.\n",q_r_n,qbase, cle&0xffff);
 to = 250;  // time-out à 0.25 seconde
 signal( SIGPIPE, gest_sigpipe );

 /* création du SAP des clients */
 socklen_t      cltsSAPlen;
 struct sockaddr cltsSAP;
 getTCPsap(&cltsSAP,&cltsSAPlen,NULL,service);

 /* création de l'automate de connexion */
 int sock;
 if ( (sock=socket(AF_INET,SOCK_STREAM,0))==-1 ) {
  fprintf(stderr,"%s: pb socket: %s\n",argv[0],strerror(errno));
  exit(1);
 }
 if ( bind(sock,&cltsSAP,cltsSAPlen)<0 ) {
  fprintf(stderr,"%s: pb bind: %s\n",argv[0],strerror(errno));
  exit(1);
 }
 if ( listen(sock,100)!=0 ) {
   fprintf(stderr,"%s: pb listen:%s\n",argv[0],strerror(errno));
   exit(1);
 }

 while (1) {
  int cx;
  struct sockaddr cltSAP;
  socklen_t      cltSAPlen=sizeof(cltSAPlen);

  /* creation du flux de communication (cx) */
  if ( (cx=accept(sock,&cltSAP,&cltSAPlen))==-1 ) {
    fprintf(stderr,"%s: pb accept : sock=%d :
%s\n",argv[0],sock,strerror(errno));
    exit(1);
  }

  unsigned int n = rand()%1000;
  char pdu[100];
```

```
char pdu[100];
   /* dialogue: etat REPOS */
   statut = lire_PDU(pdu,cx);
   if (statut != 'H') goto fin_dialogue_inattendue ;
   gen_PDUcrq(pdu, 'C', n ^ cle);
   statut = write(cx, pdu, 3);
   /* dialogue: etat ATTS */
   statut = lire_PDU(pdu,cx);
   if (statut != 'R') goto fin_dialogue_inattendue ;
   unsigned int n_client = extrait_N_de_PDUcrq(pdu);
   printf("envoyé : %d et reçu : %d\n", n, n_client);
   if (n_client != n + 1) {
     statut = write(cx, "F", 2);
   } else {
     statut = write(cx, "O", 2);
   }
   /* dialogue: etat ACCEPT */
   statut = lire_PDU(pdu,cx);
   if (statut == 'Q') {
     int nquestion = extrait_N_de_PDUcrq(pdu);
     int idx = nquestion - 1;
     if (idx < 0 || idx >= q_r_n) {
       printf("erreur: numero de question inconnue\n");
       goto fin_dialogue_inattendue ;
     }
     char * nom = q_r_t[nquestion].r;
     gen_PDUm8(pdu, nom);
     write(cx, pdu, 10);
   } else {
     statut = write(cx, "F", 2);
     goto fin_dialogue_inattendue ;
   }

   /* dialogue: etat REPOS (FIN) */
   close(cx);
   continue;

fin_dialogue_inattendue:
   fprintf(stderr,"%s: message de type '%c' est
inattendu\n",
          prgname,statut);
fin_dialogue_erreur:
   close(cx);

  } // while (1)

  close(sock);
  return 0;
}
```

```
#ifndef FILE_QRAC_H
#define FILE_QRAC_H
#include <poll.h>
#ifndef CLE
#   define CLE 0xAA99   // valeur de la clé par défaut.
#endif

const char* prgname; // le nom du programme pour les message d'erreur.
int   silence;      // 1: ne pas afficher les mess de debug et/ou d'information
int   cle = CLE;    // la clé de cryptage/décrytage
int   to  = 0;      // timeout pour la lecture des PDU en ms (0: pas de timeout)

/*=================================================================
======*/

/**
 * Si host est non nul getTCPsap initialise *sap et *saplen pour un client
 * TCP/IPv4 sur le serveur host:port.
 * Si host est nul elle initialise *sap et *saplen pour un serveur TCP/IPv4
 * acceptant les connexions des clients *:port.
 *
 * host peut être donné en doted format ou en nom "humain" (host ou
 * host.doman).
 * port peut être un nom de service ou un entier.
 *
 * En cas d'échec, getTCPsap affiche un message d'erreur et termine le
 * processus avec un statut de 1.
**/
void getTCPsap(struct sockaddr* sap, socklen_t*saplen,
          const char *host, const char* port)
{
 int status;
 struct addrinfo hints,*found;
 memset(&hints,0,sizeof(hints));
 hints.ai_flags    = host!=0 ? 0 : AI_PASSIVE;
 hints.ai_family   = AF_INET;
 hints.ai_socktype = SOCK_STREAM;
 status=getaddrinfo(host,port,&hints,&found);
 if ( status!=0 ) {
    fprintf(stderr,"%s: pb getaddrinfo : %s\n",prgname,gai_strerror(status
    exit(1);
 }
 *sap    = *found->ai_addr;
 *saplen =  found->ai_addrlen;
 freeaddrinfo(found);

}
/**
 * Lit n octets sur le flux cx et les range dans le tampon buf.
 *
* Si la variable globale to est supérieure à zéro, la lecture est
 * abandonnée au bout de to millisecondes.
 *
 * Le tampon buf doit être alloué par l'appelant et sa taille doit
 * être d'au moins n octets.
 *
 * Retour:
 *  La fonction renvoie un nombre négatif ou nul si le timeout est
 *  atteint ou si une erreur de lecture s'est produite.
 *  Elle renvoie n si les n octets ont été lus.
**/
int lire_data(char*buf, int cx, int n)
{
   int i,statut;
   for (i=0 ; i<n ; i++) {
      if ( to>0 ) {
         struct pollfd fds = { cx, POLLIN, 0 };
         if ( (statut=poll(&fds, 1, to))<=0 ) {
           if ( ! silence )
            fprintf(stderr,"%s: pb lecture : %s\n",prgname,
               statut<0 ? strerror(errno) : "timeout");
           return -i;
         }
      }
      if ( (statut=read(cx,buf+i,1))!=1 ) {
```

```
return (N);       /* convertir base 256 en base 10*/
}

/**
 * Génère dans le tampon pdu le PDU M8 contenant le message msg.
 *
 * Si msg est trop grand, il est tronqué.
 *
 * Le tampon buf doit être alloué par l'appelant et être assez grand
 * pour contenir un PDU M8.
**/
void gen_PDUm8(char* pdu, const char* msg)
{
   pdu[0] = '8';
   int i;
   for (i = 0; msg[i] && i < 8 ; i++) {
     pdu[i + 1] = msg[i];
   }
   for ( ; i < 9 ; i++) {
     pdu[i + 1] = 0;
   }
}

/*=========================================================================*/
#endif // FILE_QRAC_H
/*=========================================================================*/
```

```
#include <stdio.h>#include <errno.h>#include <string.h>#include <stdlib.h>#include
<unistd.h>#include <sys/types.h>#include <signal.h>#include <sys/socket.h>#include
<netdb.h>#include <arpa/inet.h>#include "qrac.h"
void gest_sigpipe(int sig)

  fprintf(stderr,"%s: serveur déconnecté\n",prgname);
  exit(1);

int main(int argc, char** argv)

 prgname = argv[0];
 int statut;

 /* vérification des arguments */
 if ( argc!=4 ) {
    fprintf(stderr,"%s:usage %s serveur port nq\n",prgname,prgname);
    exit(1);
 }
 char* namesvr   = argv[1];
 char* service   = argv[2];
 int   nquestion = atoi(argv[3]);
 if ( nquestion<0 ) {
    fprintf(stderr,"%s: %d est un numéro de question invalide\n",prgname,nquestion);
    exit(1);
 }
 signal( SIGPIPE, gest_sigpipe );
* création du SAP du serveur */
 socklen_t       svrSAPlen;
 struct sockaddr svrSAP;
 getTCPsap(&svrSAP,&svrSAPlen,namesvr,service);

 /* connexion au serveur */
 int cx;
 if ( (cx=socket(AF_INET, SOCK_STREAM, 0))==-1 ) {
  fprintf(stderr,"%s: pb socket: %s\n",argv[0],strerror(errno));
  exit(1);
 }
 if ( connect(cx,&svrSAP,svrSAPlen)==-1 ) {
  fprintf(stderr,"%s: pb connect: %s\n",argv[0],strerror(errno));
  exit(1);
 }
 char pdu[100];
/* dialogue: etat REPOS */
  statut = write(cx,"H",2);

  /* dialogue: etat ATTC */
  statut = lire_PDU(pdu,cx);
  if ( statut!='C' ) goto fin_dialogue_inattendue;
  unsigned short v = extrait_N_de_PDUcrq(pdu) ^ cle;
  gen_PDUcrq(pdu,'R', v+1);
  statut = write(cx,pdu,3);

  /* dialogue: etat CHALLENGE */
  statut = lire_PDU(pdu,cx);
  if(statut == 'F'){
    fprintf(stderr, "Vous n'etes pas autorisé!\n");
    goto fin_dialogue_erreur;
  }else if (statut != 'O') goto fin_dialogue_inattendue;

  gen_PDUcrq(pdu,'Q', nquestion); /* ON envoie le numéro de la question dans l
*/
  statut = write(cx,pdu,3);


  /* dialogue: etat ACCEPT */
  statut = lire_PDU(pdu,cx);
  if (statut == 'F') {
    printf("Fin.\n");
    goto fin_dialogue_erreur;
  } else if (statut == '8') {
      printf("Réponse à la question %d: %s\n", nquestion, pdu + 1);
  }
  else{
    goto fin_dialogue_inattendue;
  }
    /* dialogue: etat REPOS (FIN) */
  close(cx);
  return 0;

fin_dialogue_inattendue:
  fprintf(stderr,"%s: message de type '%c' est inattendu\n",
        prgname,statut);
fin_dialogue_erreur:
  close(cx);
  return 1;
}
```