```c
/* INFO-F-201 : Systèmes d'exploitation I
 * Projet 2 : Le super serveur et les attaques DoS
 * Quentin Delhaye, IRCI3-T, 2012-2013
 * * * *
 * Client
 * * * *
 */
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <errno.h>
#include <netdb.h>
#include <sys/types.h>
#include <sys/socket.h>
#include <sys/wait.h>
#include <netinet/in.h>
#include <string.h>
#include <time.h>

#define PORT 42042

int getRandom(int min,int max);

int main(int argc, char *argv[])
{
    int sockfd;//socket descriptor
    char buf[100];
    int numbytes;
    struct sockaddr_in server;
    struct hostent *he;
    short int valServer, valTest, testResult;
    /* valServeur : valeur envoyé par le serveur, vaut 1, 2 ou 3.
     * valTest : valeur éventuellement générée par le client, vaut 1, 2 ou 3.
     * testResult : réponse du cliet à la requête du serveur, vaut 1 ou 0.
     */
    pid_t pid;
    short int clientStatus = -1;
    /* clientStatus donne le statut du client décrété par le serveur : -1 s'il n'est pas déterminé,
     * 0 s'il est méchant, 1 s'il gentil.
     */
    int badBoy;//Gentillesse du client déterminé par le processus père du client. 0 : gentil, 1 : méchant.

    if (argc != 2){
        fprintf(stderr,"Usage : <commande> <server>");
        return EXIT_FAILURE;
    }

    if ((he=gethostbyname(argv[1])) == NULL){
        perror("Client dit : erreur gethostbyname");
        return EXIT_FAILURE;
    }

    int k=0;
    //Boucle générant les cinq fils itérativement
    for(k=0;k<5;++k){
        srandom(time(NULL)+k);
        /* Pour avoir des nombres aléatoires indépendants d'un processus à l'autre,
         * on change le seed de srandom pour avoir une nouvelle base de génération de
         * nombres pseudo aléatoires. Le comportement sera toujours le même d'une exécution
         * à l'autre. Pour le rendre plus aléatoire, on ajoute une composante dépendant
         * du moment d'exécution: time(NULL)
         */
        badBoy = getRandom(1,2)-1;//Détermination  de la gentillesse du fils. 0 = gentil, 1 = méchant
        pid = fork();
        if ( pid == 0){//Processus fils
            if ((sockfd = socket(PF_INET,SOCK_STREAM,0)) == -1){
                perror("Client dit : erreur socket");
                return EXIT_FAILURE;
            }

            server.sin_family = AF_INET;
            server.sin_port = htons(PORT);
            server.sin_addr = *((struct in_addr*)he->h_addr);
            memset(&(server.sin_zero),'\0',8);

            if (connect(sockfd,(struct sockaddr *)&server,sizeof(struct sockaddr)) == -1){
```

```
                perror("Client dit : erreur connect");
                return EXIT_FAILURE;
            }

            while (clientStatus == -1){//Boucle d'authentification
            /* Boucle d'authentification.
             * Tant que son statut n'est pas déterminé, le client reste dans cette boucle.
             * Il reçoit une valeur du serveur, puis s'il est gentil, il génère une valeur
             * aléatoire comprise entre 1 et 3 et la compare au test du serveur.
             * Il envoie ensuite le résultat au serveur (1 si les deux valeurs correspondent, 0 sinon).
             * Dans le cas où le client est méchant, il envoie d'office 0.
             * Le client attend enfin une réponse du serveur lui indiquant son statut (-1, 0 ou 1)
             */
                if ((numbytes=recv(sockfd,&valServer,sizeof(short int),0)) == -1){
                    perror("Client dit : erreur recv");
                    return EXIT_FAILURE;
                }

                testResult = 0;//Par défaut, le résultat du test est 0
                /* Dans le cas où le client est gentil, on calcule une valeur de test,
                 * sinon on ne fait rien, "testResult" reste à sa valeur par défaut,
                 * à savoir 0.
                 */
                if (badBoy == 0){
                    valTest = getRandom(1,3);
                    valServer = ntohs(valServer);
                    testResult=(valTest==valServer)?1:0;
                }
                testResult=htons(testResult);

                //Envoi de la réponse au serveur
                if (send(sockfd,&testResult,sizeof(short int),0) == -1){
                    perror("Client : Erreur de send du résultat");
                    return EXIT_FAILURE;
                }

                //Réception du statut décrété par le serveur
                if (recv(sockfd,&clientStatus,sizeof(short int),0) == -1){
                    perror("Client : erreur recv ACK");
                    return EXIT_FAILURE;
                }
                clientStatus = ntohs(clientStatus);
            }

            //Réception du message du serveur
            if ((numbytes=recv(sockfd,buf,99,0)) == -1){
                perror("Client dit : erreur recv message");
                return EXIT_FAILURE;
            }
            buf[numbytes]='\0';
            printf("Le serveur me dit : %s\n",buf);
            close(sockfd);
            return EXIT_SUCCESS;//Fin du fils
        }
        else if (pid == -1){
            perror("Client dit : errur à la création du processus");
            return EXIT_FAILURE;
        }
    }
    return EXIT_SUCCESS;
}

int getRandom(int min,int max)
{
    return (random()%max)+min;
}
```