```c
/* 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[])
{
    //srandom(time(NULL));// Initialisation de la base de génération de nombres pseudo aléatoires
    int sockfd;
    char buf[100];
    int numbytes;
    struct sockaddr_in server;
    struct hostent *he;
    short int valServer, valTest, testResult;
    pid_t pid;
    //int status;
    short int clientStatus = -1;
    int badBoy;

    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;
    for(k=0;k<5;++k){
        //printf("Père : itération #%i\n",k);
        srandom(time(NULL)+k);//Pour avoir des nombres aléatoires indépendants d'un processus à l'autre, on cha
ngele 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, je pourrais faire un srandom(ti
me(NULL)+k).

        badBoy = getRandom(1,2)-1;
        pid = fork();
        if ( pid == 0){
            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

                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){//Changer en ternaire
                    valTest = getRandom(1,3);
                    valServer = ntohs(valServer);
                //Déclaration ternaire de TestResult
                    if (valTest == valServer){
                        testResult = 1;
                    }
                }
                testResult=htons(testResult);

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

                if (recv(sockfd,&clientStatus,sizeof(short int),0) == -1){
                    perror("Client : erreur recv ACK");
                    return EXIT_FAILURE;
                }
                clientStatus = ntohs(clientStatus);
            }

            numbytes=recv(sockfd,buf,99,0);
            buf[numbytes]='\0';
            printf("Le serveur me dit : %s\n",buf);
            close(sockfd);
            return EXIT_SUCCESS;
        }
    }
    return EXIT_SUCCESS;
}

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

```