```c
/* INFO-F-201 : Systèmes d'exploitation I
 * Projet 2 : Le super serveur et les attaques DoS
 * Quentin Delhaye, IRCI3-T, 2012-2013
 * * * *
 * Serveur
 * * * *
 */

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

#define PORT 42042
#define BACKLOG 20
#define MAX_COUNT 5
//#define __VERBOSE

int getRandom(int min,int max);

int main()
{
    int sockfd, new_fd;//socket descriptors
    struct sockaddr_in server;
    struct sockaddr_in client;
    unsigned int sin_size = sizeof(struct sockaddr_in);
    int yes = 1;
    fd_set readfds;
    pid_t pid;
    short int valTest;//Valeur de test comprise entre 1 et 3 à envoyer au client
    short int clientAnswer;//Réponse du client au test, vaut 0 ou 1 (échec ou succès)
    short int clientStatus = -1;//Statut du client : -1 (en cours), 0(refusé), 1(accepté)
    int count = 0;

    //Initialisation du socket
    if ((sockfd = socket(PF_INET,SOCK_STREAM,0)) == -1){
        perror("Serveur dit : erreur socket.");
        return EXIT_FAILURE;
    }

    if (setsockopt(sockfd,SOL_SOCKET,SO_REUSEADDR,&yes,sizeof(int)) == -1){
        perror("Serveur dit : erreur setsockopt.");
        return EXIT_FAILURE;
    }

    server.sin_family = AF_INET;
    server.sin_port = htons(PORT);
    server.sin_addr.s_addr = INADDR_ANY;
    memset(&(server.sin_zero),'\0',8);

    if (bind(sockfd,(struct sockaddr *)&server,sizeof(struct sockaddr)) == -1){
        perror("Serveur dit : erreur de bind.");
        return EXIT_FAILURE;
    }

    if (listen(sockfd,BACKLOG) == -1){
        perror("Serveur dit : erreur listen");
        return EXIT_FAILURE;
    }

    while (1){

        FD_ZERO(&readfds);// On vide l'ensemble readfs
        FD_SET(sockfd, &readfds);// On ajoute sockfd à l'ensemble readfs
        if (FD_ISSET(sockfd,&readfds)){// On vérifie qu'il en fait bien partie
            new_fd = accept(sockfd,(struct sockaddr *)&client,&sin_size);
            if (new_fd == -1){
                perror("Serveur dit : erreur accept");
```

```
                return EXIT_FAILURE;
            }

            pid = fork();
            if (pid == 0){//fils
                close(sockfd);

                #ifdef __VERBOSE
                printf("Serveur dit : connexion acceptée.\n");
                #endif

                while (count<MAX_COUNT && clientStatus!=1){
                    /* Phase de vérification du client.
                     * Le serveur détermine aléatoirement une valeur de test comprise entre 1 et 3.
                     * Il envoie cette valeur au client et attend sa réponse.
                     * Il lui envoie ensuite son statut : -1 pour indéterminé, le test continue,
                     * 0 : méchant client, statut déterminé si le serveur a envoyé MAX_COUNT tests au client,
                     * 1 : gentil, le client a répondu positivement et le serveur peut ainsi
                     * mettre fin à l'authentification.
                     */
                    ++count;
                    srandom(count);//Changement de seed
                    valTest = getRandom(1,3);
                    valTest=htons(valTest);
                    if (send(new_fd,&valTest,sizeof(short int),0) == -1){//Envoi de la valeur de test
                        perror("Serveur dit : erreur send");
                        return EXIT_FAILURE;
                    }
                    if (recv(new_fd,&clientAnswer,sizeof(short int),0) == -1){//Réception réponse du client
                        perror("Serveur : erreur de réception de la réponse du client");
                        return EXIT_FAILURE;
                    }

                    if (ntohs(clientAnswer) == 1){
                        /* Si le client répond 1, c'est gagné, on peut lui renvoyer sa propre réponse,
                         * puisque "1" représente une reconnaissance du statut "gentil".
                         */
                        clientStatus=htons(clientAnswer);
                    }
                    else if(ntohs(clientAnswer) == 0){
                        /* Si le client répond négativement, sont statut vaut soit -1, soit 0 si
                         * on arrive à la fin de la boucle,
                         * c'est-à-dire si [count-MAX_COUNT = 0]
                         */
                        clientStatus = (count-MAX_COUNT < -1)? -1 : count-MAX_COUNT;
                        clientStatus=htons(clientStatus);
                    }
                    if (send(new_fd,&clientStatus,sizeof(short int),0) == -1){
                        perror("Serveur dit : erreur de send clientStatus");
                        return EXIT_FAILURE;
                    }
                }

                if (clientStatus == 1) //Le client est gentil
                    send(new_fd,"Please make yourself at home, my friend : ]\n",45,0);
                else  //Le client est méchant
                    send(new_fd,"Leave me alone, you bad boy >: [\n",34,0);

                return EXIT_SUCCESS;//fin du fils
            }
            else if (pid == -1){
                perror("Client dit : Erreur de fork");
                return EXIT_FAILURE;
            }
        }
        close(new_fd);
    }

    return EXIT_SUCCESS;
}

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

```