

```

1 /* INFO-F-201 : Systèmes d'exploitation I
2 * Projet 2 : Le super serveur et les attaques DoS
3 * Quentin Delhayé, IRCI3-T, 2012-2013
4 * * * *
5 * Client
6 * * * *
7 */
8 #include <stdio.h>
9 #include <stdlib.h>
10 #include <unistd.h>
11 #include <errno.h>
12 #include <netdb.h>
13 #include <sys/types.h>
14 #include <sys/socket.h>
15 #include <sys/wait.h>
16 #include <netinet/in.h>
17 #include <string.h>
18 #include <time.h>
19
20 #define PORT 42042
21
22 int getRandom(int min,int max);
23
24 int main(int argc, char *argv[])
25 {
26     int sockfd;//socket descriptor
27     char buf[100];
28     int numbytes;
29     struct sockaddr_in server;
30     struct hostent *he;
31     short int valServer, valTest, testResult;
32     /* valServer : valeur envoyée par le serveur, vaut 1, 2 ou 3.
33     * valTest : valeur éventuellement générée par le client, vaut 1, 2 ou 3.
34     * testResult : réponse du client à la requête du serveur, vaut 1 ou 0.
35     */
36     pid_t pid;
37     short int clientStatus = -1;
38     /* clientStatus donne le statut du client décrété par le serveur : -1 s'il n'est pas déterminé,
39     * 0 s'il est méchant, 1 s'il gentil.
40     */
41     int badBoy;//Gentillesse du client déterminé par le processus père du client. 0 : gentil, 1 : méchant.
42
43     if (argc != 2){
44         fprintf(stderr,"Usage : <commande> <server>");
45         return EXIT_FAILURE;
46     }
47
48     if ((he=gethostbyname(argv[1])) == NULL){
49         perror("Client dit : erreur gethostbyname");
50         return EXIT_FAILURE;
51     }
52
53     int k=0;
54     //Boucle générant les cinq fils itérativement
55     for(k=0;k<5;++k){
56         srandom(time(NULL)+k);
57         /* Pour avoir des nombres aléatoires indépendants d'un processus à l'autre,
58         * on change le seed de srandom pour avoir une nouvelle base de génération de
59         * nombres pseudo aléatoires. Le comportement sera toujours le même d'une exécution
60         * à l'autre. Pour le rendre plus aléatoire, on ajoute une composante dépendant
61         * du moment d'exécution: time(NULL)
62         */
63         badBoy = getRandom(1,2)-1;//Détermination de la gentillesse du fils. 0 = gentil, 1 = méchant
64         pid = fork();
65         if ( pid == 0){//Processus fils
66             if ((sockfd = socket(PF_INET,SOCK_STREAM,0)) == -1){
67                 perror("Client dit : erreur socket");
68                 return EXIT_FAILURE;
69             }
70
71             server.sin_family = AF_INET;
72             server.sin_port = htons(PORT);
73             server.sin_addr = *((struct in_addr*)he->h_addr);
74             memset(&(server.sin_zero), '\0',8);
75
76             if (connect(sockfd,(struct sockaddr *)&server,sizeof(struct sockaddr)) == -1){

```

```

77     perror("Client dit : erreur connect");
78     return EXIT_FAILURE;
79 }
80
81 while (clientStatus == -1){//Boucle d'authentification
82 /* Boucle d'authentification.
83 * Tant que son statut n'est pas déterminé, le client reste dans cette boucle.
84 * Il reçoit une valeur du serveur, puis s'il est gentil, il génère une valeur
85 * aléatoire comprise entre 1 et 3 et la compare au test du serveur.
86 * Il envoie ensuite le résultat au serveur (1 si les deux valeurs correspondent, 0 sinon).
87 * Dans le cas où le client est méchant, il envoie d'office 0.
88 * Le client attend enfin une réponse du serveur lui indiquant son statut (-1, 0 ou 1)
89 */
90     if ((numbytes=recv(sockfd,&valServer,sizeof(short int),0)) == -1){
91         perror("Client dit : erreur recv");
92         return EXIT_FAILURE;
93     }
94
95     testResult = 0;//Par défaut, le résultat du test est 0
96     /* Dans le cas où le client est gentil, on calcule une valeur de test,
97     * sinon on ne fait rien, "testResult" reste à sa valeur par défaut,
98     * à savoir 0.
99     */
100    if (badBoy == 0){
101        valTest = getRandom(1,3);
102        valServer = ntohs(valServer);
103        testResult=(valTest==valServer)?1:0;
104    }
105    testResult=htons(testResult);
106
107    //Envoi de la réponse au serveur
108    if (send(sockfd,&testResult,sizeof(short int),0) == -1){
109        perror("Client : Erreur de send du résultat");
110        return EXIT_FAILURE;
111    }
112
113    //Réception du statut décrété par le serveur
114    if (recv(sockfd,&clientStatus,sizeof(short int),0) == -1){
115        perror("Client : erreur recv ACK");
116        return EXIT_FAILURE;
117    }
118    clientStatus = ntohs(clientStatus);
119 }
120
121 //Réception du message du serveur
122 if ((numbytes=recv(sockfd,buf,99,0)) == -1){
123     perror("Client dit : erreur recv message");
124     return EXIT_FAILURE;
125 }
126 buf[numbytes]='\0';
127 printf("Le serveur me dit : %s\n",buf);
128 close(sockfd);
129 return EXIT_SUCCESS;//Fin du fils
130 }
131 else if (pid == -1){
132     perror("Client dit : erreur à la création du processus");
133     return EXIT_FAILURE;
134 }
135 }
136 return EXIT_SUCCESS;
137 }
138
139 int getRandom(int min,int max)
140 {
141     return (random()%max)+min;
142 }
143

```

```

1 /* INFO-F-201 : Systèmes d'exploitation I
2 * Projet 2 : Le super serveur et les attaques DoS
3 * Quentin Delhayé, IRCI3-T, 2012-2013
4 * * * *
5 * Serveur
6 * * * *
7 */
8
9 #include <stdio.h>
10 #include <stdlib.h>
11 #include <unistd.h>
12 #include <errno.h>
13 #include <sys/types.h>
14 #include <sys/socket.h>
15 #include <netinet/in.h>
16 #include <arpa/inet.h>
17 #include <sys/wait.h>
18 #include <netdb.h>
19 #include <string.h>
20 #include <time.h>
21
22 #define PORT 42042
23 #define BACKLOG 20
24 #define MAX_COUNT 5
25 //#define __VERBOSE
26
27 int getRandom(int min,int max);
28
29 int main()
30 {
31     int sockfd, new_fd;//socket descriptors
32     struct sockaddr_in server;
33     struct sockaddr_in client;
34     unsigned int sin_size = sizeof(struct sockaddr_in);
35     int yes = 1;
36     fd_set readfds;
37     pid_t pid;
38     short int valTest;//Valeur de test comprise entre 1 et 3 à envoyer au client
39     short int clientAnswer;//Réponse du client au test, vaut 0 ou 1 (échec ou succès)
40     short int clientStatus = -1;//Statut du client : -1 (en cours), 0(refusé), 1(accepté)
41     int count = 0;
42
43     //Initialisation du socket
44     if ((sockfd = socket(PF_INET,SOCK_STREAM,0)) == -1){
45         perror("Serveur dit : erreur socket.");
46         return EXIT_FAILURE;
47     }
48
49     if (setsockopt(sockfd,SOL_SOCKET,SO_REUSEADDR,&yes,sizeof(int)) == -1){
50         perror("Serveur dit : erreur setsockopt.");
51         return EXIT_FAILURE;
52     }
53
54     server.sin_family = AF_INET;
55     server.sin_port = htons(PORT);
56     server.sin_addr.s_addr = INADDR_ANY;
57     memset(&(server.sin_zero),'\0',8);
58
59     if (bind(sockfd,(struct sockaddr *)&server,sizeof(struct sockaddr)) == -1){
60         perror("Serveur dit : erreur de bind.");
61         return EXIT_FAILURE;
62     }
63
64     if (listen(sockfd,BACKLOG) == -1){
65         perror("Serveur dit : erreur listen");
66         return EXIT_FAILURE;
67     }
68
69     while (1){
70
71         FD_ZERO(&readfds);// On vide l'ensemble readfs
72         FD_SET(sockfd, &readfds);// On ajoute sockfd à l'ensemble readfs
73         if (FD_ISSET(sockfd,&readfds)){// On vérifie qu'il en fait bien partie
74             new_fd = accept(sockfd,(struct sockaddr *)&client,&sin_size);
75             if (new_fd == -1){
76                 perror("Serveur dit : erreur accept");

```

```

77     return EXIT_FAILURE;
78 }
79
80 pid = fork();
81 if (pid == 0){//fils
82     close(sockfd);
83
84     #ifdef __VERBOSE
85     printf("Serveur dit : connexion acceptée.\n");
86     #endif
87
88     while (count<MAX_COUNT && clientStatus!=1){
89         /* Phase de vérification du client.
90          * Le serveur détermine aléatoirement une valeur de test comprise entre 1 et 3.
91          * Il envoie cette valeur au client et attend sa réponse.
92          * Il lui envoie ensuite son statut : -1 pour indéterminé, le test continue,
93          * 0 : méchant client, statut déterminé si le serveur a envoyé MAX_COUNT tests au client,
94          * 1 : gentil, le client a répondu positivement et le serveur peut ainsi
95          * mettre fin à l'authentification.
96          */
97         ++count;
98         srand(count);//Changement de seed
99         valTest = getRandom(1,3);
100        valTest=htons(valTest);
101        if (send(new_fd,&valTest,sizeof(short int),0) == -1){//Envoi de la valeur de test
102            perror("Serveur dit : erreur send");
103            return EXIT_FAILURE;
104        }
105        if (recv(new_fd,&clientAnswer,sizeof(short int),0) == -1){//Réception réponse du client
106            perror("Serveur : erreur de réception de la réponse du client");
107            return EXIT_FAILURE;
108        }
109
110        if (ntohs(clientAnswer) == 1){
111            /* Si le client répond 1, c'est gagné, on peut lui renvoyer sa propre réponse,
112             * puisque "1" représente une reconnaissance du statut "gentil".
113             */
114            clientStatus=htons(clientAnswer);
115        }
116        else if(ntohs(clientAnswer) == 0){
117            /* Si le client répond négativement, son statut vaut soit -1, soit 0 si
118             * on arrive à la fin de la boucle,
119             * c'est-à-dire si [count-MAX_COUNT = 0]
120             */
121            clientStatus = (count-MAX_COUNT < -1)? -1 : count-MAX_COUNT;
122            clientStatus=htons(clientStatus);
123        }
124        if (send(new_fd,&clientStatus,sizeof(short int),0) == -1){
125            perror("Serveur dit : erreur de send clientStatus");
126            return EXIT_FAILURE;
127        }
128    }
129
130    if (clientStatus == 1) //Le client est gentil
131        send(new_fd,"Please make yourself at home, my friend : ]\n",45,0);
132    else //Le client est méchant
133        send(new_fd,"Leave me alone, you bad boy >: [\n",34,0);
134
135    return EXIT_SUCCESS;//fin du fils
136 }
137 else if (pid == -1){
138     perror("Client dit : Erreur de fork");
139     return EXIT_FAILURE;
140 }
141 }
142 close(new_fd);
143 }
144
145 return EXIT_SUCCESS;
146 }
147
148 int getRandom(int min,int max)
149 {
150     return (rand()%max)+min;
151 }
152

```

```
1 # INFO-F-201 : Systèmes d'exploitation I
2 # Projet 2 : Le super serveur et les attaques DoS
3 # Quentin Delhayé, IRCI3-T, 2012-2013
4
5 TARGET=client serveur
6
7 normal: $(TARGET)
8
9 client: client.c
10     gcc -Wall client.c -o client
11
12 serveur: serveur.c
13     gcc -Wall serveur.c -o serveur
14
15 clean:
16     $(RM) $(TARGET)
17
```



