

# Protocoles IP - TP5

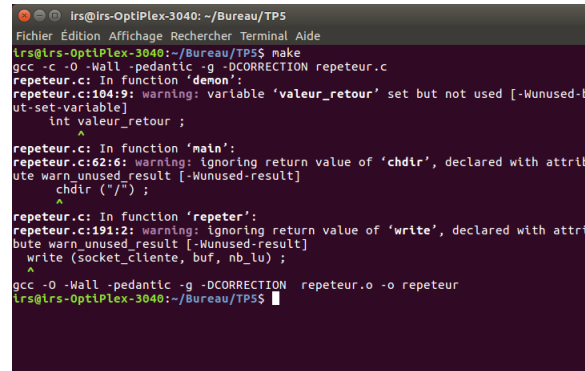
CAUMES Clément  
LAMMAMRA Aicha  
MERIMI Mehdi  
RAMAROSON Andritsalama

Master 1 Informatique - Master 1 CHPS

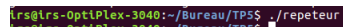
# 1 Exercice 1

On utilise un hub pour connecter les 4 PCs. On utilise les adresses IP 192.168.1.1, 192.168.1.2, 192.168.1.3 et 192.168.1.4.

Pour exécuter ce programme, 192.168.1.1 compile et exécute le programme :

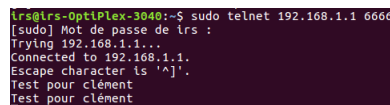


```
lrs@lrs-OptiPlex-3040: ~/Bureau/TP5
Fichier Edition Affichage Rechercher Terminal Aide
lrs@lrs-OptiPlex-3040:~/Bureau/TP5$ make
gcc -c -O -Wall -pedantic -g -DCORRECTION repeteur.c
repeteur.c: In function 'demon':
repeteur.c:104:9: warning: variable 'valeur_retour' set but not used [-Wunused-but-set-variable]
    int valeur_retour ;
    ^
repeteur.c: In function 'main':
repeteur.c:62:6: warning: ignoring return value of 'chdir', declared with attribute warn_unused_result [-Wunused-result]
    chdir ("/");
    ^
repeteur.c: In function 'repeteur':
repeteur.c:191:2: warning: ignoring return value of 'write', declared with attribute warn_unused_result [-Wunused-result]
    write (socket_client, buf, nb_lu) ;
    ^
gcc -O -Wall -pedantic -g -DCORRECTION repeteur.o -o repeteur
lrs@lrs-OptiPlex-3040:~/Bureau/TP5$
```



```
lrs@lrs-OptiPlex-3040:~/Bureau/TP5$ ./repeteur
```

Tandis que 192.168.1.2 fait la connexion à 192.168.1.1 sur le port 6666 :



```
lrs@lrs-OptiPlex-3040:~$ sudo telnet 192.168.1.1 6666
[sudo] Mot de passe de lrs :
Trying 192.168.1.1...
Connected to 192.168.1.1.
Escape character is '^['.
Test pour clément
Test pour clément
```

On voit que le programme répète ce que le client écrit sur le port 6666.

## 2 Exercice 2

Pour détecter que l'utilisateur tape "stop", on modifie la fonction repeter. On compare la chaîne tapée avec "stop". Si c'est le cas, on écrit dans le fichier "var/log/local0.log" l'adresse IP du client :

```
irs@irs-OptiPlex-3040:~$ sudo telnet 192.168.1.1 6666
Trying 192.168.1.1...
Connected to 192.168.1.1.
Escape character is '^]'.
coucou
coucou
stop
Connection closed by foreign host.
```

Côté serveur, on modifie le fichier "/etc/syslog.conf" :

```
GNU nano 2.5.3 Fichier : syslog.conf
# /etc/syslog.conf Configuration file for inetutils-syslogd.
# For more information see syslog.conf(5) manpage.
#
# First some standard logfiles. Log by facility.
#
auth,authpriv.* /var/log/auth.log
*.;auth,authpriv.none -/var/log/syslog
#cron.* /var/log/cron.log
daemon.* -/var/log/daemon.log
kern.* -/var/log/kern.log
lpr.* -/var/log/lpr.log
mail.* -/var/log/mail.log
user.* -/var/log/user.log
uucp.* -/var/log/uucp.log
local0.* /var/log/local0.log
```

On peut voir dans le fichier "var/log/local0.log" dès que le client se déconnecte :

```
GNU nano 2.5.3 Fichier : local0.log
Apr 19 11:38:07 irs-OptiPlex-3040 repeater[5901]: deconnexion
Apr 19 11:53:25 irs-OptiPlex-3040 repeater[6094]: Deconnexion
```

Le code source est le suivant :

```
1 /*
2  * Fichier 'repeteur.c' utilise pour le TD 5 de programmation IP.
3  * Le programme 'repeteur' est un programme demon "repetant" tout
4  * ce qu'on lui dit. Il tourne sur le port 6666 et reste a l'ecoute
5  * d'une eventuelle connexion.
6  *
7  * Historique
8  *    1999/10/15 : dntt : creation
9  */
10
11 #include <unistd.h>
12 #include <stdlib.h>
13 #include <sys/stat.h>
14 #include <string.h>
15 #include <stdio.h>
16
17 #include <sys/types.h>
18 #include <sys/socket.h>
19 #include <sys/wait.h>
20 #include <netinet/in.h>
```

```

21 #include <netdb.h>
22 #include <syslog.h>
23 #include <arpa/inet.h>
24
25 #include <signal.h>
26
27 #define MAXLEN 1024
28 #define PORT_REPETEUR 6666
29 #define NB_CONN_MAX 5
30
31 void demon () ;
32
33 void repeter (int socket_client, struct sockaddr_in sock_addr) ;
34 int lecture_socket (int socket_client, char *buf) ;
35
36
37 /*****
38  *
39  * Fonction principale :
40  * On cree le demon en dupliquant le processus afin de detacher le
41  * programme du processus pere
42  *
43  *****/
44 int main ()
45 {
46     switch (fork ())
47     {
48         case -1 :
49             perror ("*** erreur : fork impossible : demon non cree") ;
50             exit (1) ;
51
52         case 0 :
53             /*
54              * Ce processus fils est celui qui deviendra le demon.
55              * Ingredients pour en faire un vrai et bon demons :
56              */
57             /* 1 - On cree une nouvelle session. Pas de terminal de controle */
58             setsid () ;
59
60             /* 2 - le repertoire courant est change a la racine afin de ne pas
61              * causer de probleme lors de demontage eventuel de partition
62              */
63             chdir ("/") ;
64
65             /* 3 - Le masque sur les droits lors de creation de fichiers */
66             umask (0) ;
67
68             /* 4 - Comme c'est un demon qui est lance et qui est "detache" du
69              * term, l'entree, la sortie et l'erreur standard n'ont pas de
70              * terminal ou s'afficher. On peut donc les fermer.
71              */
72             //close (0) ;
73             //close (1) ;
74             //close (2) ;
75
76

```

```

77      /* 6 - On lance le demon proprement dit */
78      demon () ;
79
80      /* 7 - Si le demon s'arrete, alors on sort du processus fils. Le
81      *      programme est alors termine. */
82      exit (1) ;
83
84      default :
85      /* Le processus pere est tue. On rend la main au shell appelant. */
86      exit (0) ;
87  }
88  return 0 ;
89 }
90
91  /* *****
92  *
93  * Le programme 'demon'
94  *
95  * Structure
96  *      sockaddr_in : /usr/include/netinet/in.h
97  *
98  *
99  *
100  ***** */
101 void demon ()
102 {
103     int socket_ecoute ;
104     int socket_client ;
105     int valeur_retour ;
106     int salong ;
107     int opt = 1 ;
108     struct sockaddr_in monadr, sonadr ;
109     int status ;
110
111
112     /*
113     * Creation de la socket d'ecoute
114     */
115     socket_ecoute = socket (PF_INET, SOCK_STREAM, 0) ;
116
117     /*
118     * Modification des options associees a la socket d'ecoute.
119     */
120     setsockopt (socket_ecoute, SOL_SOCKET, SO_REUSEADDR, (char *) &opt, \
121                sizeof (opt)) ;
122
123     /*LOG_INFO
124     * Initialisation des parametres internet de la socket d'ecoute.
125     */
126     bzero ((char *) &monadr, sizeof monadr) ;
127     monadr.sin_family = AF_INET ;
128     monadr.sin_port = htons (PORT_REPETEUR) ;
129     monadr.sin_addr.s_addr = INADDR_ANY ;
130     valeur_retour = bind (socket_ecoute, (struct sockaddr *) &monadr, \
131                          (int) sizeof monadr) ;
132

```

```

133  /*
134  * Socket mise en position d'ecoute ('listen'). On specifie aussi le
135  * nombre de connexions simultanees acceptees.
136  */
137  valeur_retour = listen (socket_ecoute, NB_CONN_MAX) ;
138
139  /**
140  * Le demon tourne indefiniment a l'ecoute d'eventuelles connections
141  * 'accept'
142  * Lorsqu'un client se connecte, une socket est cree (socket_client).
143  * on duplique le processus et le processus fils prend en charge
144  * cette socket par la fonction 'repetier'.
145  */
146  while (1)
147  {
148      salong = sizeof sonadr ;
149
150      socket_client = accept (socket_ecoute, (struct sockaddr *) &sonadr, \
151                             (socklen_t *) &salong) ;
152
153      switch (fork ())
154      {
155          case -1 :
156              break ;
157
158          case 0 :
159              repeter (socket_client, sonadr) ;
160              close (socket_client) ;
161              kill (getpid(),SIGTERM);
162
163          default :
164              waitpid (-1, &status, WNOHANG);
165              close (socket_client) ;
166      }
167  }
168 }
169
170  /*****
171  *
172  * La fonction qui repete
173  *
174  *
175  *
176  *****/
177  void repeter (int socket_client, struct sockaddr_in sock_addr)
178  {
179      int nb_lu ;
180      char buf [MAXLEN] ;
181      char* addr;
182
183      addr=inet_ntoa(sock_addr.sin_addr);
184
185      while ((nb_lu = lecture_socket (socket_client, buf)) > 0)
186      {
187          // Si probleme de lecture sur la socket (fermee brutalement
188          // par exemple)

```

```

189     if (nb_lu == -1 || nb_lu == 0)
190     {
191         return ;
192     }
193     if (strncmp("stop\r\n", buf, 6)==0){
194         openlog("repeater", LOG_PID | LOG_CONS, LOG_LOCAL0);
195         syslog(LOG_INFO, "%s", addr);
196         closelog();
197         kill(getpid(), SIGTERM);
198     }
199
200
201     // On repete ce qui a ete dit et on renvoie au client
202     write (socket_client, buf, nb_lu) ;
203 }
204 }
205
206 int lecture_socket (int socket_client, char *buf)
207 {
208     int nb_lu ;
209
210     nb_lu = read (socket_client, buf, MAXLEN) ;
211     return (nb_lu) ;
212 }

```

Listing 1 – repeteur.c

### 3 Exercice 3

Le programme suivant va lire ligne par ligne le fichier et l'envoyer aux clients. Pour le tester, on va créer un fichier "test.txt" coté serveur :



Ensuite, on va le remplir :

```
test 1  
test 2  
test 3
```

Puis, on va compiler et exécuter le programme avec :

```
lrs@lrs-OptiPlex-3040:~/Bureau/tp5_client$ ./parleur 192.168.1.2 22 test.txt  
j'ai envoye : test 1  
  
entre temps :  
j'ai reçu : SSH-2.0-OpenSSH_7.2p2 Ubuntu-4ubuntu2.2  
Protocol mismatch.  
  
j'ai envoye : test 2  
  
entre temps :  
j'ai reçu :  
j'ai envoye : test 3  
  
entre temps :  
j'ai reçu :
```

Côté client, on vérifie bien que l'utilisateur a reçu :

Wireshark capture showing network traffic between 192.168.1.1 and 192.168.1.2. The capture includes ICMP echo requests and replies, and an SSH connection attempt. The SSH attempt fails with a 'Protocol mismatch' error. The packet list shows the SSH handshake and the encrypted packet. The packet details show the SSH version and the encrypted packet. The packet bytes show the raw data of the encrypted packet.



Concernant le code source, il a été fourni sur e-campus :

```
1 //
2 // usage:
3 //
4 //
5 //
6 // ip : ip du serveur "repeteur"
7 // port : port du serveur
8 // nomfichier: nom du fichier a envoyer
9 /*****
10 *****/
11 #include <unistd.h>
12 #include <stdlib.h>
13 #include <sys/stat.h>
14 #include <string.h>
15 #include <stdio.h>
16 #include <sys/types.h>
17 #include <sys/socket.h>
18 #include <sys/wait.h>
19 #include <netinet/in.h>
20 #include <netdb.h>
21 #include <syslog.h>
22 #include <signal.h>
23 // #include <error.h>
24 #define MAXLEN 1024
25 #define PORT_REPETEUR 6666
26 #define NB_CONN_MAX 5
27 int main(int argc, char **argv){
28
29     FILE *fp;
30     int sockfd, portno, n;
31     struct sockaddr_in serv_addr;
32     struct hostent *server;
33
34     /* ouverture fichier */
35     fp=fopen(argv[3], "r");
36
37     /* ouverture socket */
38     portno = atoi(argv[2]);
39     sockfd = socket(AF_INET, SOCK_STREAM, 0);
40     if (sockfd < 0)
41         fprintf(stderr, "ERROR opening socket\n");
42
43     //recuperer l'ip via le premier argument
44     server = gethostbyname(argv[1]);
45
46     if (server == NULL) {
47         fprintf(stderr, "ERROR, no such host\n");
48         exit(0);
49     }
50
51     // initialisation a zero du buffer
52     bzero((char *) &serv_addr, sizeof(serv_addr));
53     // remplir ip et port
54
```

```

55     serv_addr.sin_family = AF_INET;
56     bcopy((char *)server->h_addr, (char *)&serv_addr.sin_addr.s_addr, server->
h_length);
57     serv_addr.sin_port = htons(portno);
58
59     // socket connect
60     if (connect(sockfd, (struct sockaddr *)&serv_addr, sizeof(serv_addr)) < 0)
61         fprintf(stderr, "ERROR connecting\n");
62     if (fp != NULL)
63     {
64         char line [ 128 ];
65
66         /* or other
67          suitable maximum line size */
68         // boucle de lecture de lignes
69
70         while ( fgets ( line , sizeof line , fp ) != NULL ) /* read a line */
71         {
72             // ecriture dans le socket
73             n = write(sockfd, line, strlen(line));
74             if (n < 0)
75                 fprintf(stderr, "ERROR writing to socket\n");
76             printf(" j'ai envoye : %s \n", line);
77             bzero(line, 128);
78             printf("entre temps : %s \n", line);
79             // lecture du socket
80             n = read(sockfd, line, 128);
81             if (n < 0)
82                 fprintf(stderr, "ERROR reading from socket\n");
83             printf(" j'ai reçu : %s \n", line);
84             // printf("ligne : %s", line);
85         }
86         fclose ( fp);
87     }
88     return 0;
89 }

```

Listing 2 – parleur.c

## 4 Exercice 4

Pour tracer l'interruption de l'administrateur, on crée une nouvelle fonction "void signal\_administrateur(int signal)". En effet, quand l'administrateur tue le processus "repeteur", le client se déconnecte automatiquement :

```
lrs@lrs-OptiPlex-3040:~/Bureau/tp5$ sudo telnet 192.168.1.1 6666
[sudo] mot de passe de lrs :
Trying 192.168.1.1...
Connected to 192.168.1.1.
Escape character is '^]'.
C
C
Connection closed by foreign host.
```

Le serveur reçoit sur "var/log/local0.log" :

```
Apr 26 10:56:43 lrs-OptiPlex-3040 repeater[6597]: 192.168.1.2
Apr 26 10:56:43 lrs-OptiPlex-3040 repeater: interruption de l'administrateur
```

Le code source est le suivant, on crée une fonction signal\_administrateur qui sera appelé au début du programme et qui sera appelée quand l'administrateur arrêtera le programme.

```
1  /*
2  * Fichier 'repeteur.c' utilise pour le TD 5 de programmation IP.
3  * Le programme 'repeteur' est un programme demon "repetant" tout
4  * ce qu'on lui dit. Il tourne sur le port 6666 et reste l'ecoute
5  * d'une eventuelle connexion.
6  *
7  * Historique
8  *   1999/10/15 : dntt : creation
9  */
10
11 #include <unistd.h>
12 #include <stdlib.h>
13 #include <sys/stat.h>
14 #include <string.h>
15 #include <stdio.h>
16
17 #include <sys/types.h>
18 #include <sys/socket.h>
19 #include <sys/wait.h>
20 #include <netinet/in.h>
21 #include <netdb.h>
22 #include <syslog.h>
23 #include <arpa/inet.h>
24
25 #include <signal.h>
26
27 #define MAXLEN 1024
28 #define PORT_REPETEUR 6666
29 #define NB_CONN_MAX 5
30
31 void demon () ;
32 void signal_administrateur(int signal);
33 void repeter (int socket_client, struct sockaddr_in sock_addr) ;
34 int lecture_socket (int socket_client, char *buf) ;
35
```

```

36
37  /*****
38  *
39  * Fonction principale :
40  * On cree le demon en dupliquant le processus afin de detacher le
41  * programme du processus pere
42  *
43  *****/
44  int main ()
45  {
46
47      signal (SIGTERM, signal_administrateur);
48
49      switch (fork ())
50      {
51          case -1 :
52              perror ("*** erreur : fork impossible : demon non cree") ;
53              exit (1) ;
54
55          case 0 :
56              /*
57               * Ce processus fils est celui qui deviendra le demon.
58               * Ingredients pour en faire un vrai et bon demons :
59               */
60              /* 1 - On cree une nouvelle session. Pas de terminal de controle */
61              setsid () ;
62
63              /* 2 - le repertoire courant est change a la racine afin de ne pas
64               *      causer de probleme lors de demontage eventuel de partition
65               */
66              chdir ("/") ;
67
68              /* 3 - Le masque sur les droits lors de creation de fichiers */
69              umask (0) ;
70
71              /* 4 - Comme c'est un demon qui est lance et qui est "detache" du
72               *      term, l'entree, la sortie et l'erreur standard n'ont pas de
73               *      terminal ou s'afficher. On peut donc les fermer.
74               */
75              //close (0) ;
76              //close (1) ;
77              //close (2) ;
78
79
80              /* 6 - On lance le demon proprement dit */
81              demon () ;
82
83              /* 7 - Si le demon s'arrete, alors on sort du processus fils. Le
84               *      programme est alors termine. */
85              exit (1) ;
86
87          default :
88              /* Le processus pere est tue. On rend la main au shell appelant. */
89              exit (0) ;
90      }
91      return 0 ;

```

```

92 }
93
94 /*****
95 *
96 * signal_administrateur qui laisse un message quand l'administrateur
97 * arrete le programme.
98 *
99 *
100 *
101 *
102 *****/
103 void signal_administrateur(int signal){
104     openlog("repeateur", 0, LOG_LOCAL0);
105     syslog(LOG_INFO, "interruption de l'administrateur");
106     closelog();
107     exit(0);
108 }
109
110 /*****
111 *
112 * Le programme 'demon'
113 *
114 * Structure
115 *      sockaddr_in : /usr/include/netinet/in.h
116 *
117 *
118 *
119 *****/
120 void demon ()
121 {
122     int socket_ecoute ;
123     int socket_client ;
124     int valeur_retour ;
125     int salong ;
126     int opt = 1 ;
127     struct sockaddr_in monadr, sonadr ;
128     int status ;
129
130
131     /*
132     * Creation de la socket d'ecoute
133     */
134     socket_ecoute = socket (PF_INET, SOCK_STREAM, 0) ;
135
136     /*
137     * Modification des options associees a la socket d'ecoute.
138     */
139     setsockopt (socket_ecoute, SOL_SOCKET, SO_REUSEADDR, (char *) &opt, \
140         sizeof (opt)) ;
141
142     /*LOG_INFO
143     * Initialisation des parametres internet de la socket d'ecoute.
144     */
145     bzero ((char *) &monadr, sizeof monadr) ;
146     monadr.sin_family = AF_INET ;
147     monadr.sin_port = htons (PORT_REPETEUR) ;

```

```

148 monadr.sin_addr.s_addr = INADDR_ANY ;
149 valeur_retour = bind (socket_ecoute, (struct sockaddr *) &monadr, \
150                      (int) sizeof monadr) ;
151
152 /*
153  * Socket mise en position d'ecoute ('listen'). On specifie aussi le
154  * nombre de connexions simultanees acceptees.
155  */
156 valeur_retour = listen (socket_ecoute, NB_CONN_MAX) ;
157
158 /**
159  * Le demon tourne indefiniment a l'ecoute d'eventuelles connections
160  * 'accept'
161  * Lorsqu'un client se connecte, une socket est cree (socket_client).
162  * on duplique le processus et le processus fils prend en charge
163  * cette socket par la fonction 'repete'.
164  */
165 while (1)
166 {
167     salong = sizeof sonadr ;
168
169     socket_client = accept (socket_ecoute, (struct sockaddr *) &sonadr, \
170                           (socklen_t *) &salong) ;
171
172     switch (fork ())
173     {
174         case -1 :
175             break ;
176
177         case 0 :
178             repete (socket_client, sonadr) ;
179             close (socket_client) ;
180             kill (getpid(), SIGTERM);
181
182         default :
183             waitpid (-1, &status, WNOHANG);
184             close (socket_client) ;
185     }
186 }
187 }
188
189 /*****
190  *
191  * La fonction qui repete
192  *
193  *
194  *
195  *****/
196 void repete (int socket_client, struct sockaddr_in sock_addr)
197 {
198     int nb_lu ;
199     char buf [MAXLEN] ;
200     char* addr;
201
202     addr=inet_ntoa(sock_addr.sin_addr);
203

```

```

204 while ((nb_lu = lecture_socket (socket_client, buf)) > 0)
205 {
206     // Si probleme de lecture sur la socket (fermee brutalement
207     // par exemple)
208     if (nb_lu == -1 || nb_lu == 0)
209     {
210         return ;
211     }
212     if (strncmp("stop\r\n", buf, 6)==0){
213         openlog("repeater", LOG_PID | LOG_CONS, LOG_LOCAL0);
214         syslog(LOG_INFO, "%s", addr);
215         closelog();
216         kill(getpid(), SIGTERM);
217     }
218
219     // On repete ce qui a ete dit et on renvoie au client
220     write (socket_client, buf, nb_lu) ;
221 }
222
223 }
224
225 int lecture_socket (int socket_client, char *buf)
226 {
227     int nb_lu ;
228
229     nb_lu = read (socket_client, buf, MAXLEN) ;
230     return (nb_lu) ;
231 }

```

Listing 3 – repeteur.c