Functions

pipingchild	1:54
sigint_handler	1:46
sigusr1_handler	1:40


```
1   /*****************************************************************
2
3    SERVER su socket di tipo STREAM
4    per compito del 4/9/02
5
6    (come cliente utilizzare telnet)
7
8    *****************************************************************/
9
10
11  #include <sys/types.h>
12
13  #include <signal.h>
14
15  #include <sys/socket.h>
16  #include <netinet/in.h>
17  #include <netdb.h>
18  #include <stdio.h>
19  #include <sys/time.h>
20  #include <string.h>
21  #include <errno.h>
22
23
24
25
26  #define DEFAULTPORT 10101
27  #define DEFAULTTIMEOUT 3660
28
29
30  #define FAILAUTHMESG  "You are outside my network and cannot be authorized to this se
    rvice\n"
31  #define FAILAUTHLEN strlen(FAILAUTHMESG)
32
33  sigset_t sigmask, oldmask,zeromask;
34  int nreq;
35  int pid1,pid2;
36
37  int sock;
38
39
40  void sigusr1_handler()
41  {
42    printf("%d requests received so far\n",nreq);
43  }
44
45
46  void sigint_handler()
47  {
48    printf("Whoops, got SIGINT: process exits\n");
49    shutdown(sock,2);
50    exit(0);
51  }
52
53
54  pipingchild(int msgsock)
55  {
56    int i,ns,piped[2];
57    char *com1[32];
58    char *com2[32];
59    char command1[1024];
60    char command2[1024];
61    char *p;
62    sigset_t  sigmask;
63
64  sigemptyset( &sigmask);
65  sigaddset(&sigmask, SIGINT);
66
67  sigprocmask(SIG_BLOCK, &sigmask, &oldmask);
68
69   if (pipe(piped) < 0)
70     return 2;
71
72   read(msgsock,command1,1024);
73   read(msgsock,command2,1024);
74
```

Functions

main	2:147
pipingchild	1:54
sigint_handler	1:46
sigusr1_handler	1:40

```
75
76      // Per semplicita' non viene gestita l'estrazione degli argomenti
77
78      p=command1;
79      while(*p != '\r') p++;
80      *p=0;
81
82      p=command2;
83      while(*p != '\r') p++;
84      *p=0;
85
86      com1[0]=command1;
87      com1[1]=NULL;
88      com2[0]=command2;
89      com2[1]=NULL;
90
91      if((pid2=fork())==0)
92        {
93          /* Lo stdin corrente non interessa a com2 (lettore): chiude il descrittore 0 */
94          close(0);
95          /* Si richiede un nuovo descrittore per la lettura dalla pipe:
96           il primo descrittore libero e' lo 0 (stdin) che viene
97           associato alla lettura dalla pipe */
98          dup(piped[0]);
99
100         /* Gli altri descrittori delle pipe non servono piu':
101          il lettore usa stdin per leggere dalla pipe */
102         close(piped[0]);
103         close(piped[1]);
104
105         /* Esecuzione del comando 2 (che ha la pipe come stdin)  */
106
107         //     printf("Executing %s (%s)\n",com2[0],command2);
108         execvp(com2[0], com2);
109         perror("exec com2");
110         return 4;
111       }
112     else
113       if((pid1=fork())==0)
114           {
115             /* Lo stdout corrente non interessa a com1 (scrittore): chiude il descrittor
    e 1 */
116             close(1);
117         /* Si richiede un nuovo descrittore per la scrittura sulla pipe:
118          il primo descrittore libero e' 1 (stdout) che viene
119          associato alla scrittura sulla pipe */
120             dup(piped[1]);
121
122             /* Gli altri descrittori delle pipe non servono piu':
123              lo scrittore usa stdout per scrivere sulla pipe */
124             close(piped[0]);
125             close(piped[1]);
126
127             /* Esecuzione del comando 1 (che ha la pipe come stdout)  */
128             //     printf("Executing %s (%s)\n",com1[0],command1);
129             execvp(com1[0], com1);
130             perror("exec com1");
131             return 5;
132           }
133
134       else
135         {
136
137         close(piped[0]);
138         close(piped[1]);
139
140         sigprocmask(SIG_UNBLOCK, &sigmask, &oldmask);
141
142         }
143   }
144
145
146
147   main()
148   {
```

MODIFIED

Functions	
main	2:147
pipingchild	1:54
sigint_handler	1:46
sigusr1_handler	1:40

```
149  int     length;
150  struct  sockaddr_in     server,client;
151  int     s,msgsock,retval,rval,rval2,i;
152  struct  hostent *hp,*gethostbyname();
153   char servername[256];
154
155  struct sigaction sig, osig;
156   fd_set accept_fds;
157   struct timeval timeRec;
158
159  /* Gestione segnali */
160
161  sig.sa_handler= sigusr1_handler;
162  sigemptyset( &sig.sa_mask);
163  sig.sa_flags= SA_RESTART;        /* attenzione: select ignora SA_RESTART in Linux */
164
165  sigaction(SIGUSR1, &sig, &osig); /* il figlio la ereditera' */
166
167
168  sig.sa_handler= sigint_handler;
169  sigemptyset( &sig.sa_mask);
170  sig.sa_flags= 0;
171
172  sigaction(SIGINT, &sig, &osig); /* il figlio la ereditera' */
173
174    /* Crea la  socket STREAM */
175    sock= socket(AF_INET,SOCK_STREAM,0);
176    if(sock<0)
177      {   perror("opening stream socket");
178      exit(1);
179      }
180  /* Utilizzo della wildcard per accettare connessioni su qualunque interfaccia */
181    server.sin_family =   AF_INET;
182    server.sin_addr.s_addr= INADDR_ANY;
183    server.sin_port = htons(DEFAULTPORT);
184    if (bind(sock,(struct sockaddr *)&server,sizeof server)<0)
185      {
186        server.sin_port = htons(0);
187        if (bind(sock,(struct sockaddr *)&server,sizeof server)<0)
188          {
189            perror("binding stream socket");
190            exit(1);
191          }
192      }
193
194
195    length= sizeof server;
196    if(getsockname(sock,(struct sockaddr *)&server,&length)<0)
197      {
198        perror("getting socket name");
199        exit(1);
200      }
201
202    printf("Socket port #%d\n",ntohs(server.sin_port));
203
204    gethostname(servername,256);
205  hp=     gethostbyname(servername);
206
207  if(hp==0)
208          {
209          fprintf(stderr,"%s: unknown host",servername);
210          exit(2);
211          }
212
213
214  memcpy( (char *)&server.sin_addr, (char *)hp->h_addr ,hp->h_length);
215
216
217    /* Pronto ad accettare connessioni */
218
219    listen(sock,2);
220
221    do
222      {
223        /* Attesa di una connessione */
```

MODIFIED

Functions	
main	2:147
pipingchild	1:54
sigint_handler	1:46
sigusr1_handler	1:40

```
224
225         FD_ZERO(&accept_fds);
226         FD_SET(sock,&accept_fds);
227
228         timeRec.tv_sec = DEFAULTTIMEOUT; timeRec.tv_usec = 0;
229
230         retval= select(sock+1,&accept_fds, NULL,NULL,&timeRec);
231
232         if(retval < 0 && errno != EINTR) /* Per evitare  */
233           {
234             perror("select");
235             exit(-3);
236           }
237         else
238           if(retval == 0)
239           {
240           /* Timeout */
241             printf("No new connection since %d seconds: server exits\n",DEFAULTTIMEOUT)
    ;
242             close(sock);
243             exit(0);
244           }
245           else if (retval >0)
246             {
247               /* C'e' una connessione  */
248               if(FD_ISSET(sock,&accept_fds))
249                 {
250                   msgsock= accept(sock,(struct sockaddr *)&client,(int *)&length);
251                   if(msgsock ==-1)
252               {
253                 perror("accept"); exit(-6);}
254                   else
255                     {
256                       printf("Serving connection from %s, port %d\n",
257                              inet_ntoa(client.sin_addr), ntohs(client.sin_port));
258
259                       /* Verifica autenticazione del client */
260
261                       if((ntohl(client.sin_addr.s_addr) & 0xFFFFFF00) != (ntohl(server.
    sin_addr.s_addr)& 0xFFFFFF00))
262                     {
263                       fprintf(stderr,"Client authentication fails %x %x.\n", server.sin_a
    ddr.s_addr,client.sin_addr.s_addr);
264                       write(msgsock,FAILAUTHMESG,FAILAUTHLEN);
265                       shutdown(msgsock,2);
266                       close(msgsock);
267                     }
268                         else
269                           {
270                       /* Client autenticato */
271
272                           nreq++;
273
274                           if(fork()==0)
275                             {
276                               close(sock);
277                               pipingchild(msgsock);
278                               close(msgsock);
279                               exit(0);
280                             }
281                           else
282                             close(msgsock);
283
284                          }
285
286                       }
287                 }
288             }
289       }
290
291     while(1);
292
293     exit(0);
294   }
295
```

296
297
298

Functions	
main	2:147
pipingchild	1:54
sigint_handler	1:46
sigusr1_handler	1:40