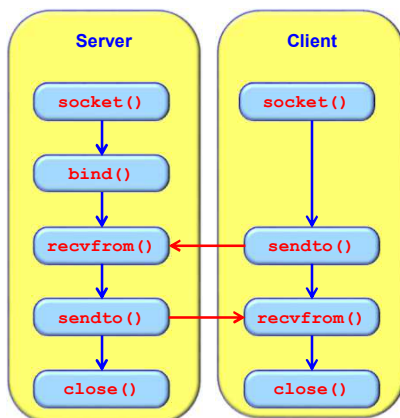




I socket sono definiti in modo univoco da un insieme di 5 elementi:

- il protocollo utilizzato
- l'indirizzo IP locale
- la porta locale
- l'indirizzo IP remoto
- la porta remota

Basta che uno solo di questi elementi sia differente per rendere diverso il collegamento.



```
struct sockaddr
{ u_short sa_family;
  char     sa_data[14];
}
```

Gestisce varie famiglie di socket (anche IPv4 e IPv6).

La famiglia **sa_family** può essere di uno dei seguenti tipi:

AF_UNIX, AF_NS, AF_IMPLINK, AF_INET, AF_INET6

Tipo **AF_INET**

```
struct sockaddr_in
{ short int          sin_family; //AF_INET
  unsigned short int sin_port;
  struct in_addr      sin_addr;
  unsigned char       sin_zero[8];
};

struct in_addr
{ u_int32_t s_addr;
};
```

Tipo **AF_INET6**

```
struct sockaddr_in6
{
    u_int16_t    sin6_family; //AF_INET6
    u_int16_t    sin6_port;
    u_int32_t    sin6_flowinfo; //IPv6 flow info
    struct in6_addr sin6_addr;
    u_int32_t    sin6_scope_id; //scope ID
};

struct in6_addr
{
    unsigned char    s6_addr[16];
};
```

Conversione degli indirizzi

pton = Presentation to Network
ntop = Network to Presentation

```
struct sockaddr_in sa; // IPv4
struct sockaddr_in6 sa6; // IPv6

inet_pton(AF_INET, "192.0.2.1", &(sa.sin_addr)); //IPv4

inet_pton(AF_INET6, "2001:db8:63b3:1::3490",
&(sa6.sin6_addr)); // IPv6

inet_ntop(AF_INET, &(sa.sin_addr), ip4,
INET_ADDRSTRLEN);
inet_ntop(AF_INET6, &(sa6.sin6_addr), ip6,
INET6_ADDRSTRLEN);
```

Reti di Calcolatori 7

Conversione degli indirizzi

int inet_aton(const char *name, struct in_addr *addr)

converte un indirizzo IP in formato "xx.xx.xx.xx" nel corrispondente numero esadecimale e lo inserisce nella struttura *addr*.

unsigned long inet_addr(const char *name)

converte un indirizzo IP in un *unsigned long*

char *inet_ntoa(struct in_addr addr)

fa l'operazione inversa di *inet_aton()*

```
unsigned short htons(unsigned short int n)
unsigned short ntohs(unsigned short int n)
unsigned long htonl(unsigned long int n)
unsigned long ntohl(unsigned long int n)
```

Reti di Calcolatori 8

Socket

```
#include <sys/types.h>
#include <sys/socket.h>

int socket(int family, int type, int protocol);
```

La chiamata *socket()* restituisce un identificatore di *socket*. Il *socket* viene creato e viene definito il suo tipo e il protocollo utilizzato.

L'utilizzo dei descrittori di socket è simile a quello dei file.

Reti di Calcolatori 9

Socket

Family

PF_INET PF_INET6 (AF_INET)

Type (solo per PF_INET):

SOCK_DGRAM, SOCK_STREAM

Il protocollo dipende dalla famiglia (di seguito per AF_INET):

IPPROTO_UDP, IPPROTO_TCP, IPPROTO_ICMP, IPPROTO_RAW

Reti di Calcolatori 10

Bind

```
int bind(int socket,
struct sockaddr *addr,
int addrlen);
```

La chiamata *bind()* restituisce zero in caso di successo. La *bind* serve per inserire i dati locali (indirizzo e porta) nel *socket*.

L'effetto della *bind* è duplice: per il traffico in *input* serve per dire al sistema a chi deve consegnare pacchetti entranti, mentre per il traffico in *output* serve per inserire il mittente nell'istestazione dei pacchetti.

Reti di Calcolatori 11

Bind

Nella definizione dell'indirizzo Internet, è possibile usare alcuni valori definiti tramite macro:

INADDR_LOOPBACK

indica l'host stesso (localhost 127.0.0.1).

INADDR_ANY

serve per accettare le connessioni da qualunque indirizzo

INADDR_BROADCAST

serve per mandare messaggi in broadcast

INADDR_NONE

viene restituito da alcune funzioni in caso di errore.

Reti di Calcolatori 12

Queste due funzioni operano come la `send()` e la `recv()` ma sono utilizzate per le trasmissioni senza connessione. Nella `sendto()` deve essere specificato l'indirizzo di destinazione, mentre nella `recvfrom()` il campo indirizzo viene riempito con quello del mittente.

Chiude un socket aperto.

The diagram illustrates the structure of a network packet. It shows a flow from a **Sender (client)** to a **Receiver (server)**. The packet is represented as a sequence of five segments: **IP_s**, **IP_r**, **Port_s**, **Port_r**, and **Data**.

[illegible]

Reti di Calcolatori 15

[illegible]

Bed di Calcolandi 16

[illegible]

© 2001 Blackwell Science Ltd *Journal of Internal Medicine* 250: 439–447

```

1  UDP Receiver - Receiver/
2  #include <stdio.h>
3  #include <stdlib.h>
4  #include <string.h>
5  #include <sys/types.h>
6  #include <sys/socket.h>
7  #include <unistd.h>
8  #include <arpa/inet.h>
9
10 int main(int argc, char** argv)
11 {
12     int sockfd;
13     struct sockaddr_in local_addr, remote_addr;
14     struct sockaddr_in *sin_addr(remote_addr);
15     char buf[1024];
16
17     if (argc > 2)
18     {
19         printf("Usage: receiver listening_port\n");
20         return 0;
21     }
22
23     if (argc < 2)
24     {
25         printf("Usage: local_ip remote_ip\n");
26         return 0;
27     }
28
29     if (argc == 2)
30     {
31         printf("Usage: local_ip remote_ip\n");
32         return 0;
33     }
34
35     if (argc == 3)
36     {
37         printf("Usage: local_ip remote_ip\n");
38         return 0;
39     }
40
41     if (argc == 4)
42     {
43         printf("Usage: local_ip remote_ip\n");
44         return 0;
45     }
46
47     if (argc == 5)
48     {
49         printf("Usage: local_ip remote_ip\n");
50         return 0;
51     }
52
53     if (argc == 6)
54     {
55         printf("Usage: local_ip remote_ip\n");
56         return 0;
57     }
58
59     if (argc == 7)
60     {
61         printf("Usage: local_ip remote_ip\n");
62         return 0;
63     }
64
65     if (argc == 8)
66     {
67         printf("Usage: local_ip remote_ip\n");
68         return 0;
69     }
70
71     if (argc == 9)
72     {
73         printf("Usage: local_ip remote_ip\n");
74         return 0;
75     }
76
77     if (argc == 10)
78     {
79         printf("Usage: local_ip remote_ip\n");
80         return 0;
81     }
82
83     if (argc == 11)
84     {
85         printf("Usage: local_ip remote_ip\n");
86         return 0;
87     }
88
89     if (argc == 12)
90     {
91         printf("Usage: local_ip remote_ip\n");
92         return 0;
93     }
94
95     if (argc == 13)
96     {
97         printf("Usage: local_ip remote_ip\n");
98         return 0;
99     }
100
101     if (argc == 14)
102     {
103         printf("Usage: local_ip remote_ip\n");
104         return 0;
105     }
106
107     if (argc == 15)
108     {
109         printf("Usage: local_ip remote_ip\n");
110         return 0;
111     }
112
113     if (argc == 16)
114     {
115         printf("Usage: local_ip remote_ip\n");
116         return 0;
117     }
118
119     if (argc == 17)
120     {
121         printf("Usage: local_ip remote_ip\n");
122         return 0;
123     }
124
125     if (argc == 18)
126     {
127         printf("Usage: local_ip remote_ip\n");
128         return 0;
129     }
130
131     if (argc == 19)
132     {
133         printf("Usage: local_ip remote_ip\n");
134         return 0;
135     }
136
137     if (argc == 20)
138     {
139         printf("Usage: local_ip remote_ip\n");
140         return 0;
141     }
142
143     if (argc == 21)
144     {
145         printf("Usage: local_ip remote_ip\n");
146         return 0;
147     }
148
149     if (argc == 22)
150     {
151         printf("Usage: local_ip remote_ip\n");
152         return 0;
153     }
154
155     if (argc == 23)
156     {
157         printf("Usage: local_ip remote_ip\n");
158         return 0;
159     }
160
161     if (argc == 24)
162     {
163         printf("Usage: local_ip remote_ip\n");
164         return 0;
165     }
166
167     if (argc == 25)
168     {
169         printf("Usage: local_ip remote_ip\n");
170         return 0;
171     }
172
173     if (argc == 26)
174     {
175         printf("Usage: local_ip remote_ip\n");
176         return 0;
177     }
178
179     if (argc == 27)
180     {
181         printf("Usage: local_ip remote_ip\n");
182         return 0;
183     }
184
185     if (argc == 28)
186     {
187         printf("Usage: local_ip remote_ip\n");
188         return 0;
189     }
190
191     if (argc == 29)
192     {
193         printf("Usage: local_ip remote_ip\n");
194         return 0;
195     }
196
197     if (argc == 30)
198     {
199         printf("Usage: local_ip remote_ip\n");
200         return 0;
201     }
202
203     if (argc == 31)
204     {
205         printf("Usage: local_ip remote_ip\n");
206         return 0;
207     }
208
209     if (argc == 32)
210     {
211         printf("Usage: local_ip remote_ip\n");
212         return 0;
213     }
214
215     if (argc == 33)
216     {
217         printf("Usage: local_ip remote_ip\n");
218         return 0;
219     }
220
221     if (argc == 34)
222     {
223         printf("Usage: local_ip remote_ip\n");
224         return 0;
225     }
226
227     if (argc == 35)
228     {
229         printf("Usage: local_ip remote_ip\n");
230         return 0;
231     }
232
233     if (argc == 36)
234     {
235         printf("Usage: local_ip remote_ip\n");
236         return 0;
237     }
238
239     if (argc == 37)
240     {
241         printf("Usage: local_ip remote_ip\n");
242         return 0;
243     }
244
245     if (argc == 38)
246     {
247         printf("Usage: local_ip remote_ip\n");
248         return 0;
249     }
250
251     if (argc == 39)
252     {
253         printf("Usage: local_ip remote_ip\n");
254         return 0;
255     }
256
257     if (argc == 40)
258     {
259         printf("Usage: local_ip remote_ip\n");
260         return 0;
261     }
262
263     if (argc == 41)
264     {
265         printf("Usage: local_ip remote_ip\n");
266         return 0;
267     }
268
269     if (argc == 42)
270     {
271         printf("Usage: local_ip remote_ip\n");
272         return 0;
273     }
274
275     if (argc == 43)
276     {
277         printf("Usage: local_ip remote_ip\n");
278         return 0;
279     }
280
281     if (argc == 44)
282     {
283         printf("Usage: local_ip remote_ip\n");
284         return 0;
285     }
286
287     if (argc == 45)
288     {
289         printf("Usage: local_ip remote_ip\n");
290         return 0;
291     }
292
293     if (argc == 46)
294     {
295         printf("Usage: local_ip remote_ip\n");
296         return 0;
297     }
298
299     if (argc == 47)
300     {
301         printf("Usage: local_ip remote_ip\n");
302         return 0;
303     }
304
305     if (argc == 48)
306     {
307         printf("Usage: local_ip remote_ip\n");
308         return 0;
309     }
310
311     if (argc == 49)
312     {
313         printf("Usage: local_ip remote_ip\n");
314         return 0;
315     }
316
317     if (argc == 50)
318     {
319         printf("Usage: local_ip remote_ip\n");
320         return 0;
321     }
322
323     if (argc == 51)
324     {
325         printf("Usage: local_ip remote_ip\n");
326         return 0;
327     }
328
329     if (argc == 52)
330     {
331         printf("Usage: local_ip remote_ip\n");
332         return 0;
333     }
334
335     if (argc == 53)
336     {
337         printf("Usage: local_ip remote_ip\n");
338         return 0;
339     }
340
341     if (argc == 54)
342     {
343         printf("Usage: local_ip remote_ip\n");
344         return 0;
345     }
346
347     if (argc == 55)
348     {
349         printf("Usage: local_ip remote_ip\n");
350         return 0;
351     }
352
353     if (argc == 56)
354     {
355         printf("Usage: local_ip remote_ip\n");
356         return 0;
357     }
358
359     if (argc == 57)
360     {
361         printf("Usage: local_ip remote_ip\n");
362         return 0;
363     }
364
365     if (argc == 58)
366     {
367         printf("Usage: local_ip remote_ip\n");
368         return 0;
369     }
370
371     if (argc == 59)
372     {
373         printf("Usage: local_ip remote_ip\n");
374         return 0;
375     }
376
377     if (argc == 60)
378     {
379         printf("Usage: local_ip remote_ip\n");
380         return 0;
381     }
382
383     if (argc == 61)
384     {
385         printf("Usage: local_ip remote_ip\n");
386         return 0;
387     }
388
389     if (argc == 62)
390     {
391         printf("Usage: local_ip remote_ip\n");
392         return 0;
393     }
394
395     if (argc == 63)
396     {
397         printf("Usage: local_ip remote_ip\n");
398         return 0;
399     }
400
401     if (argc == 64)
402     {
403         printf("Usage: local_ip remote_ip\n");
404         return 0;
405     }
406
407     if (argc == 65)
408     {
409         printf("Usage: local_ip remote_ip\n");
410         return 0;
411     }
412
413     if (argc == 66)
414     {
415         printf("Usage: local_ip remote_ip\n");
416         return 0;
417     }
418
419     if (argc == 67)
420     {
421         printf("Usage: local_ip remote_ip\n");
422         return 0;
423     }
424
425     if (argc == 68)
426     {
427         printf("Usage: local_ip remote_ip\n");
428         return 0;
429     }
430
431     if (argc == 69)
432     {
433         printf("Usage: local_ip remote_ip\n");
434         return 0;
435     }
436
437     if (argc == 70)
438     {
439         printf("Usage: local_ip remote_ip\n");
440         return 0;
441     }
442
443     if (argc == 71)
444     {
445         printf("Usage: local_ip remote_ip\n");
446         return 0;
447     }
448
449     if (argc == 72)
450     {
451         printf("Usage: local_ip remote_ip\n");
452         return 0;
453     }
454
455     if (argc == 73)
456     {
457         printf("Usage: local_ip remote_ip\n");
458         return 0;
459     }
460
461     if (argc == 74)
462     {
463         printf("Usage: local_ip remote_ip\n");
464         return 0;
465     }
466
467     if (argc == 75)
468     {
469         printf("Usage: local_ip remote_ip\n");
470         return 0;
471    
```

```
*) &remote_addr,  
r.sin_port), line);
```

Esempio UDP bidirezionale

Il seguente esempio mostra invece una connessione UDP *bidirezionale*. Un modulo (**transponder**) rimane residente su un host e restituisce al mittente tutto ciò che arriva alla sua porta d'ascolto.

Sender Porta d'ascolto Transponder

Reti di Calcolatori 19

Esempio UDP bidirezionale

```

/* Sample UDP server */

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

int main(int argc, char**argv)
{
    int sockfd, n;
    struct sockaddr_in local_addr, remote_addr;
    socklen_t len = sizeof(remote_addr);
    char msg[1000];

    if (argc < 2)
    {
        printf("Usage: server listening_PORT");
        return 0;
    }

    /* Sample UDP server */
    /* ... (code continues) ... */
}

```

Reti di Calcolatori 20

Esempio UDP bidirezionale

```

if ((sockfd=socket(AF_INET, SOCK_DGRAM, 0)) < 0)
{
    printf("\nErrore nell'apertura del socket");
    return -1;
}

memset(&local_addr, 0, sizeof(local_addr));
local_addr.sin_family = AF_INET;
local_addr.sin_addr.s_addr = htonl(INADDR_ANY);
local_addr.sin_port = htons(atoi(argv[1]));

if (bind(sockfd, (struct sockaddr *)
    &local_addr, sizeof(local_addr)) < 0)
{
    printf("\nErrore nel binding. Errore %d\n",
        errno);
    return -1;
}

for (;;)
{
    n = recvfrom(sockfd, msg, 1000, 0, (struct sockaddr *)
        &remote_addr, &len);
    msg[n] = 0;
    printf("From IP:%s Port:%d msg:%s\n",
        inet_ntoa(remote_addr.sin_addr),
        ntohs(remote_addr.sin_port), msg);
    sendto(sockfd, msg, n, 0, (struct sockaddr *)
        &remote_addr, len);
}

```

Reti di Calcolatori 21

Esempio UDP bidirezionale

```

/* Sample UDP client */

#include <sys/socket.h>
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <sys/types.h>
#include <sys/socket.h>
#include <netinet/in.h>
#include <arpa/inet.h>

int main(int argc, char** argv)
{
    int sockfd, n;
    struct sockaddr_in remote_addr, local_addr;
    char sendline[1000];
    char recvline[1000];
    socklen_t len = sizeof(remote_addr);

    if (argc != 3)
    {
        printf("usage: UDPclient <remote_IP remote_port>\n");
        return 1;
    }

    /* Sample UDP client */
    /* ... (code continues) ... */
}

```

Reti di Calcolatori 22

Esempio UDP bidirezionale

```

if ((sockfd=socket(AF_INET, SOCK_DGRAM, 0)) < 0)
{
    printf("\nErrore nell'apertura del socket");
    return -1;
}

memset(&remote_addr, 0, sizeof(remote_addr));
remote_addr.sin_family = AF_INET;
remote_addr.sin_addr.s_addr = inet_addr(argv[2]);
remote_addr.sin_port = htons(atoi(argv[3]));

while (fgets(sendline, 1000, stdin) != NULL)
{
    sendto(sockfd, sendline, strlen(sendline), 0,
        (struct sockaddr *) &remote_addr,
        sizeof(remote_addr));

    n = recvfrom(sockfd, recvline, 1000, 0, &remote_addr,
        &len);
    printf("From IP:%s Port:%d msg:%s\n",
        inet_ntoa(remote_addr.sin_addr),
        ntohs(remote_addr.sin_port), recvline);
}

```

Reti di Calcolatori 23

Connessioni TCP

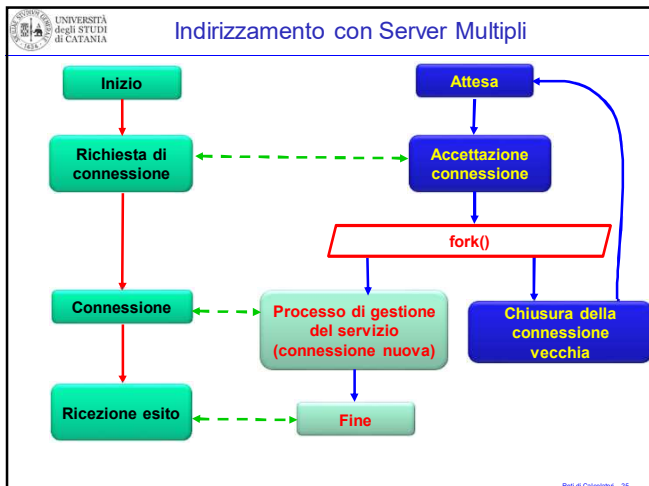
Server

- socket()
- bind()
- listen()
- accept()
- recv()
- send()
- close()

Client

- socket()
- connect()
- send()
- recv()
- close()

Reti di Calcolatori 24



Esempio fork()

```

/* Sample fork */

#include <stdio.h>

int main(int argc, char**argv)
{ int n=0, pid;

  for(; n<5; ++n) printf("%d \n",n);

  pid = fork();

  if (pid == 0)
  { for(; n<10; ++n) printf("%d %d \n", pid, n*2);
    return 0;
  }
  else
  { for(; n<10; ++n) printf("%d %d \n", pid, n*3);
    return 0;
  }
}
  
```

Reti di Calcolatori 26

Lato server – listen() – accept()

```

int listen(int socket, int backlog);
  
```

La chiamata `listen()` abilita il socket a ricevere connessioni, rendendolo quindi un server socket. Il parametro `n` indica quante richieste in sospeso devono essere accodate.

```

int accept(int socket, struct sockaddr *addr,
           socklen_t *addrlen);
  
```

La chiamata `accept()` è bloccante. Non appena arriva una richiesta di connessione, crea un nuovo socket e ne restituisce il descrittore. Il vecchio socket rimane aperto e non connesso. L'indirizzo restituito è quello di chi ha effettuato la `connect()`

Reti di Calcolatori 27

Lato client – connect()

```

int connect(int socket, struct sockaddr *addr,
            int addrlen);
  
```

La chiamata `connect()` inizializza una connessione con un socket remoto. L'indirizzo che viene passato è relativo ad un host remoto.

La `connect()` è bloccante, finché non vengono negoziati i parametri della trasmissione (il protocollo è il TCP).

La funzione viene richiamata da un client che vuol connettersi ad un server (il cui socket deve già essere aperto).

Reti di Calcolatori 28

send() – recv()

```

int send(int socket, void *buffer, size_t size,
         int flags);

int recv(int socket, void *buffer, size_t size,
         int flags);
  
```

Queste due funzioni operano come la `write()` e la `read()` per i file normali. Sono utilizzate per trasmissioni con connessione.

Reti di Calcolatori 29

Client Server TCP - Modulo server

```

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

int main(int argc, char**argv)
{ int sockfd, newsockfd, n;
  struct sockaddr_in local_addr, remote_addr;
  socklen_t len;
  char msg[1000];

  if (argc < 2)
  { printf("Use: server listening_PORT");
    return 0;
  }
}
  
```

Reti di Calcolatori 30

Client Server TCP - Modulo server

```
if((sockfd=socket(AF_INET,SOCK_STREAM,0)) <0)
{ printf("\nErrore nell'apertura del socket");
return -1;
}
memset((char *) &local_addr,0,sizeof(local_addr));
local_addr.sin_family = AF_INET;
local_addr.sin_addr.s_addr = htonl(INADDR_ANY);
local_addr.sin_port = htons(atoi(argv[1]));

if(bind(sockfd, (struct sockaddr *) &local_addr,
sizeof(local_addr))<0)
{ printf("\nErrore nel binding. Errore %d \n",
errno);
return -1;
}

listen(sockfd,5);
```

```
/* Sample TCP server */
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <sys/types.h>
#include <sys/socket.h>
#include <netinet/in.h>
#include <arpa/inet.h>
#include <errno.h>

int main(int argc, char**argv)
{
    struct sockaddr_in local_addr, remote_addr;
    int sockfd, newsockfd, n;
    char msg[1024];

    if (argc != 2)
    {
        printf("Usage: %s <port>\n", argv[0]);
        return -1;
    }

    sockfd = socket(AF_INET, SOCK_STREAM, 0);
    if (sockfd < 0)
    {
        perror("socket");
        return -1;
    }

    memset(&local_addr, 0, sizeof(local_addr));
    local_addr.sin_family = AF_INET;
    local_addr.sin_addr.s_addr = htonl(INADDR_ANY);
    local_addr.sin_port = htons(atoi(argv[1]));

    if (bind(sockfd, (struct sockaddr *) &local_addr,
        sizeof(local_addr)) < 0)
    {
        perror("bind");
        return -1;
    }

    listen(sockfd, 5);

    while (1)
    {
        remote_addr.sin_family = AF_INET;
        remote_addr.sin_port = 0;
        newsockfd = accept(sockfd, (struct sockaddr *) &remote_addr,
            &remote_addr);
        if (newsockfd < 0)
        {
            perror("accept");
            continue;
        }

        bzero(msg, 1024);
        n = recv(newsockfd, msg, 1023, 0);
        if (n < 0)
        {
            perror("recv");
            continue;
        }

        printf("Received %d bytes from %s\n", n,
            inet_ntoa(remote_addr.sin_addr));

        if (n > 0)
        {
            send(newsockfd, msg, n, 0);
        }

        close(newsockfd);
    }

    close(sockfd);
    return 0;
}
```

Reti di Calabrese 31

Client Server TCP - Modulo server

```
for(;;)
{ len = sizeof(remote_addr);
  newsockfd = accept(sockfd, (struct sockaddr *)
    &remote_addr, &len);

  if (fork() == 0)
  { close(sockfd);
    for(;;)
    { n = recv(newsockfd, msg, 1024, 0);
      if (n == 0) return 0;
      msg[n] = 0;
      printf("\nPid=%d: received from %s: %s\n",
        getpid(), inet_ntoa(remote_addr.sin_addr),
        ntohs(remote_addr.sin_port), msg);
      send(newsockfd, msg, n, 0);
    }
    return 0;
  }
  else
  { close(newsockfd);
  }
}
```

```
/* Sample TCP server */
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <sys/types.h>
#include <sys/socket.h>
#include <netinet/in.h>
#include <arpa/inet.h>
#include <errno.h>

int main(int argc, char**argv)
{
    struct sockaddr_in local_addr, remote_addr;
    int sockfd, newsockfd, n;
    char msg[1024];

    if (argc != 2)
    {
        printf("Usage: %s <port>\n", argv[0]);
        return -1;
    }

    sockfd = socket(AF_INET, SOCK_STREAM, 0);
    if (sockfd < 0)
    {
        perror("socket");
        return -1;
    }

    memset(&local_addr, 0, sizeof(local_addr));
    local_addr.sin_family = AF_INET;
    local_addr.sin_addr.s_addr = htonl(INADDR_ANY);
    local_addr.sin_port = htons(atoi(argv[1]));

    if (bind(sockfd, (struct sockaddr *) &local_addr,
        sizeof(local_addr)) < 0)
    {
        perror("bind");
        return -1;
    }

    listen(sockfd, 5);

    while (1)
    {
        remote_addr.sin_family = AF_INET;
        remote_addr.sin_port = 0;
        newsockfd = accept(sockfd, (struct sockaddr *) &remote_addr,
            &remote_addr);
        if (newsockfd < 0)
        {
            perror("accept");
            continue;
        }

        bzero(msg, 1024);
        n = recv(newsockfd, msg, 1023, 0);
        if (n < 0)
        {
            perror("recv");
            continue;
        }

        printf("Received %d bytes from %s\n", n,
            inet_ntoa(remote_addr.sin_addr));

        if (n > 0)
        {
            send(newsockfd, msg, n, 0);
        }

        close(newsockfd);
    }

    close(sockfd);
    return 0;
}
```

Reti di Calabrese 32

Client Server TCP - Modulo client

```
/* Sample TCP client */
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <sys/types.h>
#include <sys/socket.h>
#include <netinet/in.h>
#include <arpa/inet.h>
#include <errno.h>

int main(int argc, char**argv)
{
    int sockfd, n;
    struct sockaddr_in local_addr, dest_addr;
    char sendline[1024];
    char recvline[1024];

    if (argc != 3)
    { printf("usage: client IP_address <Port>\n");
      return 1;
    }
}
```

```
/* Sample TCP client */
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <sys/types.h>
#include <sys/socket.h>
#include <netinet/in.h>
#include <arpa/inet.h>
#include <errno.h>

int main(int argc, char**argv)
{
    struct sockaddr_in dest_addr;
    int sockfd, n;
    char sendline[1024];
    char recvline[1024];

    if (argc != 3)
    {
        printf("Usage: %s <IP> <Port>\n", argv[0]);
        return -1;
    }

    sockfd = socket(AF_INET, SOCK_STREAM, 0);
    if (sockfd < 0)
    {
        perror("socket");
        return -1;
    }

    memset(&dest_addr, 0, sizeof(dest_addr));
    dest_addr.sin_family = AF_INET;
    dest_addr.sin_addr.s_addr = inet_addr(argv[1]);
    dest_addr.sin_port = htons(atoi(argv[2]));

    if (connect(sockfd, (struct sockaddr *) &dest_addr,
        sizeof(dest_addr)) < 0)
    {
        perror("connect");
        return -1;
    }

    while (1)
    {
        bzero(sendline, 1024);
        fgets(sendline, 1024, stdin);
        n = send(sockfd, sendline, strlen(sendline), 0);
        if (n < 0)
        {
            perror("send");
            continue;
        }

        bzero(recvline, 1024);
        n = recv(sockfd, recvline, 1024, 0);
        if (n < 0)
        {
            perror("recv");
            continue;
        }

        printf("Received %d bytes from %s: %s\n",
            n, inet_ntoa(dest_addr.sin_addr),
            ntohs(dest_addr.sin_port), recvline);
    }

    close(sockfd);
    return 0;
}
```

Reti di Calabrese 33

Client Server TCP - Modulo client

```
sockfd=socket(AF_INET,SOCK_STREAM,0);

memset(&dest_addr, 0, sizeof(dest_addr));
dest_addr.sin_family = AF_INET;
dest_addr.sin_addr.s_addr = inet_addr(argv[1]);
dest_addr.sin_port = htons(atoi(argv[2]));

connect(sockfd, (struct sockaddr *) &dest_addr,
sizeof(dest_addr));

while (fgets(sendline,1024,stdin) != NULL)
{ send(sockfd,sendline,strlen(sendline),0);
  n=recv(sockfd,recvline,1024,0);
  recvline[n]=0;
  printf("\nPid=%d: received from %s: %s\n",
    getpid(), inet_ntoa(dest_addr.sin_addr),
    ntohs(dest_addr.sin_port), recvline );
}
```

```
/* Sample TCP client */
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <sys/types.h>
#include <sys/socket.h>
#include <netinet/in.h>
#include <arpa/inet.h>
#include <errno.h>

int main(int argc, char**argv)
{
    struct sockaddr_in dest_addr;
    int sockfd, n;
    char sendline[1024];
    char recvline[1024];

    if (argc != 3)
    {
        printf("Usage: %s <IP> <Port>\n", argv[0]);
        return -1;
    }

    sockfd = socket(AF_INET, SOCK_STREAM, 0);
    if (sockfd < 0)
    {
        perror("socket");
        return -1;
    }

    memset(&dest_addr, 0, sizeof(dest_addr));
    dest_addr.sin_family = AF_INET;
    dest_addr.sin_addr.s_addr = inet_addr(argv[1]);
    dest_addr.sin_port = htons(atoi(argv[2]));

    if (connect(sockfd, (struct sockaddr *) &dest_addr,
        sizeof(dest_addr)) < 0)
    {
        perror("connect");
        return -1;
    }

    while (1)
    {
        bzero(sendline, 1024);
        fgets(sendline, 1024, stdin);
        n = send(sockfd, sendline, strlen(sendline), 0);
        if (n < 0)
        {
            perror("send");
            continue;
        }

        bzero(recvline, 1024);
        n = recv(sockfd, recvline, 1024, 0);
        if (n < 0)
        {
            perror("recv");
            continue;
        }

        printf("Received %d bytes from %s: %s\n",
            n, inet_ntoa(dest_addr.sin_addr),
            ntohs(dest_addr.sin_port), recvline);
    }

    close(sockfd);
    return 0;
}
```

Reti di Calabrese 34