Laboratorio di Reti di Calcolatori

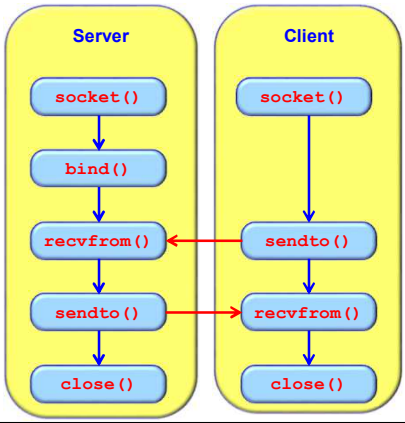
Socket di Berkeley

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.

Connessioni UDP



Definizione di un socket

```
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`

Definizione di un socket AF_INET

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;
};
```


Definizione di un socket AF_INET6

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);
```


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)
```


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.

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`

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'intestazione dei pacchetti.

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.

sendto - recvfrom

```
int sendto(int socket, void *buffer, size_t size
   int flags, struct sockaddr *addr, size_t length);

int recvfrom(int socket, void *buffer, size_t size
  int flags, struct sockaddr *addr, size_t *length);
```

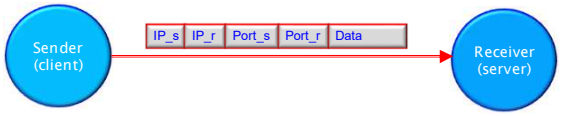
Queste due funzioni operano come la `send()` e la `recv()` ma sono utilizzate per le trasmissioni senza connessione.
Nella `sendto()` deve essere specificato l'indirizzo di destinazione, mentre nella `recvfrom()` il campo indirizzo viene riempito con quello del mittente.

```
int close(int socket);
```

Chiude un socket aperto.

Esempio UDP – sender to receiver

Il seguente programma è il più semplice immaginabile. È composto da un sender che spedisce pacchetti e da un receiver che li riceve. La comunicazione è unidirezionale. Il protocollo usato è UDP.



Esempio UDP – Sender (client)

```
/* UDP Sender - 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>

int main(int argc, char* argv[])
{ int sockfd;
  struct sockaddr_in dest_addr;
  extern int errno;
  int i,n;
  char line[100];

  if(argc < 3)
  {  printf("Use: sender IP_dest PORT_dest");
     return 0;
  }
```


Esempio UDP – Sender (client)

```
   if((sockfd=socket(AF_INET,SOCK_DGRAM,0)) <0)
  { printf("\nErrore nell'apertura del 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]));

  printf("\nInserisci un numero intero ");
  scanf("%d",&n);

  for(i=0; i<10; ++i)
  { sprintf(line,"%d",i+n);
    printf("%s\n",line);
    sendto(sockfd, line, strlen(line)+1, 0, &dest_addr,
sizeof(dest_addr));
  }
}
```


Esempio UDP – Receiver (server)

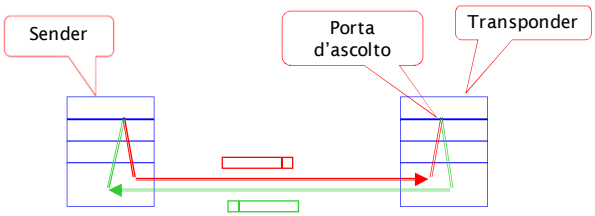
```
/* UDP Receiver - Server*/
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <sys/types.h>
#include <sys/socket.h>
#include <netinet/in.h>
#include <arpa/inet.h>
#include <errno.h>

int main(int argc, char* argv[])
{ int sockfd;
  struct sockaddr_in local_addr, remote_addr;
  socklen_t len = sizeof(remote_addr);
  char line[100];

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


Esempio UDP – Receiver (server)

```
  if((sockfd=socket(AF_INET,SOCK_DGRAM,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",er
    return -1;
  }

  for(;;)
  { recvfrom(sockfd, line, 99,0, (struct sockaddr *) &remote_addr,
&len);
    printf("From IP:%s Port:%d msg:%s \n",
inet_ntoa(remote_addr.sin_addr),  ntohs(remote_addr.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.



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 mesg[1000];

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


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",e
    return -1;
  }
  for (;;)
  { n = recvfrom(sockfd,mesg,1000,0,(struct sockaddr *) &remote_addr,
&len);
    mesg[n] = 0;
    printf("From IP:%s Port:%d msg:%s \n",
inet_ntoa(remote_addr.sin_addr), ntohs(remote_addr.sin_port), mesg);
    sendto(sockfd,mesg,n,0,(struct sockaddr *)&remote_addr,len);
  }
}
```

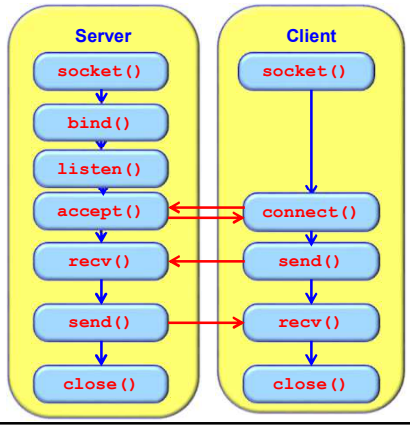

Esempio UDP bidirezionale

```
/* 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;
   }
```

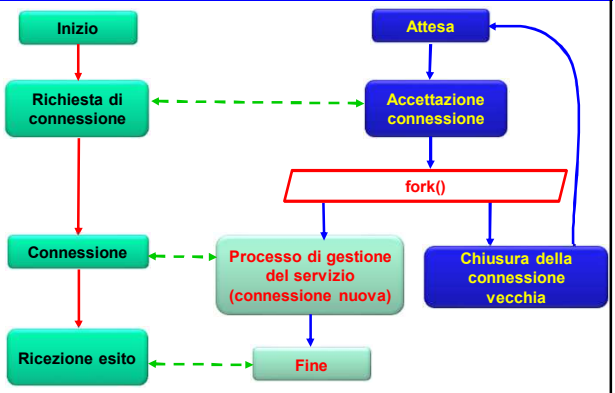

Esempio UDP bidirezionale

```
 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_add
    recvline[n]=0;
    printf("From IP:%s Port:%d msg:%s \n",
           inet_ntoa(remote_addr.sin_addr),
           ntohs(remote_addr.sin_port), recvline);
  }
}
```


Connessioni TCP



Indirizzamento con Server Multipli



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;
  }
}
```


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()`**

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).

send( ) – recv( )

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

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

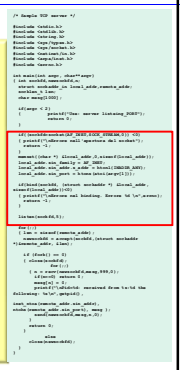
Queste due funzioni operano come la **`write()`** e la **`read()`** per i file normali. Sono utilizzate per trasmissioni con connessione.

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 mesg[1000];

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

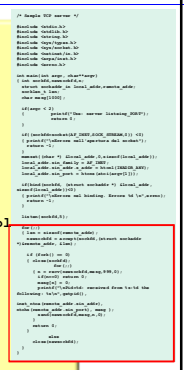
Client Server TCP - Modulo server

```
 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);
```



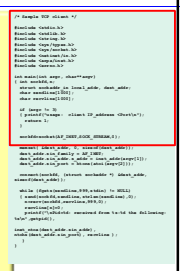
Client Server TCP - Modulo server

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

    if (fork() == 0)
    { close(sockfd);
      for(;;)
      { n = recv(newsockfd,mesg,999,0);
        if(n==0) return 0;
        mesg[n] = 0;
        printf("\nPid=%d: received from %s:%d the fol
%s\n", getpid(), inet_ntoa(remote_addr.sin_addr),
ntohs(remote_addr.sin_port), mesg );
        send(newsockfd,mesg,n,0);
      }
      return 0;
    }
    else
      close(newsockfd);
  }
}
```



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[1000];
  char recvline[1000];

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



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,999,stdin) != NULL)
   { send(sockfd,sendline,strlen(sendline),0);
     n=recv(sockfd,recvline,999,0);
     recvline[n]=0;
     printf("\nPid=%d: received from %s:%d the following:
%s\n",getpid(), inet_ntoa(dest_addr.sin_addr),
          ntohs(dest_addr.sin_port), recvline );
    }
}
```