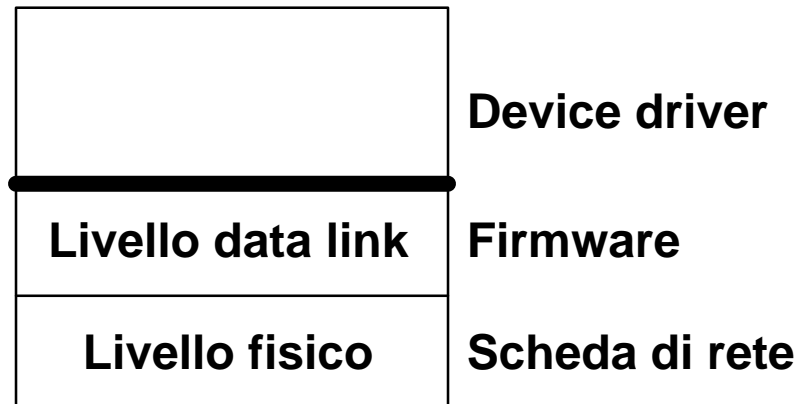


Programmazione di rete

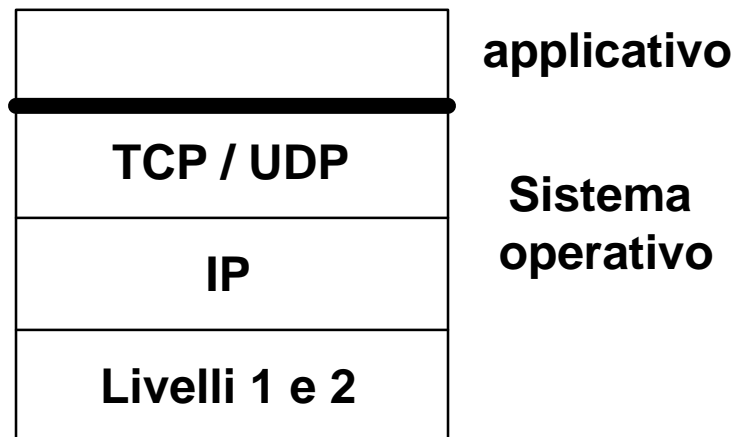
- Un programma che utilizza servizi di comunicazione di rete *interagisce* con il modulo che implementa i servizi
- L'interazione avviene attraverso un'**interfaccia** (API - *Application Program Interface*)
- Generalmente gli standard di protocollo
 - specificano le funzionalità dei protocolli (il servizio offerto)
 - non specificano le interfacce (come il servizio viene fruito)
- Per facilitare la portabilità del software si tende a utilizzare interfacce standardizzate

Esempi di interfacce

- **NDIS**



- **Socket**



Tipi di interfacce

- **Interfaccia procedurale:**

è costituita da un insieme di procedure che realizzano le funzionalità richieste (per esempio aprire una connessione o inviare un datagram)

- **Interfaccia basata su messaggi**

è costituita da un insieme di messaggi che fruitore e fornitore si scambiano secondo un certo protocollo

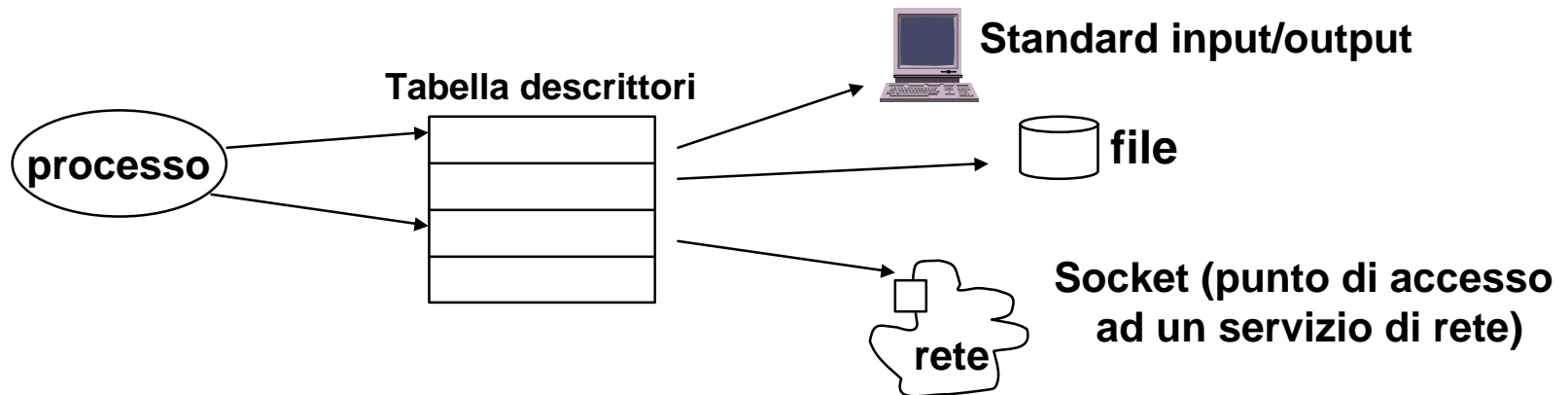
- **Interfaccia basata su meccanismi specifici** (p. es. interrupt software)

Interfaccia di rete bsd4.3

- E` diventata in pratica lo standard per accedere ai servizi della rete ARPA/Internet
- Nata in ambito Unix, è stata implementata per tutti i principali sistemi (per esempio *Pathworks for DOS*, *Windows sockets*)
- Essendo basata su astrazioni chiamate **socket**, è nota anche come *socket API* o *interfaccia dei socket*
- Documentazione:
 - W. R. Stevens, "Unix Network Programming", Prentice Hall, 1990
 - D. E. Comer, D. L. Stevens, "Internetworking with TCP/IP", Vol. 3, Prentice Hall, 1997

Caratteristiche generali


- Interfaccia Procedurale in linguaggio C
- Nasce con l'idea di estendere il modello di I/O convenzionale di Unix:




- L'adesione al modello però non può essere totale (il paradigma *open-read-write-close* si adatta bene solo alle comunicazioni connection-oriented)

Caratteristiche generali (cont.)

- Definisce procedure **generali** che prevedono l'utilizzo di diversi stack di protocollo (TCP/IP=caso particolare)

`maketcpconnection(...)`  `makeconnection(tcp,...)`



- In realtà è stata definita riflettendo i meccanismi dei protocolli TCP/IP
 - modello asimmetrico di connessione
 - stream senza delimitazione dei messaggi
 - ...

I socket

- Un **socket** e` l'astrazione di un'interfaccia di comunicazione tra processi (endpoint, sap).
- I socket “vivono” in **domini**, ciascuno con la propria **famiglia di protocolli** e **famiglia di indirizzi**
- La comunicazione tra domini diversi è impossibile.
- Esempi di domini:

Dominio	Protocol Family	Address Family
ARPA Internet	PF_INET	AF_INET
Reti ISO/OSI	PF_ISO	AF_ISO
Unix pipes	PF_UNIX	AF_UNIX

Caratteristiche dei socket : il TIPO

- Identifica la *tipologia di servizio* accessibile tramite il socket. I tipi supportati normalmente sono:

- **SOCK_STREAM**

flusso **bidirezionale continuo** (senza delimitazioni) di byte, trasmessi in modo affidabile, in sequenza e senza duplicazioni (servizio **connection-oriented**, tipicamente di livello 4).

- **SOCK_DGRAM**

trasmissione **bidirezionale** di **messaggi** (datagram) senza garanzia di affidabilità, ordinamento o assenza di duplicazioni (servizio **connectionless**, tipicamente di livello 4).

- **SOCK_RAW**

accesso diretto ai protocolli di rete di basso livello (per esempio livello di rete)

Caratteristiche dei socket : il PROTOCOLLO

- In ogni dominio, e per ogni tipo di socket, e` possibile selezionare un particolare protocollo da utilizzare. Per esempio, nel dominio PF_INET:

tipo **SOCK_STREAM**

- Protocollo TCP (IPPROTO_TCP)

tipo **SOCK_DGRAM**

- Protocollo UDP (IPPROTO_UDP)

tipo **SOCK_RAW**

- Protocollo ICMP (IPPROTO_ICMP)
- Pacchetti IP (IPPROTO_RAW)

Caratteristiche dei socket: le OPZIONI

- Specificano altre caratteristiche varie del socket.

Esempi:

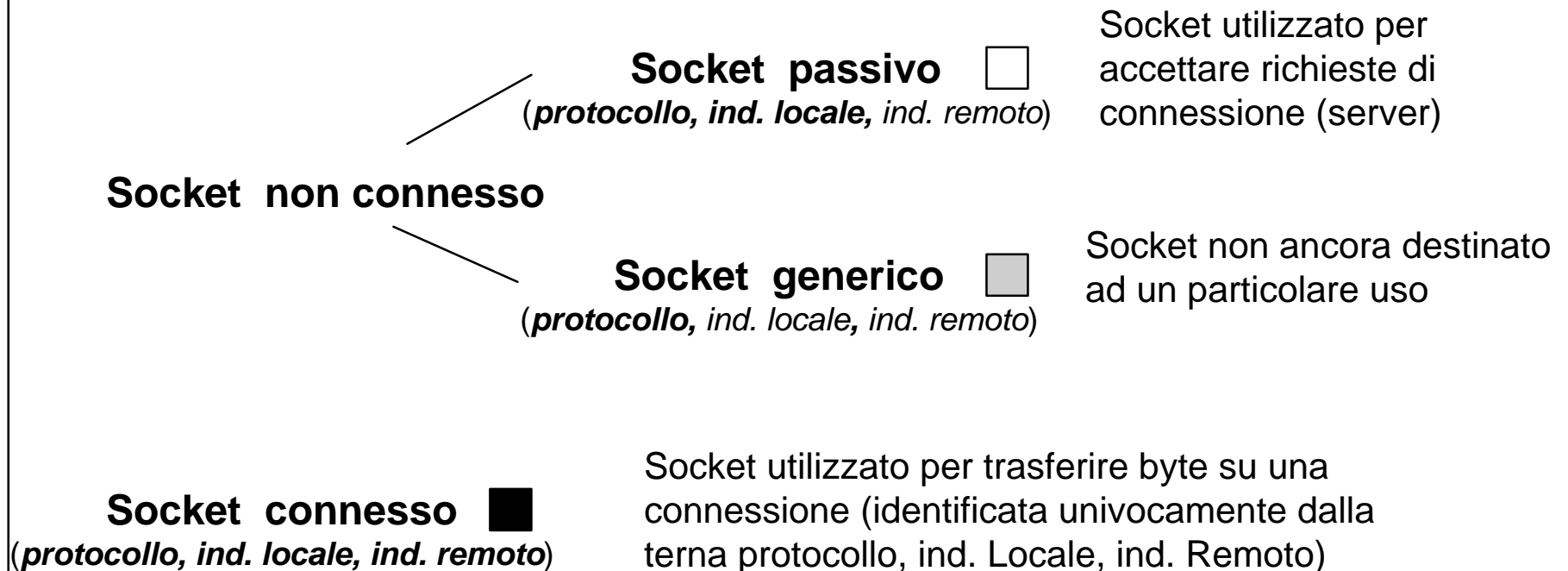
- `SO_RCVBUF` (dimensione del buffer per la ricezione)
- `SO_SNDBUF` (dimensione del buffer per la trasmissione)
- `SO_LINGER` (se abilitata, ritarda la chiusura delle connessioni in presenza di dati nei buffer)
- `SO_TYPE` (e` il tipo del socket)
- `SO_KEEPALIVE` (se abilitata, tra i socket connessi vengono scambiati messaggi periodici. Nel caso in cui non arrivi la risposta, la connessione viene chiusa)

Struttura dati associata ad un socket generico

dominio
tipo
protocollo
Indirizzo locale
Indirizzo remoto
Opzioni

Utilizzo dei socket STREAM

- Per realizzare i diversi tipi di interazione servono diversi tipi di endpoint:



Primitive dell'interfaccia

- Sono di natura sincrona (bloccanti)
- Segnalano condizioni di errore restituendo il valore -1
- Il codice relativo all'errore verificatosi si può leggere:
 - in ambiente UNIX leggendo la variabile `errno`

```
void err_fatal(char *mes) {  
    printf("%s, errno=%d\n", mes, errno);  
    perror(""); exit(1);  
}
```

- in ambiente Windows Sockets chiamando la funzione `WSAGetLastError()`

```
void err_fatal(char *mes) {  
    printf("%s, errno=%d\n", mes, WSAGetLastError());  
    perror(""); exit(1);  
}
```

Creazione di un socket

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

- **family** dominio del socket
- **type** tipo di socket
- **protocol** protocollo utilizzato dal socket

Valore di ritorno :

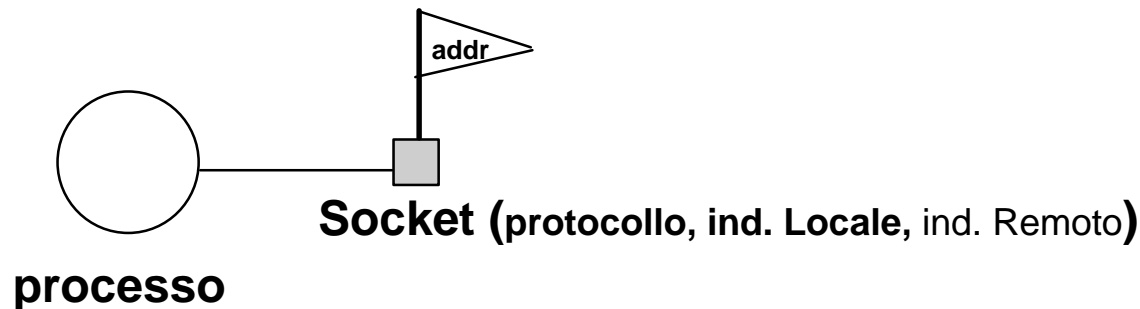
- l'identificativo locale (in Unix il file descriptor) del socket creato



Assegnazione di un indirizzo locale di rete ad un socket

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

- **socket** socket cui assegnare l'indirizzo
- **addr** puntatore all'indirizzo che si vuole assegnare
- **addrlen** lunghezza della struttura sockaddr utilizzata
(sockaddr varia a seconda della address family)



Struct sockaddr

- E` costituita da 2 campi:
 - sa_family (la famiglia di indirizzi utilizzata - 2byte)
 - sa_data (l'indirizzo, specificato secondo le regole della famiglia di indirizzi - massimo 14 byte)
- Nel dominio Internet si usa la famiglia AF_INET. L'indirizzo comprende:
 - 2 byte: port number
 - 4 byte: IP address (indirizzo dell'host)

Di fatto viene utilizzata la struttura **sockaddr_in** (che è la specializzazione di sockaddr per la famiglia AF_INET)

Esempio di uso di struct sockaddr_in

```
struct in_addr {
    unsigned long s_addr;
}

struct sockaddr_in {
    short sin_family;
    u_short sin_port;
    struct in_addr sin_addr;
    char sin_zero[8];
};
```

```
struct sockaddr_in saddr;
saddr.sin_family      = AF_INET;
saddr.sin_port        = 80;
saddr.sin_addr.s_addr = INADDR_ANY;
result = bind(s, (struct sockaddr *) &saddr, sizeof(saddr));
```

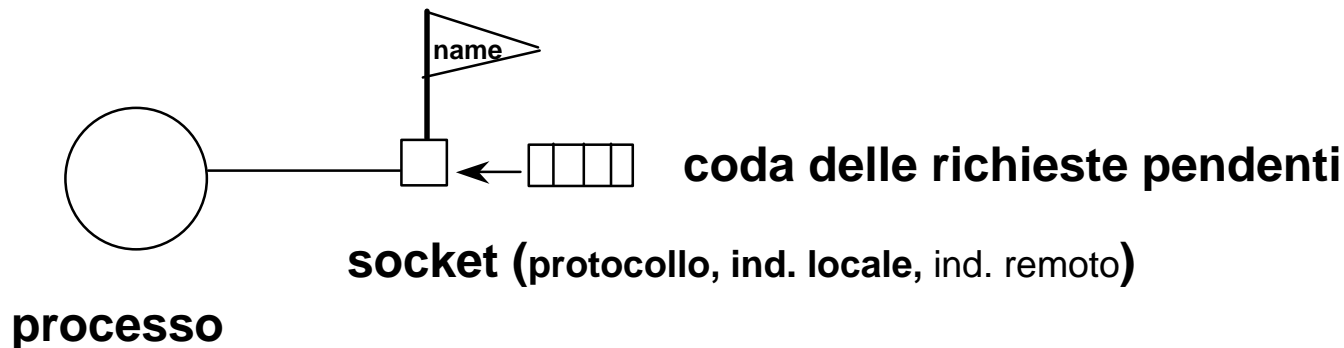
Connessione di socket **STREAM**

- Segue il modello Client/Server tipico del TCP:
 - Il Server comunica l'intenzione di accettare richieste di connessione (Open Passiva)
 - Il Client richiede di instaurare una connessione (Open Attiva)
 - Il Server accetta la richiesta di connessione, rendendo possibile il completamento dell'handshake.
 - Completato l'handshake, Client e Server si sbloccano e possono trasmettere/ricevere dati. In qualunque istante possono decidere di chiudere la connessione, oppure chiudere soltanto uno dei due flussi di dati (shutdown).

Predisposizione del socket per ricevere richieste di connessione (server)

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

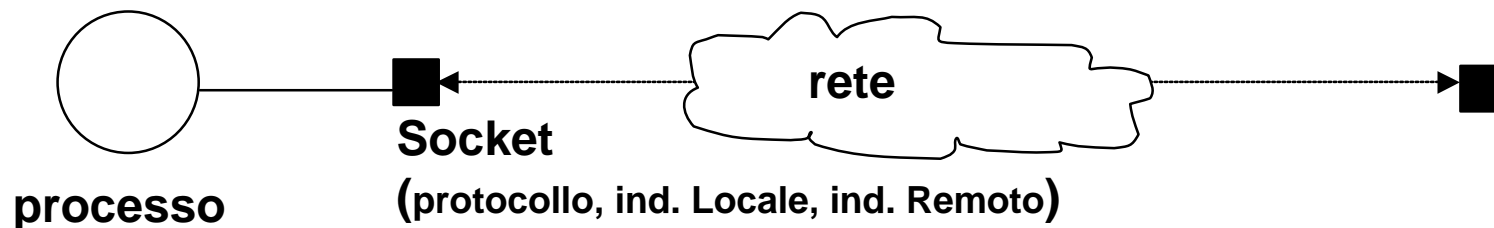
- **socket** socket al quale mettersi in attesa
- **backlog** lunghezza massima della coda delle richieste pendenti



Richiesta di connessione (client)

```
int connect (int socket, struct sockaddr  
            *destaddr, int addrlen)
```

- **socket** socket che si vuole connettere
- **destaddr** puntatore all'indirizzo del server remoto
cui si vuole indirizzare la richiesta
- **addrlen** lunghezza dell'indirizzo



Accettazione di una richiesta di connessione (server)

```
int accept (int socket, struct sockaddr  
*srcaddr, int *addrlen)
```

- **socket** socket dove si riceve la richiesta
- **srcname** puntatore all'indirizzo del socket remoto con cui viene stabilita la connessione
- **namelen** puntatore alla lunghezza dell'indirizzo

Valore di ritorno:

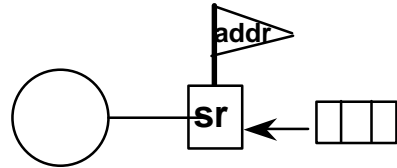
- l'identificativo locale di una copia del socket locale, connessa al socket remoto.

SERVER

CLIENT



`listen(sr, backlog)`

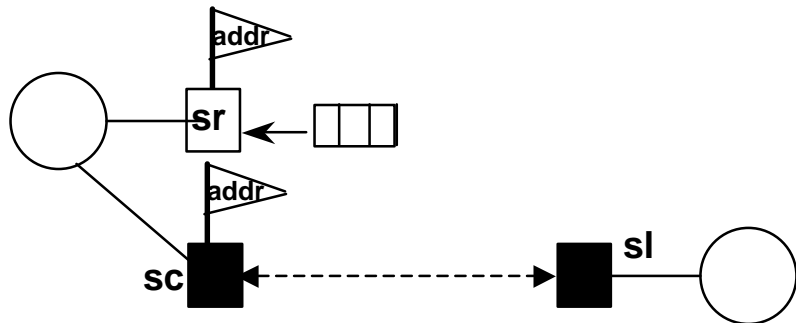
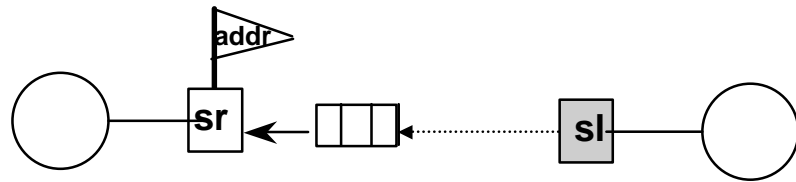


`sc = accept(sr, srcname, sizeof(srcname))`

`connect(sl, addr, sizeof(addr))`

attesa

attesa



Apertura Connessione (Client)

```
int                taddr_n, tport_n;
struct sockaddr_in saddr;
SOCKET            s;
int               result;
. . .
s = socket(AF_INET, SOCK_STREAM, IPPROTO_TCP);
if (s == INVALID_SOCKET)
    err_fatal("socket() failed");

saddr.sin_family      = AF_INET;
saddr.sin_port        = tport_n;
saddr.sin_addr.s_addr = taddr_n;

result = connect(s, (struct sockaddr *) &saddr, sizeof(saddr));
if (result == -1)
    err_fatal("connect() failed");
```

Apertura Connessione (Server)

```
int                lport_n;        /* porta utilizzata */
struct sockaddr_in saddr, caddr;
SOCKET            s, s1;
int               result;
. . .
s = socket(AF_INET, SOCK_STREAM, IPPROTO_TCP);

saddr.sin_family      = AF_INET;
saddr.sin_port        = lport_n;
saddr.sin_addr.s_addr = INADDR_ANY;

result = bind(s, (struct sockaddr *) &saddr, sizeof(saddr));

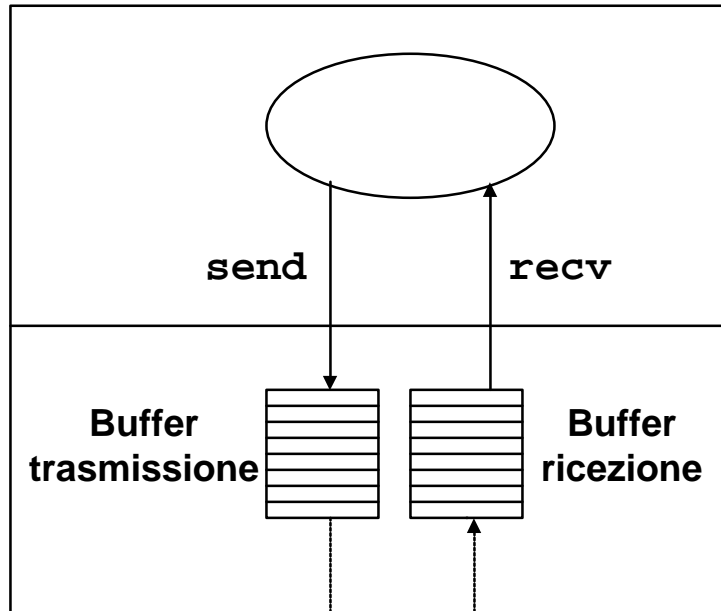
result = listen(s, backlog);

addrlen = sizeof(struct sockaddr_in);
s1 = accept(s, (struct sockaddr *) &caddr, &addrlen);
```

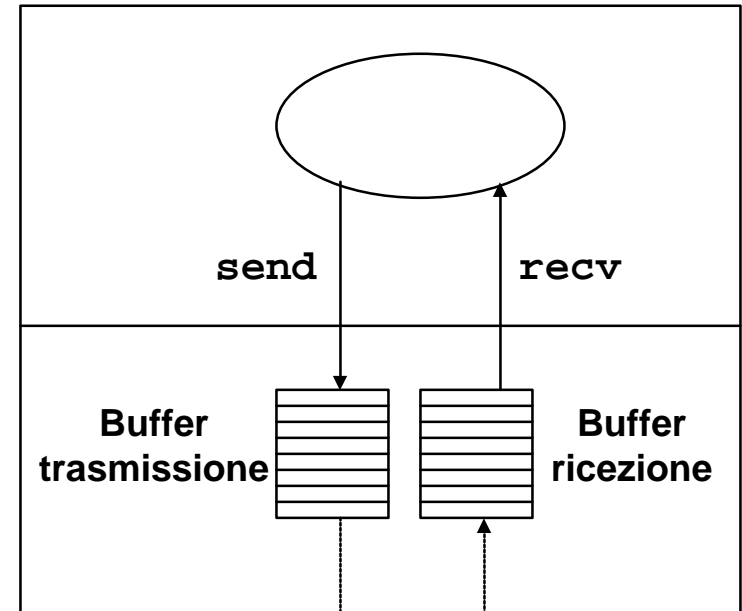
Trasferimento dati su connessioni

- Avviene secondo il seguente modello:

Host 1



Host 2



rete

Invio dati su una connessione

```
int send (int socket, char *data, int datalen,  
         int flags)
```

- **socket** socket connesso attraverso cui si inviano i dati
- **data** buffer contenente i dati da inviare
- **datalen** lunghezza del blocco di dati da inviare
- **flags** specifica eventuali opzioni, come per esempio "out-of-band data"

Valore di ritorno:

- numero di byte effettivamente inviati

Ricezione dati su una connessione

```
int recv (int socket, char *buffer, int  
         buflen, int flags)
```

- **socket** socket connesso attraverso cui si ricevono i dati
- **buffer** buffer di ricezione
- **buflen** lunghezza del buffer di ricezione
- **flags** specifica eventuali opzioni, come per esempio "out-of-band data"

Valore di ritorno:

numero di byte ricevuti (0 se è stata chiusa la connessione).

Esempio: invio di un buffer

```
int writen(SOCKET s, char *ptr, int nbytes)
{
    int nleft, nwritten;

    for (nleft=nbytes; nleft > 0; )
    {
        nwritten = send(s, ptr, nleft, 0);
        if (nwritten <=0)
            return (nwritten);
        else
        {
            nleft -= nwritten;
            ptr += nwritten;
        }
    }
    return (nbytes - nleft);
}
```

Esempio: ricezione di n byte

```
int readn (SOCKET s, char *ptr, int len)
{
    int nread, nleft;

    for (nleft=len; nleft > 0; )
    {
        nread=recv(s, ptr, nleft, 0);
        if (nread > 0)
        {
            nleft -= nread;
            ptr += nread;
        }
        else if (nread == 0) /* conn. closed by party */
            break;
        else /* error */
            return (nread);
    }
    return (len - nleft);
}
```

Chiusura di una connessione

- Normalmente per chiudere una connessione si chiude il socket (vengono anche rilasciate le risorse relative):
 - In ambiente Unix, dove i socket sono trattati esattamente come i file, si usa la `close()`
 - Nell'interfaccia Windows Sockets viene fornita una specifica funzione:

```
int closesocket (int socket)
```

- L'opzione `SO_LINGER` specifica se la chiusura deve essere "graceful" (cioè se prima di chiudere la connessione devono essere svuotati i buffer).

Chiusura parziale

- Una connessione può essere chiusa anche solo limitatamente ad una direzione con la primitiva:

```
int shutdown (int socket, int how)
```

- La modalità di chiusura è controllata dal parametro `how`:
 - 0 disabilita solo la ricezione di dati dal socket
 - 1 disabilita solo l'invio di dati sul socket
 - 2 disabilita entrambe le operazioni sul socket
- L'operazione di `shutdown` non rilascia le risorse del socket. Per rilasciarle occorre chiudere il socket.

Invio/ricezione di datagram su socket di tipo SOCK_DGRAM

- Poiché il servizio è connectionless/datagram:
 - non servono operazioni preliminari
 - si trasmette/riceve un datagram per volta
 - ad ogni trasmissione si deve specificare il destinatario
- Normalmente si usa la bind per assegnare un indirizzo locale ben preciso al socket.
- Terminate le operazioni sul socket il socket deve essere chiuso (per rilasciare le risorse)

Invio di datagram

```
int sendto (int socket, char *data,  
           int datalen, int flags,  
           struct sockaddr *addr, int addrlen)
```

- **socket** socket attraverso cui inviare il datagram
- **data** buffer contenente il datagram da inviare
- **datalen** lunghezza datagram da inviare
- **flags** eventuali opzioni
- **addr** puntatore all'indirizzo del destinatario
- **addrlen** lunghezza dell'indirizzo

Ricezione di datagram

```
int recvfrom (int socket, char *buffer,  
             int buflen, int flags,  
             struct sockaddr *addr, int *addrlen)
```

- **socket** socket attraverso cui ricevere il datagram
- **buffer** buffer in cui verrà depositato il datagram ricevuto
- **addrlen** lunghezza del buffer
- **flags** eventuali opzioni
- **addr** puntatore indirizzo del mittente
- **addrlen** puntatore lunghezza dell'indirizzo del mittente

Il problema del Blocking

- Un processo bloccato su una primitiva (p. es. `recv`) non è in grado di reagire al verificarsi di altri eventi
- In alcuni casi (p. es. se non so da quale socket arriverà il prossimo input) è necessario un meccanismo per evitare il blocking indefinito
- Nell'interfaccia dei socket questo meccanismo è fornito dalla primitiva **`select`**, che consente di:
 - attendere il verificarsi di un qualsiasi evento (p. es. arrivo di dati) tra quelli cui si è interessati
 - attendere al massimo per un certo tempo

Select: Eventi controllabili

Un evento può corrispondere al verificarsi di

- una condizione che garantisce di poter eseguire una determinata operazione **senza bloccarsi**:
 - La condizione di **Leggibilità** di un socket è l'OR delle seguenti:
 - » vi è almeno una richiesta di connessione pendente (socket passivi)
 - » sono disponibili dati nel buffer di ricezione (socket connessi)
 - » si è verificato un errore sul socket - p. es. chiusura connessione
 - La condizione di **Scrivibilità** di un socket è l'OR delle seguenti:
 - » il socket è connesso e il buffer di trasmissione del socket non è pieno
 - » si è verificato un errore sul socket - p. es. chiusura connessione
- un'**eccezione**:
 - presenza di dati urgenti

SELECT: Sintassi

```
int select (
int nfd, _____
fd_set *readfd, _____
fd_set *writefd, _____
fd_set *exceptfd, _____
struct timeval *timeout _____
)
```

Numero di socket su cui fare la selezione

Insiemi di socket di cui interessa il verificarsi di condizioni di leggibilità, scrivibilità e di eccezioni

Tempo massimo di bloccaggio (se NULL il tempo è infinito)

Select: Funzionamento

- Il chiamante si blocca in attesa che qualcuno dei socket da selezionare diventi selezionabile.
- Non appena questo si verifica, la select si sblocca e:
 - rimpiazza i 3 insiemi di socket da selezionare con i corrispondenti sottoinsiemi di socket selezionati
 - restituisce il numero complessivo di socket selezionati
- Se nessun socket è diventato selezionabile entro il tempo massimo di bloccaggio, la select si sblocca e restituisce il valore 0.

Select: Macro per operare sugli insiemi di socket

- **FD_SET(int s, fd_set *fd)**

(Inserisce il socket s nell'insieme puntato da fd)

- **FD_CLR(int s, fd_set *fd)**

(Elimina il socket s dall'insieme puntato da fd)

- **FD_ZERO(fd_set *fd)**

(Azzera l'insieme puntato da fd)

- **int FD_ISSET(int s, fd_set *fd)**

(Restituisce valore TRUE se il socket s appartiene all'insieme puntato da fd)

Esempio: ricezione di datagram

```
FD_ZERO(&cset);
FD_SET(s1, &cset); FD_SET(s2, &cset);
tval.tv_sec = 15; tval.tv_usec = 0;
if ((n = select(FD_SETSIZE, &cset, NULL, NULL, &tval)) == -1)
    err_fatal("select() failed");
if (n>0)
{
    if (FD_ISSET(s1, &cset))
        s = s1;
    else
        s = s2;
    fromlen = sizeof(struct sockaddr_in);
    n=recvfrom(s,buf,BUFLen,0,(struct sockaddr *)&from,&fromlen);
    if (n != -1)
        printf("Received message from socket %d\n", s);
    else printf("Error in receiving response\n");
}
else printf("No response after %d seconds\n",tval.tv_sec);
```

Informazioni sullo stato di un socket

- `int getsockname (int s, struct sockaddr * addr, int * addrlen)`

(Scrive in `addr` l'indirizzo associato al socket `s`)

- `int getpeername (int s, struct sockaddr * addr, int * addrlen)`

(Scrive in `addr` l'indirizzo del socket cui `s` è connesso)

- `int getsockopt (int s, int level, int opt_name, char *opt_val, int *val_len)`

(Scrive in `opt_val` il valore dell'opzione specificata dalla coppia `(opt_name, level)`)

Altre funzioni di utilità

- Conversione degli interi da ordine di rete a ordine dell'host e viceversa:
 - `int htons (int hostaddr)`
 - `unsigned long htonl (long hostaddr)`
 - `int ntohs (int netaddr)`
 - `unsigned long ntohl (long netaddr)`
- Conversione da notazione "dotted decimal" a notazione di rete e viceversa:
 - `unsigned long inet_addr (char * string)`
 - `char * inet_ntoa (struct in_addr)`

- Impostazioni delle opzioni di un socket:

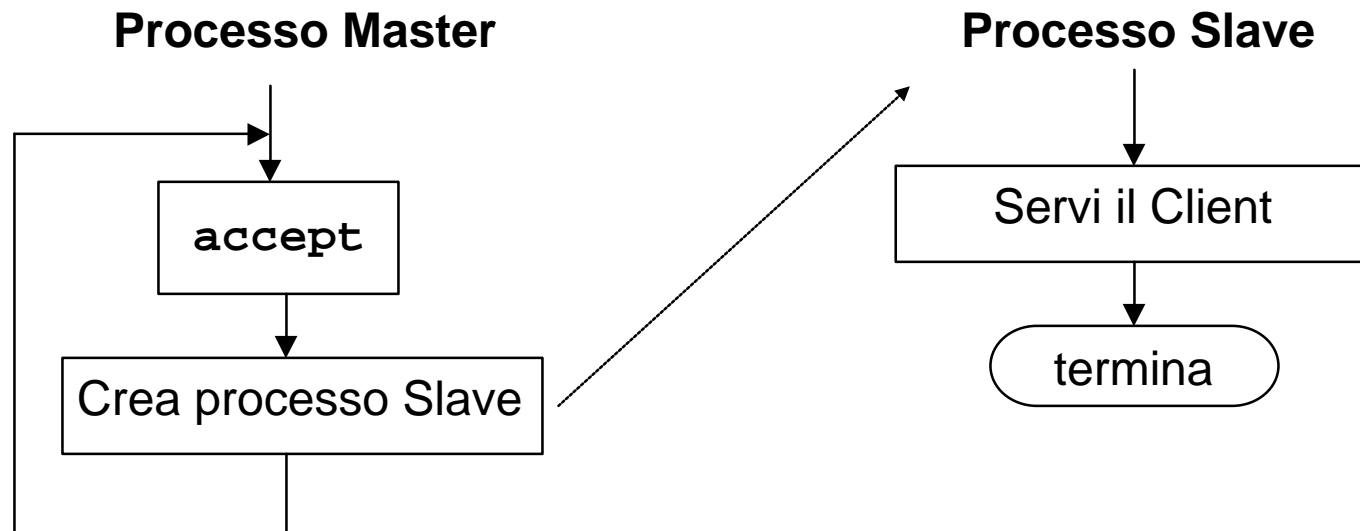
```
int setsockopt(int socket, int level, int  
    op_name, char *opt_val, int optlen)
```

- Accesso a informazioni di rete:

- gethostbyaddr
- gethostbyname
- gethostname
- getprotobyname
- getprotobynumber
- getservbyname
- getservbyport

Server concorrenti

- Sono utilizzati per ridurre i tempi di risposta del server
- Ogni volta che viene accettata una connessione viene lanciato un nuovo processo per servirla



Esempio: Server Concorrente (UNIX)

```
/* creazione socket s, bind, listen */
...
for (;;)
{
    addrlen = sizeof(struct sockaddr_in);
    new = accept(s, (struct sockaddr *) &c_addr, &addrlen);
    if (new == INVALID_SOCKET) err_fatal("accept() failed");

    if((childpid=fork())<0) err_fatal("fork() failed");
    else if (childpid > 0)
    {
        /* processo padre */
        close(new);          /* chiudo nuovo socket */
    }
    else
    {
        /* processo figlio */
        close(s);           /* chiudo socket del padre */
        service(new);      /* servo il client */
    }
}
}
```

Aspetti specifici Unix

- I file di include da utilizzare sono:

```
#include <sys/types.h>
#include <sys/socket.h>
#include <sys/time.h>          /* serve per select */
#include <netinet/in.h>       /* rete internet */
```

- Per accedere ad un socket stream si possono usare anche le system call **read** e **write**.

Aspetti specifici Windows Sockets

- L'interfaccia Windows Sockets implementa tutte le principali funzioni dei socket BSD4.3
- Possiede in più un altro insieme di primitive analoghe, basate su **interazione asincrona** (orientate ai messaggi).
- Non permette di trattare omogeneamente file e socket.
- Richiede in più (a causa dell'uso di librerie dinamiche):
 - Inizializzazione dell'interfaccia con aggancio alla DLL all'avvio dell'applicazione (WSAStartup())
 - Rilascio delle risorse e sgancio dalla DLL al termine dell'applicazione (WSACleanup())

Uso di Windows Sockets

```
...  
#include <winsock.h>  
  
...  
void main()  
{  
...  
/* Inizializzazione Windows Sockets */  
... WSStartup(...);  
  
/* Uso delle primitive bsd4.3 */  
  
/* Rilascio risorse Windows Sockets */  
WSACleanup();  
}
```

Interfaccia socket Java

- È basata sui socket BSD
- Maschera le primitive BSD con classi e metodi di più alto livello:
 - package java.net
 - » **class Socket**
 - Realizza un socket di tipo STREAM **connesso**
 - » **class ServerSocket**
 - Realizza un socket di tipo STREAM **passivo**
 - » **class DatagramSocket**
 - Realizza un socket di tipo DATAGRAM

Esempio: Ricezione dati da server

```
import java.io.*;
import java.net.*;

class SocketTest
{ public static void main(String[] args)
  { try
    { Socket s = new Socket("cclix1.polito.it", 13);
      DataInputStream is =
      new DataInputStream(s.getInputStream());
      boolean more = true;
      while (more)
      { String str = is.readLine();
        if (str == null)
          more = false;
        else
          System.out.println(str);
        }
      } catch(IOException e) {System.out.println("Error"+e);
    }
  }
```

Esempio1: servizio “echo” (stream / character-oriented)

- **Server:** una volta instaurata una connessione, fa l'eco di tutti i caratteri ricevuti.
- **Client:** si connette al server, invia i caratteri acquisiti da tastiera e visualizza le risposte del server.
- Il Client termina quando termina lo standard input.

Esempio2: servizio “echo” (stream / line-oriented)

- **Server:** una volta instaurata una connessione, fa l'eco di tutti i messaggi ricevuti.
- **Client:** si connette al server, invia messaggi acquisiti da tastiera e visualizza le risposte del server.
- Il Server riconosce due messaggi speciali:
 - “close” Provoca la chiusura della connessione
 - “stop” Provoca la terminazione del Server
- Il newline viene usato come terminatore di messaggio.

Migliorie su echo client e server

- Modificare il client in modo che rimanga bloccato in attesa dell'eco al massimo per un certo tempo (usare la select)
- Modificare il server in modo che possa servire più client contemporaneamente (usare la select e/o la fork)

Esempio 3: Servizio “echo” (datagram)

- **Server:** Fa l’eco di tutti i datagram ricevuti.
- **Client:** Invia come datagram i messaggi acquisiti da tastiera e visualizza le risposte del server.