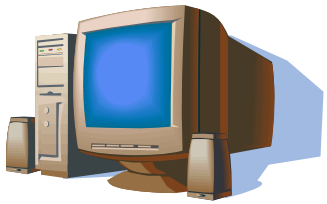
I Socket di Berkeley

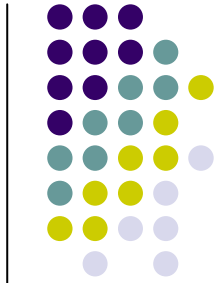


di
Francesco Licandro

Dipartimento di Ingegneria Informatica e delle Telecomunicazioni
Università degli studi di Catania

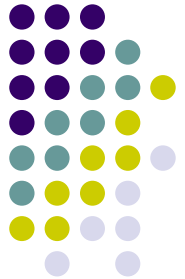
A.A. 2004-2005

Programmazione su rete



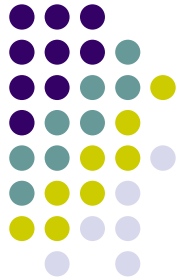
- Le applicazioni di rete consistono di diverse componenti, in esecuzione su macchine differenti (in generale), che operano in modo indipendente e che possono scambiare informazioni.
  - Forniscono i servizi di alto livello utilizzati dagli utenti
- Le applicazioni sono processi comunicanti e distribuiti.
  - La comunicazione avviene utilizzando i servizi offerti dal sottosistema di comunicazione
    - Comunicazione a scambio di messaggi.
  - La cooperazione può essere implementata secondo varimodelli.

Modello Client-Server



- Client:
  - E' l'applicazione che richiede il servizio
    - Inizia il contatto con il server
- Server:
  - E' l'applicazione che fornisce il servizio richiesto.
    - Attende di essere contattato dal client
- Come fa un'applicazione ad identificare in rete l'altra applicazione con la quale vuole comunicare?
  - Indirizzo IP dell'host su cui è in esecuzione l'altra applicazione
  - Numero di porta
    - L'host ricevente può così determinare a quale applicazione locale deve essere consegnato il messaggio.

Il paradigma Client-Server (C/S)



- Il chiamato è il server:
  - deve aver divulgato il proprio indirizzo
  - resta in attesa di chiamate
  - in genere viene contattato per fornire un servizio
- Il chiamante è il client:
  - conosce l'indirizzo del partner
  - prende l'iniziativa di comunicare
  - usufruisce dei servizi messi a disposizione dal server

Il concetto di indirizzo

- Una comunicazione può quindi essere identificata attraverso la quintupla:

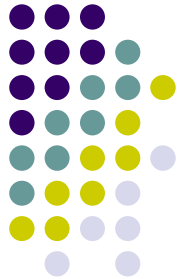
{protocol, local-addr, local-port, foreign-addr, foreign-port}

- Una coppia {addr, process} identifica univocamente un terminale di comunicazione (end-point).
- Nel mondo IP, ad esempio:
  - local-addr e foreign-addr rappresentano indirizzi IP
  - local-port e foreign-port rappresentano numeri di porta

Server concorrenti ed iterativi

- Server iterativo
  - Gestisce una richiesta client per volta
- Server ricorsivo
  - Gestisce più richieste client contemporaneamente
    - Creato processo/thread di servizio in grado di gestire la richiesta client.

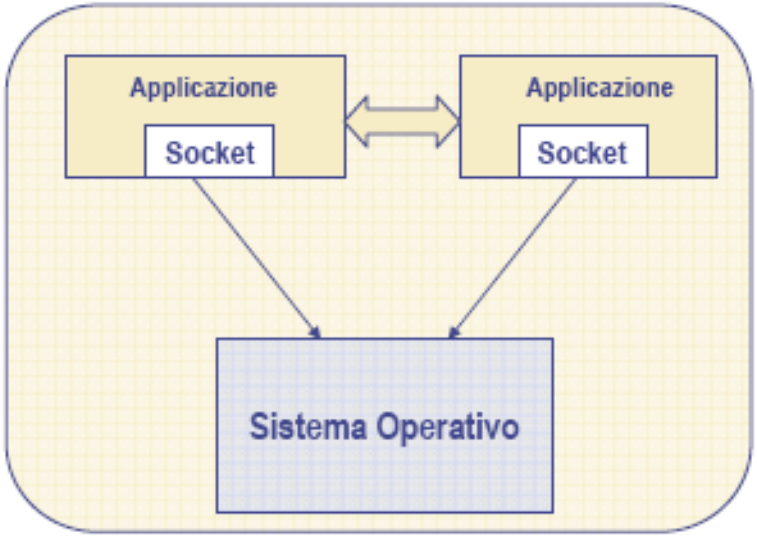
Interazione livello applicativo-trasporto



- Le applicazioni client e server utilizzano TCP o UDP come protocollo di trasporto
- Il software di gestione del protocollo di trasporto si trova all'interno del sistema operativo
- Il software dell'applicazione si trova all'esterno del sistema operativo
- Per poter comunicare 2 applicazioni devono interagire con i rispettivi sistemi operativi.
  - Ogni applicazione deve chiedere al suo SO di inviare o ricevere dati tramite la rete.

Come???

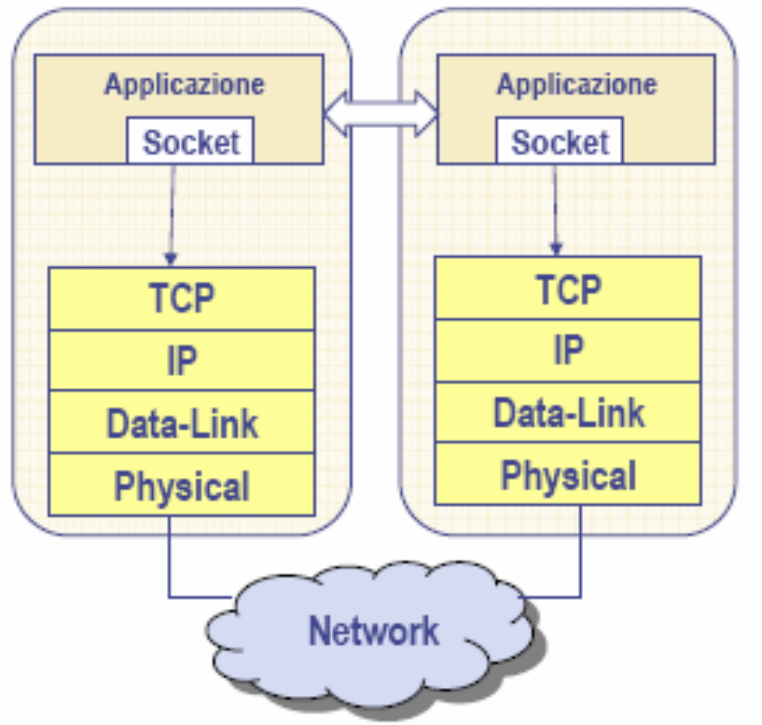
Comunicazione locale



Esempio di comunicazione locale

Due applicazioni, localizzate sulla stessa macchina, scambiano dati tra di loro utilizzando l'interfaccia delle socket.
Le socket utilizzate a questo scopo vengono comunemente definite Unix-domain socket.

Comunicazione remota



Esempio di comunicazione remota.

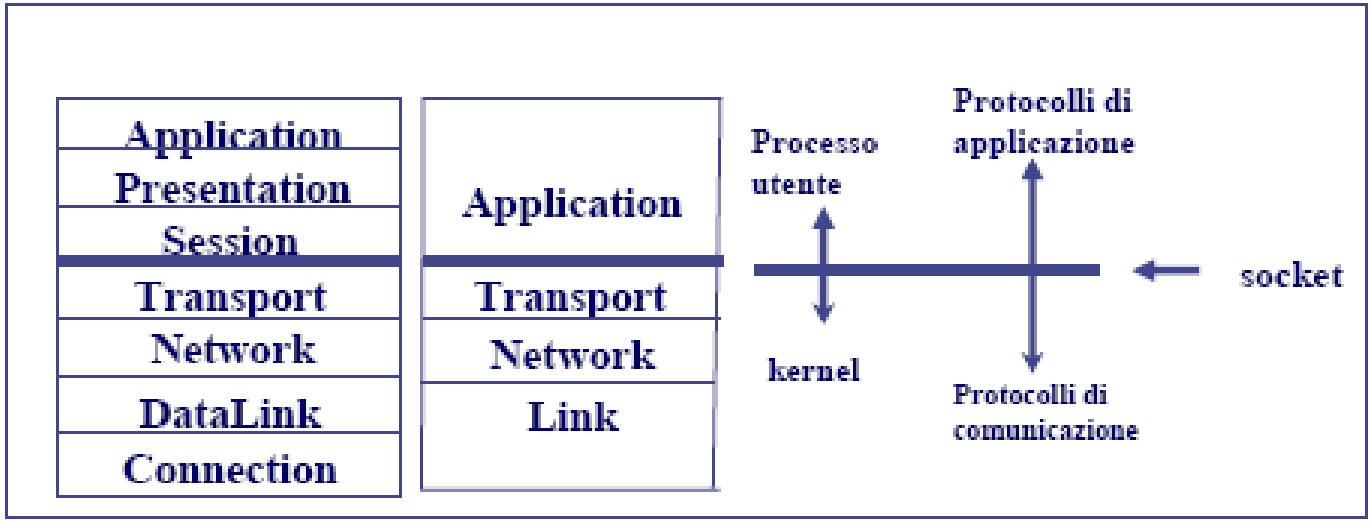
Anche due applicazioni situate su macchine distinte possono scambiare informazioni secondo gli stessi meccanismi. Così funzionano telnet, ftp, ICQ, Napster.

Application Programming Interface (API)

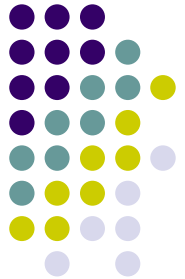
- Si utilizza un meccanismo che svolge il ruolo di interfaccia fra il Sistema Operativo e l'applicazione di rete:
  - Application Programming Interface (API)
    - Insieme delle funzioni che possono essere invocate per effettuare chiamate di sistema
      - Le interfacce delle funzioni sono indipendenti dalla piattaforma
    - La più diffusa è la **Berkeley Sockets**.

Application Programming Interface

- Nei sistemi Unix definiscono la divisione fra kernel space e user space.

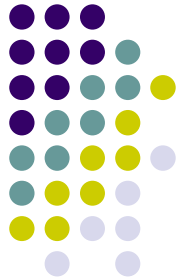


Socket di Berkeley



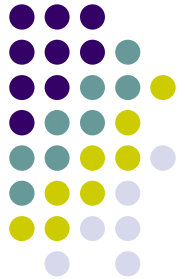
- E’ una interfaccia locale all’host, controllata dal sistema operativo, creata/posseduta dall’applicazione tramite la quale il processo applicativo può inviare/ricevere messaggi a/da un altro processo applicativo (locale o remoto).
  - Creato dinamicamente dal SO su richiesta del processo applicativo utente
  - Persiste solo durante l’esecuzione dell’applicazione
  - Il suo ciclo di vita è simile a quello di un file:
    - Apertura
    - Collegamento ad un endpoint
    - Lettura/scrittura
    - Chiusura

Le origini



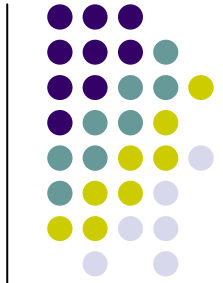
- Inizialmente nasce in ambiente UNIX
  - Negli anni ‘80 la Advanced Research Project Agency finanziò l’università di Berkeley per implementare la suite TCP/IP nel sistema operativo Unix.
  - I ricercatori di Berkeley svilupparono il set originario di funzioni che fu chiamato *interfaccia socket*.
- Rappresentano una estensione delle API di UNIX per la gestione dell’I/O su periferica standard (files su disco, stampanti, etc).
- Rappresentano lo standard di riferimento per tutta la programmazione su reti.

Interazione tra Applicazione e SO



- L’applicazione chiede al sistema operativo di utilizzare i servizi di rete
- Il sistema operativo crea un socket e lo restituisce all’applicazione
  - restituito un socket descriptor
- L’applicazione utilizza il socket
  - Open, Read,Write,Close.
- L’applicazione chiude il socket e lo restituisce al sistema operativo

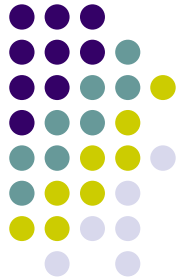
Tipi di Socket



- **SOCK_STREAM**
  - Una socket STREAM stabilisce una **connessione**.
  - La comunicazione è **affidabile**, **bidirezionale** e i byte sono consegnati in **sequenza**.
  - (presenza di meccanismi out-of-band)

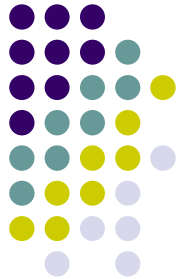
- **SOCK_DGRAM**
  - Una socket DATAGRAM non stabilisce alcuna connessione (**connectionless**).
  - Ogni messaggio è indirizzato individualmente a un destinatario.
  - **NON** c'è garanzia di consegna del messaggio.
  - Non è garantito **l'ordine** di consegna dei messaggi.

Comunicazione “Connection-Oriented”



- In una comunicazione dati Connection-Oriented, i due endpoints dispongono di un canale di comunicazione che:
  - trasporta flussi
  - è affidabile
  - è dedicato
  - preserva l’ordine delle informazioni

Progettazione di un Server TCP

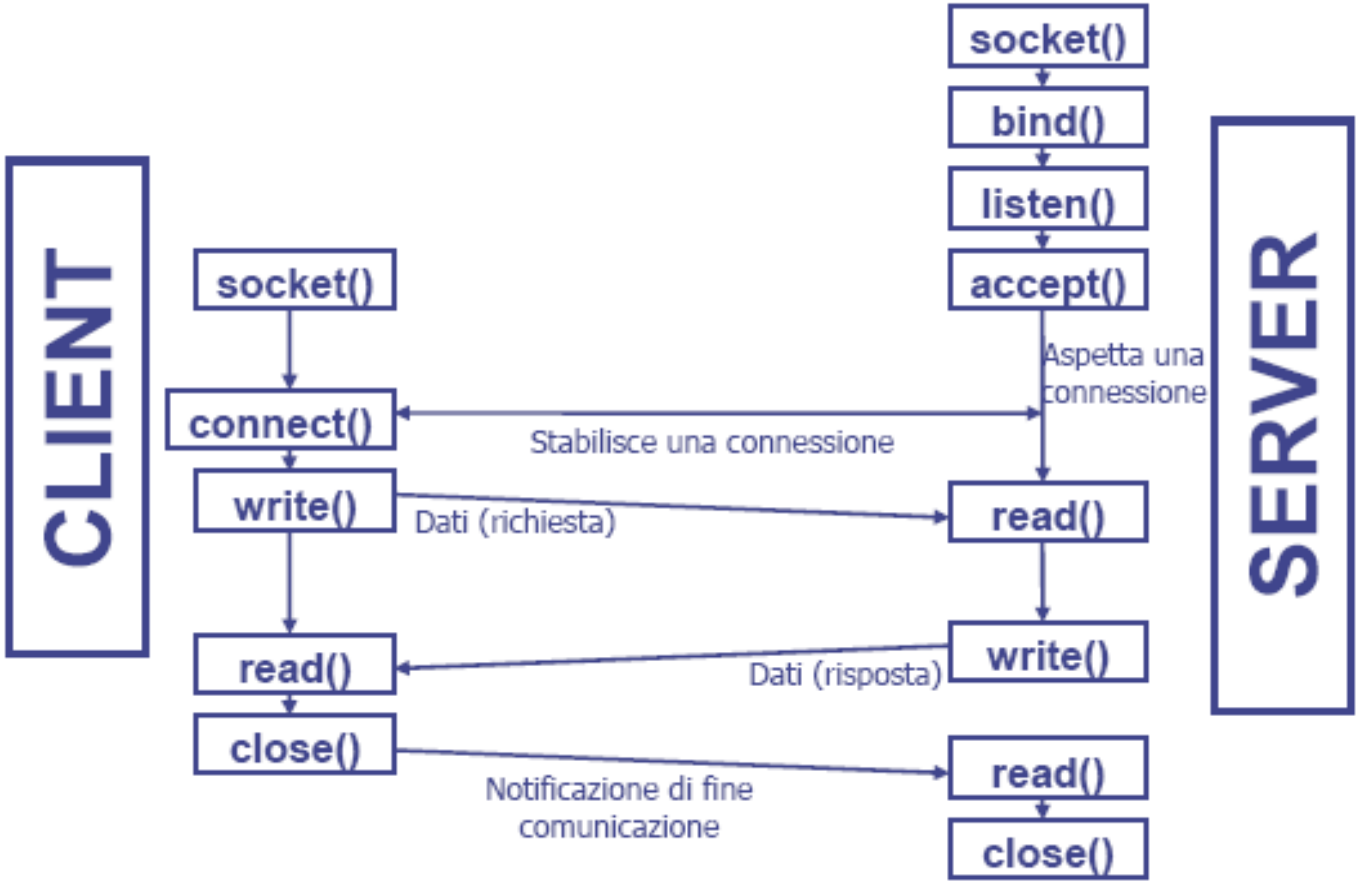


- Creazione di un endpoint
  - Richiesta al sitema operativo
- Collegamento dell'endpoint ad una porta
  - Ascolto sulla porta
    - Processo sospeso in attesa
- Accettazione della richiesta di un client
- Letture e scritture sulla connessione
- Chiusura della connessione

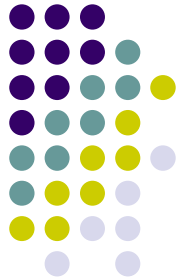
Progettazione di un Client TCP

- Creazione di un endpoint
  - Richiesta al sistema operativo
- Creazione della connessione
  - Implementa open di TCP (3-way handshake)
- Lettura e scrittura sulla connessione
  - Analogo a operazioni su file in Unix
- Chiusura della connessione
  - Implementa close di TCP (4-way handshake)

Comunicazione “Connection-Oriented”

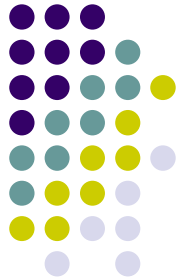


Comunicazione “Connectionless o Datagram”



- In una comunicazione dati Datagram il canale:
  - trasporta messaggi
  - non è affidabile
  - è condiviso
  - non preserva l’ordine delle informazioni

Progettazione di un Server UDP

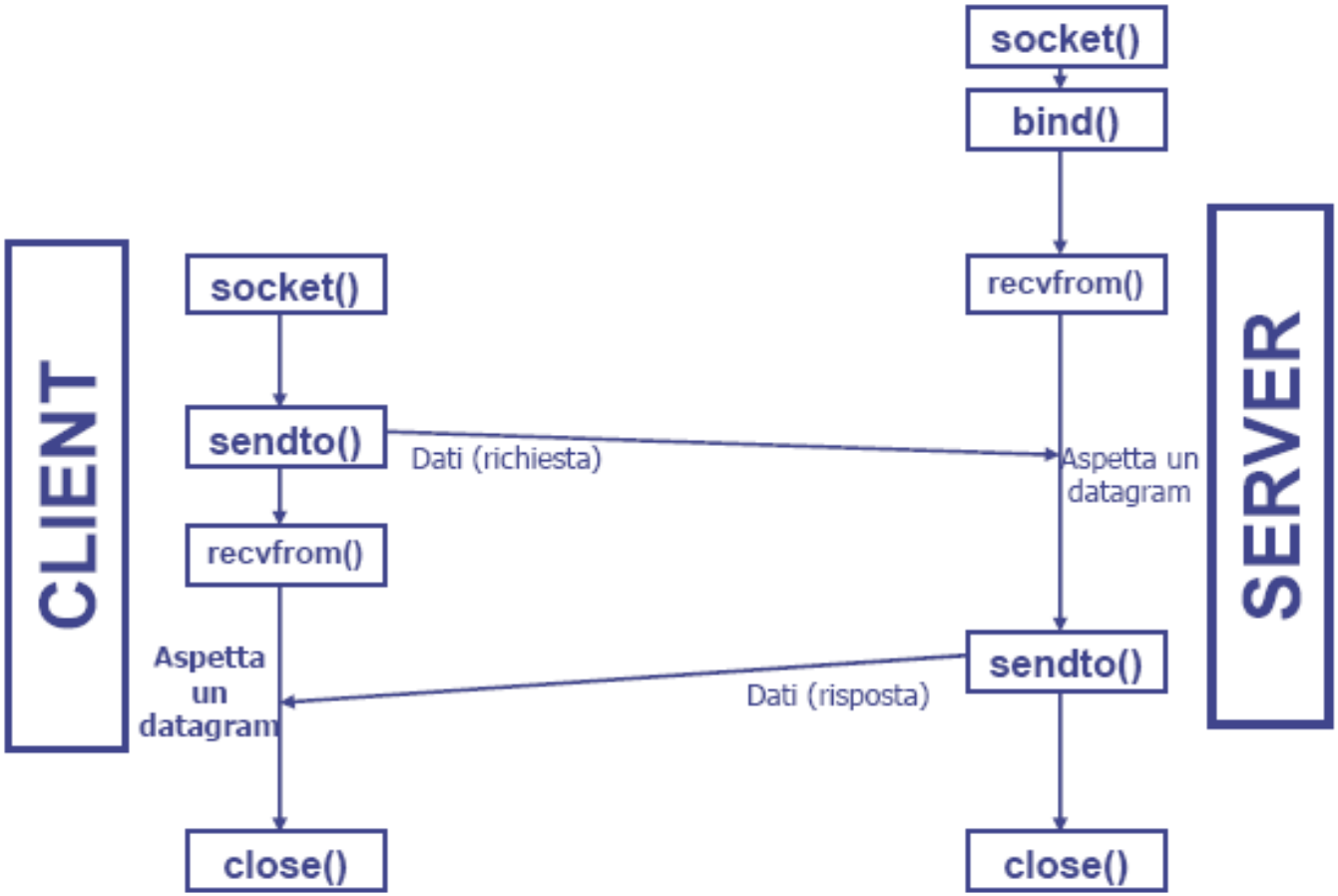


- Creazione di un endpoint
  - Richiesta al sistema operativo
- Collegamento dell'endpoint ad una porta
  - open passiva in attesa di ricevere datagram
- Ricezione ed invio di datagram
- Chiusura dell'endpoint

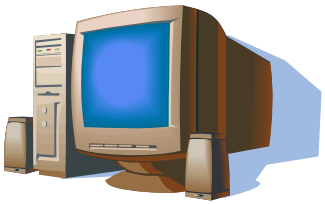
Progettazione di un Client UDP

- Creazione di un endpoint
  - Richiesta al sistema operativo
- Invio e ricezione di datagram
- Chiusura dell'endpoint

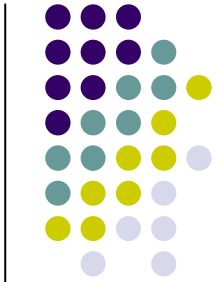
Struttura di un'applicazione UDP



La programmazione dei Socket



Le strutture dati per le socket: il trattamento degli indirizzi

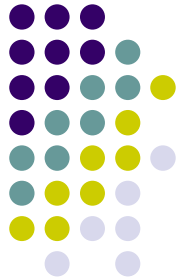


```
<sys/socket.h>
struct sockaddr {
    u_short         sa_family;      /* address family: AF_xxx value */
    char            sa_data[14];    /* up to 14 bytes of protocol-specific
    address */
};

<netinet/in.h>
    struct in_addr {
    u_long          s_addr;         /* 32-bit netid/hostid network byte ordered
    */
};

struct sockaddr_in {
    short           sin_family;     /* AF_INET */
    u_short         sin_port;       /* 16-bit port number network byte ordered
    */
    struct in_addr sin_addr;
    char            sin_zero[8];            /* unused */
};
```

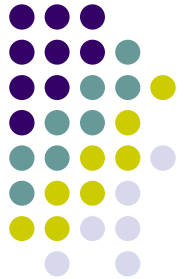
Indirizzo di socket



- Definizioni del C e definizioni dei tipi dati (typedef) utilizzati in tutto il sistema sono

Tipo di dati in C	**4.3BSD**
unsigned char	u_char
unsigned short	u_short
unsigned int	u_int
usigned long	u_long

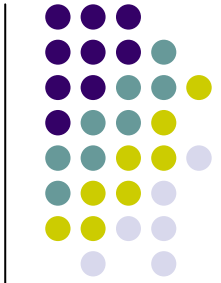
Funzione *socket()*



- Prima funzione eseguita dal client e dal server
- Crea un endpoint
- Restituisce
  - -1 se la creazione non è riuscita
  - il descrittore del socket se la creazione è riuscita

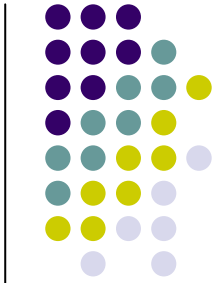
```
#include <sys/socket.h>
int socket(int family, int type, int protocol);
```

Parametri della funzione *socket()*



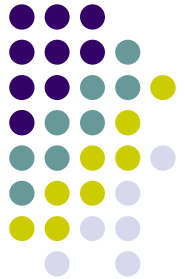
- int family specifica la famiglia di protocolli da usare:
  - **AF_INET IPv4**
  - AF_INET6 IPv6
  - AF_LOCAL prot. locale (client e server sullo stesso host)
  - Altri
- int type identifica il tipo di socket
  - SOCK_STREAM per uno stream di dati (TCP)
  - SOCK_DGRAM per datagrammi (UDP)
  - SOCK_RAW per applicazioni dirette su IP

Parametri della funzione *socket()*



- int protocol
  - 0 per specificare il protocollo di default indotto dalla coppia family e type (tranne che per SOCK_RAW)
  - AF_INET + SOCK_STREAM determinano una trasmissione TCP (IPPROTO_TCP)
  - AF_INET + SOCK_DGRAM determinano una trasmissione UDP (IPPROTO_UDP)

Funzione *socket()*



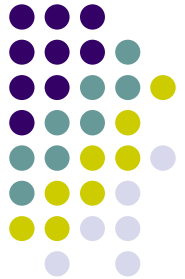
- Esempio di invocazione della funzione **socket():**

```
if ((sd = socket(AF_INET, SOCK_STREAM, 0) < 0)
{
        perror("socket");
        exit(1);
```

- Della quintupla, dopo la chiamata socket(), resta specificato solo il primo campo:

{**protocollo**, indirizzo locale, porta locale, indirizzo remoto, porta remota}

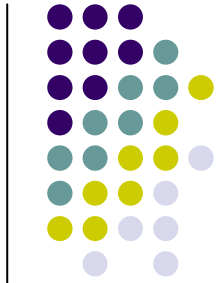
Funzione *connect()*



```
#include <sys/types.h>
#include <sys/socket.h>
int connect(int sd, const SA *servaddr,
socklen_t addrlen);
```

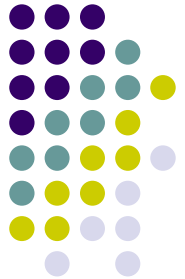
- Permette ad un client di aprire una connessione con il server
  - Il SO sceglie una porta effimera ed effettua una open attiva
  - la funzione termina solo dopo che la connessione è stata creata
- Restituisce
  - -1 in caso di errore
  - 0 se la connessione è stata creata

Parametri della funzione *connect()*



- sd è il socket descriptor
- servaddr è un puntatore all'indirizzo dell'endpoint a cui ci si vuole collegare
  - indirizzo IP + numero di porta
  - puntatore di tipo sockaddr (SA)
- addrlen è la lunghezza in byte di servaddr
- In caso di errore restituisce
  - ETIMEDOUT è scaduto il time out del SYN
  - ECONNREFUSED il server ha rifiutato il SYN
  - EHOSTUNREACH errore di indirizzamento

Funzione *connect()*



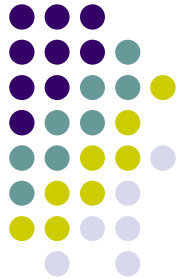
- Esempio di invocazione della funzione **connect():**

```
if (connect(sd, (struct sockaddr *)&servaddr,
sizeof(servaddr)) < 0) {
perror("connect");
exit(1);
}
```

- Della quintupla, dopo la chiamata connect(), restano specificati tutti i campi relativi agli indirizzi:

- {**protocollo**, **indirizzo locale, porta locale**, **indirizzo remoto, porta remota**}

Funzione *bind()*



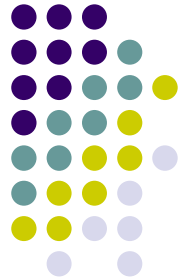
- Serve a far sapere al kernel a quale processo vanno inviati i dati ricevuti dalla rete
- Permette di assegnare uno specifico indirizzo al socket
  - se non si esegue la bind il S.O. assegnerà al socket unaporta effimera ed uno degli indirizzi IP dell'host dipende dall'interfaccia utilizzata
  - in genere eseguito solo dal server per usare una porta prefissata

Parametri della funzione *bind()*

```
#include <sys/types.h>
#include <sys/socket.h>
int bind(int sd, const SA*myaddr, socklen_t addrlen);
```

- L'oggetto puntato da myaddr **può** specificare
  - l'indirizzo IP
  - il numero di porta (indirizzo locale)
- Un campo non specificato è messo a 0
  - Se la porta è 0 ne viene scelta una effimera
  - Se l'indirizzo IP è INADDR_ANY (0) il server accetterà richieste su ogni interfaccia
    - quando riceve un segmento SYN utilizza come indirizzo IP quello specificato nel campo destinazione del segmento
- La funzione restituisce un errore se l'indirizzo non è utilizzabile
  - EADDRINUSE

Funzione *bind()*



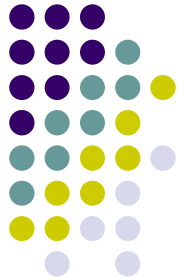
- Esempio di invocazione della funzione **bind():**

```
name.sin_family = AF_INET
name.sin_port = htons(0);
name.sin_addr.s_addr = htonl(INADDR_ANY);
if (bind(sd, (struct sockaddr *)&name, sizeof(name)) < 0) {
perror("bind");
exit(1);}
```

- Della quintupla, dopo la chiamata bind(), restano specificati il secondo ed il terzo campo, cioè gli estremi locali della comunicazione:

{**protocollo**, **indirizzo locale, porta locale**, indirizzo remoto, porta remota}

Funzione *listen()*

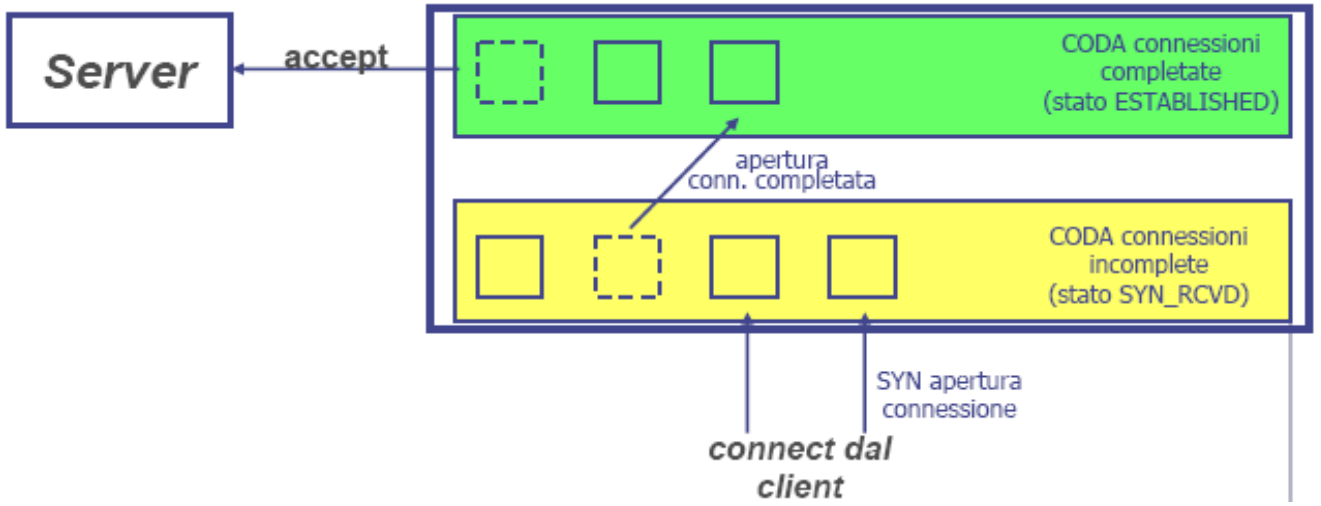


- Utilizzata per rendere un socket passivo

```
#include <sys/socket.h>
int listen(int sd, int backlog);
```

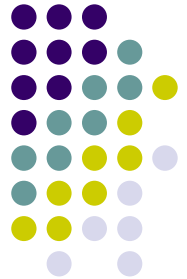
- Specifica quante connessioni possono essere accettate e messe in attesa di essere servite
  - le connessioni sono accettate o rifiutate dal S.O. senza interrogare il server

Backlog



- Nel backlog ci sono sia le richieste di connessione in corso di accettazione che quelle accettate ma non ancora passate al server
- La somma degli elementi in entrambe le code non può superare il *backlog*

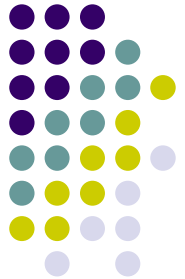
Funzione *accept()*



```
#include <sys/types.h>
#include <sys/socket.h>
int accept(int sd, SA*cliaddr, socklen_t* addrlen);
```

- Permette ad un server di prendere la prima connessione completata dal backlog
  - Se il backlog è vuoto il server rimane bloccato sulla chiamata a funzione fino a quando non viene accettata una connessione
- Restituisce
  - -1 in caso di errore
  - un nuovo descrittore di socket assegnato automaticamente dal S.O. e l'indirizzo del client
    - la porta del nuovo descrittore è effimera

Socket di Ascolto e Socket Connesso



- Il server quindi utilizza due socket diversi per ogni connessione con un client
  - il socket di ascolto (listening socket) è quello creato dalla funzione **socket()**
    - utilizzato per tutta la vita del processo
    - in genere usato solo per accettare richieste di connessione
  - il socket connesso (connected socket) è quello creato dalla funzione **accept()**
    - usato solo per la connessione con un certo client
    - usato per lo scambio dei dati con il client
- I due socket identificano due connessioni distinte

Funzione *close()*

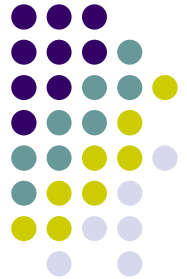
```
#include <unistd.h>
int close(int sd);
```

- Marca il descrittore come chiuso
- il processo non può più utilizzare il descrittore ma la connessione non viene chiusa subito
  - TCP continua ad utilizzare il socket trasmettendo i dati che sono eventualmente nel buffer
- Restituisce
  - -1 in caso di errore 0 se OK
- Più processi possono condividere un descrittore
  - un contatore mantiene il numero di processi associati al descrittore
  - la procedura di close della connessione viene avviata solo quando il contatore arriva a 0

Ricevere ed inviare i dati

- Una volta utilizzate le precedenti chiamate, la “connessione” è stata predisposta.
- La quintupla risulta completamente impostata.
- A questo punto chiunque può inviare o ricevere dati.
- Per questo, è necessario aver concordato un protocollo comune.

Le system-call **send()** e **sendto()**

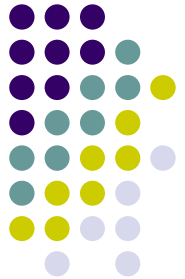


- send()e sendto() si utilizzano per inviare dati verso l'altro terminale di comunicazione.

```
int send(int sockfd, char *buff, int nbytes, int
   flags);
int sendto(int sockfd, char *buff, int nbytes, int
   flags, struct sockaddr *to, int addrlen);
```

- ***sockfd*** è il descrittore restituito dalla chiamata socket().
- ***buff*** punta all'inizio dell'area di memoria contenente i dati da inviare.
- ***nbytes*** indica la lunghezza in bytes del buffer, e quindi, il numero di bytes che devono essere inviati.
- ***to*** e ***addrlen*** indicano l'indirizzo del destinatario, con la sua lunghezza.
- ***flags*** abilita particolari opzioni. In generale è pari a 0.
- entrambe restituiscono il numero di bytes effettivamente inviati.

Le system-call **recv()** e **recvfrom()**



- recv()e recvfrom() si utilizzano per ricevere dati dall'altro terminale di
- comunicazione.

```
int recv(int sockfd, char *buff, int nbytes, int flags);
int recvfrom(int sockfd, char *buff, int nbytes, int flags,
   struct sockaddr *from, int *addrlen);
```

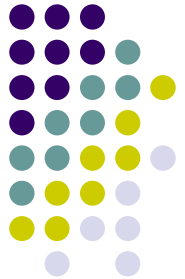
- ***sockfd*** è il descrittore restituito dalla chiamata socket().
- ***buff*** punta all'inizio dell'area di memoria in cui devono essere ricopiati i dati ricevuti.
- ***nbytes*** è un parametro di ingresso che indica la lunghezza del buffer.
- ***from*** e ***addrlen*** contengono, dopo la chiamata, l'indirizzo del mittente con la sua lunghezza.
- ***flags*** abilita particolari opzioni. In generale è pari a 0.
- entrambe restituiscono il numero di bytes ricevuti (minore o uguale a nbytes).
- in assenza di dati da leggere, la chiamata è bloccante.

Il marshalling dei parametri

- Una serie di funzioni è stata prevista, nell'ambito della comunicazione su Internet, proprio a questo scopo.
- Vanno sotto il nome di Byte Ordering Routines.
- Sui sistemi che adottano la stessa convenzione fissata per Internet, queste routines sono implementate come "null-macros" (funzioni 'vuote').

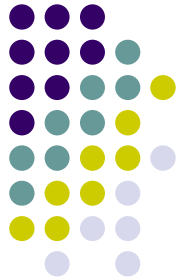
```
u_long htonl(u_long hostlong);  /* host to net long */
u_short htons(u_short hostshort);     /* host to net
   short */
u_long ntohl(u_long netlong);   /* net to host long */
u_short ntohs(u_short netshort);      /* net to host
   short */
```

Operazioni sui buffer



- Le classiche funzioni standard del C per operare sulle stringhe (strcpy(), strcmp(), etc.) non sono adatte per operare sui buffer di trasmissione e ricezione.
- Esse infatti ipotizzano che ogni stringa sia terminata dal carattere null e, di conseguenza, non contenga caratteri null.
- Ciò non può essere considerato vero per i dati che si trasmettono e che si ricevono sulle socket.
- E' stato necessario quindi prevedere altre funzioni per le operazioni sui buffer.

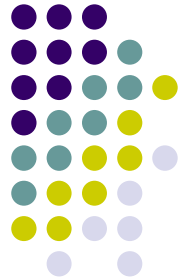
Operazioni sui buffer



```
bcopy(char *src, char *dest, int nbytes);
bzero(char *dest, int nbytes);
int bcmp(char *ptr1, char *ptr2, int nbytes);
```

- bcopy() copia nbytes bytes dalla locazione src alla locazione dest.
- bzero() imposta a zero nbytes a partire dalla locazione dest.
- bcmp() confronta nbytes bytes a partire dalle locazioni ptr1 e ptr2, restituendo 0 se essi sono identici, altrimenti un valore diverso da 0.

Conversione di indirizzi

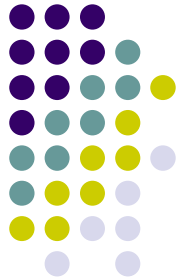


- Poiché spesso nel mondo Internet gli indirizzi vengono espressi in notazione dotted-decimal (p.es. 192.168.1.1), sono state previste due routines per la conversione tra questo formato e il formato in_addr.

```
u_long      inet_addr(char *ptr);
char        *inet_ntoa(struct in_addr inaddr);
```

- inet_addr() converte una stringa (C-style) dalla notazione dotted-decimal alla notazione in_addr (che è un intero a 32 bit).
- inet_ntoa() effettua la conversione opposta.

Nomi logici dei nodi e indirizzi fisici

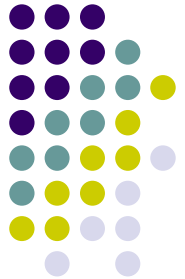


- struct hostent *__gethostbyname__(const char *name);
- struct hostent *__gethostbyaddr__(const char *addr, int len, int type);
- **gethostbyname** riceve in ingresso il nome logico di un host Internet e restituisce una struttura **hostent** (contenente l'indirizzo fisico IP del nodo).
- **gethostbyaddr** riceve in ingresso l'indirizzo fisico di un host Internet e restituisce una struttura **hostent** (contenente in nome della macchina).

- **Nomi logici dei nodi e indirizzi fisici**

```
struct hostent {
char *h_name;         /* canonical name of host */
char **h_aliases;     /* alias list */
int h_addrtype;       /* host address type */
int h_length;         /* length of address */
char **h_addr_list;    /* list of addresses */
#define h_addr h_addr_list[0] /* address, for backward
   compatiblity */
};
```

Opzioni di socket

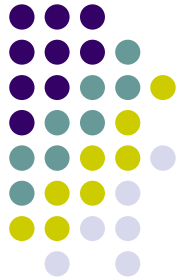


- getsockopt(), setsockopt()

```
int getsockopt(int sockfd, int livello, int nomeopz,
   char *opzval, int *lunghopz)
int setsockopt(int sockfd, int livello, int nomeopz,
   char *opzval, int *lunghopz)
```

- livello: specifica l'entità che deve interpretare l'opzione nel sistema.

Opzioni di socket



livello	nimeopz	*get*	*set*	*descrizione*	*tipo di dato*
IPPROTO_IP	IP_OPTIONS	*	*	Opzioni di intestazione di IP	
IPPROTO_TCP	TCP_MAXSEG	*		Legge la max dim. del segmento in TCP	int
	TCP_NODELAY	*	*	Non ritarda la trasmissione per riunire i pacchetti	int
SOL_SOCKET	SO_BROADCAST	*	*	Permette l'invio in broadcast	int
	SO_DEBUG	*	*	Attiva il debug	int
	SO_DONTROUTE	*	*	Uso solo indirizzo di interfaccia	int
	SO_ERROR	*		leggo lo stato di errore	int
	SO_KEEPALIVE	*	*	tiene vive le connessioni	int
	SO_LINGER	*	*	indugia sulla chiusura se ci sono dati	int
	SO_RCVBUF	*	*	Riceve le dim. del buffer di ricezione	int
	SO_SNDBUF	*	*	Riceve le dim. del buffer di trasm.	int
	SO_TYPE	*		legge il tipo di socket	int
	…				