# Programmazione di Rete

Socket

Ing. Carlo Nobile



## Sommario

- Berkeley's socket
- Socket UDP: funzioni fondamentali
- Esempio applicazione:
  - Listener
  - Sender
- Socket non bloccanti
- Indirizzo IP e "Porta"
- Sicurezza
  - Buffer Overflow
- Programmi "Robusti"



## Introduzione: Socket

- Punto terminale di una "comunicazione di rete"
  - connection oriented:
    - » TCP
  - connection less
    - » UDP
- TCP: Sistema Telefonico
- UDP: Sistema Postale

- Socket: Buffer di memoria accessibile all'utente (software) in contrapposizione a Interfaccia accessibile al sistema operativo



## Socket UDP: Fasi

- Allocazione risorse e apertura
  - struct sockaddr_in
  - socket()
- Collegamento
  - bind
- Trasmissione e Ricezione
  - sendto
  - recvfrom
- Chiusura e rilascio risorse
  - close

## Socket TCP: Fasi

- Allocazione risorse e apertura
  - come socket UDP
- Collegamento
  - bind
  - listen
  - connect
- Trasmissione e Ricezione
  - send
  - recv
- Chiusura e rilascio risorse
  - close



## Socket: File

- Forte analogia con “modus operandi” FILE
- In sistemi *nix tutto viene trattato come file:
  - semplicità
  - standard



## Funzione: Socket()

- Prototipo
  - int socket(int dominio, int tipo, int protocollo);
    - » ritorna un descrittore maggiore di 0 se va tutto bene
    - » ritorna un valore minore di 0 se ci sono problemi
- dominio:
  - AF_INET (macro) utilizzo nella suite TCP-IP (Internet protocol)
  - AF_INET6
- tipo: categoria protocollo
  - UDP: SOCK_DGRAM
  - TCP: SOCK_STREAM
- protocollo:
  - 0 (“automatico” di solito ne esiste uno per tipo socket)



## Struttura: sockaddr

- Campi:
  - unsigned short sin_family
    - » address family, AF_xxxx
  - char sa_data[14];
    - » protocol address

sa_data conterrà
- » indirizzo destinazione
- » indirizzo di *port*

## Struttura: sockaddr_in

- Campi:
  - sin_family
    vedi dominio
  - sin_addr
    - » .s_addr
      Indirizzo IP a cui "legare" il socket;
      si usa la macro INADDR_ANY per tutte;
  - sin_port
    numero di port (tradotto con porta)
    se vale 0 allora sceglie il sistema la prima libera
    NB: > 1024



## Funzione: bind()

- Prototipo:
  - int bind(int descrittore, (struct sockaddr *) indirizzo, int lunghezza_struct_sockaddr);
    - » ritorna un descrittore maggiore di 0 se vatutto bene
    - » ritorna 0 se ci sono problemi
      - esempio porta già occupata
  - la struct sockaddr deve già contenere i dati fondamentali, ovvero protocollo, indirizzo IP e porta



## Funzione: close()

- int close(int descrittore_socket)
  - bisogna passare il descrittore del socket da terminare;
  - ritorna 0 se la chiusura è riuscita;
  - ritorna -1 in caso di errore.



## Esempio

- Programma con le funzioni base, ma senza alcuna finalità se non mostrare la sitassi di base.

Nome file: base.c

# Approfondimenti

- campo di strut sockaddr_in
  - unsigned char sin_zero[8]

  deve essere azzerato quindi utilizzo
    - » bzero()
    - » memset()
- Per ragioni storiche

```
struct in_addr {
        unsigned long s_addr;
}
```



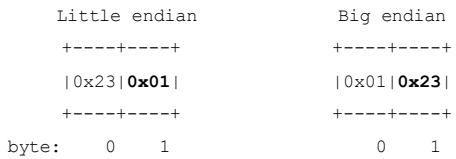
# Funzione htons()

- Host to Network short: converte gli interi nel formato corretto per la rete (Big Endian) dal formato macchina.
- Big Endian:
  - HP
  - IBM
  - Motorola
- Little Endian:
  - Intel
  - Dec
  - Vax

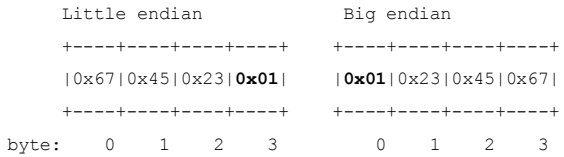


# Standard Memorizzazione Numeri

Nel caso di una WORD, il numero esadecimale 0x0123 sarà memorizzato




Nel caso di una DWORD, il numero esadecimale 0x01234567 sarà memorizzato




(Negli esempi il valore in grassetto è il byte più significativo)



# Altre funzioni di conversione

- htonl()

  Host to Network Long
- ntohs()

  Network to Host Short
- ntohl()

  Network to Host Short

## Indirizzo IP

- Come convertire l'indirizzo IP dal formato stringa
  inet_addr("130.251.1.4")

- E viceversa ?
  inet_ntoa(ina.sin_addr)

se lo devo conservare ricordarsi di farne una copia !!!

In Linguaggio C: ... strcpy() ...



## UDP: invio

- int sendto(int descrittore_socket,
  const void *messaggio, int len_messaggio,
  unsigned int flags,
  const struct sockaddr *remoto, int len_sockaddr)

restituisce il numero di byte inviati oppure -1 in presenza di errore

len_sockaddr = sizeof (struct sockaddr)



## UDP: ricezione

- int recvto(int descrittore_socket,
  void *buffer, int len_buffer, unsigned int flags,
  const struct sockaddr *remoto, int *len_sockaddr)

restituisce il numero di byte inviati oppure -1 in presenza di errore

len_sockaddr = sizeof (struct sockaddr) ⇨ Attenzione è un puntatore ritorna un valore !!!



## TCP: Effettuare la Connessione

- int connect(ind descrittore_socket,
  struct sockaddr * remote, int len_sockaddr);

ritorna -1 in caso di problemi: per gestire l'errore meglio utilizzare la variabile errno

- bind potrebbe non servire
  (dipende dall'applicazione)
- non scelgo la "port" di partenza ma solo IP e "port" di destinazione

## TCP: restare in attesa connessione

- int listen(int descrittore_socket, int num_connessioni)

num_connessioni:

numero di connessioni da “tenere” in coda

- Ritorna 0 se operazione conclusa con successo; -1 in caso di errore !!!
- Ricordarsi di utilizzare variabile errno



## TCP: Accettare una connessione

- int accept(int descrittore_socket,
  struct sockaddr *remoto, int *len_sockaddr);

ritorna -1 in caso di errore e un valore > 0 come nuovo descrittore del socket

attenzione al parametro int *len_sockaddr che è un puntatore



## TCP: invio

- int send(int descrittore_socket, char *messaggio,
  int len_messaggio, int flags);

Ritorna il numero di byte inviati, -1 in caso di errore (da usare errno)

se flags = 0 equivalente a una write

altre possibili opzioni

- non blocking
- msg_confirm
- msg_more



## TCP: ricezione

- int recv(int descrittore_socket, char *buffer,
  int len_buffer, int flags);

Ritorna il numero di byte ricevuti, -1 in caso di errore (da usare errno)

len_buffer è la lunghezza massima accettabile per il messaggio

## Funzione: shutdown()

- int shutdown(int descrittore_socket, int flag)

simile a close, ma con un maggior controllo

  - bisogna passare il descrittore del socket da terminare;
  - flag specifica alcune azioni da intraprendere successivamente:
    - » 0 disabilita pendenti ricezioni
    - » 1 disabilita pendenti invii
    - » 2 disabilita invi e ricezioni pendenti
  - ritorna 0 se la chiusura è riuscita;
  - ritorna -1 in caso di errore.



## Funzione getpeername()

- Scopo: sapere chi è connesso all'altro capo del socket;
- int getpeername(int descr_socket, struct sockaddr *remote, int *len_addr);
- Ritorna 0 se operazione conclusa con successo; -1 in caso di errore !!!

Ricordarsi di utilizzare inet_ntoa()



## Funzione: gethostname()

- Scopo: ottenere il “nome” del proprio host
- int gethostname(char *nome_host, int len_array)
- si trova nella libreria <unistd.h>
- 0 se operazione conclusa correttamente
- -1 in caso di errore



## Struct hostent

- struct hostent {

```
char    *h_name; /* nome ufficiale host */
char    **h_aliases; /* vettore nomi alternativi terminante
   con un NULL */
int     h_addrtype; /* tipo di indirizzo: AF_INET */
int     h_lenght; /* lunghezza indirizzo */
char    **h_addr_list; /* vettore indirizzi degli host con 0
   come terminatore, Network Byte Order */
}
#define h_addr h_addr_list[0]
```

## Funzione: gethostbyname()

- struct hostent *gethostbyname(const char *name);
- Ritorna NULL in caso di errore
- utilizzo libreria netdb.h
- Esempio:

```
h=gethostbyname(nameHost);
printf("Nome host : %s\n", h->h_name);
printf("IP Address: %s\n",inet_ntoa(*((struct in_addr *) h->h_addr)));
```



## Struttura per indirizzo IP

- struct in_addr {
  unsigned long s_addr;
  }
- struttura storica per memorizzare indirizzo IP in formato

Order By Network



## Gestione multi-connessione

- int fork()
- genera un processo figlio e ritorna 0 per il processo figlio, il PID nel padre;
- se ritorna -1 errore



## Blocking

- per rendere non bloccanti i socket esistono due alternative:
  - fnctl
  - select

Esaminiamo brevemente la prima alternativa; è necessario includere "le librerie"

- unistd.h, fcntl.h

int fcntl(int descrittore_socket, F_SETFL, O_NONBLOCK);

l'alternativa select fornisce:

- soluzione più complessa
- maggiore flessibilità: scelta tempi attesa