Sistemi Operativi

Laurea in Ingegneria Informatica

Università di Roma Tor Vergata

Docente: Francesco Quaglia

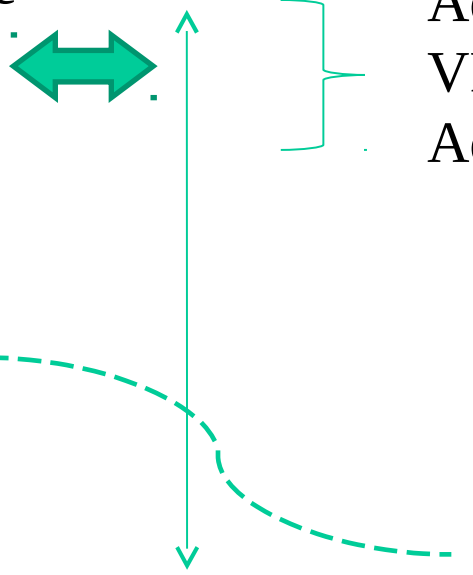
Sockets

1. Concetti basici
2. Domini e tipi di comunicazione
3. Protocolli
4. Sockets in sistemi UNIX/Windows

Sockets - concetti basici

- un socket è un oggetto di I/O associato a un canale di I/O

descrittore
oppure
handle



Ad-hoc API per la creazione
VFS API per la trasmissione/ricezione
Ad-hoc API per la trasmissione/ricezione

Un driver di livello kernel (o una pila di drivers)
implementa le reali operazioni (associate alle system-call)

Domini – Tipi di comunicazione – Protocolli

- Il “dominio” determina il modo in cui un socket è identificato
- Quindi il dominio è un insieme di identificatori (detti indirizzi) che possono essere:
  - ✓ locali rispetto ad uno specifico sistema
  - ✓ locali rispetto ad una sottorete (e.g. una LAN)
  - ✓ globali
- I tipi di comuncazione attuabili tramite socket sono quelli classici
  - ✓ stream
  - ✓ block (in particolare “packet”)
- I protocolli sono le istanze dei driver di I/O da associare ai socket

Domini classici

- AF_INET (AF_INET6) — Internet protocols
- AF_UNIX — Unix internal protocols (not really communication, but IPC)
- AF_NS — Xerox NS protocols
- AF_IMPLINK — IMP link layer (Interface Message Processor)

Notazioni equivalenti

- PF_INET (PF_INET6)
- PF_UNIX
- PF_NS
- PF_IMPLINK

AF = address family

PF = protocol family

Dalla man-page di Linux

Name	Purpose
PF_UNIX, PF_LOCAL	Local communication
PF_INET	IPv4 Internet protocols
PF_INET6	IPv6 Internet protocols
PF_IPX	IPX - Novell protocols
PF_NETLINK	Kernel user interface device
PF_X25	ITU-T X.25 / ISO-8208 protocol
PF_AX25	Amateur radio AX.25 protocol
PF_ATMPVC	Access to raw ATM PVCs
PF_APPLETALK	Appletalk
PF_PACKET	Low level packet interface

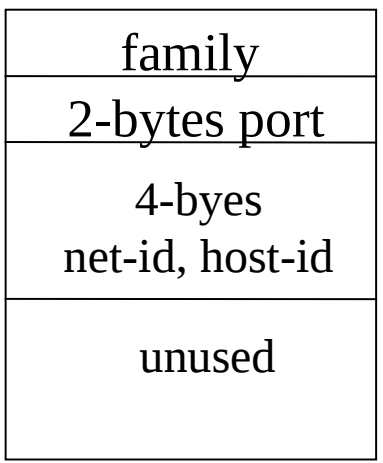
Formato degli indirizzi

per Posix definito in <sys/socket.h>

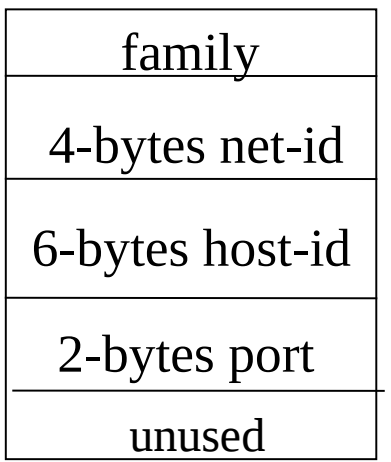
```
struct sockaddr{
    u_short  sa_family;   /* address family */
    char     sa_data[14]; /* up to 14 bytes of protocol specific address  */
}
```

adeguato per AF_INET e AF_NS

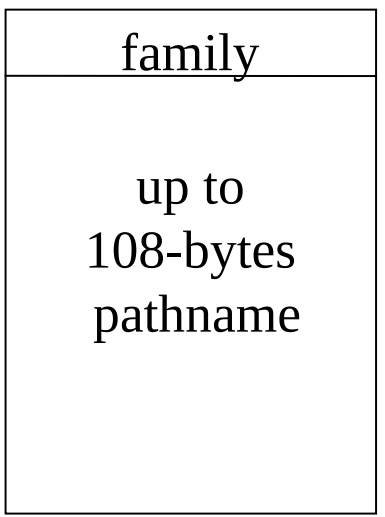
`struct sockaddr_in`



`struct sockaddr_ns`



`struct sockaddr_un`



Buffer strutturato per AF_INET

Per Posix efinito in <netinet/in.h>

```
struct in_addr {
      u_long  s_addr;   /* 32net-id/host-id */
              /* network byte ordered */
}

struct sockaddr_in {
      short            sin_family; /* domain   */
      u_short          sin_port; /* 2-bytes port number */
      struct in_addr   sin_addr; /* 4-bytes host-id  */
      char             sin_zero[8]; /* unused */
}
```

- se **sin_ port** è pari a 0, il sistema usa ephemeral port (non adeguato uin caso di canali per applicazioni server)
- usare **bzero()** per evitare comportamenti non deterministici

Classici tipi di comunicazione

- SOCK_STREAM streaming
- SOCK_DGRAM datagram

← i più comuni

- SOCK_RAW raw data
- SOCK_SEQPACKET sequenced packet
- SOCK_RDM reliable delivery of messages (non implementato)

Protocolli di default	AF_UNIX	AF_INET	AF_NS
SOCK_STREAM	yes	TCP	SPP
SOCK_DGRAM	yes	UDP	IDP
SOCK_RAW		IP	yes
SOCK_SEQPACKET			SPP

nessun acronimo

Sequenced packet protocol

UNIX sockets - creazione on demand

int socket(int domain, int type, int protocol)

Descrizione invoca la creazione di un socket

Argomenti 1) domain: specifica del dominio di comunicazione relativamente al quale può operare il socket

2) type: specifica la semantica della comunicazione associata al socket

3) protocol: specifica il particolare protocollo di comunicazione per il socket

Restituzione un intero non negativo (descrittore di socket) in caso di successo;

-1 in caso di fallimento

Selezione di protocollo

- alcune volte, fissata la coppia (domain,type), è possibile scegliere tra più protocolli di comunicazione
- altre volte invece fissata tale coppia esiste un solo protocollo di comunicazione valido
- il parametro protocol specifica quale protocollo si vuole effettivamente usare una volta fissata tale coppia qualora esista una possibilità di scelta
- il valore 0 per il parametro protocol indica che si vuole utilizzare il protocollo di default, o eventualmente l'unico disponibile per quella coppia (domain,type)

Combinazioni ammissibili per AF_INET

domain	type	protocol	actual protocol
AF_INET	SOCK_DGRAM	IPPROTO_UDP	UDP
AF_INET	SOCK_STREAM	IPPROTO_TCP	TCP
AF_INET	SOCK_RAW	IPPROTO_ICMP	ICMP
AF_INET	SOCK_RAW	IPPROTO_RAW	(raw)

ICMP = Internet Control Management Protocol
ha funzioni di monitoring/gestione del livello IP

Per Posix ls definizione delle macro `IPPROTO_xxx` è nell'header <`netinet/in.h`>

Assegnazione di un indirizzo on-demand

int bind(int ds_sock, struct sockaddr *my_addr, int addrlen)

Descrizione invoca l'assegnazione di un indirizzo al socket

Argomenti 1) ds_sock: descrittore di socket
2) *my_addr: puntatore al buffer che specifica l'indirizzo
3) addrlen: lunghezza (in byte) dell'indirizzo

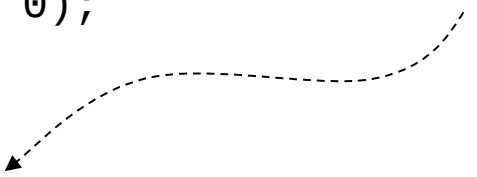
Restituzione -1 in caso di fallimento

- il terzo parametro serve per specificare la **taglia esatta** dell'indirizzo rispetto al dominio di interesse
- il buffer strutturato di tipo **sockaddr** è dimensionato in modo da poter contenere indirizzi appartenenti al dominio per cui la loro specifica richiede il massimo numero di byte (unica eccezione è il dominio AF_UNIX)

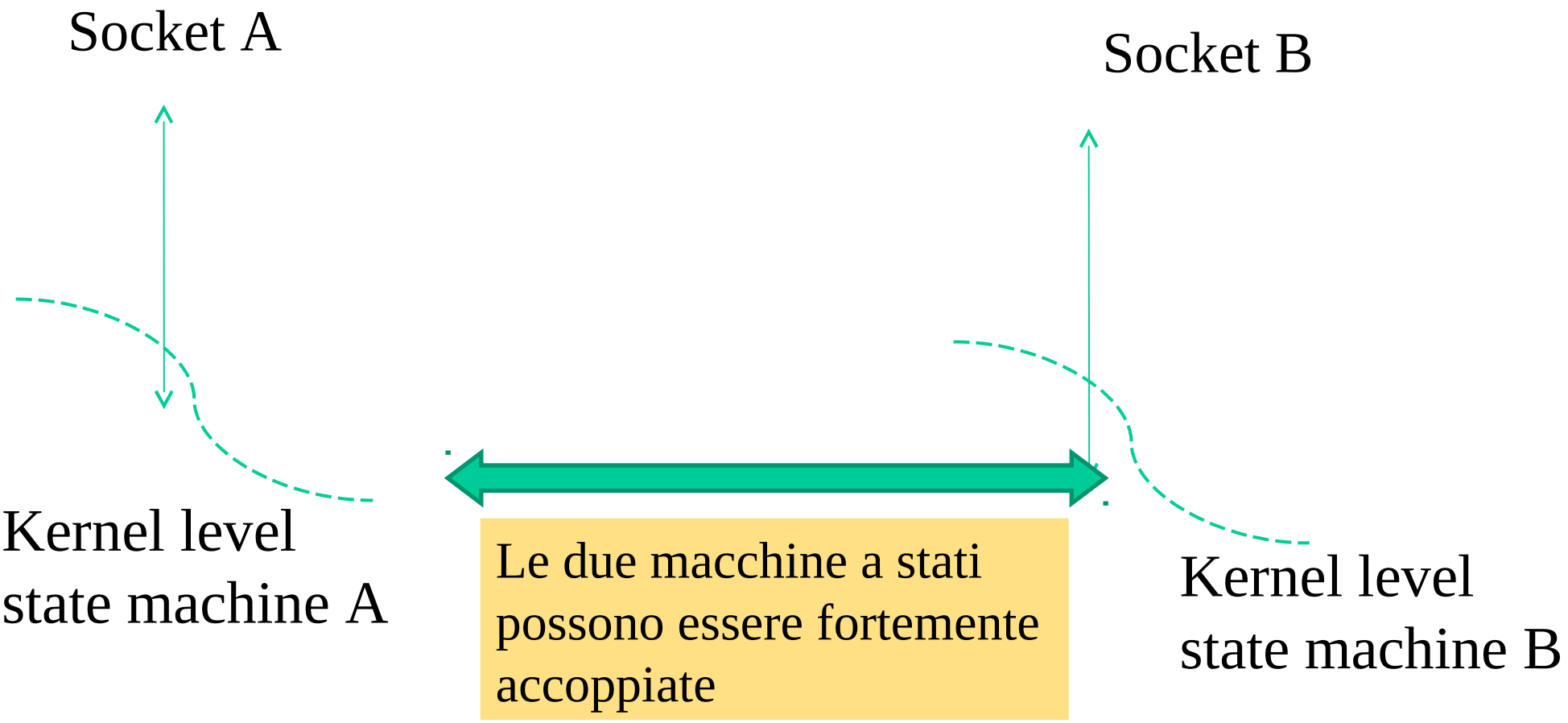
Un esempio di assegnazione di indirizzo in AF_UNIX

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

void main() {
        int ds_sock; int len;
        struct sockaddr_un my_addr;

        ds_sock = socket(AF_UNIX, SOCK_STREAM, 0);

        my_addr.sun_family = AF_UNIX;
        strcpy(my_addr.sun_path,"my_name");

       len = sizeof(my_addr.sun_path) + sizeof(my_addr.sun_family);

        bind(ds_sock, &my_addr, len);
}
```

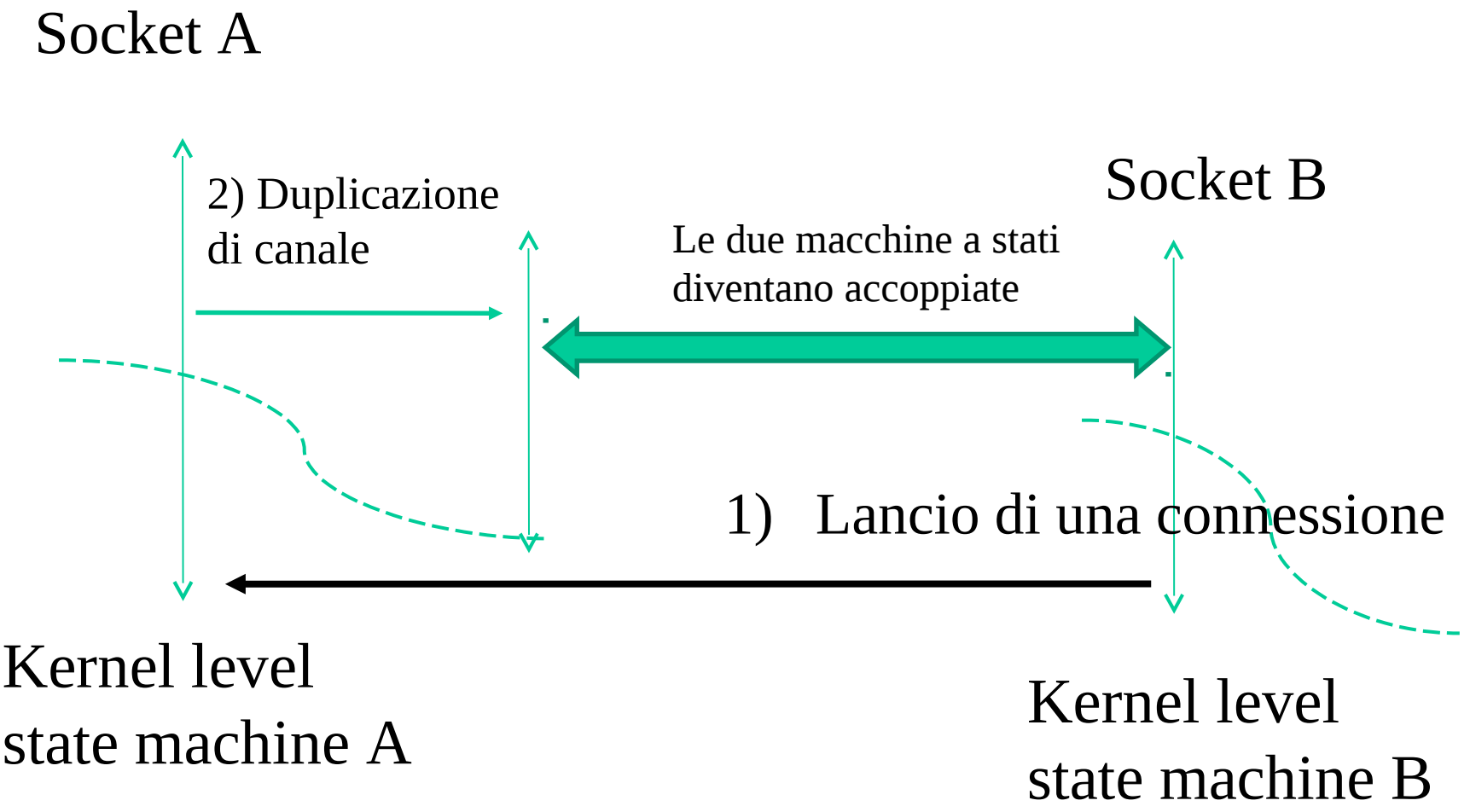
campi sun_family e sun_path



Relazioni tra socket



SOCK_STREAM - connessioni



Attesa di connessioni (SOCK_STREAM)

int accept(int ds_sock, struct sockaddr *addr, int *addrlen)

Descrizione invoca l'accettazione di una connesione su un socket

Argomenti 1) ds_sock: descrittore di socket

2) *addr: puntatore al buffer su cui si copierà l'indirizzo del chiamante

3) *addrlen: puntatore al buffer su cui si scriverà la taglia dell'indirizzo del chiamante (compatibilità per domini)

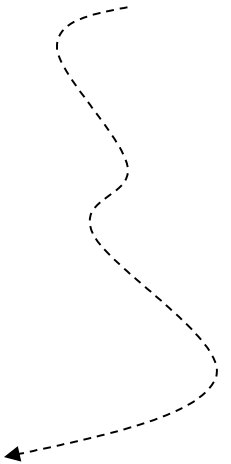
Restituzione un intero positivo indicante il descrittore di un nuovo socket in caso di successo; -1 in caso di errore

- l’accettazione effettua lo switch della connessione su un nuovo socket
- per AF_INET il port number per il nuovo socket e’ lo stesso del socket originale

Un esempio nel dominio AF_INET (accept fallirà deterministicamente)

```
#include <sys/types.h>
#include <sys/socket.h>
#include <netinet/in.h>
#include <netdb.h>
#include <stdio.h>

void main() {
    int ds_sock, ds_sock_acc;
    struct sockaddr_in my_addr;
    struct sockaddr addr;
    int addrlen;

    ds_sock = socket(AF_INET, SOCK_STREAM, 0);

    my_addr.sin_family      = AF_INET;
    my_addr.sin_port        = 25000;
    my_addr.sin_addr.s_addr = INADDR_ANY;

    bind(ds_sock, &my_addr, sizeof(my_addr));
    ds_sock_acc = accept(ds_sock, &addr, &addrlen);

    close(ds_sock);
    close(ds_sock_acc);
}
```

switch della connessione



Modalità listening e backlog di connessioni

int listen(int ds_sock, int backlog)

Descrizione invoca l'impostazione orientativa del numero di connessioni pendenti

Argomenti 1) sock_ds: descrittore di socket
2) backlog: numero di connessioni da mantenere sospese

Restituzione -1 in caso di errore

- una connessione è pendente quando non può essere associata ad un socket destinazione che però esiste
- il backlog specificato tramite questa system call è orientativo nel senso che il sistema operativo potrebbe decidere di mantenere un backlog più ampio
- pendenti un numero più alto di connessioni rispetto al richiesto è necessario impostare un backlog prima di attendere qualsiasi connessione

Un esempio nel dominio AF_INET

```
#include <sys/types.h>
#include <sys/socket.h>
#include <netinet/in.h>
#include <netdb.h>
#include <stdio.h>
#define  BACKLOG 10

void main() {
    int ds_sock, ds_sock_acc, addrlen;
    struct sockaddr_in my_addr; struct sockaddr addr;

    ds_sock = socket(AF_INET, SOCK_STREAM, 0);

    my_addr.sin_family      = AF_INET;
    my_addr.sin_port        = 25000;
    my_addr.sin_addr.s_addr = INADDR_ANY;

    bind(ds_sock, &my_addr, sizeof(my_addr));
    listen(ds_sock, BACKLOG);

    while(1) {
        ds_sock_acc = accept(ds_sock, &addr, &addrlen);
        close(ds_sock_acc);
    }
}
```

Lancio di connessioni

int connect(int ds_socks, struct sockaddr *addr, int addrlen)

Descrizione invoca la connessione di un socket su un indirizzo

Argomenti 1) ds_sock: descrittore del socket da connettere

2) *addr: puntatore al buffer contenente l'indirizzo al quale connettere il socket

3) addrlen: la taglia dell'indirizzo al quale ci si vuole connettere

Restituzione -1 in caso di errore

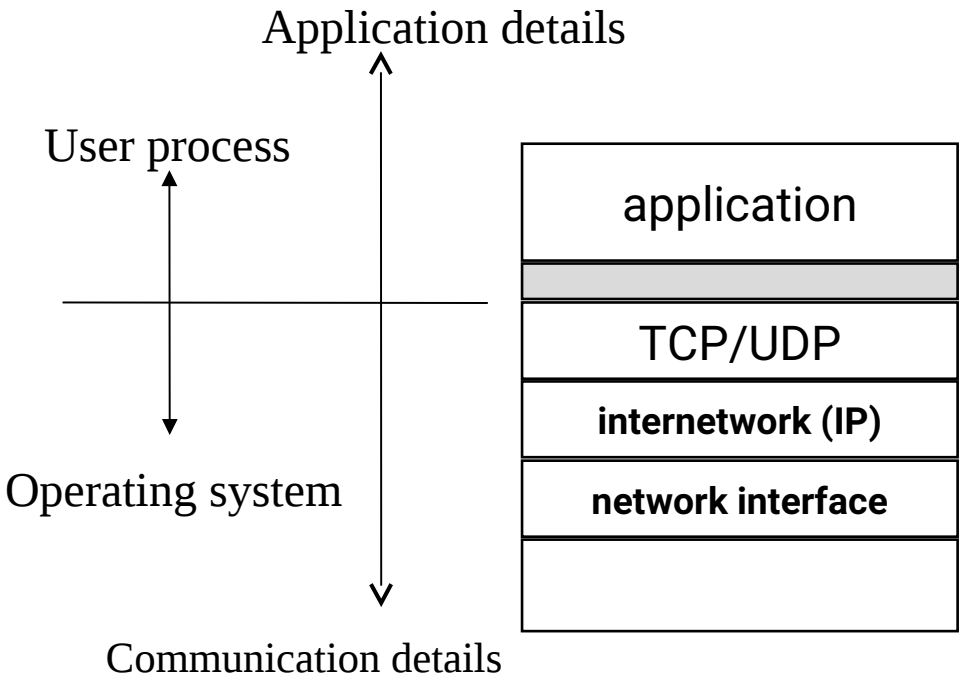
- necessaria in caso di tipo di comunicazione SOCK_STREAM
- può essere usata anche in caso di comunicazione ``connectionless'', ovvero SOCK_DGRAM, SOCK_RAW

Connessione su comunicazione DGRAM

Vantaggi

- non c'è necessità di reimpostare ogni volta l'indirizzo del destinatario nello spedire un nuovo datagram
- le system call per la spedizione avranno quindi bisogno di identificare solo il punto di uscita dal sistema
- otteniamo come una "post box" univocamente associata ad una destinazione
- se il protocollo datagram usato supporta notifica di indirizzi invalidi allora la connessione permette di riportare indirizzi invalidi al mittente
- i messaggi di errore (ad esempio "port unreachable") sono riportati tramite appositi acchetti ICMP

AF_INET sockets



Interfaccia socket
TLI (Transport Layer Interface)

- end-point determinati da:
  - indirizzo dell'host (IP): livello rete
  - Port number: livello di trasporto

Trasporto AF_INET

Supporta realmente trasferimento di dati **tra processi** ovvero tra canali accessibili a questi (il livello rete supporta solo trasferimento tra host)

Protocolli standard

TCP (Transmission Control Protocol)

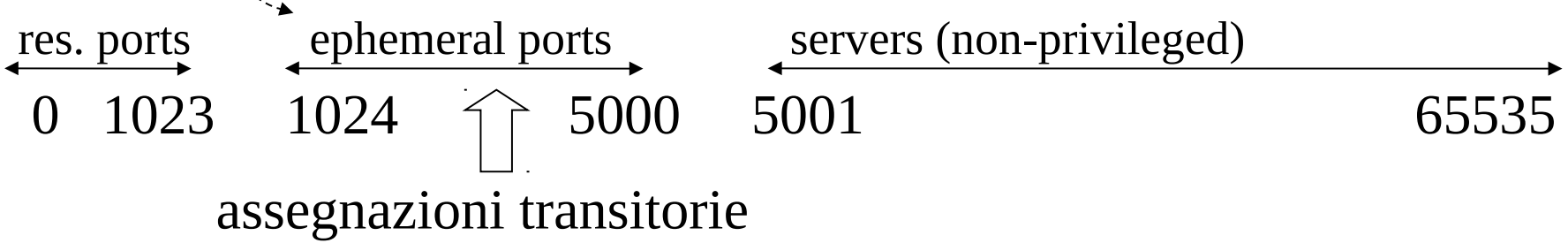
- orientato alla connessione (instaurazione e chiusura esplicita)
- connessione full duplex (e' possibile trasferimento contemporaneo nelle due direzioni della connessione)
- consegna affidabile ed ordinata delle informazioni

UDP (User Datagram Protocol)

- non orientato alla connessione
- consegna non affidabile
- consegna non ordinata

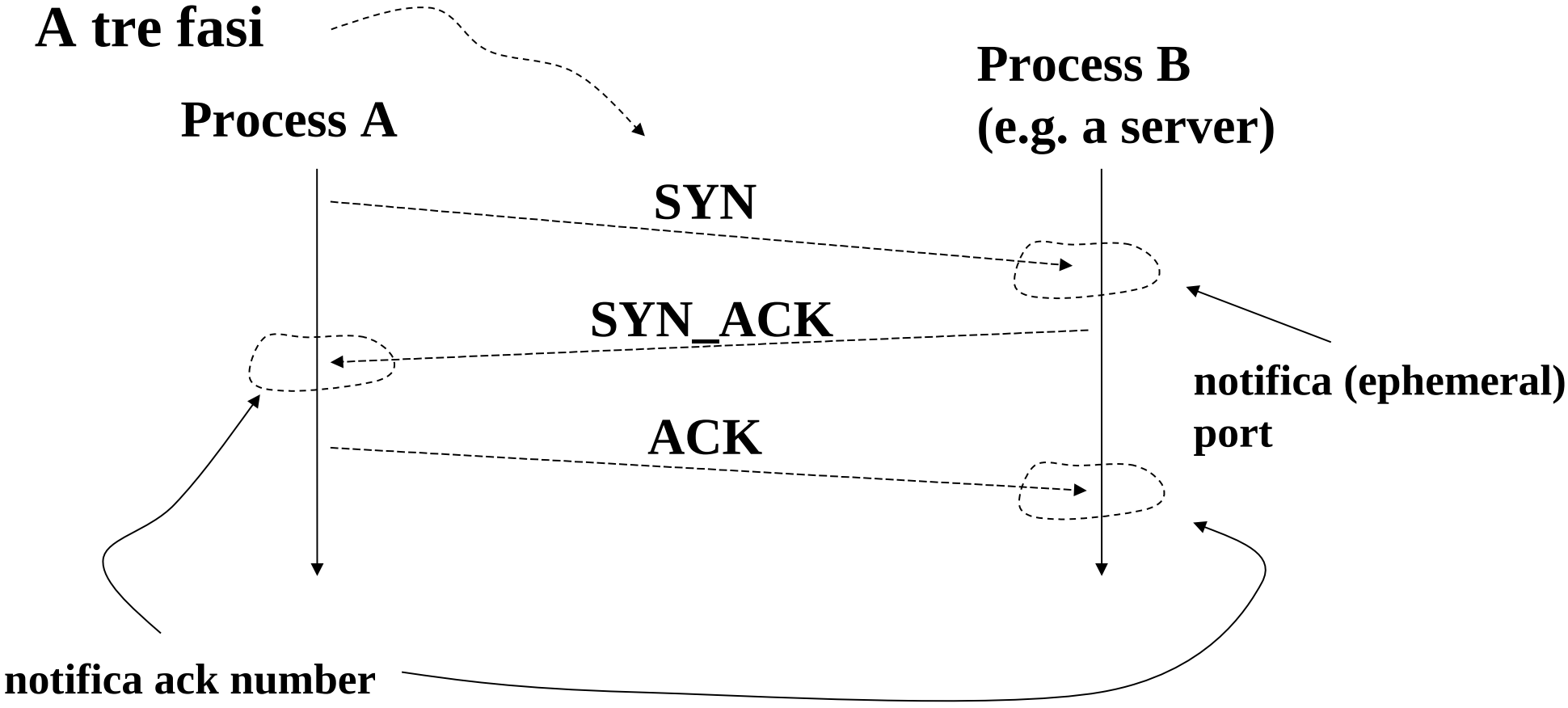
Port numbers

- l’utilizzo dei numeri di porto da parte del sistema operativo varia con la versione del sistema
- in BSD



- servizi (sever) ben noti lavorano sui seguenti port numbers:
  - ftp 21/tcp
  - telnet 23/tcp
  - snmp 161/udp
  - HTTP 80/tcp

Connessione di socket AF_INET



Associazione completa:

<source-port, source-IP, destination-port, destination-IP, protocol>

Un esempio nel dominio AF_INET

```
#include <sys/types.h>
#include <sys/socket.h>
#include <netinet/in.h>
#include <netdb.h>
#include <stdio.h>

void main() {
      int ds_sock, length, ret; struct sockaddr_in addr;
      struct hostent *hp; /* utilizzato per la restituzione
               della chiamata gethostbyname() */

      ds_sock = socket(AF_INET, SOCK_STREAM, 0);

      addr.sin_family = AF_INET;
      addr.sin_port   = 25000; // non canonico (network-order)

      hp = gethostbyname(“claudius.ce.uniroma2.it");
      memcpy(&addr.sin_addr, hp->h_addr, 4);

      ret = connect(ds_sock, &addr, sizeof(addr));
      if ( ret == -1 ) printf("Errore nella chiamata connect\n");

      close(ds_sock);
}
```

struct hostent

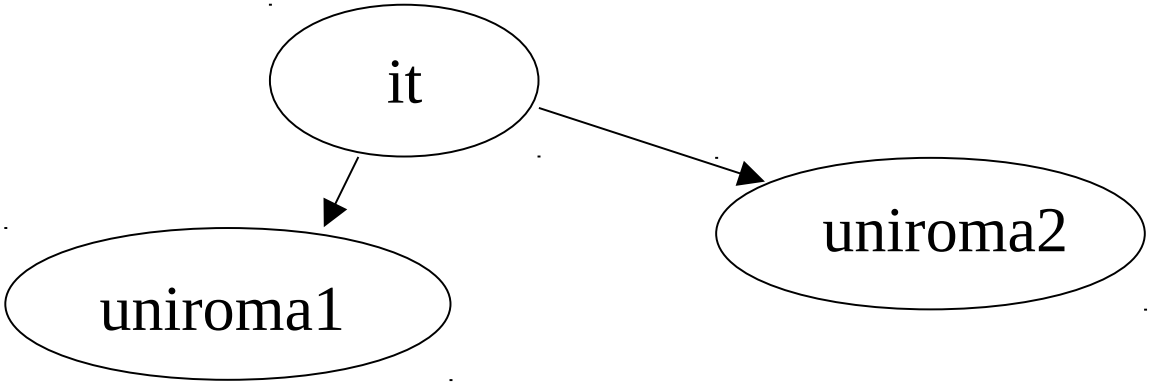
```
#define h_addr h_addr_list[0]; /* indirizzo del buffer
                                  di specifica del numero IP */

struct hostent {
   char   *h_name;      /* nome ufficiale dell'host  */
   char   **h_aliases;  /* lista degli alias  */
   int    h_addrtype;   /* tipo di indirizzo dell'host */
   int    h_length;     /* lunghezza (in byte) dell'indirizzo */
   char   **h_addr_list; /* lista di indirizzi dal name server */
}
```

- `gethostbyname()` è non completamente adeguata per il multithread poichè non è rientrante
- in tal caso usare `gethostbyname_r()` che invece è rientrante

Servizio DNS

- ad ogni host viene associato un nome, definito come stringhe separate da punti
- la prima stringa identifica il nome dell'host vero e proprio
- le stringhe rimanenti identificano la rete di appartenenza, detta anche dominio
- esistono host dislocati in rete che ospitano i name server, ovvero dei server che implementano un meccanismo distribuito su scala geografica per
- risalire all'indirizzo IP di un host a partire dal nome (e viceversa)
- l'organizzazione è gerarchica, basata su ripartizione per domini

Gerarchia di domini



Ogni livello gerarchico ha almeno un NS <u>autoritativo</u>

Alcuni domini di massimo livello

com -> organizzazioni commerciali
edu -> istituzioniUSA per l'istruzione
gov -> istituzioni governative USA
mil -> istituzioni militari USA
net -> maggiori organizzazioni di supporto ad Internet
org -> organizzazioni senza scopo di lucro diverse dalle precedenti
it,fr,.. -> domini nazionali

Caso speciale per AF_UNIX - coppie di sockets

int socketpair(int domain, int type, int protocol, int sockvec[2])

Descrizione invoca la creazione di una coppia di socketonnessi

Argomenti 1) domain: specifica del dominio di comunicazione relativamente al quale può operare il socket

2) type: specifica la semantica della comunicazione associata al socket

3) protocol: specifica il particolare protocollo di comunicazione per il socket

4) sockvec[2]: coppia di descritori di socket restituiti

Restituzione -1 in caso di fallimento

- opera sia su SOCK_STREAM che SOCK_DGRAM, in ogni caso solo sul domino AF_UNIX

Chiusura di un socket

- quando un processo non ha più bisogno di un dato socket per la comunicazione può chiudere tramite la chiamata close()

- il parametro della chiamata sarà il descrittore del socket che si vuole chiudere

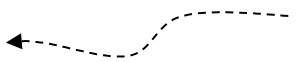
- è da notare che quando un processo chiude un socket, il socket stesso viene rimosso solo qualora non vi sia alcun altro processo che possieda un descrittore valido per quel socket

- i descrittori di socket vengono trattati alla stregua di descrittori di file (tabella locale per processo)

- descrittori validi multipli possono essere originati per effetto della system call fork()

Un esempio

```
#include <sys/types.h>
#include <sys/socket.h>
#include <netinet/in.h>
#include <netdb.h>
#include <stdio.h>

void main() {
      int ds_sock;  char c;
      struct sockaddr_in my_addr;

      ds_sock = socket(AF_INET, SOCK_STREAM, 0);

      my_addr.sin_family      = AF_INET;
      my_addr.sin_port        = 25000;
      my_addr.sin_addr.s_addr = INADDR_ANY;

      bind(ds_sock, &my_addr, sizeof(my_addr));

      if ( fork()!=0 ) close(ds_sock)
      else {
          while ( read(0,&c,1) != -1 );
          close(ds_sock)
      }
}
```



socket ancora attivo

Spedizione e ricezione dati

Previo uso di connect()

- si possono utilizzare le system call read() write()
- alternativamente si possono utilizzare le seguenti system call

```
int send(int sock_ds, const void *buff, int size, int flag)
int recv(int sock_ds, void *buff, int size, int flag)
```

Descrizione invocano spedizione/ricezione di dati tramite socket

Argomenti 1) sock_ds: descrittore di socket locale
2) *buff: puntatore al buffer destinato ai dati
3) size: taglia dei dati
4) flag: specifica delle opzioni di spedizione

Restituzione -1 in caso di errore

Spedizione e ricezione dati

Modalità sconnessa

- si possono utilizzare le seguenti system call

```
int sendto(int sock_ds, const void *buff, int size, int flag, struct sockaddr *addr, int addrlen)
int recvfrom(int sock_ds, void *buff, int size, int flag, struct sockaddr *addr, int *addrlen)
```

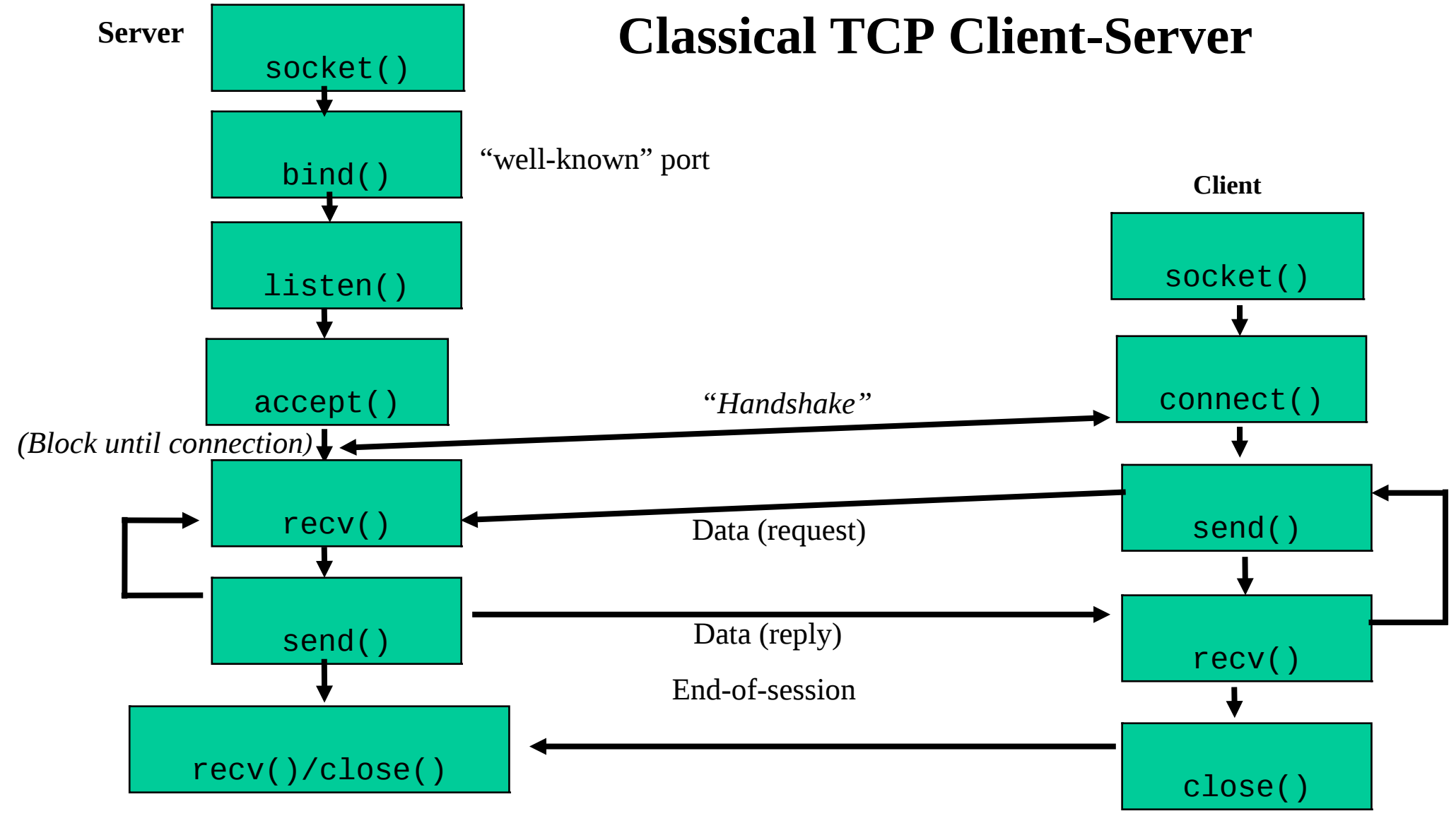
Descrizione invocano spedizione/ricezione di dati tramite socket

Argomenti

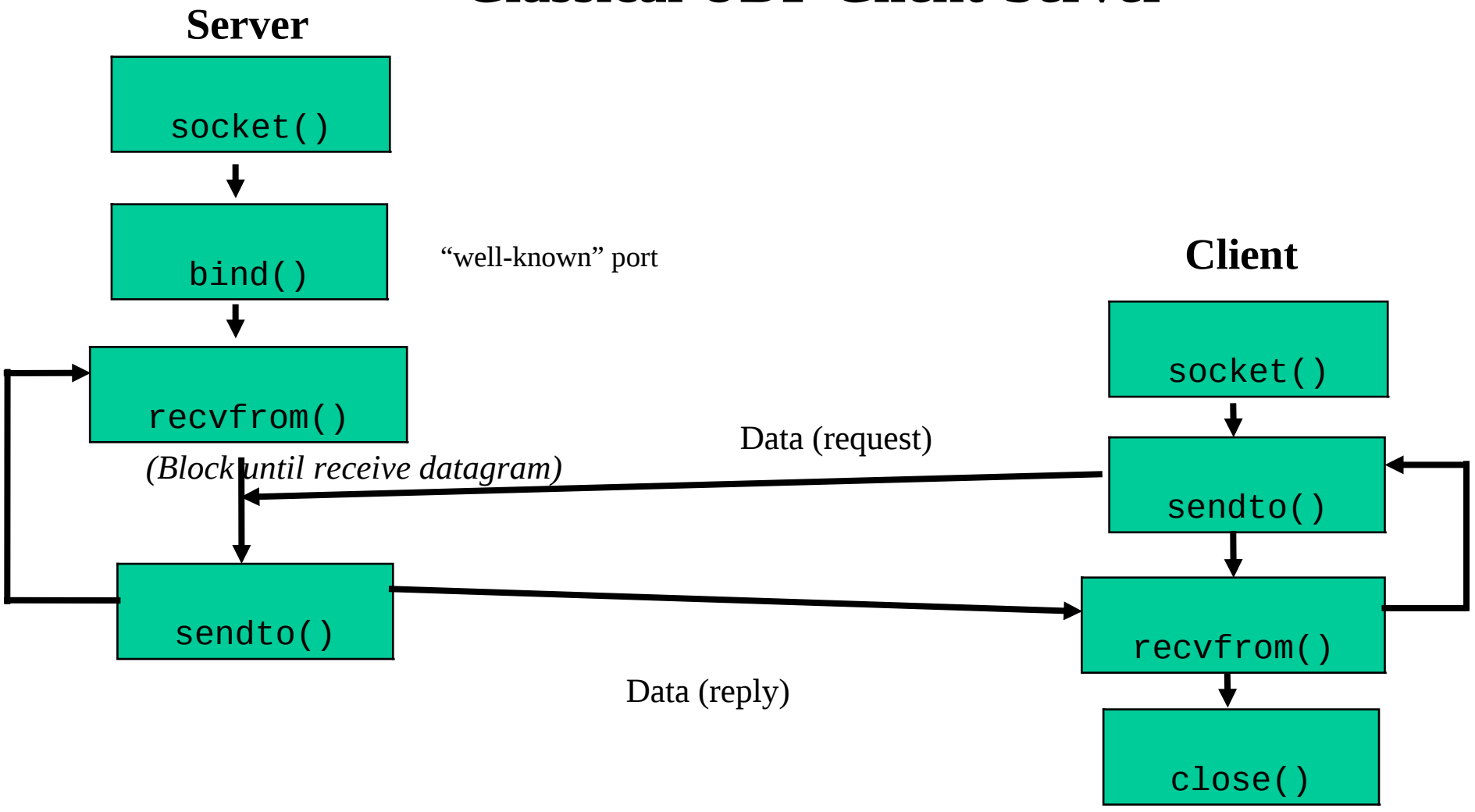
1) sock_ds: descrittore di socket locale
2) *buff: puntatore al buffer destinato ai dati
3) size: taglia dei dati
4) flag: specifica delle opzioni di spedizione
5) *addr: buffer per l'indirizzo di destinazione/sorgente
6) addrlen (*addrlen): lunghezza indirizzo destinazione/sorgente

Restituzione -1 in caso di errore

Classical TCP Client-Server



Classical UDP Client-Server



Byte Ordering

- diverse architetture hardware manipolano i dati di dimensione maggiore di un byte in maniera diversa

ES: un intero di 4 byte contenente il valore 258 può essere rappresentato in due modi differenti:

Big Endian

0	.....		3
2	1	0	0

Little Endian

0	.....		3
0	0	1	2

- i dati che vanno sulla rete sono sempre in **network order** (big endian)
- tuttavia i dati usati sulla macchina sono in **host order** (little o big endian dipendente dall'architettura hardware)

Funzioni di conversione

- alcune system call richiedono che certi dati vengano forniti in **network order** (ES: il contenuto di `struct sockaddr_in` in `bind()`)
- un programma che usi i socket può funzionare su una architettura HW ma non su altre, **anche se si usa lo stesso sistema operativo!**

Soluzione: funzioni di conversione
(mascherano differenze architetturali)

uint16_t htons (uint16_t host16bitvalue);
uint32_t htonl (uint32_t host32bitvalue);
Prendono come parametro un intero in host order a 16 o 32 bit rispettivamente e restituiscono lo stesso intero in network order

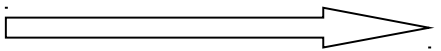
uint16_t ntohs (uint16_t network16bitvalue);
uint32_t ntohl (uint32_t network32bitvalue);
Prendono come parametro un intero in network order a 16 o 32 bit rispettivamente e restituiscono lo stesso intero in host order

Un esempio di applicazione di TCP

Applicazione server
(please check with bugs!)

```
#include <sys/types.h>
#include <sys/socket.h>
#include <sys/signal.h>
#include <netinet/in.h>
#include <netdb.h>
#include <stdio.h>

#define MAX_DIM 1024
#define CODA 3

void main() {
    int ds_sock, ds_sock_a, rval;
    struct sockaddr_in server;
    struct sockaddr client;
    char buff[MAX_DIM];

    sock = socket(AF_INET, SOCK_STREAM, 0);

    bzero((char*)&server, sizeof(server));
    server.sin_family      = AF_INET;
    server.sin_port        = htons(25000);
    server.sin_addr.s_addr = INADDR_ANY;
```

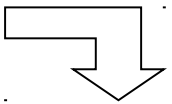
continua



```
    bind(ds_sock, &server, sizeof(server));
    listen(ds_sock, CODA);

    length = sizeof(client);
    signal(SIGCHLD, SIG_IGN);

    while(1) {

        while( (ds_sock_a = accept(ds_sock, &client, &length)) == -1);

        if (fork()==0) {
                close(ds_sock);
            do {
                read(ds_sock_a, buff, MAX_DIM);
                printf("messaggio del client = %s\n", buff);
            } while(strcmp(buff,"quit") != 0);
            write(ds_sock_a, “letto", strlen(“letto”)+1);
            close(ds_sock_a);
            exit(0);
        }
        else close(ds_sock_a);
    }
}
```

Applicazione client

```
 #include <sys/types.h>
#include <sys/socket.h>
#include <netinet/in.h>
#include <netdb.h>
#include <stdio.h>

#define  MAX_DIM  1024

void main(){
        int ds_sock, length, res;
        struct sockaddr_in client;
        struct hostent *hp;
        char buff[MAX_DIM];

        ds_sock = socket(AF_INET, SOCK_STREAM, 0);

        client.sin_family =      AF_INET;
        client.sin_port   =      htons(25000);

        hp = gethostbyname("hermes.dis.uniroma1.it");
        bcopy(hp->h_addr, &client.sin_addr, hp->h_length);

        res = connect(ds_sock, &client,sizeof(client));
```

continua 

```
            if ( res ==      -1 ) {
                    printf("Errore nella connect \n");
                    exit(1);
            }

            printf("Digitare le stringhe da trasferire (quit per terminare): ");
            do {
                    scanf("%s", buff);
                    write(ds_sock, buff, MAX_DIM);
            } while(strcmp(buff,"quit") != 0);

    read(ds_sock, buff, MAX_DIM);
    printf(“Risposta del server: %s\n", buff);

    close(ds_sock);
}
```

Opzioni su socket

- un socket può avere una serie di opzioni
- ogni opzione permette di controllare il comportamento di alcuni livelli del protocollo di comunicazione (e della relativa pila)

```
int setsockopt(int sockfd, int level, int optname, void *optval, socklen_t optlen)
int getsockopt(int sockfd, int level, int optname, const void *optval, socklen_t *optlen)
```

Descrizione system call che permettono di cambiare una delle opzioni sul socket o di leggerne lo stato, rispettivamente

Argomenti
1) sockfd: descrittore di un socket
2) level: identificatore del tipo di opzione (opzione relativa al socket o relativa a qualche protocollo specifico)
3) optname: identificatore della specifica opzione da cambiare/leggere
4) optval: puntatore a una variabile contenente il valore da cambiare o in cui verrà registrato il valore da leggere.
5) oplen: puntatore alla lunghezza della variabile puntata da optval o lunghezza della variabile puntata da optval

Ritorno 0 se tutto OK, -1 in caso di errore

Alcuni tipi di opzioni

SO_DEBUG
Turns on recording of debugging information. This option enables or disables debugging in the underlying protocol modules. This option takes an **int** value. This is a Boolean option.

SO_BROADCAST
Permits sending of broadcast messages, if this is supported by the protocol. This option takes an **int** value. This is a Boolean option.

SO_REUSEADDR
Specifies that the rules used in validating addresses supplied to *bind()* should allow reuse of local addresses, if this is supported by the protocol. This option takes an **int** value. This is a Boolean option.

SO_KEEPALIVE
Keeps connections active by enabling the periodic transmission of messages, if this is supported by the protocol. This option takes an **int** value.

If the connected socket fails to respond to these messages, the connection is broken and threads writing to that socket are notified with a SIGPIPE signal. This is a Boolean option.

SO_LINGER
Lingers on a *close()* if data is present. This option controls the action taken when unsent messages queue on a socket and *close()* is performed. If SO_LINGER is set, the system shall block the calling thread during *close()* until it can transmit the data or until the time expires. If SO_LINGER is not specified, and *close()* is issued, the system handles the call in a way that allows the calling thread to continue as quickly as possible. This option takes a **linger** structure, as defined in the *<sys/socket.h>* header, to specify the state of the option and linger interval.

SO_OOBINLINE
Leaves received out-of-band data (data marked urgent) inline. This option takes an **int** value. This is a Boolean option.

SO_SNDBUF
Sets send buffer size. This option takes an **int** value.

SO_RCVBUF
Sets receive buffer size. This option takes an **int** value.

SO_DONTROUTE
Requests that outgoing messages bypass the standard routing facilities. The destination shall be on a directly-connected network, and messages are directed to the appropriate network interface according to the destination address. The effect, if any, of this option depends on what protocol is in use. This option takes an **int** value. This is a Boolean option.

SO_RCVLOWAT
Sets the minimum number of bytes to process for socket input operations. The default value for SO_RCVLOWAT is 1. If SO_RCVLOWAT is set to a larger value, blocking receive calls normally wait until they have received the smaller of the low water mark value or the requested amount. (They may return less than the low water mark if an error occurs, a signal is caught, or the type of data next in the receive queue is different from that returned; for example, out-of-band data.) This option takes an **int** value. Note that not all implementations allow this option to be set.

SO_RCVTIMEO
Sets the timeout value that specifies the maximum amount of time an input function waits until it completes. It accepts a **timeval** structure with the number of seconds and microseconds specifying the limit on how long to wait for an input operation to complete. If a receive operation has blocked for this much time without receiving additional data, it shall return with a partial count or *errno* set to [EAGAIN] or [EWOULDBLOCK] if no data is received. The default for this option is zero, which indicates that a receive operation shall not time out. This option takes a **timeval** structure. Note that not all implementations allow this option to be set.

SO_SNDLOWAT
Sets the minimum number of bytes to process for socket output operations. Non-blocking output operations shall process no data if flow control does not allow the smaller of the send low water mark value or the entire request to be processed. This option takes an **int** value. Note that not all implementations allow this option to be set.

SO_SNDTIMEO

comunemente usate

Riferimenti

Rago,S.: UNIX System V Network Programming, Addison-Wesley, 1993.

Stevens, W.R.: UNIX Network Programming, Prentice Hall, 1998.

Peterson – Davie: “Computer Networks: A system approach” Morgan Kaufmann 2000.

Windows sockets (molto simili alle UNIX sockets)

```
SOCKET socket(int address_family, int type, int protocol)
```

Descrizione

- invoca la creazione di un socket

Argomenti

- address_family: specifica la famiglia di indirizzi con cui il socket può operare (un elenco completo può essere trovato nel file `winsock2.h` di Visual C++), il dominio di indirizzi di interesse per la nostra trattazione è AF_INET
- type: specifica la semantica della comunicazione associata al socket SOCK_STREAM e SOCK_DGRAM
- protocol: specifica il particolare protocollo di comunicazione per il socket (usare 0 per il default)

Restituzione

- un descrittore di socket in caso di successo; INVALID_SOCKET in caso di fallimento

Associazione di indirizzi

```
int bind(SOCKET ds_sock, struct sockaddr *my_addr,
      int addrlen)
```

Descrizione

invoca l'assegnazione di un indirizzo al socket

Argomenti

- ds_sock: descrittore di socket
- *my_addr: puntatore al buffer che specifica l'indirizzo
- addrlen: lunghezza (in byte) dell'indirizzo

Restituzione

- 0 in caso di successo; SOCKET_ERROR in caso di fallimento

Attesa di connessioni

```
SOCKET accept(SOCKET ds_sock, struct sockaddr
                *addr, int *addrlen)
```

Descrizione

- invoca l'accettazione di una connesione su un socket

Argomenti

- `ds_sock`: descrittore di socket
- `*addr`: puntatore al buffer su cui si copierà l'indirizzo del chiamante
- `*addrlen`: puntatore al buffer su cui si scriverà la taglia dell'indirizzo del chiamante

Restituzione

- il descrittore di un nuovo socket in caso di successo; `INVALID_SOCKET` in caso di errore

Connessioni

```
int connect(SOCKET ds_socks, struct sockaddr *addr, int addrlen)
```

Descrizione

- invoca la connessione su un socket

Argomenti

- `ds_sock`: descrittore del socket locale
- `*addr`: puntatore al buffer contenente l'indirizzo del socket al quale ci si viole connettere
- `addrlen`: la taglia dell'indirizzo del socket al quale ci si vuole connettere

Restituzione

- 0 per una connessione corretta, `SOCKET_ERROR` in caso di errore

Backlog e chiusura di socket

```
listen(SOCKET ds_sock, int backlog)

closesocket(SOCKET ds_socket)
```

Comunicazione

```
int recv(SOCKET sock_ds, const char *buff, int size, int flag)

int send(SOCKET sock_ds, const char *buff, int size, int flag)


int recvfrom(SOCKET sock_ds, char *buff, int size, int flag,
                  struct sockaddr *addr, int *addrlen)


int sendto(SOCKET sock_ds, const void *buff, int size, int flag,
                  struct sockaddr *addr, int addrlen)
```

Inizializzazione interfaccia Winsocket

```
int WSAStartup( WORD wVersionRequested,
                LPWSADATA lpWSAData )
```

Parametri

- `wVersionRequested`: la piu' alta versione delle Window Sockets che il processo chiamante può supportare. Il byte piu' significativo specifica il numero di “minor version”; il byte piu' significativo specifica la “major version”. Vedi funzione `MAKEWORD(x,y)`
- `lpWSAData`: puntatore ad una struttura WSADATA che riceve i dettagli sull'implementazione

Restituzione

0 se ha successo, altrimenti un codice di errore

Ausilii di programmazione

```
int inet_aton(const char *cp, struct in_addr *inp);

in_addr_t inet_addr(const char *cp);

in_addr_t inet_network(const char *cp);

char *inet_ntoa(struct in_addr in);

struct in_addr inet_makeaddr(int net, int host);
```