La Programmazione di Rete e di Sistema (ii)

Giuseppe Pozzi

Impianti di Elaborazione (allievi Gestionali – Como)
Facoltà di Ingegneria dell'Informazione
Politecnico di Milano

giuseppe.pozzi@polimi.it
- versione del 26 novembre 2003 -

La programmazione di rete

Architettura delle reti

Tipologia di reti

- La comunicazione tra due calcolatori avviene attraverso un mezzo trasmissivo, come ad es.:
  - doppino di rame;
  - cavo coassiale;
  - fibra ottica;
  - onde elettromagnetiche;
  - porte a infrarossi;
  - …. o una loro combinazione.

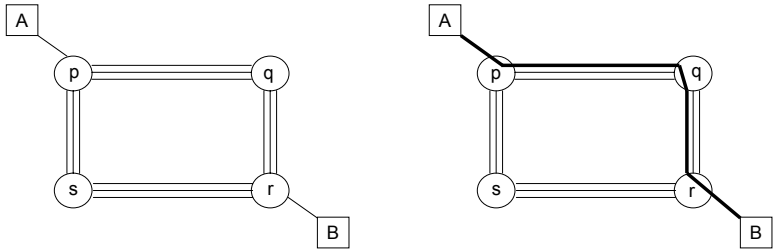
La tecnologia di rete

- E' definita da un insieme di tipi di mezzi trasmissivi e di regole di connessione dei calcolatori e degli altri apparati di rete, tenuti insieme da un protocollo di trasmissione comune.
- I parametri della tecnologia di rete sono:
  - la distanza;
  - la velocità di trasmissione (bit al secondo);
  - il costo.

Connessione a commutazione di circuito

- è una connessione diretta, punto a punto;
- garantisce una banda (ad es. 64 kbps);
- tipicamente basata su linee telefoniche;
- lo sfruttamento della banda non è genericamente continuo:
  (ad es. occupo la linea senza sfruttarne la banda quando leggo una pagina Web che ho appena scaricato in locale);
- elevati costi.

Commutazione di circuito



A seguito della richiesta di connessione, viene creato un circuito fisico da A a B.

Connessione a commutazione di pacchetto

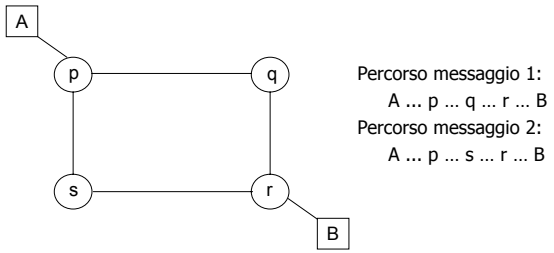
- il canale è unico, ma il traffico è diviso in piccoli messaggi (pacchetti) di poche centinaia di byte;
  - quando un calcolatore collegato in rete (host) non utilizza la banda, questa può essere utilizzata da altri calcolatori, consentendo più comunicazioni simultanee;
  - non è garantito il percorso effettuato;
  - presenta lo svantaggio di comunicazione **frammentata**.

Connessione a commutazione di pacchetto

- Non garantendo il percorso seguito dai messaggi, si realizza un servizio che può essere:
  - a circuito virtuale:
    - mantiene l'ordinamento dei messaggi durante la trasmissione;

  oppure
  - a datagramma:
    - ogni messaggio è trattato in modo indipendente.

Commutazione di pacchetto



Percorso messaggio 1:
A ... p ... q ... r ... B
Percorso messaggio 2:
A ... p ... s ... r ... B

A seguito della richiesta di connessione, la rete non crea nessun collegamento.

Definizioni

- Tipo di rete: tecnologia di rete utilizzata.
- Istanza di rete: una specifica rete di un certo tipo.
- Categoria di reti:
  - LAN (local area network): velocità 4 Mbps - 2 Gbps;
  - MAN (metropolitan area network): velocità 56 kbps - 100 Mbps;
  - WAN (wide area network): velocità 9.6 kbps - 45 Mbps.

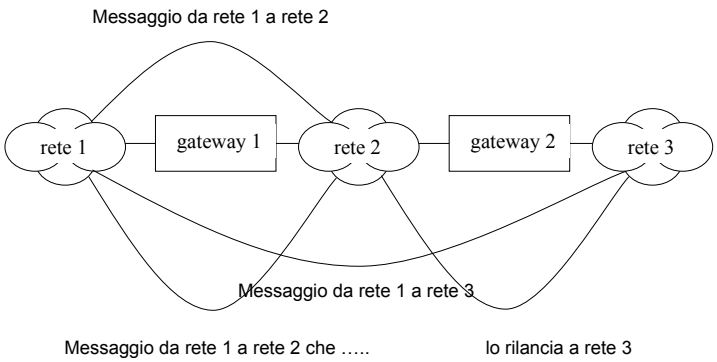
Reti standard

- La maggior parte delle reti è stata standardizzata:
  - IEEE 802.3 (Ethernet): di tipo LAN;
  - Fast Ethernet: di tipo LAN;
  - Token-Ring (IBM): di tipo LAN.
- La comunicazione tra calcolatori anche se su reti differenti e' difficile ma irrinunciabile. L'utente vuole vedere tutto come una **unica** rete.

internet

- Una rete di reti è detta internet (con la *i* minuscola).
- Una internet è definita fornendo i protocolli per trasferire le informazioni tra le varie reti. Il TCP/IP serve proprio a questo.

Interconnessione tra reti



Indirizzo IP

- Ogni host collegato ad una internet ha un suo proprio indirizzo (detto indirizzo IP):
  - univoco: non esistono cioè due macchine di una stessa internet che abbiano indirizzo IP uguale;
  - composto da netid e hostid, per un totale di 32 bit;
  - tutte le macchine di una rete hanno lo stesso netid.

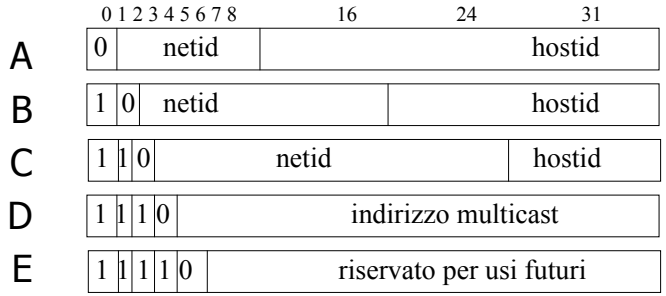
Classi di reti internet

- Classe A: reti con più di 65536 host;
- Classe B: reti con host tra 256 e 65536;
- Classe C: reti con meno di 256 host;
- Classe D: indirizzo multicast;
- Classe E: per usi futuri.

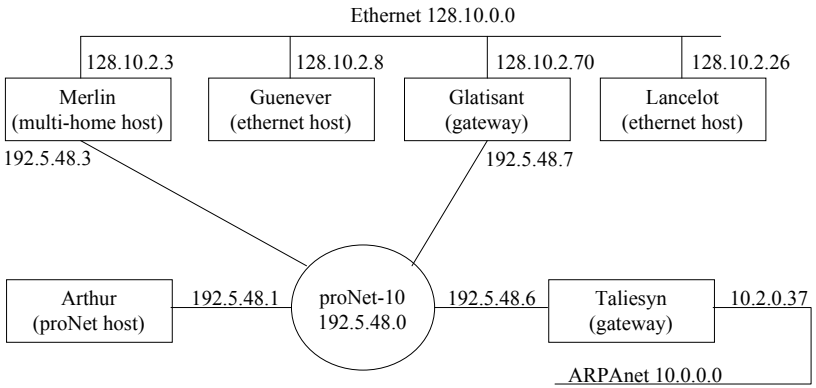
Tramite il broadcast, i messaggi sono inviati a tutti gli host della rete.

Tramite il multicast, i messaggi vengono inviati ad un gruppo di host anche se appartenenti a LAN fisiche separate.

Classi di reti internet



Esempio di internet (Purdue University, 1980 circa)



Il protocollo IP

- Il servizio realizzato da IP è la consegna del datagramma.
- Il datagramma è un pacchetto di bit contenente:
  - i dati;
  - le informazioni ausiliare quali ad es:
    - indirizzo del mittente;
    - indirizzo del destinatario.

Il protocollo IP

- Il servizio realizzato da IP è la consegna del datagramma:
  - senza connessione. Ogni pacchetto è trattato indipendentemente dagli altri;
  - inaffidabile. La consegna del messaggio al destinatario non è garantita, possono esservi pacchetti persi, duplicati, ritardati o consegnati fuori sequenza. Il mittente non ha riscontro dell'avvenuta consegna;
  - best effort. Il software compie ogni tentativo per consegnare i pacchetti correttamente.

Il protocollo IP

- Fornisce:
  - formato esatto di tutti i dati;
  - funzioni di istradamento (routing), realizzato proprio in base all'indirizzo IP;
  - ogni gateway dispone di opportune tabelle (di routing) per l'istradamento;
  - insieme di regole che inglobano l'idea di consegna non affidabile.

Stratificazione del protocollo

- Viene utilizzata una famiglia (o suite) di protocolli.
- La suddivisione in protocolli consente di dividere un problema complesso in più sottoproblemi ed una migliore analisi di ogni singolo protocollo.
- I vari protocolli gestiscono differenti problemi.

Problemi gestiti dai vari protocolli

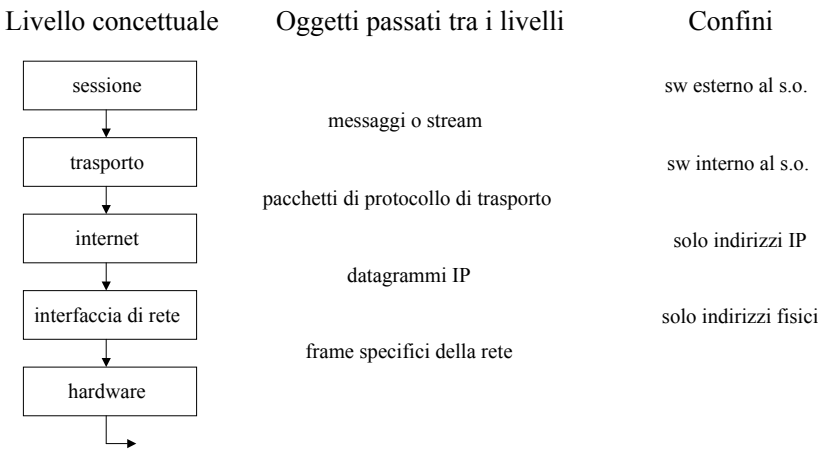
- malfunzionamenti hardware (host o gateway);
- congestione della rete: una macchina congestionata può sopprimere ulteriore traffico;
- ritardo o perdita di pacchetti: il software deve riconoscere ed adattarsi a ritardi lunghi e di durata variabile;
- alterazione dei dati dovuta a possibili errori di trasmissione;
- duplicazione dei dati o errore nella sequenza di trasmissione, ad es. dovuti a reti che offrono più percorsi alternativi.

I protocolli stratificati

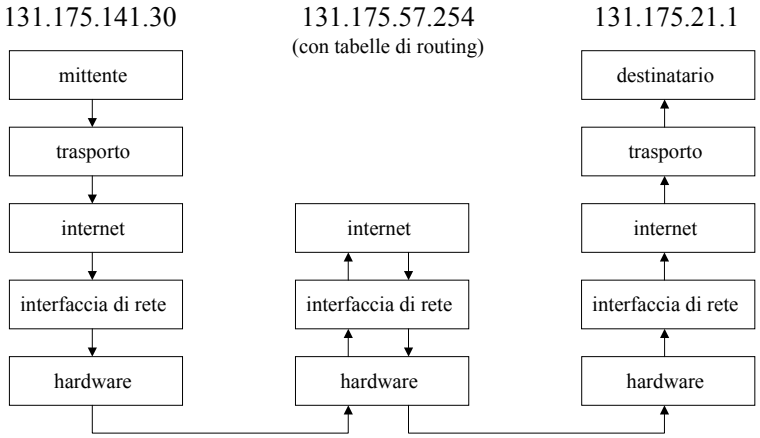
Il modello ISO/OSI prevede la definizione di 7 livelli di protocollo. Partendo dal più alto:

(ISO/OSI)	Internet Protocol Suite
• 7) di applicazione;	• http, smtp, ftp, telnet ...
• 6) di presentazione;	• (compressione pacchetti)
• 5) di sessione;	• (DNS)
• 4) di trasporto	• TCP e UDP
• 3) di rete;	• IP e routing
• 2) di collegamento dei dati;	• non specificati
• 1) fisico.	• non specificati

I protocolli stratificati



Comunicazione tra due host



Svantaggi della stratificazione

- Se la trasmissione è nella stessa sottorete, conviene utilizzare pacchetti di grandi dimensioni: se la trasmissione è tra sottoreti differenti, conviene utilizzare pacchetti di dimensioni inferiori.
- La stratificazione rende difficile l'ottimizzazione delle comunicazioni.
- Ma … è possibile utilizzare dei buffer.

Terminologia

- internet: una rete di reti;
- Internet (INTERconnected NETwork): la più diffusa internet del mondo.
- TCP/IP: il più diffuso protocollo per creare internet, ed usato da Internet.
- intranet: una rete privata basata sulle stesse tecnologie di Internet.
- extranet: diverse intranet collegate tra loro.

User Datagram Protocol

- IP non consente di distinguere più destinazioni di datagrammi all'interno della stessa macchina (vede **solo** l'indirizzo IP).
- E' una limitazione molto pesante: basti pensare che un semplice *telnet* su un host, lo isolerebbe da altre connessioni.
- Per superare tale limite è stato definito il protocollo User Datagram Protocol.

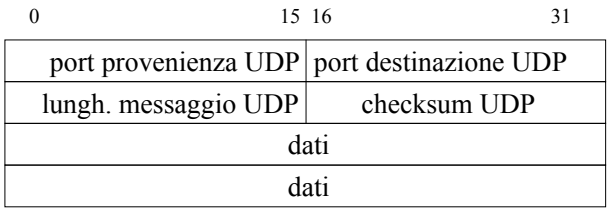
UDP

- Per distinguere più connessioni verso uno stesso calcolatore, si introduce il concetto di *port*, inteso come punto di destinazione di una connessione.
- Ogni port è identificato univocamente da un *unsigned int*.
- Un processo accede ad un port in modo sincrono e bufferizzato.
- La connessione UDP richiede quindi un indirizzo IP ed un numero di port.

UDP

- A parte la specifica di un port, UDP ha gli stessi vincoli di IP:
  - senza connessione;
  - inaffidabile;
  - best-effort.

Il datagramma UDP



Domanda: perchè manca l'indirizzo IP nel datagramma UDP?

Alcuni port riservati

num.	nome	descrizione	
11	USERS	systat	utenti attivi
13	DAYTIME	daytime	ora del giorno
15		netstat	stato della rete
17	QUOTE	quotd	citazione del giorno
20	FTP-DATA	ftp-data	dati FTP
21	FTP	ftp	FTP
23	TELNET	telnet	connessione di terminale
25	SMTP	smtp	simple mail transport protocol
37	TIME	time	tempo
42	NAMESERVER	name	server di nomi dell'host
53	DOMAIN	nameserver	server di nomi del dominio
79	FINGER	finger	
80	WEBSERVER	web server	server web
93	DCP	device control protocol	

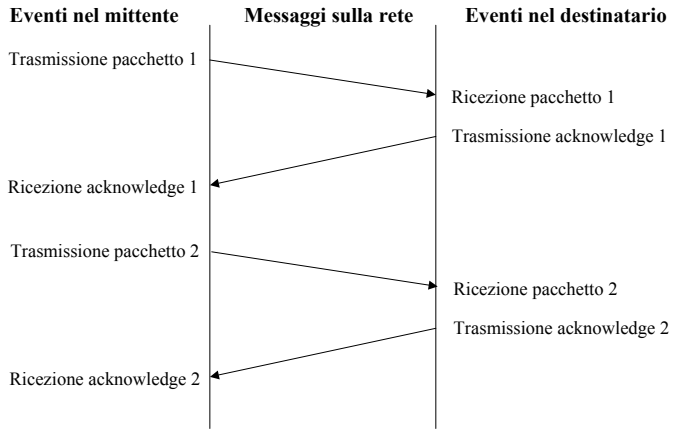
Servizio affidabile di trasporto di stream

- TCP (Trasmission Control Protocol)/IP:
  - orientamento allo stream: trasmette sequenze di byte;
  - connessione di circuito virtuale: chiamante e chiamato stabiliscono una connessione full-duplex utilizzando un circuito virtuale. In caso di fallimento, entrambi se ne accorgono;
  - trasferimento bufferizzato;
  - stream non strutturato: ad es. un record può essere frammentato su più pacchetti.

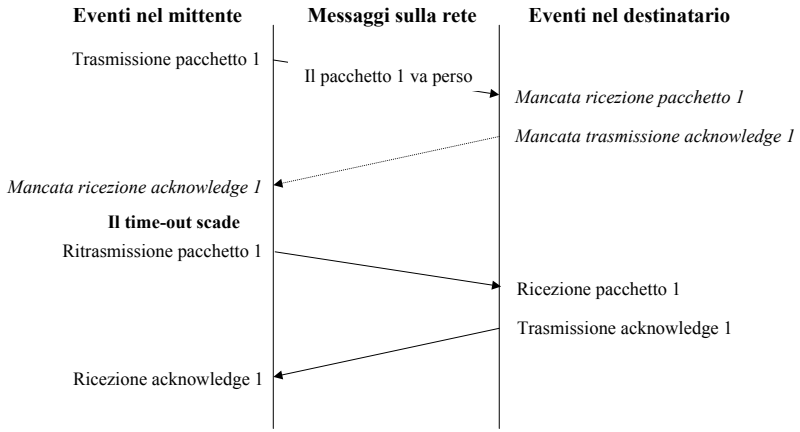
L'affidabilità di TCP/IP

- E' una caratteristica imprescindibile.
- E' basata sul riscontro positivo di ricezione (PAR - Positive Acknowledge with Retransmission).
- Il destinatario informa il mittente della ricezione del messaggio.
- Il mittente se non ottiene riscontro dal destinatario entro un certo tempo (time-out), arguisce la perdita del pacchetto.

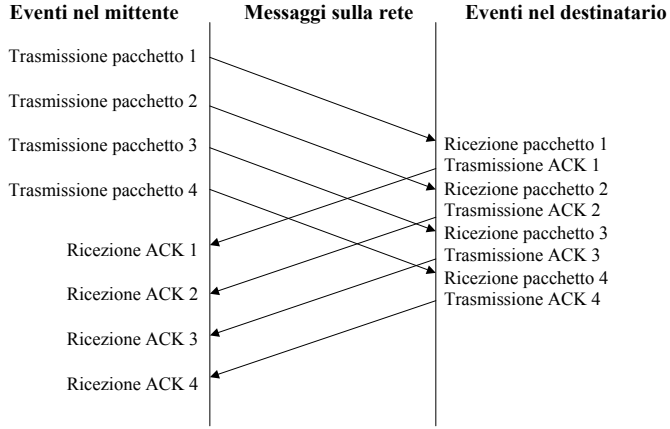
Protocollo con riscontro positivo di ricezione



Trasmissione con perdita di pacchetto



Trasmissione contemporanea di più pacchetti



Controllo di flusso

- Ogni riscontro dal ricevente indica anche il numero di byte che il ricevente è in grado di accettare.
- In tal modo il ricevente può indicare al trasmittente se si ha congestione sulla linea e addirittura annullare la trasmissione di altri pacchetti indicando di essere disposto a ricevere 0 byte.

Il datagramma TCP

- Il datagramma TCP descrive il formato dei pacchetti.
- Esso prevede:
  - una parte intestazione, di lunghezza fissa pari a 6 blocchi da 32 bit ciascuno (quindi 192 bit);
  - una parte dati, di lunghezza variabile.

Il datagramma TCP

- port di provenienza: port del mittente (16 bit);
- port di destinazione: port del destinatario (16 bit);
- numero sequenziale: posizione dei dati nello stream del mittente (32 bit);
- numero riscontro: numero del byte che il destinatario si aspetta di ricevere successivamente;
- hlen: lunghezza del segmento in multipli di 32 bit (4 bit). E' necessario perché il campo opzioni (vedi sotto) ha lunghezza variabile;

Il datagramma TCP

- riservato: per usi futuri (6 bit);
- codice: indica lo scopo ed il contenuto del segmento (6 bit):
  - URG: il campo puntatore urgente è valido;
  - ACK: il campo riscontro è valido;
  - PSH: il campo richiede un push;
  - RST: effettua il reset della connessione
  - SYN: sincronizza i numeri di sequenza;
  - FIN: il mittente ha raggiunto la fine del suo stream.

Il datagramma TCP

- finestra: dimensione del buffer dei dati (16 bit);
- checksum: controllo (16 bit);
- puntatore urgente: il pacchetto deve essere processato con urgenza - ad es. contiene ctrl-c ed i pacchetti residui non devono essere più inviati - (16 bit). Se il codice URG = 1, specifica la posizione della fine dei dati urgenti nella finestra;

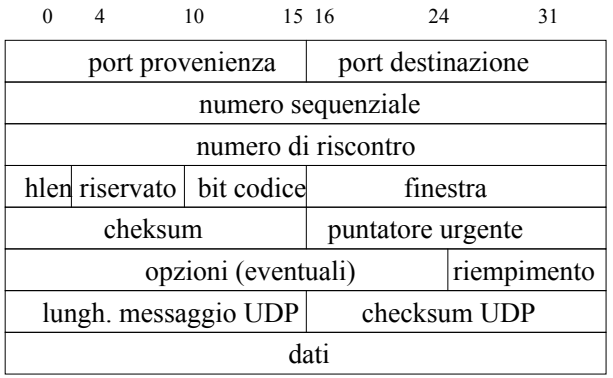
Il datagramma TCP

- opzioni: per informazioni aggiuntive, come ad es. stabilire la dimensione massima del segmento - maximum segment size - (24 bit);
- riempimento: completa le opzioni fino a 32 bit (8 bit);

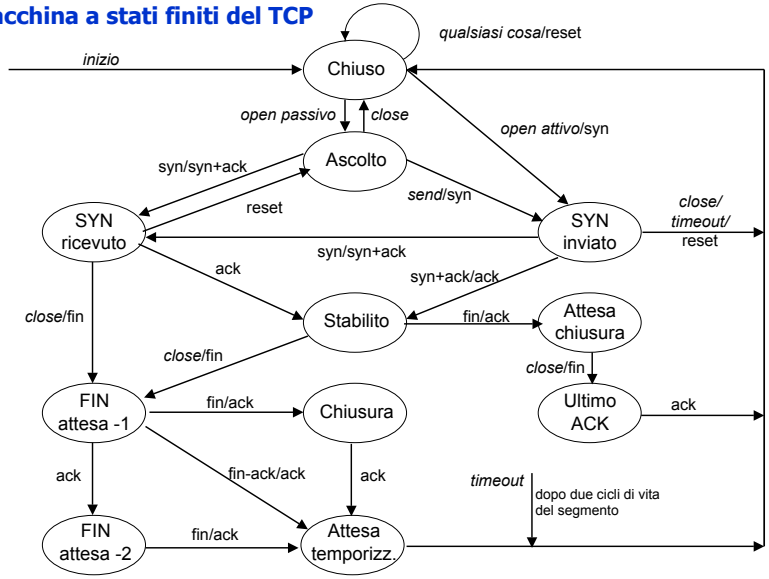
ed infine ….

- dati: i dati effettivamente trasmessi dal mittente al destinatario.

Il datagramma TCP



La macchina a stati finiti del TCP



La programmazione di rete

Introduzione alla
programmazione di rete

Applicazioni distribuite

- Applicazione: un insieme di programmi coordinati per svolgere una data funzione.
- Un'applicazione è distribuita se prevede più programmi eseguiti (o processi) su differenti calcolatori connessi tramite una rete.

  Es: Web Browser (Netscape, IE, Opera …) e Web Server (server http)

Protocollo applicativo

- Le regole per la comunicazione in una applicazione distribuita sono dette *protocollo applicativo*.

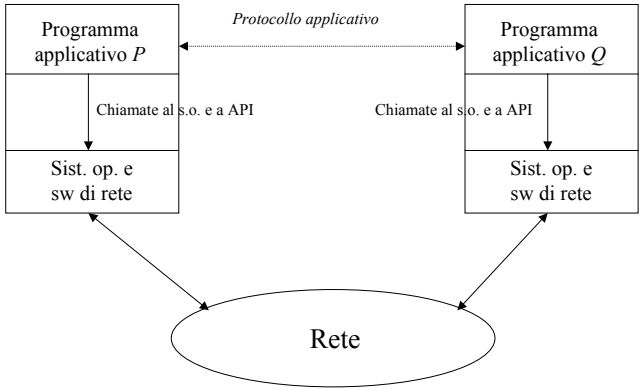
  Es. il protocollo applicativo della navigazione Web è detto HyperText Transfer Protocol - http.
- Il protocollo applicativo deve essere definito opportunamente e comune a tutti i programmi dell'applicazione.

  Es. ogni messaggio scambiato è terminato dalla stringa "\0 \0 \0".

Interfacce e protocolli

- I programmi applicativi utilizzano opportu-ne interfacce (API - application program interface), fornite dal sistema operativo e dal software di rete, per comunicare:
  - le API forniscono il canale (o supporto) di comunicazione;
  - il protocollo applicativo rappresenta le regole di comunicazione, e considera il contenuto della comunicazione.

Interfacce e protocolli



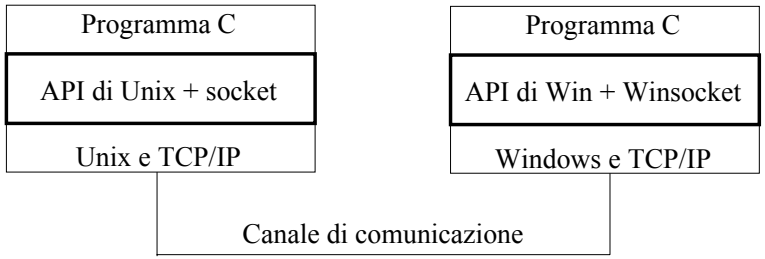
L'interfaccia di socket

- L'API standard per TCP/IP (il protocollo di internet) si chiama "interfaccia di socket" ed è stata definita a Berkley agli albori di internet (circa 1980).
- Socket funziona anche per altri protocolli differenti da TCP/IP.
- L'interfaccia di socket è in linguaggio C ed è per calcolatori Unix: per calcolatori Windows si utilizza WinSocket.

L'interfaccia di socket

- Il socket è il descrittore di una connessione.
- TCP/IP garantisce l'interoperabilità tra calcolatori anche se equipaggiati con sistemi operativi differenti.
- Calcolatori Unix e calcolatori Windows possono comunicare utilizzando, rispettivamente, le interfacce Socket e WinSocket.

Connessione tra calcolatori Unix e Windows



Il modello Client/Server

- Il software di rete TCP/IP consente la comunicazione tra calcolatori in modalità peer-to-peer (pari-a-pari) purché condividano un protocollo comune.
- Spesso i processi che cooperano sono due:
  - client: effettua la richiesta di un servizio ad un calcolatore che svolge la funzione di server;
  - server: soddisfa la richiesta del client. Il server viene pertanto attivato **prima** del client.

Il modello C/S

- Generalmente un processo conserva lo stesso ruolo (è o sempre Client o sempre Server).

Esempio:

- il Web client (Netscape, IE, Opera ….) richiede una data pagina ad un Web server;
- il Web server rintraccia la pagina specificata e la fornisce a chi ne abbia fatto richiesta (client);
- client e server utilizzano un protocollo comune (nel caso, http).

Indirizzamento

- Siano P e Q i due processi che devono comunicare. Q (client) deve richiedere un servizio a P (server).
- E' necessario che Q sappia come *raggiungere* P (per inviargli la richiesta) e che poi P sappia come raggiungere Q (per restituire quanto richiesto o segna-lare il completamento dell'operazione richiesta dallo stesso Q).

Indirizzamento

- Q indirizza P specificando il calcolatore sul quale P è in esecuzione ed il numero di *port*, che identifica il processo P sul suo calcolatore.
- Il calcolatore è specificato tramite l'indirizzo IP (Internet Protocol):
  - l'indirizzo IP è composto da 4 byte (quindi 32 bit);
  - un esempio di indirizzo IP è 131.175.21.8

Indirizzamento

- Il port:
  - identifica un processo nell'ambito del calcolatore ai fini della comunicazione;
  - generalmente non coincide con il PID del processo (che è assegnato dal sistema operativo ad ogni processo: Q non avrebbe modo di conoscerlo e quindi non potrebbe indirizzare P);
  - alcuni port (compresi tra 0 e 1023) sono assegnati a servizi standard e non possono essere utilizzati per sviluppare propri server.

Indirizzamento

- L'indirizzamento completo è quindi specificato fornendo l'indirizzo IP della macchina ed il port.

  Es. <131.175.21.8, 80> significa:
  - port 80 (di solito è il port del Web server)
  - della macchina 131.175.21.8

Il modello di comunicazione e la connessione

- La comunicazione TCP è connection-oriented (orientata alla connessione): la prima cosa da fare è stabilire una connessione, seguiranno poi lo scambio dei dati.
- E' necessario che P (server) rimanga in attesa di una richiesta di connessione da Q (client). Viene poi stabilita la connessione.

Il modello di comunicazione e la connessione

- Sequenza per la connessione:
  - P e Q si identificano come *punti terminali* della connessione;
  - ogni punto terminale è identificato dalla coppia <indirizzo IP, numero di port>;
  - ogni connessione è quindi identificata da 4 numeri: <indirizzo IP di P, numero di port di P>, <indirizzo IP di Q, numero di port di Q>;
  - dopo la connessione, il canale è bidirezionale, affidabile e orientato allo stream.

Il modello di comunicazione e la connessione

- Le principali caratteristiche del canale di comunicazione sono:
  - bidirezionale: P colloquia con Q e Q con P;
  - orientato allo stream: consente una trasmissione continua di byte (o di gruppi di byte);
  - affidabile: se il destinatario (Q o P) non riceve un byte (o un gruppo di byte), il mittente (P o Q) se ne accorge. E' la caratteristica del TCP, non presente nell'IP.

Tipi di calcolatori e formato dei dati

- I calcolatori che si affacciano sulla rete possono essere di tipi e con sistemi operativi differenti. I dati possono essere rappresentati in modi differenti.
- Il TCP/IP fornisce un formato di rete unico: ogni calcolatore dispone poi di routine per convertire i dati dal formato TCP/IP al proprio formato locale.

La programmazione di rete

La connessione

Il meccanismo accept-connect

- P (server) si pone in attesa di richieste di connessione (apertura passiva).

  Si utilizza la funzione *accept*, che è sospensiva.
- Q (client) formula richiesta a P di apertura di connessione (apertura attiva).

  Si utilizza la funzione *connect*.

Il meccanismo accept-connect

- La connessione può fallire perché:
  - il processo P (server) non esiste;
  - P esiste ma non ha ancora eseguito la *accept*;
  - P esiste ma è occupato e la coda dei processi che richiedono P è già piena;
  - Q (client) non conosce gli esatti valori dell'indirizzo IP del calcolatore di P e del port di P;
  - …..

Un semplice client: UClient1

- Le operazioni svolte da Q (client):
  - definisce una opportuna variabile di tipo *struct sockaddr_in* per contenere le informazioni su un punto terminale;
  - inizializza la struct con dati validi;
  - crea un *socket* della connessione (descrittore), facendosi assegnare un port;
  - esegue la *connect* specificando il proprio *socket*.

n.b. il cocket viene gestito come un file descriptor.

struct sockaddr_in

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

struct sockaddr_in

- sin_family: famiglia di indirizzi AF_INET
  (Address Family InterNet)
  altre famiglie: AF_UP (Xerox), AF_APPLETALK (AppleTalk), AF_UNIX (UNIX)
- sin_port: numero di port, a 16 bit (0-65535)
- sin_addr: indirizzo IP, 32 bit
- sin_zero: non utilizzato

Funzione *addr_initialize*

```
#include <sys/type.h>
#include <sys/socket.h>
#include <netinet/in.h>
void addr_initialize(struct sockaddr_in *indirizzo, int port,
       long IPaddr)
{
   indirizzo->sin_family = AF_INET;
   indirizzo->sin_port = htons((u_short) port);
        /* htons: host to network conversion, short */
   indirizzo->sin_addr.s_addr = IPaddr;
}
void addr_show(struct sockaddr_in *indirizzo)
{
   printf("IP = %s, ", inet_ntoa(indirizzo->sin_addr.s_addr));
   printf("port = %d\n", ntohs(indirizzo->sin_port));
}
```


UClient1

- Legge sulla linea di comando l'indirizzo IP del calcolatore dove c'è il processo P;
- non deve specificare il proprio indirizzo IP;
- crea un proprio socket ed ottiene un proprio port tramite l'istruzione:
  sd = socket(AF_INET, SOCK_STREAM, 0);
  di cui può conoscere il valore tramite la
  getsockname(sd, &mio_addr, &mio_addr_len).

n.b. il secondo parametro della funzione socket assume i valori:
SOCK_STREAM per protocollo TCP/IP;
SOCK_DGRAM per UDP.

UClient1

- Esegue la richiesta di connessione:
  error = connect(sd, (struct sockaddr*) &server_addr, sizeof(server_addr));
- al termine, chiude il socket rilasciando il port che può essere utilizzato da altri programmi:
  close (sd);

Sintassi di *connect*

```
int connect (int sd, struct sockaddr_in *
  server_ep, int ep_len)
/* Invia una richiesta di collegamento in qualità
  di cliente, restituisce 0 se successo, -1 se
  errore. Parametri: sd, specifica il socket (che
  deve essere già stato creato) da associare al
  canale di rete; server_ep, specifica il punto
  terminale (endpoint) del destinatario della
  richiesta di collegamento, che è il server;
  ep_len, specifica la lunghezza in byte del
  punto terminale */
```


Sintassi di *close*

```
int close (int sd)
/* Cancella un socket e ne rilascia il
  descrittore, -1 = errore. Il canale di rete
  associato al socket viene eliminato.
Parametri: sd, è il socket da cancellare. Il
  socket cancellato non è più utilizzabile per
  aprire canali. */
```


Sintassi di creazione di un *socket*

```
int socket (int family, int type, int protocol)
/* Crea un socket e ne restituisce il
  descrittore, -1 = errore.
```


Sintassi di creazione di un *socket* (ii)

```
Parametri: family, definisce la famiglia di protocolli
  (AF_INET per TCP/IP, AF_UP per Xerox, ...); type,
  specifica il tipo di comunicazione (SOCK_STREAM per
  servizio di consegna affidabile TCP, SOCK_DGRAM per
  datagramma senza connessione UDP, ...); protocol,
  specifica quale protocollo utilizzare se nella famiglia
  utilizzata ne esiste più di uno, normalmente vale 0.
Nota bene: il socket da solo non è ancora il canale di
  rete, è soltanto una struttura dati che serve per
  gestire il canale di rete; per aprire un canale di rete
  associandolo al socket occorre chiamare la primitiva
  connect oppure la primitiva accept */
```

Altre funzioni utili

unsigned long in_addr_t inet_addr (char * stringa)

/* Converte l'indirizzo IP da stringa di caratteri a formato di rete */

char * inet_ntoa (unsigned long address)

/* Converte l'indirizzo IP da formato di rete a stringa di caratteri */

unsigned short int htons (unsigned short int port)

/* Converte il numero di port TCP da numero intero a formato di rete */

unsigned short int ntohs (unsigned short int port)

/* Converte il numero di port TCP da formato di rete a numero intero */

Listato di UClient1 (i)

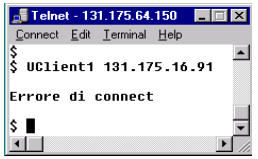
```
                    /* programma UCLIENT1  */

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

#define PORT 4000         /* port pubblico del server */

void addr_initialize();

void main(int argc, char* argv[])
{
   int sd;
   struct sockaddr_in server_addr;
   struct sockaddr_in mio_addr;
   int mio_addr_len=sizeof(mio_addr);
   int error;
```


Listato di UClient1 (ii)

```
   addr_initialize(&server_addr, PORT, inet_addr(argv[1]));

   sd=socket(AF_INET,SOCK_STREAM,0);
   error=connect(sd,(struct sockaddr*) &server_addr,
         sizeof(server_addr));
   if (error==0) {
        printf("Ho eseguito la connessione\n");
        getsockname(sd, &mio_addr, &mio_addr_len);
       printf("il mio port e': %d\n\n",ntohs(mio_addr.sin_port));
        close(sd);
   }
   else printf("%s","\nErrore di connect\n\n");
   close(sd);
}
```


Esecuzione di UClient1

Se il server non è attivo, il client riporta il fallimento della connessione:



Un semplice server: UServer1

- Le operazioni svolte da P sono analoghe a quelle svolte da Q:
  - predispone una variabile *client_addr* per memorizzare l'indirizzo IP ed il port di Q del client per potergli rispondere;
  - inizializza il proprio indirizzo, specificando il numero di port del "canale pubblico" e che intende accettare qualsiasi connessione (qualsiasi IP) a questo;
  - crea un socket *sd* per accettare connessioni;

UServer1

- indica al TCP/IP che l'indirizzo locale associato al socket *sd* è quello contenuto in *server_addr*:

  bind(sd, (struct sockaddr*) &server_addr, sizeof(server_addr));
- si pone in attesa di una richiesta di connessione:

  new_sd = accept(sd,

  (struct sockaddr)&client_addr, &client_len);*
- i dati del connesso sono nella *struct client_addr* raggiungibile tramite *new_sd*;

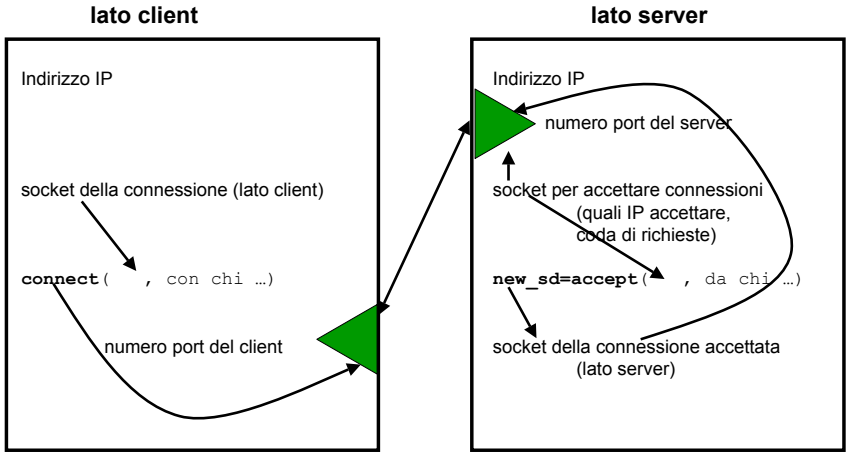
UServer1

- il server chiude la connessione con il client tramite la

  close(new_sd);
- In aggiunta il server:
  - può stabilire il numero massimo di richieste di connessione che può accodare (MAXCONN), servendone però sempre una sola per volta:

    listen(sd, MAXCONN);

Connessione lato client e lato server



Sintassi di *accept*

```
int accept (int sd, struct sockaddr_in *
  client_ep, int * ep_len)
/* Accetta una richiesta di collegamento
  in qualità di servente, restituisce un
  nuovo socket (sempre >= 0) se successo,
  -1 se errore; il nuovo socket restituito
  è quello su cui portare avanti il
  dialogo con il cliente richiedente; il
  vecchio socket è disponibile per
  ulteriori accettazioni.
```


Sintassi di *accept* (ii)

Parametri: **sd**, specifica il socket (che deve essere già stato creato) su cui ricevere la richiesta di collegamento proveniente dal cliente; **client_ep**, specifica la locazione in cui memorizzare il punto terminale (endpoint) del cliente; **ep_len**, specifica la locazione in cui memorizzare la lunghezza in byte del punto terminale */

Sintassi di *bind*

```
int bind (int sd, struct sockaddr_in * server_ep,
  int ep_len)
/* Associa un numero di porta TCP a un socket,
  restituisce 0 se successo, -1 se errore.
Parametri: sd, specifica il socket da associare
  al numero di porta TCP; server_ep, specifica il
  punto terminale (endpoint) contenente il numero
  di porta da associare (l'indirizzo ha funzione
  di filtro); ep_len, specifica la lunghezza in
  byte del punto terminale */
```


Sintassi di *listen*

```
int listen (int sd, int len)
/* Crea e dimensiona la coda di richieste di
  connessione pendenti associata al socket,
  restituisce 0 se successo, -1 se errore.
Parametri: sd, specifica il socket per cui creare
  la coda; len, specifica la lunghezza max della
  coda */
```

Sintassi di *getsockname*

int getsockname (int sd, struct sockaddr_in * local_ep, int * ep_len)

/* Cerca il punto terminale locale del canale e lo restituisce.

Parametri: **sd**, socket associato al canale il cui punto terminale locale si vuole conoscere; **local_ep**, punto terminale locale; **ep_len**, lunghezza del punto terminale */

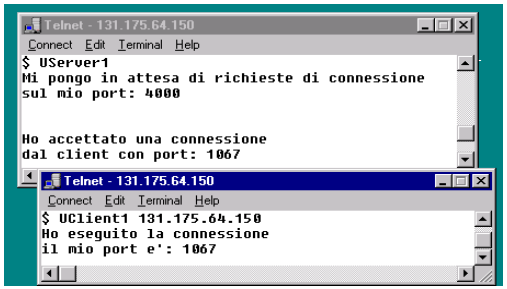
Listato di UServer1 (i)

```
                                /* programma USERVER1  */
#include <stdio.h>
#include <sys/types.h>
#include <sys/socket.h>
#include <netinet/in.h>

#define PORT 4000
#define MAXCONN 5

void addr_initialize();

void main(int argc, char * argv[])
{
   int sd,new_sd,bind_result,listen_result;
   struct sockaddr_in server_addr;
   struct sockaddr_in client_addr;
   int client_len=sizeof(client_addr);
```


Listato di UServer1 (ii)

```
   addr_initialize(&server_addr, PORT, INADDR_ANY);

   sd=socket(AF_INET,SOCK_STREAM,0);
   bind_result=bind(sd,(struct sockaddr*) &server_addr,sizeof(server_addr));
   if (bind_result!=0) {
        fprintf(stderr,"Errore nella funzione bind\n\n");
        exit(1);
   }
   listen_result=listen(sd,MAXCONN);
   printf("Mi pongo in attesa di richieste di connessione\n");
   printf("sul mio port: %d\n", ntohs(server_addr.sin_port));
   new_sd=accept(sd,(struct sockaddr*) &client_addr, &client_len);
   printf("\n\nHo accettato una connessione\n");
   printf("dal client con port: %d\n\n", ntohs(client_addr.sin_port));
   close(new_sd);
   close(sd);
}
```


UClient1 e UServer1 in esecuzione



Versioni Windows

- Analogamente alla versioni per calcolatori Unix (UClient1 e UServer1), i client ed i server possono essere scritti su calcolatori Windows, utilizzando WinSocket anziché Socket.
- Chiameremo WClient1 e WServer1 i rispettivi client e server.

Interoperabilità

- E' garantita l'interoperabilità:
  - WClient1 può collegarsi a UServer1 o a WServer1.
  - UClient1 può collegarsi a UServer1 o a WServer1.

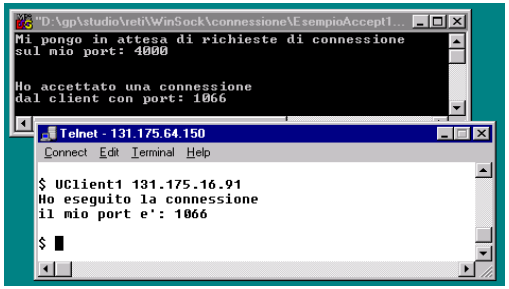
Listato di WClient1 (i)

```
                    /* programma WCLIENT1  */

#include <stdio.h>
#include <winsock2.h>
#define PORT 4000
int winDLLstart();
void addr_initialize();
void main(int argc, char* argv[])
{
    SOCKET sd;
    struct sockaddr_in server_addr;
    struct sockaddr_in mio_addr;
    int mio_addr_len=sizeof(mio_addr);
    int error;
```

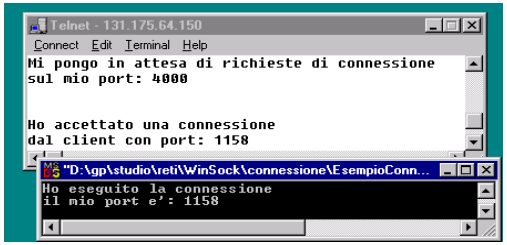

Listato di WClient1 (ii)

```
    addr_initialize(&server_addr, PORT, inet_addr(argv[1]));
    if (winDLLstart() != 0) return;
    sd=socket(AF_INET,SOCK_STREAM,0);
    error=connect(sd,(struct sockaddr*) &server_addr,sizeof(server_addr));
    if (error==0) {
         printf("Ho eseguito la connessione\n");
         getsockname(sd, &mio_addr, &mio_addr_len);
         printf("il mio port e': %d\n\n",ntohs(mio_addr.sin_port) );
         closesocket(sd);
    }
    else printf("%s","\nErrore di connect\n\n");
    closesocket(sd);
}
```

UClient1 e WServer1 in esecuzione



WClient1 e UServer1 in esecuzione



Connessione simmetrica

- Supponiamo di dover realizzare una comunicazione tra due programmi, non sapendo quale dei due venga attivato per primo. Il C/S non può essere utilizzato.
- Ognuno dei due programmi:
  - tenterà di connettersi all'altro (*connect*). Se non ci riesce, eseguirà la *accept* aspettando la richiesta di *connect* dell'altro;
  - a connessione avvenuta, ha inizio la comunicazione.

Listato di Usimmetric (i)

```
                    /* programma   USimmetric     */

#include <stdio.h>
#include <sys/types.h>
#include <sys/socket.h>
#include <netinet/in.h>
#define PORT 3000
#define MAXCONN 5

void addr_initialize();
void main(int argc, char* argv[])
{
   int con_sd, acc_sd, new_sd;
   int error,bind_result,listen_result;
   struct sockaddr_in mio_addr;
   int mio_addr_len=sizeof(mio_addr);
   struct sockaddr_in altro_addr;
   int altro_addr_len=sizeof(altro_addr);
```

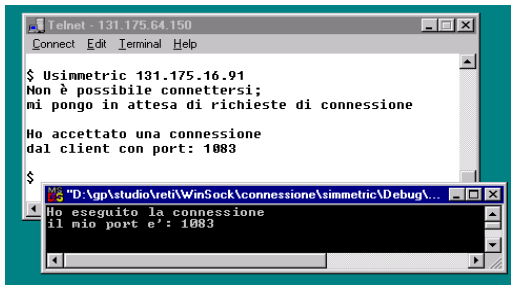
Listato di Usimmetric (ii)

```
                /*  tento la connessione    */
addr_initialize(&altro_addr, PORT, inet_addr(argv[1]));
con_sd=socket(AF_INET,SOCK_STREAM,0);
error = connect(con_sd,(struct sockaddr*)
       &altro_addr,sizeof(altro_addr));

if (error == 0)  /*connessione riuscita*/
     {
     printf("Ho eseguito la connessione\n");
     getsockname(con_sd, &mio_addr, &mio_addr_len);
     printf("il mio port e': %d\n\n\n",
         ntohs(mio_addr.sin_port) );
     close(con_sd);
}
```


Listato di Usimmetric (iii)

```
else                    /*connessione fallita*/
   {    printf("%s","Non è possibile connettersi;\n");
        printf("%s","mi pongo in attesa di richieste di
                connessione\n\n");
        addr_initialize(&mio_addr, PORT, INADDR_ANY);
        acc_sd=socket(AF_INET,SOCK_STREAM,0);
        bind_result=bind(acc_sd,(struct sockaddr*) &mio_addr,
                sizeof(mio_addr));
        listen_result=listen(acc_sd,MAXCONN);
        new_sd=accept(acc_sd,(struct sockaddr*)&altro_addr,
                &altro_addr_len);
        printf("Ho accettato una connessione\n");
        printf("dal client con port: %d\n\n",
                 ntohs(altro_addr.sin_port));
        close(new_sd);
        close(acc_sd);
   }
}
```


USimmetric in esecuzione



La programmazione di rete

La trasmissione

La trasmissione

- Dopo aver stabilito la connessione bidirezionale (*accept*/*connect*), i due processi si scambiano dati attraverso *send* e *recv*.
  - *send*:spedisce dati all'altro punto terminale della connessione;
  - *recv*:riceve dati inviati dall'altro punto terminale della connessione.

send

- Copia *numero_byte* byte da *indirizzo_buffer* a buffer di sistema;
- non è sospensiva (cioè non attende che i dati siano arrivati al destinatario), a meno che *numero_byte* sia maggiore della dimensione del buffer di sistema;
- *retval* indica i byte effettivamente inviati. Assume valore negativo se si è verificato un errore.

La sintassi di *send*

```
int send (int sd, char * message, int len,
  int flags)
/*    Spedisce,    attraverso    il    canale
  identificato da sd, len byte memorizzati
  nella stringa message.
Restituisce il numero di byte effettivamente
  inviati, -1 se errore.
Altri  param.:  flags,  specifica  funzioni
  speciali, di solito 0 */
```


Esempio di *send*

retval = send(socket, indirizzo_buffer, numero_byte, flags);

socket: identifica la connessione;

indirizzo_buffer: zona di memoria da cui prelevare i byte da inviare;

numero_byte: quanti byte inviare;

flags: per funzioni speciali. Ipotizzeremo sempre 0;

retval: esito dell'operazione di *send*.

recv

- Copia *numero_byte* byte dal buffer di sistema a *indirizzo_buffer*;
- è sospensiva, cioè attende che siano arrivati dei dati o si sia verificato un errore (ad es. chiusura connessione).

La sintassi di *recv*

```
int recv (int sd, char * message, int len,
  int flags)
/* Riceve, attraverso il canale identificato
  da sd, len byte e li memorizza nella
  stringa message.
Restituisce il numero di byte effettivamente
  ricevuti, -1 se errore.
Altri param.: flags, specifica funzioni
  speciali, di solito 0 */
```


Esempio di *recv*

retval = recv(socket, indirizzo_buffer, numero_byte, flags);

socket: identifica la connessione;

indirizzo_buffer: zona di memoria in cui scrivere i byte ricevuti;

numero_byte: numero di byte da leggere;

flags: per funzioni speciali. Ipotizzeremo sempre 0;

retval: esito dell'operazione di *receive*.

retval in *recv*

- Se arrivano byte in numero inferiore a *numero_byte*, *retval* assume il valore del numero di byte arrivati;
- se arrivano byte in numero superiore a *numero_byte*, *retval* assume il valore *numero_byte* ed i dati in eccedenza sono mantenuti nel buffer di sistema, utilizzabili per la *receive* successiva;
- *retval* assume valore negativo se si è verificato un errore o è stata chiusa la connessione.

UClient2

- UClient2:
  - stabilisce la connessione con UServer2;
  - chiede all'utente di inserire il numero di caratteri che si vogliono inviare (*num*);
  - preleva da un buffer (*char dati[12]*) tali caratteri e li invia al server;
  - ripete la richiesta di *num*. Termina, se l'utente ha indicato che non si vogliono spedire ulteriori caratteri (*num* = 0), chiudendo la connessione.

Listato di UClient2 (i)

```
                                /* programma UCLIENT2  */
#include <stdio.h>
#include <sys/socket.h>
#define PORT 4000
void addr_initialize();
void main(int argc, char* argv[])
{
   int sd;
   struct sockaddr_in server_addr;
   struct sockaddr_in mio_addr;
   int mio_addr_len=sizeof(mio_addr);
   int error, num, inviati;
   char dati[12]="abcdefghilm";

   addr_initialize(&server_addr, PORT, inet_addr(argv[1]));
   sd=socket(AF_INET,SOCK_STREAM,0);
   error=connect(sd,(struct sockaddr*) &server_addr, sizeof(server_addr));
```


Listato di UClient2 (ii)

```
if (error==0)
   {
        printf("Ho eseguito la connessione\n");
        printf("\n inserire il numero di caratteri da trasmettere: ");
        scanf("%d", &num);
        while (num > 0)
        {
                inviati=send(sd,dati,num,0);
                printf(" inserire il numero di caratteri da "
                        "trasmettere: ");
                scanf("%d", &num);
        }
        printf("\n Chiudo la connessione \n");
        close(sd);
   }
   else printf("%s","\nErrore di connect\n\n");
   close(sd);
}
```


UServer2

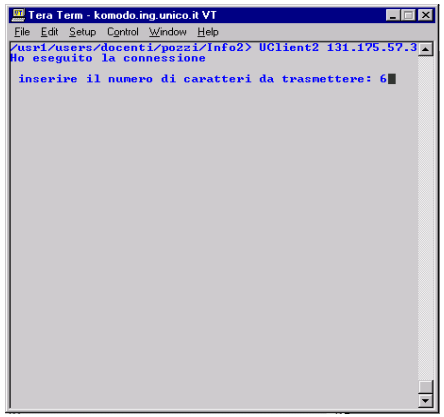
- UServer2:
  - accetta la connessione (da UClient2);
  - riceve un numero variabile di caratteri (*ric*), con un limite massimo di *DIMBUF*;
  - stampa a video il numero di caratteri ricevuti ed i caratteri stessi. Poi,
    - si prepara per una nuova ricezione di caratteri dalla stessa connessione;
    - se *ric* < 0, chiude la connessione con UClient2 e si prepara per una nuova connessione da un nuovo client.

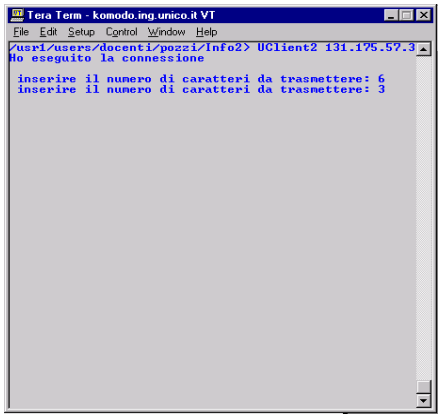
Listato di UServer2 (i)

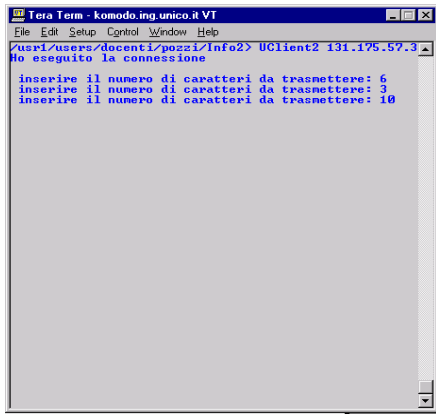
```
                    /* programma USERVER2  */
#include <stdio.h>
#include <sys/socket.h>
#define PORT 4000
#define MAXCONN 5
#define DIMBUF 6

void addr_initialize();
void main(int argc, char * argv[])
{
   int sd,new_sd,bind_result,listen_result;
   struct sockaddr_in server_addr;
   struct sockaddr_in client_addr;
   int client_len=sizeof(client_addr);
   int ric=1, i;
   char buf[DIMBUF];

   addr_initialize(&server_addr, PORT, INADDR_ANY);
```

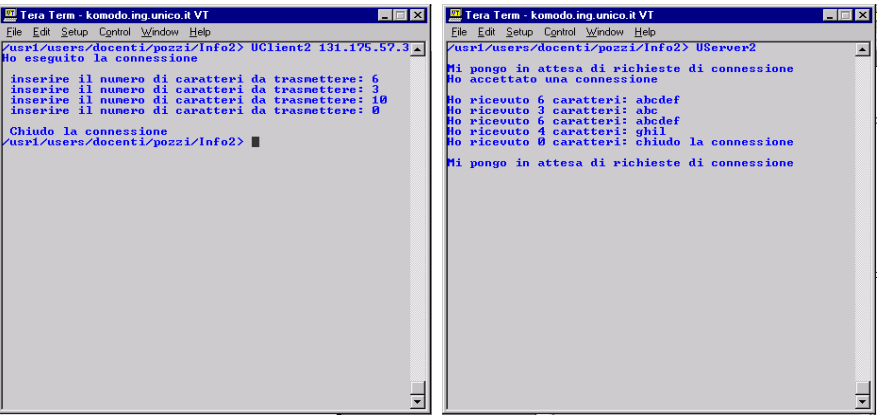

Listato di UServer2 (ii)

```
   sd=socket(AF_INET,SOCK_STREAM,0);
   bind_result=bind(sd,(struct sockaddr*) &server_addr,sizeof(server_addr));
   listen_result=listen(sd,MAXCONN);
   while (1)
   {
        printf("\nMi pongo in attesa di richieste di connessione\n");
        new_sd=accept(sd,(struct sockaddr*) &client_addr, &client_len);
        printf("Ho accettato una connessione\n");
        ric=1;
        while (ric>0)
                {
                ric=recv(new_sd,buf,DIMBUF,0);
                printf("\nHo ricevuto %d caratteri: ",ric);
                for (i=0; i<ric;i++) printf("%c", buf[i]);
        }
        close(new_sd);
        printf("chiudo la connessione\n");
   }                    /* fine del ciclo perenne */
   close(sd);    /* inutile: non si arriva mai a q.to punto */
}
```







Bufferizzazione

- Modifichiamo il codice di UServer2 e prima della *receive* inseriamo:

  printf("\ninserire un CR per iniziare la recv");
  getchar();

La sequenza di inserimento dei dati nei due programmi sarà:

WClient2:	5	3	1	3	0
WServer2:	CR	(*)	CR	CR	

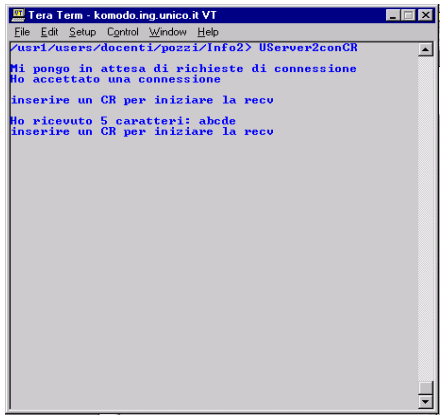
Ciò che otterremo sarà:

Listato di UServer2 con CR (ii)

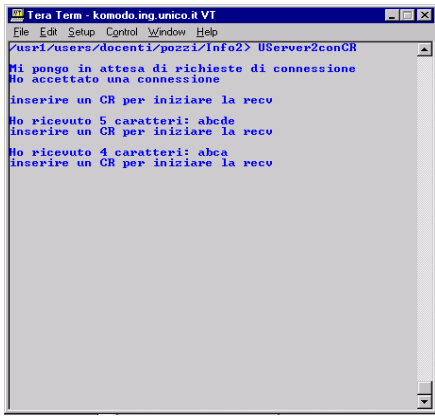
```
while (1)
{
      printf("\nMi pongo in attesa di richieste di connessione\n");
      new_sd=accept(sd,(struct sockaddr*) &client_addr, &client_len);
      printf("Ho accettato una connessione\n");
      ric=1;
      while (ric>0)
              {
              printf("\ninserire un CR per iniziare la recv");
              getchar();
              ric=recv(new_sd,buf,DIMBUF,0);
              printf("\nHo ricevuto %d caratteri: ",ric);
              for (i=0; i<ric;i++) printf("%c", buf[i]);
      }
      close(new_sd);
      printf("chiudo la connessione\n");
}                    /* fine del ciclo perenne */
close(sd);    /* inutile: non si arriva mai a q.to punto */
}
```



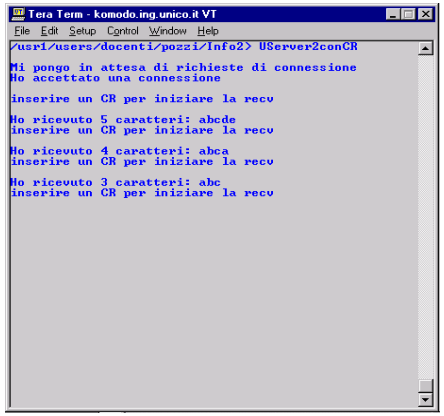
Tera Term - komodo.ing.unico.it VT
File Edit Setup Control Window Help
/usr1/users/docenti/pozzi/Info2> UServer2conCR
Mi pongo in attesa di richieste di connessione
Ho accettato una connessione
inserire un CR per iniziare la recv

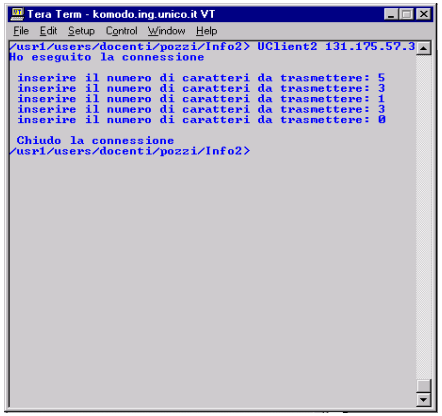

Tera Term - komodo.ing.unico.it VT
File Edit Setup Control Window Help
/usr1/users/docenti/pozzi/Info2> UServer2conCR
Mi pongo in attesa di richieste di connessione
Ho accettato una connessione
inserire un CR per iniziare la recv
Ho ricevuto 5 caratteri: abcde
inserire un CR per iniziare la recv

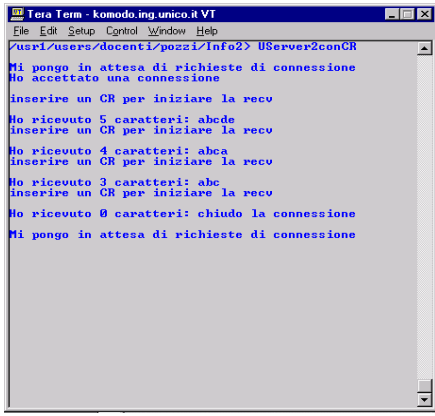

Tera Term - komodo.ing.unico.it VT
File Edit Setup Control Window Help
/usr1/users/docenti/pozzi/Info2> UClient2 131.175.57.3
Ho eseguito la connessione
inserire il numero di caratteri da trasmettere: 5
inserire il numero di caratteri da trasmettere: 3
inserire il numero di caratteri da trasmettere: 1
inserire il numero di caratteri da trasmettere:


Tera Term - komodo.ing.unico.it VT
File Edit Setup Control Window Help
/usr1/users/docenti/pozzi/Info2> UServer2conCR
Mi pongo in attesa di richieste di connessione
Ho accettato una connessione
inserire un CR per iniziare la recv
Ho ricevuto 5 caratteri: abcde
inserire un CR per iniziare la recv
Ho ricevuto 4 caratteri: abca
inserire un CR per iniziare la recv


Tera Term - komodo.ing.unico.it VT
File Edit Setup Control Window Help
/usr1/users/docenti/pozzi/Info2> UClient2 131.175.57.3
Ho eseguito la connessione
inserire il numero di caratteri da trasmettere: 5
inserire il numero di caratteri da trasmettere: 3
inserire il numero di caratteri da trasmettere: 1
inserire il numero di caratteri da trasmettere: 3
inserire il numero di caratteri da trasmettere:


Tera Term - komodo.ing.unico.it VT
File Edit Setup Control Window Help
/usr1/users/docenti/pozzi/Info2> UServer2conCR
Mi pongo in attesa di richieste di connessione
Ho accettato una connessione
inserire un CR per iniziare la recv
Ho ricevuto 5 caratteri: abcde
inserire un CR per iniziare la recv
Ho ricevuto 4 caratteri: abca
inserire un CR per iniziare la recv
Ho ricevuto 3 caratteri: abc
inserire un CR per iniziare la recv


Tera Term - komodo.ing.unico.it VT
File Edit Setup Control Window Help
/usr1/users/docenti/pozzi/Info2> UClient2 131.175.57.3
Ho eseguito la connessione
inserire il numero di caratteri da trasmettere: 5
inserire il numero di caratteri da trasmettere: 3
inserire il numero di caratteri da trasmettere: 1
inserire il numero di caratteri da trasmettere: 3
inserire il numero di caratteri da trasmettere: 0
Chiudo la connessione
/usr1/users/docenti/pozzi/Info2>


Tera Term - komodo.ing.unico.it VT
File Edit Setup Control Window Help
/usr1/users/docenti/pozzi/Info2> UServer2conCR
Mi pongo in attesa di richieste di connessione
Ho accettato una connessione
inserire un CR per iniziare la recv
Ho ricevuto 5 caratteri: abcde
inserire un CR per iniziare la recv
Ho ricevuto 4 caratteri: abca
inserire un CR per iniziare la recv
Ho ricevuto 3 caratteri: abc
inserire un CR per iniziare la recv
Ho ricevuto 0 caratteri: chiudo la connessione
Mi pongo in attesa di richieste di connessione

Il protocollo applicativo

- TCP/IP fornisce gli strumenti per realizzare un protocollo applicativo (gestisce un byte stream).
- TCP/IP non definisce il protocollo applicativo.
- Una volta definito, il protocollo applicativo gode di vita propria indipendente dalle singole implementazioni.

Esempio di protocollo applicativo

- Trasmettere stringhe di caratteri a lunghezza variabile (esempio molto comune).
- Il mittente ed il destinatario concordano che la fine della stringa è sancita dal carattere '\0'.
- Il funzionamento è del tutto analogo al meccanismo di *cin* e *cout* del C++ (!!).

Un esempio di client/server

- Interprete comandi distribuito.
- Viene utilizzato il seguente protocollo:
  - il client si connette al server e gli invia un comando (di un solo carattere):
    - 'a' è il comando senza parametri;
    - 'b' è il comando seguito da alcuni parametri, in formato stringa;
  - il server esegue il comando specificato ed invia come risposta una stringa terminata da "*\n".

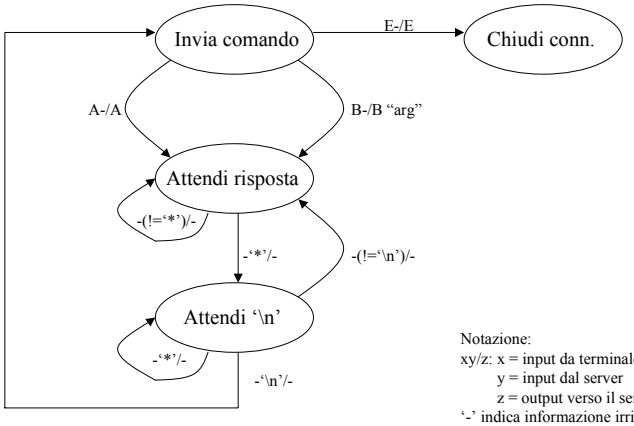
Interprete comandi distribuito

- Il client riconosce, nella risposta inviata dal server, la stringa "*\n". Deve quindi essere in grado di distinguere:
  - "*a\n";
  - "*x\n";
  - "x*a\n";
  - "a**********\n".

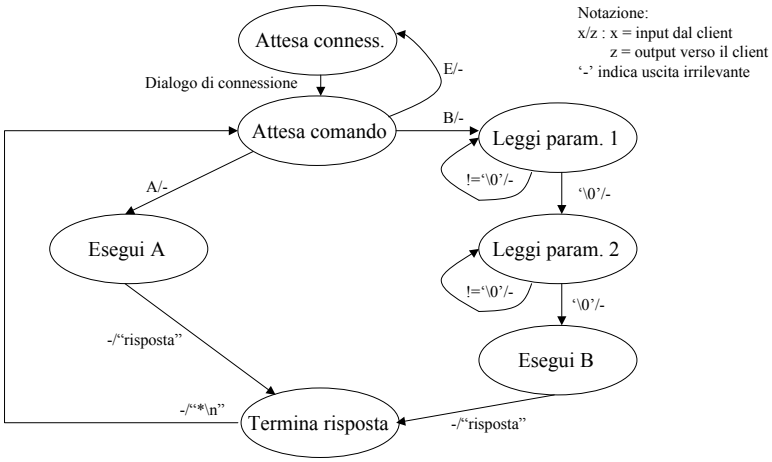
La macchina a stati finiti

- Consente di descrivere il funzionamento di un sistema individuando:
  - uno stato iniziale (a t=0);
  - gli stati nei quali il sistema può trovarsi;
  - le transizioni di stato (da uno stato **corrente** ad uno stato **successivo**);
  - gli ingressi che causano una transizione di stato;
  - uno o piu' stati finali.
- E' anche detta fsm (finite state machine).

Funzionamento del client



Funzionamento del server



Listato di UClient3 (i)

```
                    /* programma UCLIENT3  */
#include <stdio.h>
#include <sys/types.h>
#include <sys/socket.h>
#include <netinet/in.h>
#define PORT 4000
void addr_initialize();

void main(int argc, char* argv[])
{
   int sd;
   struct sockaddr_in server_addr;
   int error,fine;
   char command, combuf[2],result;
   char param1[]="alfa";
   char param2[]="beta";

   addr_initialize(&server_addr, PORT, inet_addr(argv[1]));
   sd=socket(AF_INET,SOCK_STREAM,0);
   error=connect(sd,(struct sockaddr*) &server_addr,sizeof(server_addr));
```

Listato di UClient3 (ii)

```
if (error==0)
   {
   printf("Ho eseguito la connessione\n");
   while(1)
      {
      printf("\nInserire il comando seguito da CR:");
      read(0,combuf,2);
      command=combuf[0];
      send(sd,&command,1,0);
      if (command=='e') break;
      if (command=='b')
         {
         send(sd,param1,5,0);
         send(sd,param2,5,0);
         }
      fine=0;
```


Listato di UClient3 (iii)

```
      while(fine!=1)
         {
         recv(sd,&result,1,0);
         while (result=='*')
            {
            recv(sd,&result,1,0);
            if (result=='\n') fine=1;
            else printf("*");}
            printf("%c",result);
            }
         }
      printf("\nChiudo la connessione \n");
      close(sd);
      }
   else printf("%s","\nErrore di connect\n\n");
}
```


Listato di UServer3 (i)

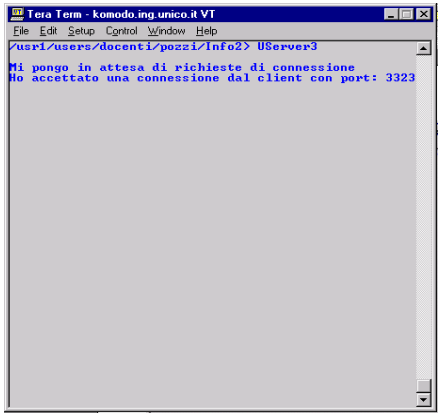
```
                    /* programma USERVER3  */
#include <stdio.h>
#include <sys/types.h>
#include <sys/socket.h>
#include <netinet/in.h>
#define PORT 4000
#define MAXCONN 5
void addr_initialize();

void main(int argc, char * argv[])
{
   int  sd,new_sd,bind_result,listen_result;
   struct sockaddr_in server_addr;
   struct sockaddr_in client_addr;
   int client_len=sizeof(client_addr);
   char inbuf;
   char rispostaA[]="**il Server ha eseguito il comando A**";
   char rispostaB[]="**il Server ha eseguito il comando B**";
   char errmessage[]="**comando errato**";
   char terminatore[2]={'*','\n'};
```

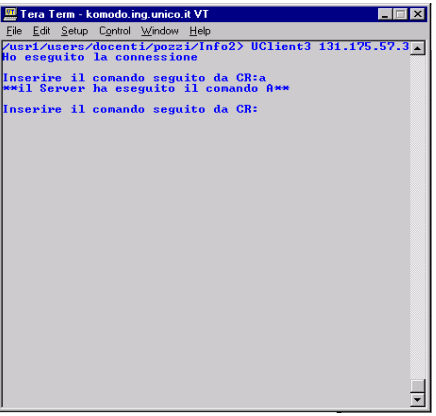

Listato di UServer3 (ii)

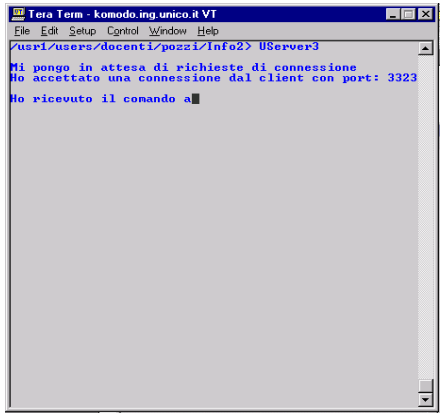
```
addr_initialize(&server_addr, PORT, INADDR_ANY);
sd=socket(AF_INET,SOCK_STREAM,0);
bind_result=bind(sd,(struct sockaddr*) &server_addr,sizeof(server_addr));
listen_result=listen(sd,MAXCONN);
while (1)
   {
   printf("\nMi pongo in attesa di richieste di connessione\n");
   new_sd=accept(sd,(struct sockaddr*) &client_addr, &client_len);
   printf("Ho accettato una connessione\n");
   do
      {
      recv(new_sd,&inbuf,1,0);
      printf("\nHo ricevuto il comando %c",inbuf);
      switch (inbuf)
         {
         case 'e':break;
         case 'a':{send(new_sd,rispostaA,sizeof(rispostaA),0);
                  send(new_sd,terminatore,2,0);break;}
```

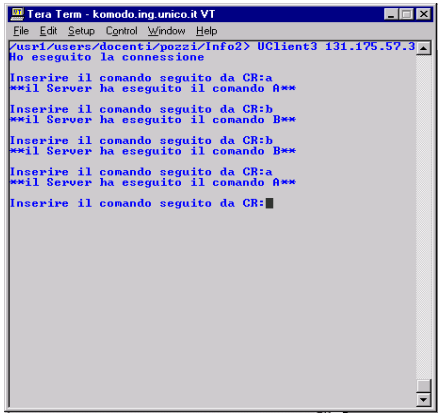
Listato di UServer3 (iii)

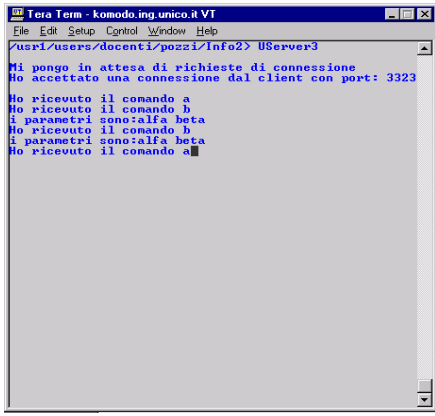
```
            case 'b':{printf("\ni parametri sono:");
                     do {recv(new_sd,&inbuf,1,0);
                        printf("%c",inbuf);
                        } while (inbuf!='\0');
                     printf(" ");
                     do {recv(new_sd,&inbuf,1,0);
                        printf("%c",inbuf);
                        } while (inbuf!='\0');
                     send(new_sd,rispostaB,sizeof(rispostaB),0);
                     send(new_sd,terminatore,2,0);break;}
            default:{send(new_sd,errmessage,sizeof(errmessage),0);
                    send(new_sd,terminatore,2,0);break;}
            }
          } while (inbuf!='e');
        close(new_sd);
        printf("\nChiudo la connessione\n");
    }
    close(sd);
}
```

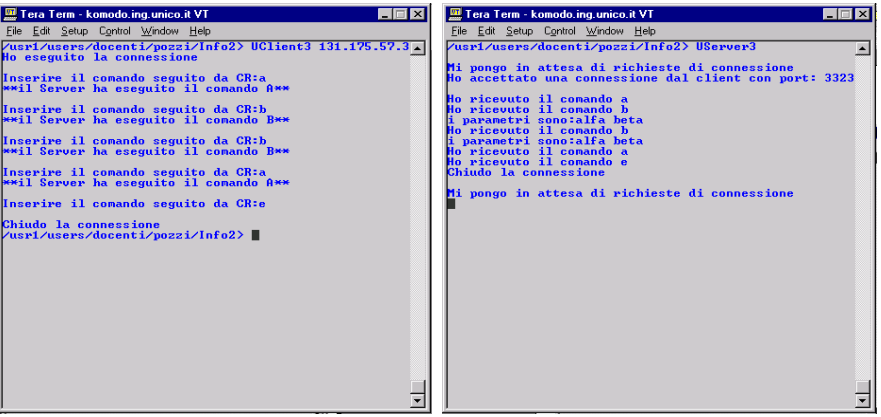












Tera Term - komodo.ing.unico.it VT
File Edit Setup Control Window Help
/usr1/users/docenti/pozzi/Info2> UClient3 131.175.57.3
Ho eseguito la connessione
Inserire il comando seguito da CR:a
il Server ha eseguito il comando A
Inserire il comando seguito da CR:b
il Server ha eseguito il comando B
Inserire il comando seguito da CR:b
il Server ha eseguito il comando B
Inserire il comando seguito da CR:a
il Server ha eseguito il comando A
Inserire il comando seguito da CR:e
Chiudo la connessione
/usr1/users/docenti/pozzi/Info2>

Tera Term - komodo.ing.unico.it VT
File Edit Setup Control Window Help
/usr1/users/docenti/pozzi/Info2> UServer3
Mi pongo in attesa di richieste di connessione
Ho accettato una connessione dal client con port: 3323
Ho ricevuto il comando a
Ho ricevuto il comando b
i parametri sono:alfa beta
Ho ricevuto il comando b
i parametri sono:alfa beta
Ho ricevuto il comando a
Ho ricevuto il comando e
Chiudo la connessione
Mi pongo in attesa di richieste di connessione