Socket TCP e UDP

Formato dei dati

- La comunicazione deve tener conto della rappresentazione dei dati
  - Rappresentazione Big Endian, Little Endian
  - Socket utilizzano network byte order (Big Endian)

- Per i byte nel messaggio scambiato?
  - Si trasmettono sequenze di caratteri
  - Un dato numerico o strutturato lo si converte in una stringa che verrà decodificata dall'host (framing and parsing)
  - Si può utilizzare una rappresentazione standard (e.g. XML)

Lettura dei dati

- Può accadere che la read() restituisca meno byte di quanti richiesti, anche se lo stream è ancora aperto.
- Accade se il buffer a disposizione del socket nel kernel è stato esaurito.
- Sarà necessario ripetere la read (richiedendo il numero dei byte mancanti) fino ad ottenere tutti i dati.

Scrittura dati

- Attenzione alla scrittura su un canale chiuso
- Nel momento in cui viene invocata la write su canale chiuso si genea un segnale SIGPIPE
- Se non gestito termina il processo:
  - signal(SIGPIPE, SIG_IGN)
  - Signal(SIGPIPE, handler)

- Si può usare la send specificando di non generare SIGPIPE, ma -1 con EPIPE
  - flag MSG_NOSIGNAL

Buffer output

- Le socket TCP hanno un buffer per l'output (send buffer) in cui vengono collocati temporaneamente i dati da trasmettere
- La dimensione di questo buffer può essere configurata mediante l'opzione SO_SNDBUF
- Se si chiama write() con n bite sul socket TCP, il kernel cerca di copiare n byte sul buffer del socket. Se il buffer del socket e' più piccolo di n byte o parzialmente occupato da dati non ancora trasmessi verranno copiati un numero minore di byte
- Se il socket è bloccante alla fine della write saranno inviati i byte indicati dal valore di ritorno della write

Scrittura completa

```
ssize_t  writen (int fd, const char *buf, size_t n)
{
  size_t nleft;   ssize_t nwritten;   char *ptr;

  ptr = buf;  nleft = n;
  while (nleft > 0)
  {
     if ( (nwritten = send(fd, ptr, nleft,  MSG_NOSIGNAL  )) < 0) {
          if (errno == EINTR)   nwritten = 0;   /* and call write() again*/
          else                          return(-1);       /* error */
     }
     nleft -= nwritten;         ptr   += nwritten;
  }
  return(n);
}
```

Scrittura completa

```
ssize_t  write_nowait (int fd, const char *buf, size_t n)
{
  int  nwritten;

 do {
    nwritten=send ( fd, buf, n, MSG_DONTWAIT|MSG_NOSIGNAL);
  }while( (nwritten<0) && (errno==EINTR) );
  return(nwritten);
}
```

Lettura completa

```
ssize_t  readn (int fd, char *buf, size_t n)
{
        size_t      nleft;    ssize_t  nread;

        nleft = n;
        while (nleft > 0) {
                if ( (nread = read(fd, buf+n-nleft, nleft)) < 0) {
                        if (errno != EINTR)
                                return(-1);   // restituisco errore
                }
                else if (nread == 0) {
                        // EOF, connessione chiusa, termino
                        // esce e restituisco il numero di byte letti
                        break;
                }
                else  // continuo a leggere
                        nleft -= nread;
        }
        return(n - nleft);      // return >= 0
}
```

Comunicazione senza connessione
socket UDP

UDP (User Datagram Protocol) è un protocollo di trasporto semplice. Se TCP è orientato alla connessione, UDP non stabilisce connessioni tra client e server e non offre garanzie di affidabilità.

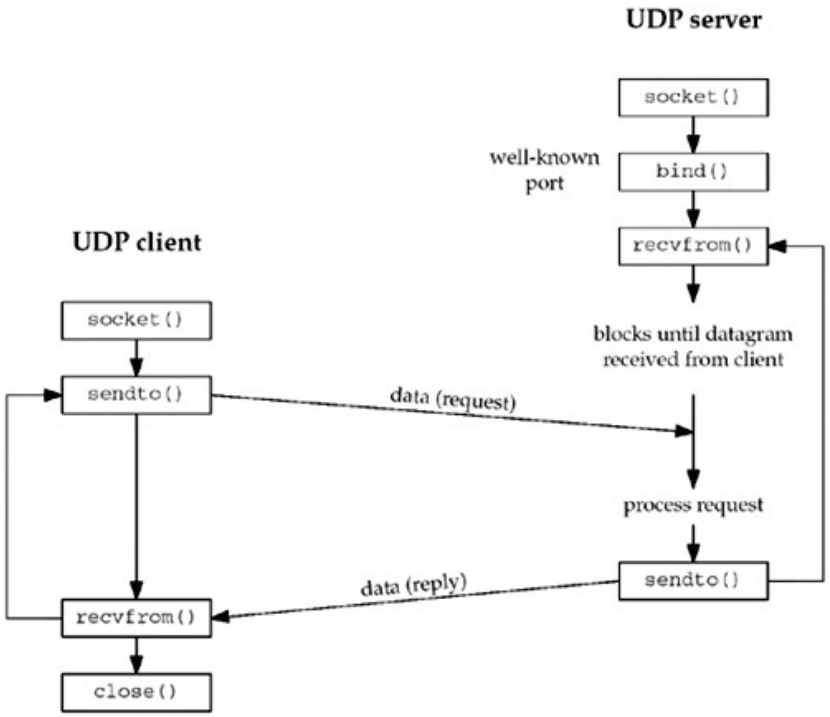
Se TCP trasmette flussi di byte, il protocollo UDP invia pacchetti di taglia fissa, detti datagram.

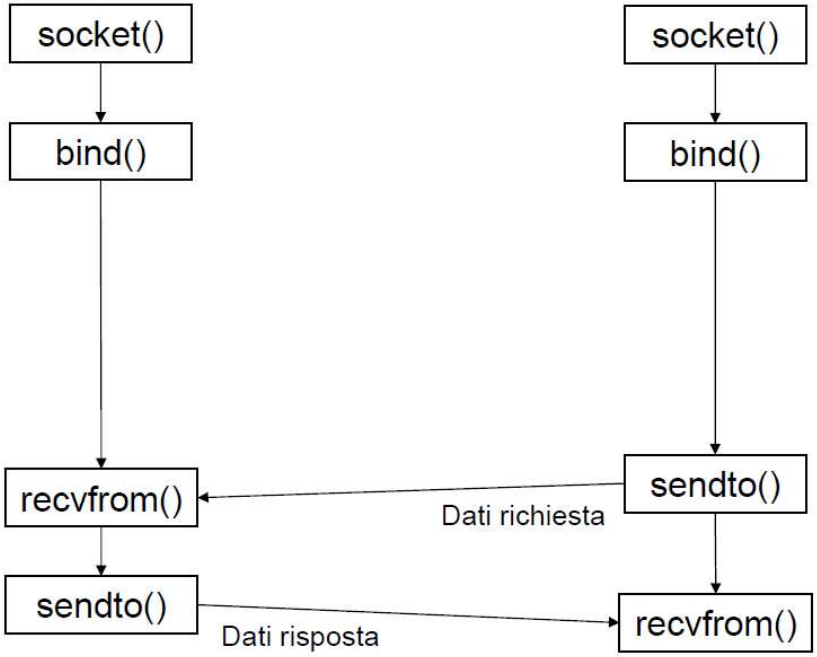
L'assenza di connessione (protocolli di handshake in 3 passi e 4 passi di chiusura) rende UDP più veloce di TCP per trasferimenti di pacchetti di byte.

Gli errori di trasmissione sono sporadici quindi se i dati non sono critici la comunicazione può essere molto rapida ed efficace

Comunicazione senza connessione socket UDP

```
(sockfd = socket(AF_INET, SOCK_DGRAM, 0)
```





Funzione sendto()

```
int sendto( int s, const void *msg, int len,
            unsigned int flags,
            const struct sockaddr *to, int tolen);
```

- Trasmissione non affidabile:
  - Non errore se pacchetto non raggiunge l'host remoto
  - Solo errori locali (e.g. dimensione pacchetto troppo grande EMSGSIZE)

- Parametri:
  - s: descrittore socket
  - msg: puntatore al buffer del messaggio
  - len: dimensione del pacchetto
  - to: indirizzo del destinatario
  - tolen: dimensione della struttura indirizzo

Funzione recvfrom()

```
int recvfrom(int s, void *buf, int len,
             unsigned int flags,
             struct sockaddr *from,
             int *fromlen);
```

- Restituisce byte ricevuto, -1 se errore:
- Parametri:
  - s: descrittore socket
  - msg: puntatore al buffer del messaggio
  - len: dimensione del buffer (se dimensione minore del pacchetto si leggono i primi len byte)
  - from: indirizzo dell'end point mittente
  - fromlen: puntatore alla dimensione (byte) della struttura sockaddr
  - Con flag MSG_PEEK non toglie pacchetto dalla coda e verifica solo indirizzo client
  - Se non arrivano messaggi rimane bloccato, si può settare timeout

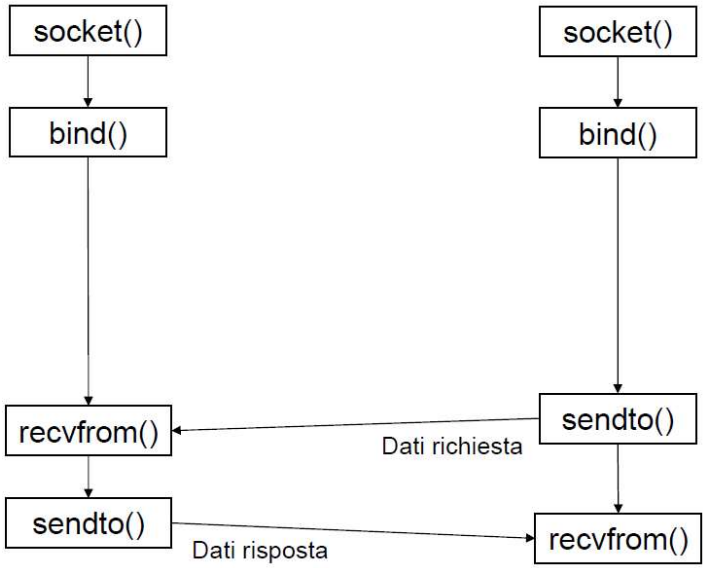
Uso di connect()

- connect() si può usare anche per comunicazione senza connessione
  - Per esempio per impostare la connessione una volta per tutte e gestire presenza di errori sulla connessione
- Se è stabilita la connect() si può usare write, send o sendto lasciando NULL gli indirizzi:

```
int sendto(sd, buf, len, flags, NULL,0)
```

Comunicazione senza connessione

- Con flusso di dati senza connessione (SOCK_DGRAM) server e client faranno bind() su indirizzo locale per poi comunicare con sendto/recvfrom



Client

```
#include <sys/types.h>
#include <sys/socket.h>
#include <arpa/inet.h>

#include <unistd.h>
#include <stdio.h>
#include <stdlib.h>
#include <string.h>

#define SERV_PORT       5193
#define MAXLINE         1024

int main(int argc, char *argv[ ])
{
  int       sockfd, n;
  char      recvline[MAXLINE + 1];
  struct    sockaddr_in   servaddr;
```

Client

```
if (argc != 2) { /* controlla numero degli argomenti */
  fprintf(stderr, "utilizzo: daytime_clientUDP <indirizzo IP server>\n");
  exit(1);
}

if ((sockfd = socket(AF_INET, SOCK_DGRAM, 0)) < 0) { /* crea il socket */
  perror("errore in socket");
  exit(-1);
}

memset((void *)&servaddr, 0, sizeof(servaddr));          /* azzera servaddr */
servaddr.sin_family = AF_INET;       /* assegna il tipo di indirizzo */
servaddr.sin_port = htons(SERV_PORT);  /* assegna la porta del server */
/* assegna l'indirizzo del server prendendolo dalla riga di comando. L'indirizzo è
   una stringa da convertire in intero secondo network byte order. */
if (inet_pton(AF_INET, argv[1], &servaddr.sin_addr) <= 0) {
            /* inet_pton (p=presentation) vale anche per indirizzi IPv6 */
  perror("errore in inet_pton");
  exit(-1);
}
```

Client

```
  /* Invia al server il pacchetto di richiesta*/
  if (sendto(sockfd, NULL, 0, 0, (struct sockaddr *) &servaddr, sizeof(servaddr)) < 0) {
    perror("errore in sendto");
    exit(-1);
  }
  /* legge dal socket il pacchetto di risposta */
  n = recvfrom(sockfd, recvline, MAXLINE, 0 , NULL, NULL);
  if (n < 0) {
    perror("errore in recvfrom");
    exit(-1);
  }
  if (n > 0) {
    recvline[n] = 0;        /* aggiunge il carattere di terminazione */
    if (fputs(recvline, stdout) == EOF)   {  /* stampa recvline sullo stdout */
      perror("errore in fputs");
      exit(-1);
    }
  }
  exit(0);
}
```

Server

```
if ((sockfd = socket(AF_INET, SOCK_DGRAM, 0)) < 0) { /* crea il socket */
  perror("errore in socket");
  exit(-1);
}

memset((void *)&addr, 0, sizeof(addr));
addr.sin_family = AF_INET;
addr.sin_addr.s_addr = htonl(INADDR_ANY); /* il server accetta pacchetti su
   una qualunque delle sue interfacce di rete */
addr.sin_port = htons(SERV_PORT); /* numero di porta del server */

/* assegna l'indirizzo al socket */
if (bind(sockfd, (struct sockaddr *) &addr, sizeof(addr)) < 0) {
  perror("errore in bind");
  exit(-1);
}
```

Funzioni bloccanti

- La funzione accept() e le funzioni per gestire I/O sono bloccanti (read, recv)
- Il server ricorsivo tradizionale
  - attende su accept() una connessione
  - Quando accetta una connessione il processo/thread principale crea un processo/thread dedicato per il controllo e si mette in attesa di altro
- Alternative:
  - Usare opzioni per le socket per impostare un timeout
  - Usare socket non bloccante con funzione fcntl (funzione di gestione fd)
    fcntl(sd, F_SETFL, O_NONBLOCK);

sd è il socket precedentemente aperto; F_SETFL significa che si vuole impostare il *file status flag* al valore dall'ultimo parametro; O_NONBLOCK costante corrispondente al valore che significa non bloccante.

Funzioni bloccanti

- La funzione accept() e le funzioni per gestire I/O sono bloccanti (read, recv)
- Il server ricorsivo tradizionale
  - attende su accept() una connessione
  - Quando accetta una connessione il processo/thread principale crea un processo/thread dedicato per il controllo e si mette in attesa di altro
- Alternative:
  - Usare opzioni per le socket per impostare un timeout
  - Usare socket non bloccante con funzione fcntl (funzione di gestione fd)
    fcntl(sd, F_SETFL, O_NONBLOCK);
  - Se la socket è non-blocking e non ci sono dati da leggere restituisce -1 ed errno è settato a EAGAIN o EWOULDBLOCK.

Opzioni SO_SNDTIMEO, SO_RCVTIMEO

int **getsockopt**(int **fd**, **level**, **option**, void ***val**, socklen_t **len**);
int **setsockopt**(int **fd**, **level**, **option**, void ***val**, socklen_t ***len**);

- val deve puntare ad una struttura timeval

```
struct timeval {
    time_t tv_sec;  /* seconds      */
    long   tv_usec; /* microseconds */
};
```

- len è pari a sizeof(timeval)
- i timeout si annullano impostando un nuovo timeout di zero secondi e zero microsecondi

Select

int select (int numfds, fd_set *readfds, fd_set *writefds, fd_set *exceptfds, struct timeval *timeout);

Header file sys/time.h, sys/types.h, unistd.h

- Controllare contemporaneamente lo stato di uno o più descrittori degli insiemi specificati
- Si blocca finché:
  - non avviene un'attività (lettura/scrittura) su un descrittore degli insiemi
  - non viene generata un'eccezione
  - non scade un timeout
- Restituisce
  - -1 in caso di errore
  - 0 se il timeout è scaduto
  - Altrimenti, il numero totale di descrittori

Parametri Select

Insiemi di descrittori da controllare

– **readfds**: pronti per operazioni di lettura

  Es: socket pronto per la lettura se c'è una connessione in attesa di accept()

– **writefds**: pronti per operazioni di scrittura

– **exceptfds**: verificare eccezioni

readfds, writefds e exceptfds sono puntatori a variabili di tipo fd_set

– fd_set insieme dei descrittori (è una bit mask implementata con un array di interi)

numfds è il numero massimo di descrittori controllati da select()

– Se maxd è il massimo descrittore usato, numfds = maxd + 1

– Può essere posto uguale alla costante FD_SETSIZE

Timeout

- timeout specifica il valore massimo che la funzione select() attende per individuare un descrittore pronto

```
struct timeval {
    int tv_sec;     // seconds
    int tv_usec;    // microseconds
};
```

- Se impostato a NULL (timeout == NULL)
  - select si blocca indefinitamente fino a quando è pronto un descrittore
- Se impostato a zero (timeout->tv_sec == 0 && timeout->tv_usec == 0 })
  - select non attende (da usare per polling dei descrittori senza bloccare)
- Se diverso da zero (timeout->tv_sec != 0 || timeout->tv_usec != 0 }),
  - select attende il tempo specificato
  - select() ritorna, se è pronto, uno (o più) dei descrittori specificati (numero positivo) oppure 0 se è scaduto il timeout

Impostazione degli insiemi

- Macro utilizzate per gestire gli insiemi di descrittori

Function	Description
`FD_SET(int fd, fd_set *set);`	Add `fd` to the `set`.
`FD_CLR(int fd, fd_set *set);`	Remove `fd` from the `set`.
`FD_ISSET(int fd, fd_set *set);`	Return true if `fd` is in the `set`.
`FD_ZERO(fd_set *set);`	Clear all entries from the `set`.

int FD_ISSET(int fd, fd_set *set);

– Al ritorno di select(), controlla se *fd* appartiene all'insieme di descrittori *set*, verificando se il bit relativo a *fd* è pari a 1 (restituisce 0 in caso negativo, un valore diverso da 0 in caso affermativo)

Select

La funzione select() rileva i descrittori pronti

– Significato diverso per i tre tipi di descrittori (lettura, scrittura, eccezione)

- Un descrittore è pronto in lettura se:
  - Nel buffer di ricezione del socket sono arrivati byte in quantità sufficiente (soglia minima per default pari a 1, modificabile con opzione del socket SO_RCVLOWAT)
  - Per il lato in lettura è stata chiusa la connessione
  - Si è verificato un errore sul socket
  - Se un socket di ascolto e ci sono delle connessione completate
- Un descrittore è pronto in scrittura se:
  - Nel buffer di invio del socket c'è spazio sufficiente per inviare (soglia minima per default pari a 2048, modificabile con opzione del socket SO_SNDLOWAT) ed il socket è già connesso (TCP) oppure non necessita di connessione (UDP)
  - Per il lato in scrittura è stata chiusa la connessione (SIGPIPE generato in scrittura)
  - Si è verificato un errore sul socket

Esempio

```c
#include <stdio.h>
#include <sys/time.h>
#include <sys/types.h>
#include <unistd.h>

#define STDIN 0  // file descriptor for standard input

int main(void)
{
    struct timeval tv;
    fd_set readfds;

    tv.tv_sec = 2;
    tv.tv_usec = 500000;

    FD_ZERO(&readfds);
    FD_SET(STDIN, &readfds);

    // don't care about writefds and exceptfds:
    select(STDIN+1, &readfds, NULL, NULL, &tv);

    if (FD_ISSET(STDIN, &readfds))
        printf("A key was pressed!\n");
    else
        printf("Timed out.\n");

    return 0;
}
```