Progettazione di un client TCP

Passi per la progettazione di un client TCP

1. Creazione di un endpoint
   - Richiesta al sistema operativo
2. Creazione della connessione
   - Implementazione del 3-way handshake per l'apertura della connessione TCP
3. Lettura e scrittura sulla connessione
   - Analogo a operazione su file in Unix
4. Chiusura della connessione
   - Implementazione del 4-way handshake per la chiusura della connessione TCP

Progettazione di un server TCP

Passi per la progettazione di un server TCP

1. Creazione di un endpoint
   - Richiesta al sistema operativo
2. Collegamento dell'endpoint ad una porta
3. Ascolto sulla porta
   - Processo sospeso in attesa
4. Accettazione della richiesta di un client
5. Letture e scritture sulla connessione
6. Chiusura della connessione

Esempio: daytime TCP

- Il client interroga il server per ottenere la data e l'ora
- Il server ottiene l'informazione dal S.O. e la invia al client
- Il client stampa l'informazione su stdout

- Assunzioni
  - Il server invia la risposta in un'unica stringa alfanumerica
  - Il client legge la stringa, la visualizza sullo schermo e termina
  - Client e server utilizzano una connessione TCP

- Per eseguire il client
  daytime_clientTCP <indirizzo IP server>
- Per eseguire il server
  daytime_serverTCP

Client TCP daytime

```
#include <sys/types.h>
#include <sys/socket.h>
#include <netinet/in.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 TCP daytime (2)

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

if ((sockfd = socket(AF_INET, SOCK_STREAM, 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) anche per indirizzi IPv6 */
  perror("errore in inet_pton");
  exit(-1);
}
```


Client TCP daytime (3)

```
/* stabilisce la connessione con il server */
if (connect(sockfd, (struct sockaddr *) &servaddr, sizeof(servaddr)) < 0) {
  perror("errore in connect");
  exit(-1);
}
/* legge dal socket fino a quando non trova EOF */
while ((n = read(sockfd, recvline, MAXLINE)) > 0) {
  recvline[n] = 0;       /* aggiunge il carattere di terminazione */
  if (fputs(recvline, stdout) == EOF)   {  /* stampa recvline sullo stdout */
    perror("errore in fputs");
    exit(-1);
  }
}
if (n < 0) {
  perror("errore in read");
  exit(-1);
}
exit(0);
}
```

Server TCP daytime

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

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

#define SERV_PORT       5193
#define BACKLOG         10
#define MAXLINE         1024

int main(int argc, char *argv[ ])
{
  int       listensd, connsd;
```


Server TCP daytime (2)

```
  struct sockaddr_in        servaddr;
  char                      buff[MAXLINE];
  time_t                    ticks;

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

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

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

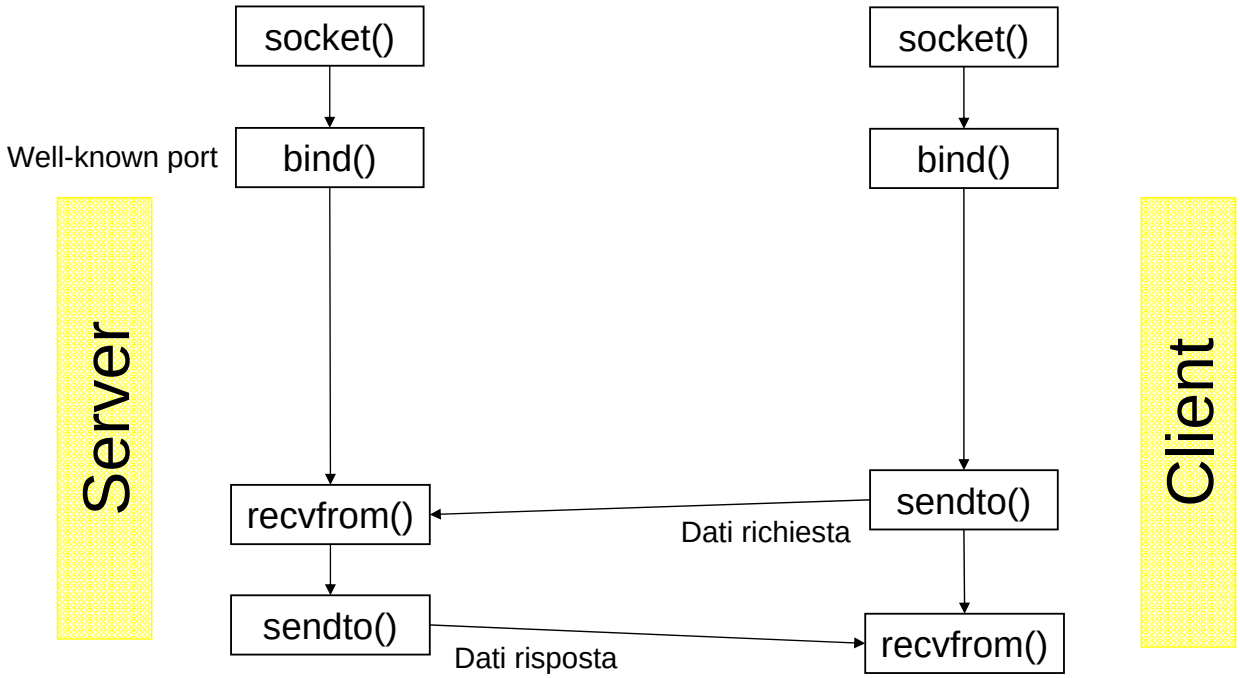
Server TCP daytime (3)

```
  if (listen(listensd, BACKLOG) < 0 ) {
    perror("errore in listen");
    exit(-1);
  }

  while (1) {
    if ( (connsd = accept(listensd, (struct sockaddr *) NULL, NULL)) < 0) {
      perror("errore in accept");
      exit(-1);
    }

    /* accetta una connessione con un client */
    ticks = time(NULL); /* legge l'orario usando la chiamata di sistema time */
    /* scrive in buff l'orario nel formato ottenuto da ctime
            snprintf impedisce l'overflow del buffer */
    snprintf(buff, sizeof(buff), "%.24s\r\n", ctime(&ticks));
     /* ctime trasforma la data e l'ora da binario in ASCII
            \r\n per carriage return e line feed */
```


Server TCP daytime (4)

```
  /* scrive sul socket di connessione il contenuto di buff */
    if (write(connsd, buff, strlen(buff)) != strlen(buff)) {
      perror("errore in write");
      exit(-1);
    }

    if (close(connsd) == -1) {  /* chiude la connessione */
      perror("errore in close");
      exit(-1);
    }
  }
  exit(0);
}
```

Comunicazione senza connessione



Comunicazione senza connessione (2)

- Per creare un socket
  socket(AF_INET, SOCK_DGRAM, IPPROTO_UDP);
  - SOCK_DGRAM è il tipo di protocollo per datagram
  - IPPROTO_UDP è lo specifico protocollo richiesto (anche 0)

- Il server invoca bind() per associare al socket l'indirizzo locale e la porta locale
  - Il numero della porta è prestabilito per l'applicazione
- Il client invoca bind() per associare al socket l'indirizzo locale e la porta locale
  - Il numero della porta è in qualche modo arbitrario
  - Il numero di porta 0 lascia al SO la scelta della porta
- Il server ed il client hanno così impostato la semi-associazione locale del socket; l'impostazione delle altre due componenti dell'associazione può essere fatta per ogni pacchetto

Funzione sendto()

```
int sendto(int sockfd, const void *buf, size_t len, int flags,
const struct sockaddr_in *to, socklen_t tolen);
```

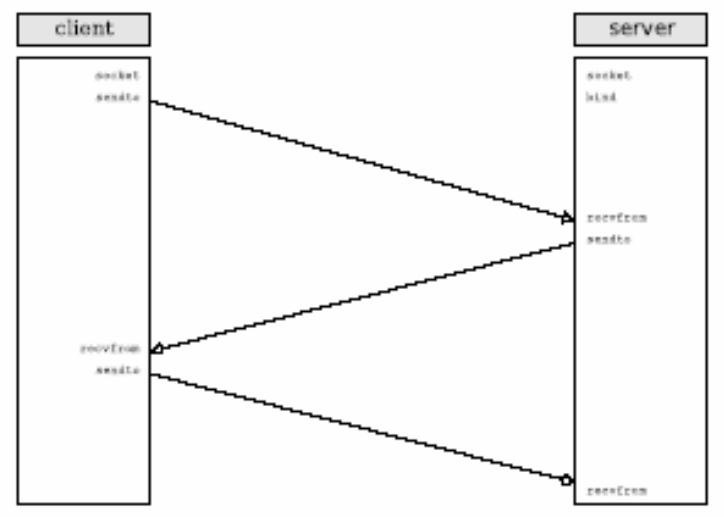
- Restituisce il numero di byte inviati (oppure -1 in caso di errore)
- Trasmissione non affidabile: sendto() non restituisce errore se il pacchetto non raggiunge l'host remoto
  - soltanto errori locali: ad es., dimensione del pacchetto maggiore della dimensione massima del datagram usato (errore EMSGSIZE)
- Parametri della funzione
  - sockfd: descrittore del socket a cui si fa riferimento
  - buf: puntatore al buffer contenente il pacchetto da inviare
  - len: dimensione (in byte) del pacchetto da inviare
  - flags: intero usato come maschera binaria per impostare una serie di modalità di funzionamento della comunicazione; per ora 0
  - to: l'indirizzo IP e la porta della macchina remota che riceverà il pacchetto inviato
  - tolen: dimensione (in byte) della struttura sockaddr

Funzione recvfrom()

```
int recvfrom(int sockfd, const void *buf, size_t len, int flags,
struct sockaddr *from, socklen_t *fromlen);
```

- Restituisce il numero di byte ricevuti (-1 in caso di errore)
- Parametri della funzione
  - sockfd: descrittore del socket a cui si fa riferimento
  - buf: puntatore al buffer contenente il pacchetto da ricevere
  - len: dimensione del buffer (numero massimo di byte da leggere)
    - Se il pacchetto ricevuto ha dimensione maggiore di len, si ottengono i primi len byte ed il resto del pacchetto è perso
  - from: puntatore alla struttura contenente l'indirizzo IP e la porta della macchina remota che ha spedito il pacchetto; usato per ottenere l'indirizzo del mittente del pacchetto
  - fromlen: puntatore alla dimensione (in byte) della struttura sockaddr
- Riceve i pacchetti da qualunque macchina remota: non è possibile rifiutare un pacchetto
  - Per verificare l'indirizzo del client, si usa il flag MSG_PEEK che non toglie il pacchetto dalla coda del SO

Scambio di pacchetti in una comunicazione UDP



Uso di connect() con i socket UDP

- connect() può essere usata anche in applicazioni client di tipo senza connessione per impostare una volta per tutte le due componenti remote dell'associazione del client e gestire la presenza di errori asincroni
  - ad es. indirizzo inesistente o su cui non è in ascolto un server
- Usando connect() il client può lasciare nulli gli ultimi campi in sendto() oppure usare la write() o la send()

```
int sendto(sockfd, buf, len, flags, NULL, 0);
int write(sockfd, buf, len);
```

- connect() impedisce la ricezione sul socket di pacchetti non provenienti dalla macchina remota registrata

```
int recvfrom(sockfd, buf, len, flags, NULL, NULL);
int read(sockfd, buf, len);
```

- Non ha senso usare connect() sul server

Associazione senza connessione

- In un flusso di dati senza connessione (tipo di protocollo SOCK_DGRAM), l'impostazione delle varie componenti dell'associazione è effettuata dalle seguenti chiamate di sistema:

End-point	Protocollo	Indirizzo locale Processo locale	Indirizzo remoto Processo remoto
Server	socket()	bind()	recvfrom()
Client	socket()	bind()	sendto()

End-point	Protocollo	Indirizzo locale Processo locale	Indirizzo remoto Processo remoto
Server	socket()	bind()	recvfrom()
Client	socket()	bind()	connect()

Esempio: daytime UDP

- Il client interroga il server per ottenere la data e l'ora
- Il server ottiene l'informazione dal SO e la invia al client
- Il client stampa l'informazione su stdout

- Assunzioni
  - Il server invia la risposta in un'unica stringa alfanumerica
  - Il client legge la stringa, la visualizza sullo schermo e termina
  - Client e server utilizzano una comunicazione UDP

- Per eseguire il client
  daytime_clientUDP <indirizzo IP server>
- Per eseguire il server
  daytime_serverUDP

Client UDP daytime

```
#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 UDP daytime (2)

```
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 UDP daytime (3)

```
  /* 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 UDP daytime

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

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


#define SERV_PORT        5193
#define MAXLINE          1024

int main(int argc, char *argv[ ])
{
  int       sockfd, len;
  struct    sockaddr_in       addr;
  char      buff[MAXLINE];
  time_t    ticks;
```

Server UDP daytime (2)

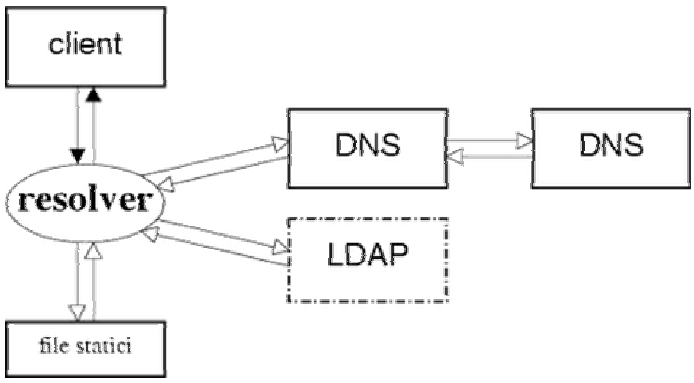
```
if ((sockfd = socket(AF_INET, SOCK_DGRAM, 0)) < 0) { /* crea il socket */
  perror("errore in socket");
  exit(-1);
}

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);
}
```


Server UDP daytime (3)

```
while (1) {
  if ( (recvfrom(sockfd, buff, MAXLINE, 0, (struct sockaddr *)&addr, &len)) < 0) {
    perror("errore in recvfrom");
    exit(-1);
  }

  ticks = time(NULL); /* legge l'orario usando la chiamata di sistema time */
  /* scrive in buff l'orario nel formato ottenuto da ctime
          snprintf impedisce l'overflow del buffer */
  snprintf(buff, sizeof(buff), "%.24s\r\n", ctime(&ticks));
  /* ctime trasforma la data e l'ora da binario in ASCII
          \r\n per carriage return e line feed*/
  /* scrive sul socket il contenuto di buff */
  if (sendto(sockfd, buff, strlen(buff), 0, (struct sockaddr *)&addr, sizeof(addr)) < 0)
   {
    perror("errore in sendto");
    exit(-1);
  }
}
exit(0);
}
```

Risoluzione dei nomi: il resolver

- Resolver: insieme di routine fornite con le librerie del C per gestire il servizio di risoluzione di nomi associati a identificativi o servizi relativi alla rete
- Nomi di: domini, servizi, protocolli, rete
  - Principali file di configurazione in Linux:
    /etc/hosts: associazioni statiche
    /etc/resolv.conf: indirizzi IP dei server DNS da contattare
    /etc/host.conf: ordine in cui eseguire la risoluzione



Risoluzione dei nomi

- Per determinare l'indirizzo IP corrispondente al nome del proprio host
  `int gethostname(char *name, size_t size);`
- Per determinare l'indirizzo IP (solo IPv4) corrispondente al nome di dominio di un host tramite DNS
  `struct hostent *gethostbyname(const char *name);`
- Dichiarazione dei prototipi delle funzioni e degli altri simboli in netdb.h
- Struttura hostent

```
struct hostent {
        char *h_name;           /* nome ufficiale dell'host */
        char **h_aliases;       /* lista di nomi alternativi dell'host */
        int h_addrtype;         /* tipo di indirizzo dell'host (AF_INET) */
        int h_length;           /* dimensione in byte dell'indirizzo */
        char **h_addr_list;     /* lista degli indirizzi corrispondenti al
                                   nome dell'host (in network byte order) */
        #define h_addr h_addr_list[0]       /* primo indirizzo dell'host */
};
```

Vedere l'esempio get_ip.c

Risoluzione dei nomi (2)

- In caso di errore, gethostbyname() restituisce NULL e imposta la variabile globale h_errno ad un valore corrispondente all'errore (ad es. HOST_NOT_FOUND)
  - Per conoscere l'errore, occorre valutare il valore di h_errno
  - In alternativa, si può usare la funzione herror()

- Per determinare il nome di un host corrispondente ad un dato indirizzo IP

```
struct hostent *gethostbyaddr(const char *addr, int length, int addrtype);
```

  - length: dimensione in byte di addr
  - addrtype: tipo di indirizzo dell'host (AF_INET)

Esempio di risoluzione dei nomi

```
#include <sys/types.h>
#include <sys/socket.h>
#include <netinet/in.h>
#include <arpa/inet.h>
#include <netdb.h>
#include <stdio.h>
…
  struct hostent *hp;
  struct sockaddr_in *sin;
  if ((hp == gethostbyname(argv[1])) == NULL) {
    herror("errore in gethostbyname");
    exit(1);
  }
  memset((void *)&sin, 0, sizeof(sin));
  sin.sin_family = hp->h_addrtype;
  memcpy(&sin.sin_addr, hp->h_addr, hp->h_length);
            /* oppure sin.sin_addr = *(struct in_addr *) hp->h_addr; */
  printf("Host name %s\n", hp->h_name);
  printf("IP address %s\n", inet_ntoa(sin.sin_addr));
…
```

Funzioni rientranti

- Una funzione è rientrante se può essere interrotta in un punto qualunque della sua esecuzione ed essere chiamata da un altro thread senza che questo comporti nessun problema nell'esecuzione della funzione
  - Problematica tipica della programmazione multi-thread
  - Una funzione che usa soltanto variabili locali è rientrante
  - Una funzione che usa memoria non nello stack (ad es. variabile globale) non è rientrante
  - Una funzione che usa un oggetto allocato dinamicamente può essere rientrante o meno
- Nella *glibc* due macro per il compilatore (_REENTRANT e _THREAD_SAFE) che attivano le versioni rientranti delle funzioni di libreria
  - Versione rientrante identificate dal suffisso _r

Funzioni rientranti (2)

- gethostbyname() non è una funzione rientrante
  - La struttura hostent è allocata in un'area statica di memoria, che può essere sovrascritta da due chiamate successive della funzione
- Per risolvere il problema si può:
  - Allocare una struttura hostent e passarne l'indirizzo usando la versione rientrante gethostbyname_r()
  - Oppure usare la funzione getipnodebyname(), che alloca dinamicamente la struttura hostent
    - Occorre invocare la funzione freehostent() per deallocare la memoria occupata dalla struttura hostent una volta che questa non serve più

Nomi dei protocolli

- I nomi dei protocolli supportati da un host sono memorizzati in un file (ad es., /ect/protocols in Linux)
- Per determinare informazioni corrispondenti al nome di un protocollo

```
struct protoent *getprotobyname(const char *name);
```

- Dichiarazione dei prototipi delle funzioni e degli altri simboli in netdb.h
- Struttura protoent

```
struct protoent {
  char *p_name;     /* nome ufficiale del protocollo */
  char **p_aliases; /* nomi alternativi del protocollo */
  int p_proto;      /* numero del protocollo (in network byte
                       order); da usare per l'argomento protocol in
                       socket() */
};
```


Funzioni per altri servizi di risoluzione

- Esistono anche altre funzioni del tipo getXXXbyname e getXXXbyaddr per interrogare gli altri servizi di risoluzione dei nomi
- XXX = serv: per risolvere i nomi dei servizi noti (ad es. smtp, http, …) definiti in Linux nel file /etc/services
  - getservbyname() risolve il nome di un servizio nel corrispondente numero di porta
  - Usa la struttura servent che contiene i relativi dati
- XXX = net: per risolvere i nomi delle reti