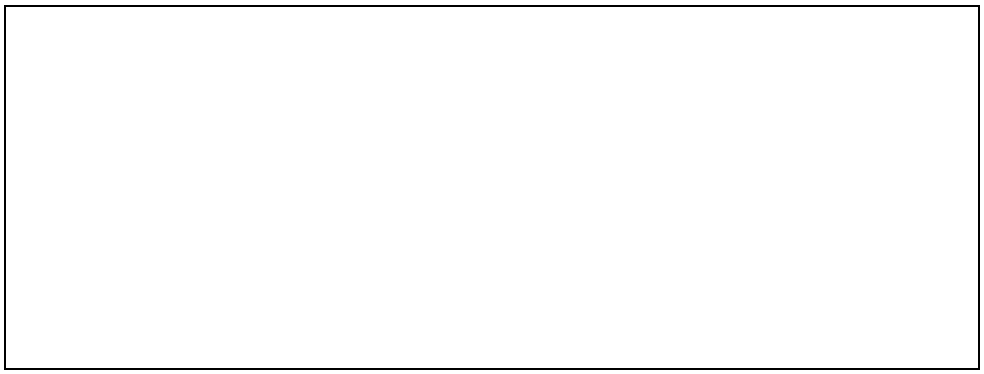
Stefano Russo
Dipartimento di Informatica e Sistemistica
Università di Napoli "Federico II"

Il modello cliente/servente in ambiente UNIX - TCP/IP

Terminologia e concetti di base

Clienti e Serventi

- In una applicazione distribuita di tipo cliente/servente, la distinzione tra i ruoli di cliente e servente viene effettuata sulla base di chi prende l'iniziativa della comunicazione
- Il programma che avvia la comunicazione è il **cliente**
- L'utente finale invoca il cliente quando vuole accedere ad un servizio disponibile su un nodo della rete
- Il programma cliente contatta il programma remoto che fornisce il servizio, gli sottomette una richiesta, ed attende la risposta
- Il **servente** è il programma che attende le comunicazioni da parte dei clienti, corrispondenti alle richieste di fornitura di un servizio
- Il servente opera ciclicamente: riceve una richiesta di un cliente, esegue le necessarie elaborazioni, e ritorna il risultato al cliente



Il modello Cliente/Servente in TCP/IP

- TCP/IP è la famiglia di protocolli con cui vengono tipicamente sviluppate le applicazioni distribuite in ambiente UNIX

- TCP/IP non fornisce di per sé meccanismi per creare automaticamente un programma all'arrivo di un messaggio su un nodo

- Perciò un servente deve essere già in attesa di comunicazioni prima che una richiesta arrivi

- Il modello cliente/servente risolve il problema della sincronizzazione (*rendez-vous*) tra due programmi che devono comunicare *peer-to-peer*

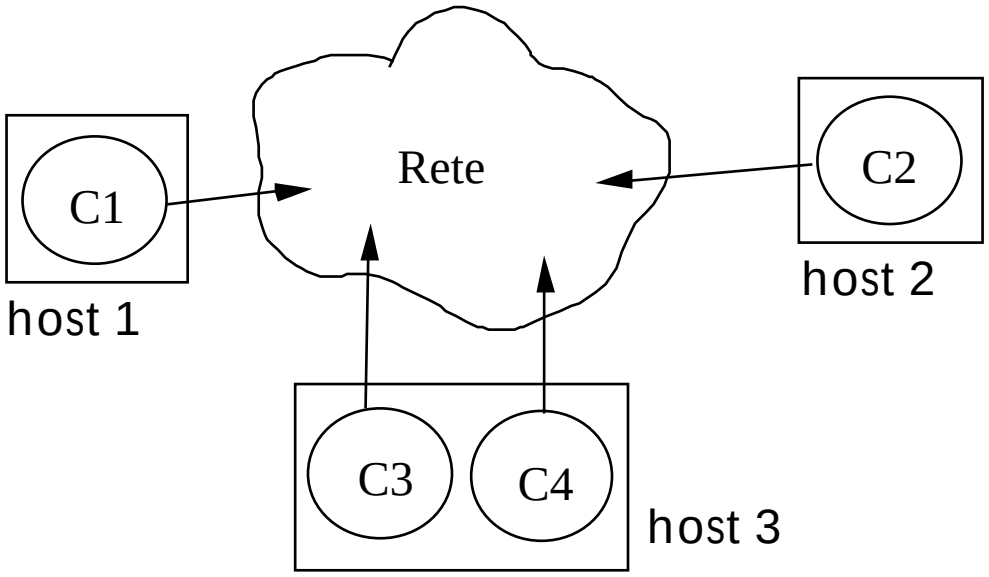
- Il modello prevede che, in un sistema costituito da una coppia di applicazioni comunicanti, un lato (servente) deve avviare l'esecuzione ed aspettare indefinitamente che l'altro lato lo contatti (cliente)

- Per questa ragione i serventi vengono tipicamente mandati in esecuzione all'atto del *bootstrap* del sistema

- Ciascun servente gira per sempre, attendendo e servendo richieste

Concorrenza tra/nei clienti

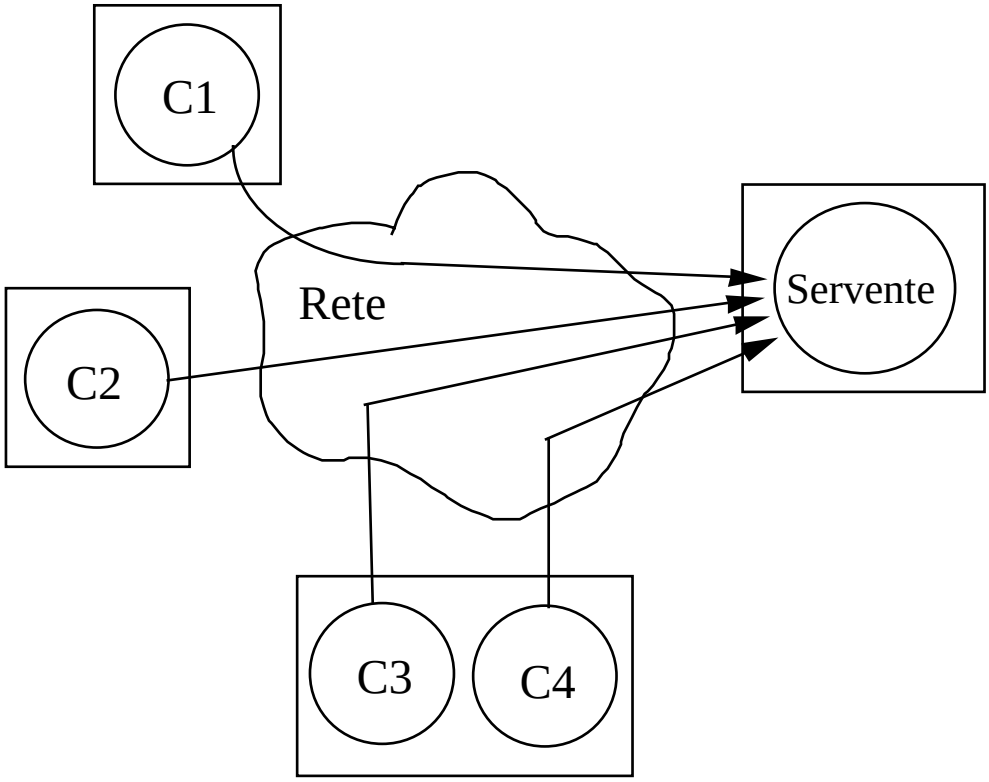
- In un sistema cliente/servente, la concorrenza tra i clienti è implicita, per via della possibile esecuzione simultanea di più clienti
  - su altrettanti nodi della rete, e/o
  - su uno stesso nodo, ad es. con sistema operativo multiutente *time-sharing*



- Talvolta può essere conveniente introdurre la concorrenza in un cliente

- Ciò accade ad es.
  - se un cliente deve operare con più serventi, non necessariamente in un ordine fissato, oppure
  - deve combinare le risposte provenienti da più serventi (es. query distribuita).

Concorrenza nei serventi



- Più clienti, allocati sulla stessa macchina o su macchine diverse, contattano uno stesso servente, usando un porto di comunicazione prefissato
- I serventi devono essere esplicitamente programmati per gestire le richieste concorrenti provenienti dai clienti
- In questo modo è possibile servire un cliente senza dover attendere la fine del servizio di un altro
- Ciò è particolarmente importante per non penalizzare i clienti che stabiliscono connessioni di breve durata

Un esempio: TELNET

- Un esempio di applicazione cliente/servente in ambiente TCP/IP è costituito dal protocollo TELNET per il servizio di terminale remoto

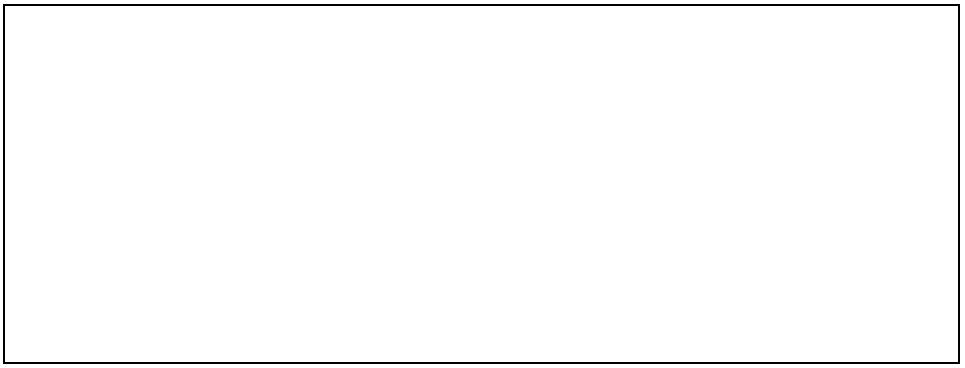
- Il protocollo TELNET specifica le modalità di interazione tra un cliente locale ed un servente su una macchina remota, sulla quale l'applicazione cliente desidera collegarsi come fosse un terminale ad essa locale

- In UNIX esiste il comando telnet per invocare l'applicazione cliente e collegarsi ad un sistema remoto:

  telnet *sistema_remoto*

  Ad es., per collegarsi remotamente da un qualunque nodo della rete Internet al sistema del DIS (Dip. di Informatica e Sistemistica) dell'Univ. di Napoli:

telnet *nadis.dis.unina.it*	
Trying...	*apertura conness.*
Connected to nadis.dis.unina.it.	*conness. stabilita - mes-*
Escape character is '^]'.	*saggi dal cliente*
Ultrix (nadis)	*messaggi dal servente*
login:	*prompt sist. remoto*



Flessibilità delle applicazioni

- TCP/IP offre l'astrazione dei **porti** di comunicazione per identificare i servizi disponibili su un elaboratore sulla rete
- Ogni servizio è associato ad un porto ed ogni porto ha un numero prefissato
- Es.: il servizio di terminale remoto usa il porto numero 23
- Quando un utente invoca il programma cliente telnet, questo si collega al porto 23 sulla macchina specificata
- In realtà, il protocollo TELNET non definisce i dettagli del servizio, ma solo le modalità di accesso ad esso, cioè il meccanismo di comunicazione base (formato dei dati scambiati, codifica dei caratteri, caratteri di controllo, ecc.)
- E' per questo che il comando telnet ammette come secondo argomento (opzionale) sulla riga di comando il numero di porto al quale ci si vuole collegare

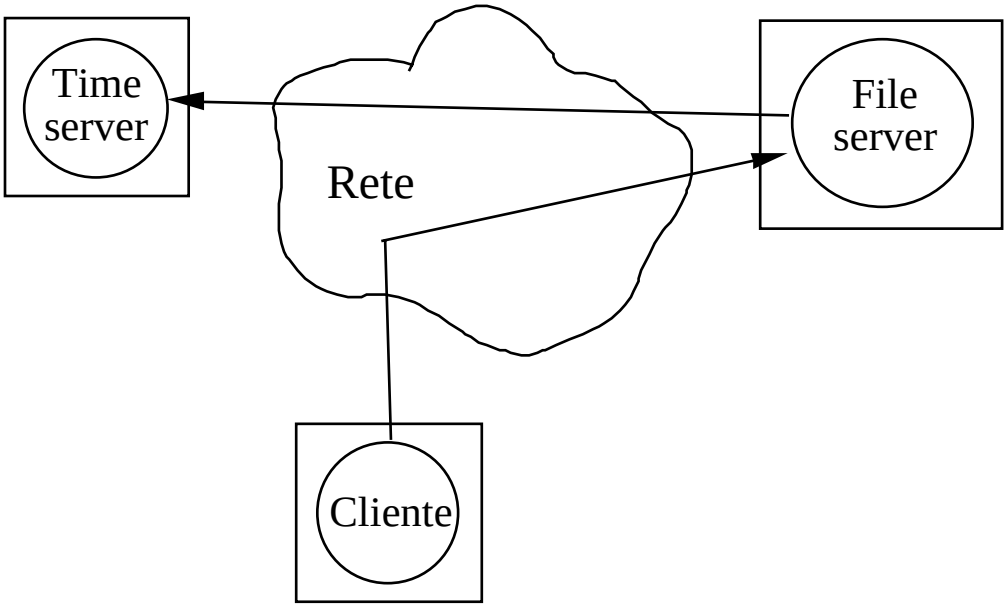
  telnet *sistema_remoto porto*

- E' quindi possibile accedere con lo stesso comando (telnet) a più servizi, associati a porti diversi
- Ad es., col meccanimo base di comunicazione fornito da TELNET si può accedere con lo stesso cliente a servizi interattivi di consultazione di banche dati, ecc.
- Il meccanismo di TCP/IP dei porti di comunicazione associati ai servizi è in grado di fornire grande **flessibilità** alle applicazioni cliente/servente

Serventi che sono clienti

- Non sempre un programma rientra esattamente nella definizione di cliente o servente

- Ad es., un *file server* può rivolgersi ad un *time server* per ricevere l'ora del giorno, da associare ad un file quando esso viene creato o modificato



- In questo caso il servente si comporta come cliente per un altro servizio, di cui ha bisogno per espletare il suo

- Quando riceve una richiesta da un cliente, il *file server* si rivolge al *time server*, riceve da questo una risposta, completa la sua elaborazione, ed infine risponde a sua volta al cliente

Servizi orientati/non orientati alla connessione

- Esistono due stili di interazione tra cliente e servente:
  - orientato alla connessione (*connection-oriented*)
  - non orientato alla connessione (*connectionless*)

- I due tipi di interazione corrispondono direttamente ai due protocolli di trasporto della famiglia TCP/IP

- L'interazione è **connection-oriented** se il cliente ed il servente comunicano usando **TCP**

- L'interazione è **connectionless** se il cliente ed il servente comunicano usando **UDP**

- Lo stile connection-oriented rende più semplice la programmazione, perchè solleva dalla responsabilità di rivelare e correggere gli errori di comunicazione

- Per via dei controlli eseguiti, TCP introduce un maggiore sovraccarico (*overhead*) e maggiori ritardi di trasmissione (*delays*)

- La scelta dello stile dipende anche dalle caratteristiche della rete. Su una rete geografica occorrerebbe sempre usare TCP, perchè le probabilità di errore, perdita di messaggi, ecc. sono non trascurabili

- Le applicazioni dovrebbero usare UDP solo se:
  - richiedono broadcast o multicast, oppure

  - non possono tollerare il sovraccarico e/o i ritardi introdotti dai circuiti virtuali TCP

- Le applicazioni che usano UDP si devono far carico di gestire l'affidabilità della comunicazione

Privilegi e complessità dei serventi

- Tipicamente i serventi accedono a risorse protette, ed hanno bisogno di speciali privilegi di sistema per essere eseguiti

- Per questo motivo occorre prestare particolare cura affinchè un cliente non vìoli le protezioni, né acquisisca maliziosamente dal servente particolari privilegi d'accesso

- Es.: file server
  - serve richieste di accesso remoto ai file
  - i file sono soggetti a delle protezioni
  - il file server viene eseguito come un programma privilegiato

- Il file server non può basarsi sui controlli d'accesso eseguiti dal sistema operativo, in quanto da programma privilegiato ne scavalca i controlli. Perciò:
  - il file server deve controllare che il cliente possa accedere al file richiesto

- In generale i serventi devono gestire:

- Autenticazione: riconoscimento dell'identità di un cliente

- Autorizzazione: controllo che ad un cliente sia consentito l'accesso al servizio

- Sicurezza dei dati: garanzia che essi non siano involontariamente rivelati né compromessi

Serventi con e senza stato

- Le informazioni che un servente mantiene circa lo stato della interazioni in corso con i clienti sono chiamate **informazioni di stato** o semplicemente stato del servente

- I serventi che non gestiscono informazioni di stato sono detti *stateless* (senza stato)

- Il motivo principale per cui si progetta un servente con stato è l'efficienza

- Infatti un servente con stato riduce le dimensioni dei messaggi che cliente e servente si devono scambiare

- Inoltre lo stato consente al servente di ricordare la precedente richiesta di un cliente e di calcolare la risposta in maniera incrementale, perciò più rapida

- Il motivo principale per cui si progetta un servente senza stato è l'affidabilità

- Le informazioni di stato possono infatti diventare scorrette o non aggiornate se la rete causa duplicazioni, ritardi e perdite di messaggi, oppure se il cliente cade e riparte

- Inoltre, se i clienti vanno ripetutamente in *crash*, un servente con stato può consumare eccessive risorse

Serventi con e senza stato (segue)

- Se un servente usa informazioni di stato scorrette, può fornire risposte non corrette, essere cioè inaffidabile

- Si consideri ad es. un file server, che soddisfa richieste di lettura/scrittura di un blocco di dati su un file da parte di clienti remoti

- Se il servente è senza stato, ogni messaggio di richiesta di lettura/scrittura da un cliente deve contenere:
  - l'intera indicazione (*pathname*) del file
  - la posizione all'interno del file
  - il numero di byte da leggere/scrivere
  - il blocco di dati, in caso di scrittura

- Se il servente è con stato, si elimina la necessità di fornire il *pathname* completo, e, per accessi sequenziali, la posizione nel file, ma ...

- se la rete duplica un messaggio, il servente ritorna due volte lo stesso blocco di dati

- Anche se il cliente se ne accorge, e ignora il secondo blocco, nel frattempo il servente ha aggiornato la posizione corrente nel file, disallineandosi col cliente

Idempotenza delle operazioni

- In generale, la presenza o assenza di stato non è solo un criterio implementativo del servente, ma dipende dal protocollo applicativo usato da cliente e servente

- Il **protocollo applicativo** è l'insieme di regole che il cliente ed il servente osservano nel loro scambio di messaggi, al fine di interagire per richiedere/fornire il servizio desiderato

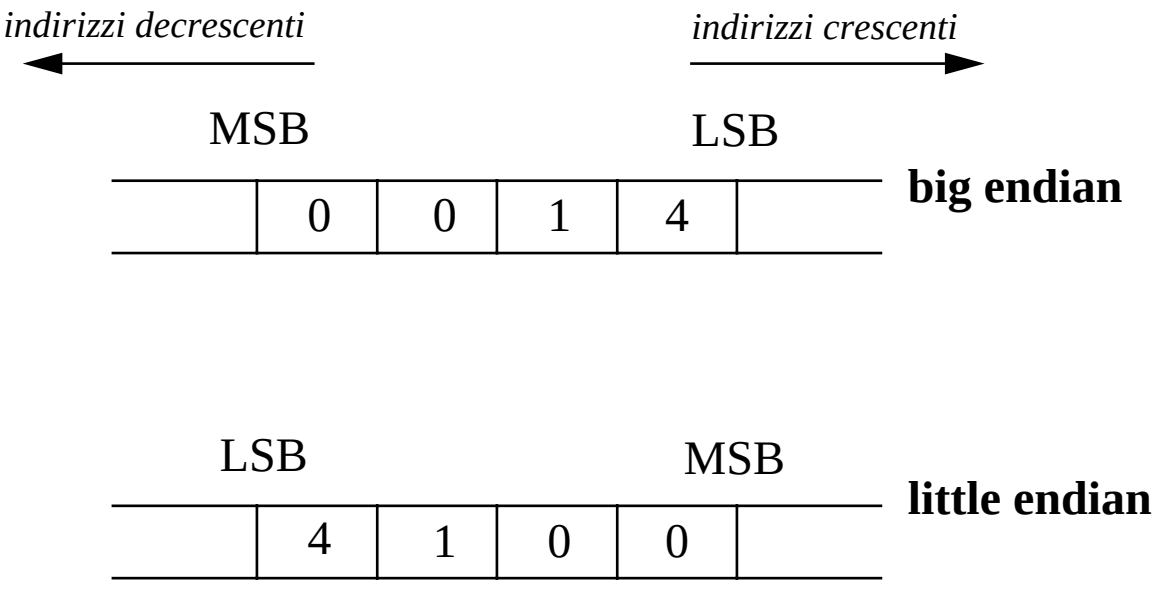
- Se il protocollo è tale che il servente fornisce sempre la stessa risposta ad una data richiesta, indipendentemente da quante volte essa giunga, si dice che l'operazione corrispondente è **idempotente**

- Nell'esempio del file server, se il servente è senza stato, esso richiede che il cliente specifichi, per una operazione di scrittura: il file, la posizione, il blocco da scrivere ed il numero di byte

- In tal caso l'operazione di scrittura è idempotente

Comunicazione tra sistemi eterogenei

- Ciascun processore definisce la propria rappresentazione dei vari tipi di dati in memoria
- Ci sono computer che memorizzano i byte più significativi agli indirizzi più bassi di memoria, altri agli indirizzi più alti, altri non memorizzano i byte in maniera contigua
- Es.: rappresentazione in memoria dell'intero decimale 260 come un intero binario su 32 bit, nelle due organizzazioni più comuni (*little endian* e *big endian*); le cifre in figura sono la rappresentazione decimale di ciascun byte:



- Quando si progetta un'applicazione, occorre prevedere il caso che le macchine dove girano il cliente ed il servente abbiano rappresentazioni native differenti
- Chiaramente, non si può operare prevedendo ogni possibile coppia di rappresentazioni tra cliente e servente

Comunicazione tra sistemi eterogenei (segue)

- La tecnica adottata nelle comunicazioni via rete tra sistemi eterogenei è quella di usare una rappresentazione intermedia comune (*machine-independent*), ed una conversione simmetrica

- Il mittente converte i dati dalla rappresentazione nativa locale a quella intermedia standard, prima di spedirli

- Il ricevente esegue la conversione dal formato intermedio a quello nativo locale

- La rappresentazione intermedia comune è detta ***external data representation***

- Naturalmente, la flessibilità e la semplicità di programmazione vengono pagate al prezzo di un sovraccarico, a volte non trascurabile
  - nel tempo di calcolo aggiuntivo per eseguire le conversioni
  - nel tempo di trasmissione (messaggi più lunghi)

- Tale sovraccarico è per di più inutile se cliente e servente si trovano realmente ad operare su nodi con la medesima rappresentazione

Lo standard XDR

- La **eXternal Data Representation** definita dalla Sun Microsystems, nota come **XDR** è diventata lo standard *de facto* per le applicazioni cliente/servente

- Tipi di dati per i quali XDR definisce una rappresentazione standard indipendente dall'hardware:

Tipo di dati	Dimens.	Descrizione
int	32 bit	intero binario con segno
unsigned int	32 bit	intero binario senza segno
bool	32 bit	val. booleano rappres. da 0 o 1
enum	variab.	tipo enumerativo con val. definiti da interi (es.: lun=1, mar=2, ...)
hyper	64 bit	intero binario con segno
unsigned hyper	64 bit	intero binario senza segno
float	32 bit	virgola mobile singola precisione
double	64 bit	virgola mobile doppia precisione
opaque	variab.	dati nella rappresentazione nativa del mittente
string	variab.	stringa di caratteri ASCII
fixed array	variab.	vettore con numero fissato di elem. di qualsiasi tipo
counted array	variab.	vettore con un numero max di elem, ma riempimento variab.
structure	variab.	come un record (struct) in C
discriminated union	variab.	come una union in C
void	0	usato dove non c'è un valore laddove un dato è opzionale
symbolic constant	variab.	cost. simb. e valore associato
optional data	variab.	ammette zero o una occorrenze di un elemento

- Lo standard XDR specifica come i tipi di dati devono essere codificati quando vengono inseriti in messaggi e trasmessi sulla rete

- Es.: XDR specifica che il tipo int usa la rappresentazione "big endian"

Sommario

- Il paradigma cliente/servente classifica i programmi applicativi come clienti o serventi a seconda di chi prende l'iniziativa della comunicazione
- Un programma che fa da servente per un servizio può comportarsi da cliente per un altro servizio

- In un sistema cliente/servente, la concorrenza tra i clienti è implicita
- I serventi vanno esplicitamente programmati per gestire la concorrenza, ovvero per servire un cliente senza dover attendere la fine del servizio di un altro
- Talvolta può essere conveniente introdurre la concorrenza in un cliente, ad es. se questo deve combinare le risposte provenienti da più serventi.

- I programmatori utilizzano solitamente TCP per spedire messaggi tra cliente e servente, perchè esso offre un servizio di comunicazione affidabile
- E' opportuno ricorrere a UDP solo per esigenze particolari, in cui TCP non possa essere adoperato

- In un mondo ideale, in cui la rete inoltra i messaggi in maniera affidabile ed i clienti non vanno mai in *crash*, progettare un servente in modo che gestisca una minima quantità di informazioni di stato circa le interazioni in corso coi clienti rende i messaggi più brevi e l'elaborazione più veloce

- Nelle comunicazioni tra sistemi eterogenei occorre fare ricorso ad una formato intermedio dei dati, indipendente

dalla rappresentazione interna adoperata dai computer del cliente e del servente

Domande ed esercizi

- Dire se i seguenti servizi di rete sono orientati o non orientati alla connessione:
  TELNET (terminale virtuale)
  FTP (trasferimento file)
  SMTP (posta elettronica)
  NFS (file system di rete)

- Quali sono i vantaggi di un servente *stateless*? E gli svantaggi? E di un servente con stato?

- Cosa si intende per idempotenza di una operazione?

- Descrivere il formato degli indirizzi del protocollo IP

- Mostrare una possibile organizzazione a domini dei nomi dei nodi di rete all'interno di una azienda su scala nazionale, organizzata in più Divisioni o Dipartimenti e con sedi in varie città

- A che serve lo standard XDR?