Lezione 9

Cenni ai sistemi operativi distribuiti
1. introduzione

- Motivazioni
- Classificazioni
- Caratteristiche e funzioni dei sistemi operativi
- Modelli di s.o. distribuiti

Un'esigenza

- Una delle poche costanti nel campo dell'informatica e' la richiesta crescente e continua di potenza computazionale
  - Previsioni meteorologiche
  - Gallerie del vento
  - Genoma umano
  - Protein folding
  - Simulazioni fenomeni naturali
  - ……

  } O (10^{12} flops = Tflops)

- Prima soluzione: progettare CPU sempre piu' veloci

Un problema

- La relativita' ristretta di Einstein impone che nessun segnale elettrico si propaga nel rame a velocita' superiore di $2x10^{10}$cm/sec
- Poiche' $s = vt \Rightarrow s = v / f$

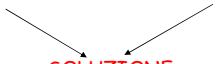
- Computer con un clock di 10 GHz i circuiti non possono essere piu' lunghi di 2 cm
- Computer con un clock di 100 GHz i circuiti non possono essere piu' lunghi di 2 mm
- Computer con un clock di 1 THz i circuiti non possono essere piu' lunghi di 0.2 mm

problema

- Circuiti cosi' piccoli dissipano una enorme quantita' di energia termica con notevoli problemi di raffreddamento
  - Es. Pentium: ~10 volte piu' delle generazioni precedenti
- Dalla meta' degli anni '80, due tecnologie rendono disponibile una alternativa:

CPU potenti e a basso costo

Reti di comunicazione efficienti e veloci



SOLUZIONE

Aggregare molte CPU in un unico ambiente di calcolo

Esempi di sistemi a processori multipli

- Processori dual core
- Server multiprocessore
- Cluster di workstation
- Supercalcolatore
- World Wide Web




vantaggi

- Condivisione delle risorse
  - Una risorsa a Roma puo' essere utilizzata da utenti a Napoli
- Accelerazione dei calcoli
  - Con due CPU si potrebbe impiegare meta' del tempo
- Scalabilita'
  - Con due CPU si potrebbe risolvere un problema piu' grande
- affidabilita'
  - Se una CPU si guasta ne posso usare un'altra
- Collaborazione
  - Team separati possono collaborare anche se distanti

Alcuni svantaggi

- L'organizzazione del software
  - Che s.o., che linguaggi cosa dovrebbe fare il sistema e cosa l'utente?
- La connessione dei processori
  - Reti: alte latenze, perdita di pacchetti
  - Condivisione delle risorse: sincronizzazione
- Sicurezza
  - Accesso solo ai dati consentiti

Classificazioni dei sistemi a processori multipli

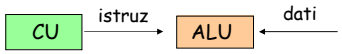
Esiste una grandissima varieta' di organizzazioni possibili

Necessarie classificazioni per evidenziare le caratteristiche

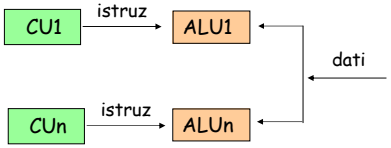
Classificazioni in base
- Al rapporto istruzioni e dati (tassonomia di Flynn)
- Alla condivisione delle risorse
- All'accesso alla memoria
- Al collegamento tra le CPU

Tassonomia di Flynn (1/2)

- La tassonomia di Flynn classifica i sistemi di calcolo in 4 categorie:
  - Single-instruction-stream, single-data-stream (SISD) computers
    - Sistemi monoprocessore

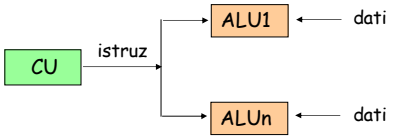


  - Multiple-instruction-stream, single-data-stream (MISD) computers
    - Non praticabile

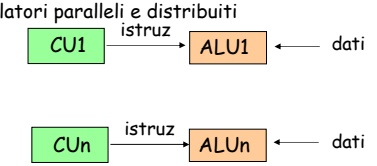


Tassonomia di Flynn (2/2)

- Single-instruction-stream, multiple-data-stream (SIMD) computers
  - array processors e calcolatori vettoriali



- Multiple-instruction-stream, multiple-data-stream (MIMD) computers
  - Calcolatori paralleli e distribuiti



Un problema della tassonomia di Flynn

- La tassonomia di Flynn ha lo svantaggio di raggruppare tutti i sistemi con molte CPU in un'unica categoria
- E' necessaria un'altra classificazione all'interno della classe MIMD

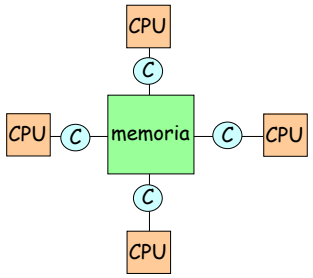
Classificazione in base alla condivisione delle risorse

- Multiprocessore
- Multicomputer
- Sistema distribuito

} Calcolatori paralleli

Sistema multiprocessore

E' un sistema dove le CPU condividono gran parte delle risorse, compresa la memoria (sistemi paralleli a memoria condivisa)



Problemi di accesso alla memoria (sincronizzazione e scarsa scalabilita')

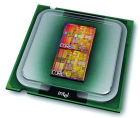
Limitato numero di CPU (2-8)

memorie locali o cache possono ridurre il problema dell'accesso alla memoria comune

esempi

Cray X-MP/48

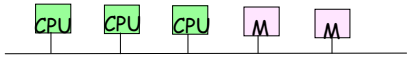
Server multiprocessore



Processori dual core

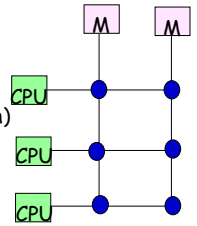
Alcuni schemi

- Bus based



  - Overhead del bus
  - Possibilita' di aggiunta di cache
  - Coerenza delle cache
  - Monitoraggio della memoria da parte delle cache (snoopy cache)

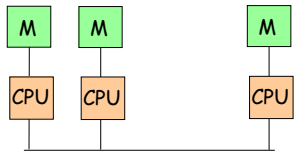
- Switch based



  - Richiesti n^2 switch
  - Possibilita' di switch a stadi (omega switch)
  - Ritardo dovuto all'attraversamento degli stadi

Sistema multicomputer

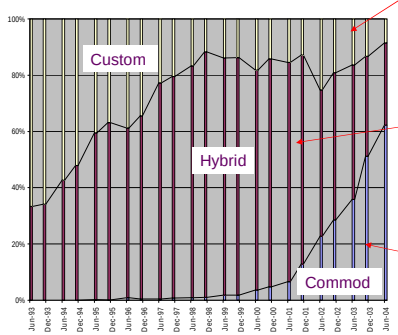
E' un sistema dove le CPU non condividono risorse (sistemi paralleli a memoria distribuita)



Ogni nodo e' un computer completo (senza periferiche) e comunica con gli altri nodi mediante uno schema a scambio di messaggi

Il modo per realizzare un multicomputer e' collegare un gruppo di CPU mediante una rete o switch

Architetture dei top500



Fonte: Jack Dongarra

CPU proprietarie con connessione proprietaria
- Cray X1
- NEC SX-7
- IBM Regatta
- IBM Blue Gene/L

CPU "commerciali" con connessione proprietaria
- SGI Altix
  - Intel Itanium 2
- Cray Red Storm
  - AMD Opteron

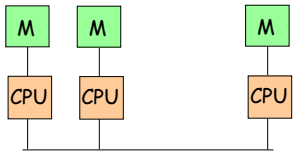
CPU "commerciali" con connessione commerciale
- Clusters
  - Pentium, Itanium, Opteron, Alpha
  - GigE, Infiniband, Myrinet, Quadrics
- IBM eServer

Topologie di connessione

- Un aspetto cruciale per le prestazioni e per l'affidabilita' dei sistemi multicomputer e' la topologia di connessione tra i nodi
  - Bus
  - Griglia 2D
  - Ipercubo
  - Switch

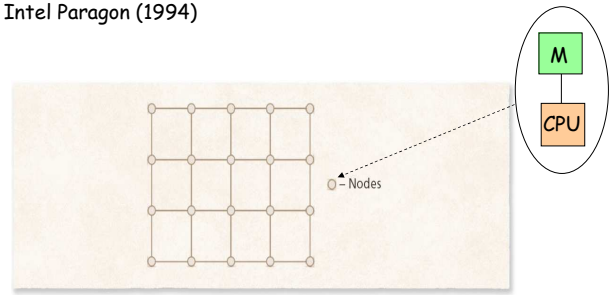
Topologia a bus

- I nodi sono connessi tramite un bus condiviso
- Topologia piu' semplice ed economica (sufficiente un cavo Ethernet)
- Scarsa efficienza ma affidabile
- Es. cluster di PC



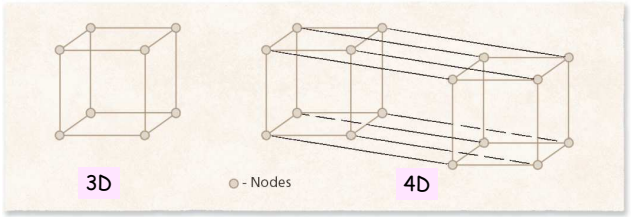
Topologia a griglia 2D

- *n* righe e *m* colonne di nodi, in cui ogni nodo e' connesso a 4 nodi vicini
- Relativamente economico e semplice da progettare per un gran numero di nodi
- scarsa performance and fault tolerance
- Es. Intel Paragon (1994)



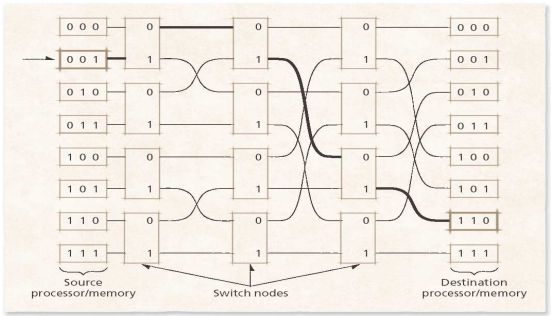
Topologia a ipercubo

- Un ipercubo *n*-dimensionale ha 2^n nodi in cui ogi nodo e' connesso a *n* vicini
- Piu' efficiente e affidabile della griglia 2D
- meno economico e piu' difficile da progettare
- Es. nCUBE (1990)



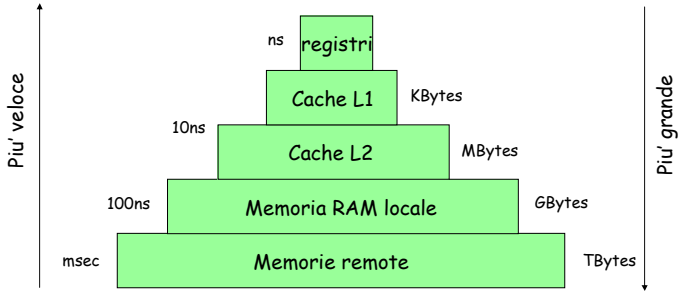
Topologia a switch a piu' stadi (omega switch)

- I messaggi viaggiano attraverso piu' livelli di switch molto semplici
- Miglior compromesso tra costi e prestazioni
- Es. IBM ASCI withe (2004)



Macchine NUMA

- I multicomputer rappresentano un esempio di macchine NUMA = Non Uniform Memory Access



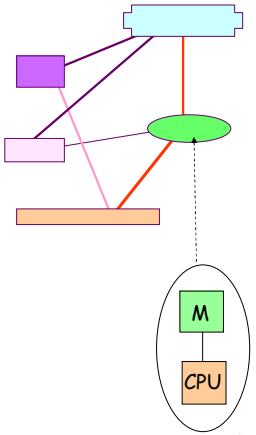
- Obiettivo: usare i dati dei livelli "alti" della memoria, ma problemi di coerenza

Sistemi distribuiti

E' un sistema dove i nodi sono sistemi completi, fisicamente distribuiti, indipendenti e eterogenei

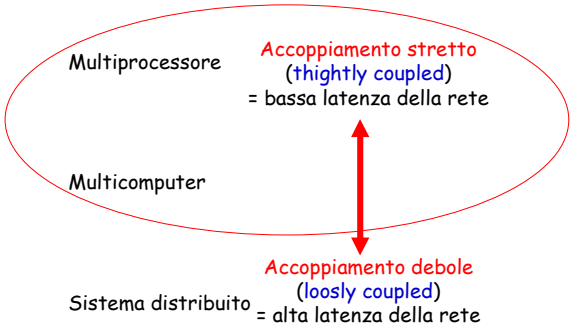
IDEA: uso dei calcolari connessi ad una rete come un unico e coerente ambiente di calcolo

I nodi comunicano mediante scambio di messaggi attraverso una rete geografica WAN



Un'altra classificazione

- E' possibile classificare i sistemi paralleli/distribuiti anche in base alla "intensita' " del collegamento tra le CPU



Sistemi operativi per sistemi con molti processori

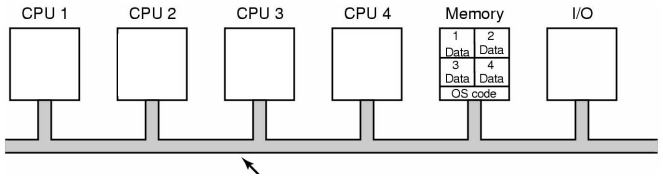
- Come l'hardware anche il software di sistema puo' avere differenti "intensita'" di accoppiamento
  - Sistemi operativi per multiprocessor (MOS)
    - Accoppiamento stretto su sistemi multiprocessor
  - Sistemi operativi distribuiti (DOS)
    - Accoppiamento stretto su sistemi multicomputer
  - Sistemi operativi di rete (NOS)
    - Accoppiamento sw debole su sistemi multicomputer

Sistemi operativi per multiprocessor (MOS)

- Accoppiamento sw stretto su multiprocessor
- stessi obiettivi di un sistema operativo monoprocessore
  - un unico sistema operativo
  - potrebbe non essere considerato un vero e proprio sistema operativo distribuito
- L'utente ha l'impressione di una sola CPU
- gestione centralizzata delle risorse
  - unica ready queue
  - memoria

s.o. per multiprocessori (1)

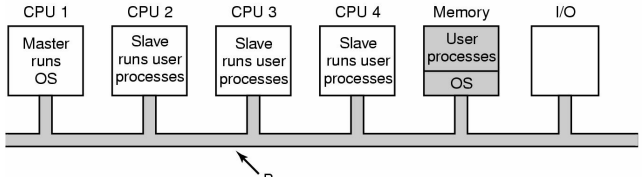
- Il modo piu' semplice per organizzare un MOS consiste nel devidiere staticamente la memoria quante sono le CPU
- Viene condiviso il codice del s.o. ma ogni CPU ha dati separati
- Le CPU lavorano come computer indipendenti



- Problema: non si sfrutta la condivisione

s.o. per multiprocessori (2)

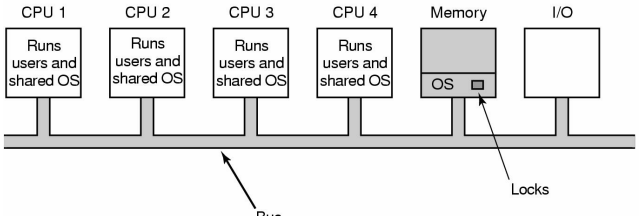
- Una seconda organizzazione si basa sul modello master/slave
- Una CPU esegue il s.o. mentre le altre eseguono i processi utente
- Risolve tutti i problemi di efficienza del modello precedente



- Problema: collo di bottiglia nella CPU1

s.o. per multiprocessori (3)

- Una terza organizzazione mira ad eliminare la asimmetria dell'approccio precedente (modello Symmetric Multi Processor SMP)
- Ogni CPU puo' eseguire sia il S.O. sia i processi utente e tutta la memoria e' a disposizione di tutte le CPU
- Miglior bilanciamento nell'uso delle risorse

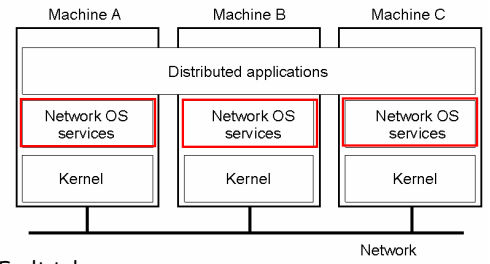


- Problema: sincronizzazione nell'accesso alle strutture dati del s.o.

s.o. per multicomputer

- Indipendentemente dal tipo di s.o. i multiprocessori hanno il collo di bottiglia rappresentato dalle risorse comuni (soprattutto l'accesso alla memoria)
- Lo stato dell'arte oggi e' rappresentato dai multicomputer
- Le CPU non comunicano attraverso la memoria ma scambiandosi informazioni
  - Sistemi operativi di rete
  - Sistemi operativi distribuiti

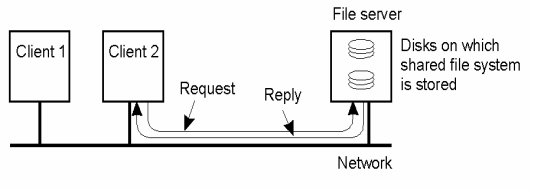
Sistemi operativo di rete



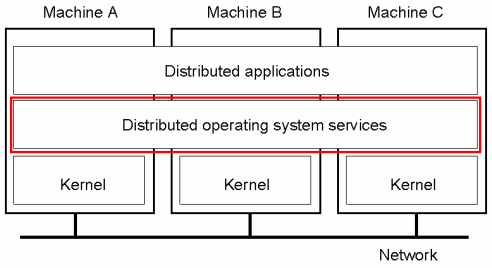
- Fault tolerance
- Accoppiamento sw debole su multicomputer
- Ogni nodo ha un proprio sistema operativo (anche diverso)
- L'utente ha coscienza della presenza di numerose CPU
- Il parallelismo e' tutto a carico dell'utente

Sistemi operativi di rete

- Offre servizi per l'accesso a risorse remote
  - telnet, ftp, rlogin per l'utilizzo di risorse remote
- Scambio di informazione solo attraverso scambio di file
- Esempio: rete di workstation
  - Unico file system di rete con montaggio automatico



Sistemi operativo distribuito



- Minore fault tolerance
- Accoppiamento sw stretto su multicomputer
- servizi uniformi e trasparenti del sistema operativo
- L'utente ha l'impressione di una sola CPU

Sistema operativo distribuito

- Architettura microkernel, dove i processi sulle varie macchine comunicano e si sincronizzano attraverso scambio di messaggi
  - Send, receive
- il kernel locale (lo stesso per tutti i computer) gestisce le risorse fisiche locali (es: paginazione della memoria, scheduling della CPU)
- Piu' difficile da realizzare

confronto

	NOS	DOS	MOS
CPU apparenti	N	1	1
Stesso s.o.	No	Si	Si
Copie del kernel	N	N	1
Che comunicazione	File	Messaggi	Memoria condivisa
Richieste sul protocollo di rete	Si	Si	no
singola ready queue	No	No	Si
Nomi di file comuni	No	Si	Si

Aspetto fondamentale

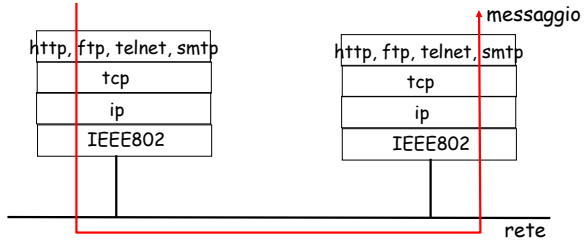
- La differenza principale tra un sistema operativo per multiprocessore e un sistema operativo per multicomputer e' la

  Comunicazione tra i processori

- Sistemi NOS:
  - Necessario un protocollo per il coordinamento di operazioni asincrone in un ambiente lento e soggetto ad errori
- Sistemi DOS
  - Oltre ad un protocollo, necessari anche meccanismi di alto livello per la comunicazione

Esempio: protocollo TCP/IP

- Usato principalmente per sessioni remote su reti WAN (telnet, ftp,…)
- 4 strati sw sovrapposti che nascondono il livelli e i dettagli sottostanti tra due nodi anche differenti

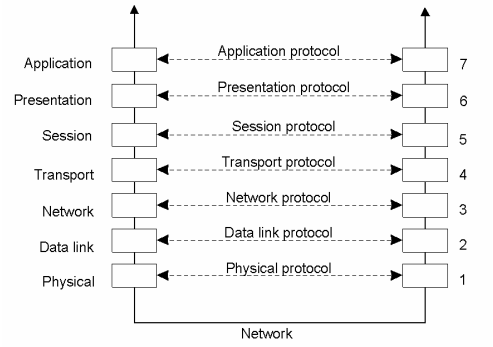


Protocollo TCP/IP

- IEEE802.X: Ha il compito di rappresentare i segnali elettrici come stringhe di bit ed e' responsabile dell'individuazione e della correzione degli errori
- Strato IP: responsabile dell'instradamento dei pacchetti, della codifica e decodifica degli indirizzi
- Strato TCP: responsabile della suddivisione, ordinamento e ricomposizione di un messaggio in pacchetti
- Strato delle applicazioni: responsabile della comunicazione tra processi, della gestione delle differenze di formato e della conversione dei caratteri e della interazione con gli utenti (trasf. File, sessioni remote, email)

Protocollo Open System Interconnection (OSI)

- Alternativa: 7 strati sw sovrapposti. Piu' flessibile ma meno efficiente



Sistemi operativi distribuiti

Tali protocolli di comunicazione hanno un elevato overhead dovuto ai numerosi strati sw.

utilizzati principalmente per ambienti di calcolo con elevate latenze della rete e kernel eterogenei➔ s.o. di rete

Una interazione basata su sessioni remote e scambio di file su protocolli di rete non permette una visione unitaria del sistema

Approccio alternativo basato su scambi di mesaggi

Message passing

- I sistemi operativi distribuiti hanno lo stesso kernel e i processi comunicano attraverso scambi di messaggi (architettura microkernel)
- In generale non sono necessari la maggior parte degli strati sw (ed es. per l'instradamento , per la gestione dei pacchetti, per la gestione della sessione e delle applicazioni)
- Necessario un solo livello con lo scopo di
  - Interfaccia con l'utente
  - Definizione dell'insieme delle richieste legali e dei valori di ritorno

Funzioni send e receive

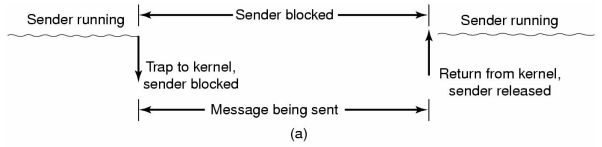
- Per una architettura sw cosi' semplice sono sufficienti poche primitive
- Send (destinatario, &mptr)
  - Manda il messaggio puntato da mptr al processo destinatario
- Receive (source, &mptr)
  - Riceve dal processo source il messaggio contenuto in mptr

Message passing

- La funzione receive e' in genere bloccante (le istruzioni successive alla chiamata non sono eseguite fino alla ricezione del messaggio)
- La funzione send puo' essere bloccante o non bloccante
- send bloccante: le istruzioni successive alla chiamata non sono eseguite fino al completamento della trasmissione
  - Facile da implementare e da controllare
- send non bloccante: la funzione ritorna subito.
  - Piu' efficiente ma non e'possibile sapere se il messaggio e' ricevuto
  - Implementata con copia in un buffer del kernel, oppure con un thread

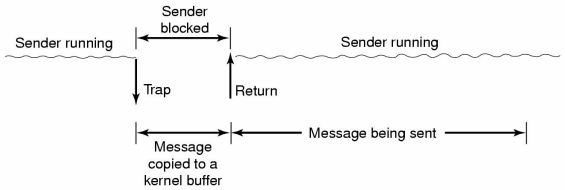
Funzioni Bloccanti vs non bloccanti

Send bloccante



(a)

Send non bloccante



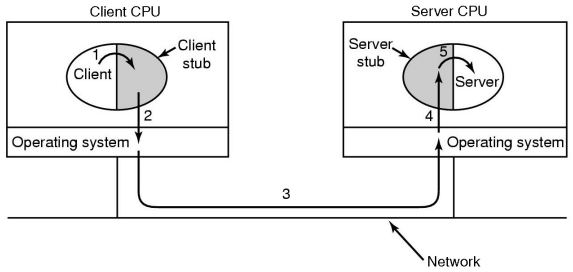
(b)

Remote Procedure Call (RPC)

- Message passing utilizza in modo pesante le funzioni l'I/O
- Una alternativa al message passing: richiamare procedure che risiedono su altre unità processanti
- L'idea è di far apparire la chiamata remota come una chiamata locale:
  - Se stiamo eseguendo una procedura su una CPU A e vogliamo invocare una procedura su una CPU B, la procedura su A viene sospesa e viene eseguita la procedura su B.
- Lo spazio degli indirizzi è diverso e quindi si deve replicare in qualche modo

Stub

- Per simulare la chiamata locale, invece di linkare la libreria locale, l'applicazione viene collegata ad una libreria chiamata stub
- Gli stub impacchettano e spacchettano i dati al fine di minimizzare la dimensione del messaggio (marshalling)



Passi di una RPC

1. Il client chiama lo stub
2. Lo stub costruisce il messaggio e fa una chiamata al s.o. locale (marshalling)
3. Il s.o. del client trasmette il messaggio al s.o. remoto
4. Il s.o. remoto consegna il messaggio allo stub del server
5. Lo stub del server spacchetta il messaggio ed effettua una chiamata di procedura locale
6. Il server fa il lavoro e ritorna il risultato allo stub
7. Lo stub del server impacchetta il messaggio e lo consegna al s.o.
8. Il s.o. del server invia il messaggio al s.o. del client
9. Il s.o. consegna il messaggio allo stub del client
10. Lo stub spacchetta il messaggio e consegna il risultato al client

Problemi delle RPC

- Gestione dei puntatori (lo spazio di indirizzamento e' differente)
  - Nello stub del client si estrae il dato che viene spedito al server, il quale passa un puntatore al server
- Dimensioni variabili degli array in C
  - lo stub non puo' fare il marshalling dei dati se non ne conosce la quantita'
- Numero variabili dei parametri (es. printf)
  - anche in questo caso lo stub non e' in grado di effettuare il marshalling
- Uso di variabili globali
  - La procedura remota fallira'