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

Progetto di applicazioni distribuite di tipo cliente/servente nel modello a scambio di messaggi in ambiente UNIX TCP/IP

Progetto di serventi

Algoritmo concettuale di un servente

- Concettualmente ogni servente esegue un algoritmo molto semplice:

- crea una socket

- la collega (*bind*) al porto locale predefinito su cui si mette in attesa di ricevere richieste

- entra in un ciclo infinito in cui
  accetta una richiesta da un cliente
  elabora la richiesta e formula la risposta
  invia la risposta al cliente

- Le complicazioni nascono dalla necessità, in generale, di dover gestire le richieste di più clienti simultaneamente (concorrenza nei serventi)

- Si consideri ad esempio un servente che offre il servizio di trasferimento file da/verso clienti su nodi remoti:

- una richiesta di trasferimento di un file molto grande non dovrebbe penalizzare troppo una richiesta analoga per un file piccolo

Tipi di servente

- I serventi vengono classificati
  - in base al tipo di protocollo di trasporto adoperato
  - alla possibilità di gestire o meno più richieste allo stesso tempo

CONCOR-RENZA	PROTOCOLLO DI TRASPORTO: iterativo - non orientato alla connessione	PROTOCOLLO DI TRASPORTO: iterativo - orientato alla connessione
	concorrente - non orientato alla connessione	concorrente - orientato alla connessione

- Il termine servente iterativo si riferisce ad un servente che elabora una richiesta alla volta
- Il termine servente concorrente si riferisce ad un servente in grado di gestire più richieste simultaneamente, ma
- non necessariamente l'implementazione si basa su più processi eseguiti in concorrenza
- La scelta del protocollo di trasporto (TCP per serventi *connection-oriented*, UDP per serventi *connectionless*) dipende dal protocollo applicativo, cioè dalle modalità di interazione previste tra cliente e servente

Scelta dell'*endpoint* locale

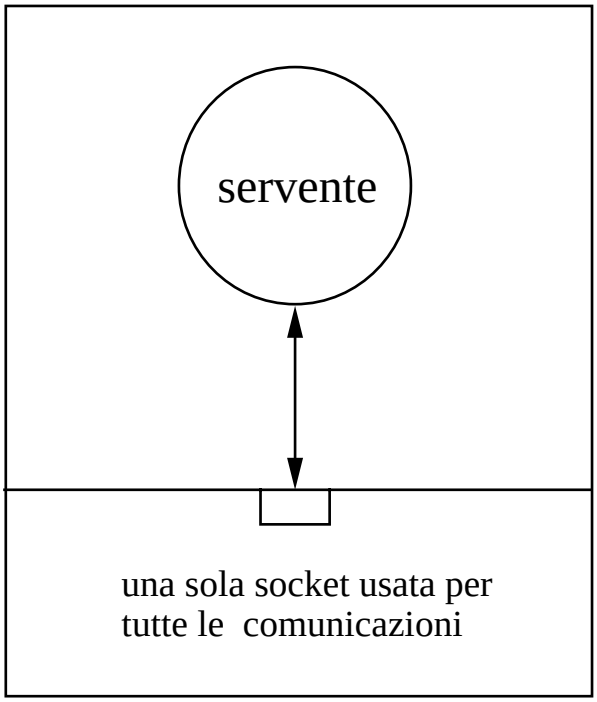
- Dopo aver allocato una socket, un servente utilizza bind per associarla ad un estremo locale nel dominio delle comunicazioni
- Come per un cliente, viene usata una struttura di tipo sockaddr_in che contiene sia un indirizzo IP che un numero di porto
- Il numero di porto del servizio può essere ottenuto, come per i clienti, con getservbyname
- L'indirizzo IP dell'host del servente può non essere unico, se esso ha più di una interfaccia di rete
- D'altra parte il servente deve poter ricevere i messaggi in entrata, qualunque sia l'indirizzo usato da un cliente
- Ciò viene ottenuto utilizzando la costante INADDR_ANY
- Un generico servente esegue le seguenti operazioni preliminari prima di allocare una socket:

1. Trova il numero di porto del servizio (*getservbyname*)
2. Trova l'identificativo del protocollo (*getprotobyname*)
3. Specifica che la socket deve accettare richieste indirizzate ad uno qualsiasi degli indirizzi IP del nodo (*INADDR_ANY*)
4. Creare la socket (*socket*)
5. Specifica l'*endpoint* locale (*bind*)

Struttura di un servente iterativo UDP

- Il servente è costituito da un solo processo che
  - esegue un ciclo infinito
  - usa un'unica socket per le richieste e le risposte

- Il servente usa l'indirizzo di provenienza della richiesta come indirizzo del destinatario della risposta



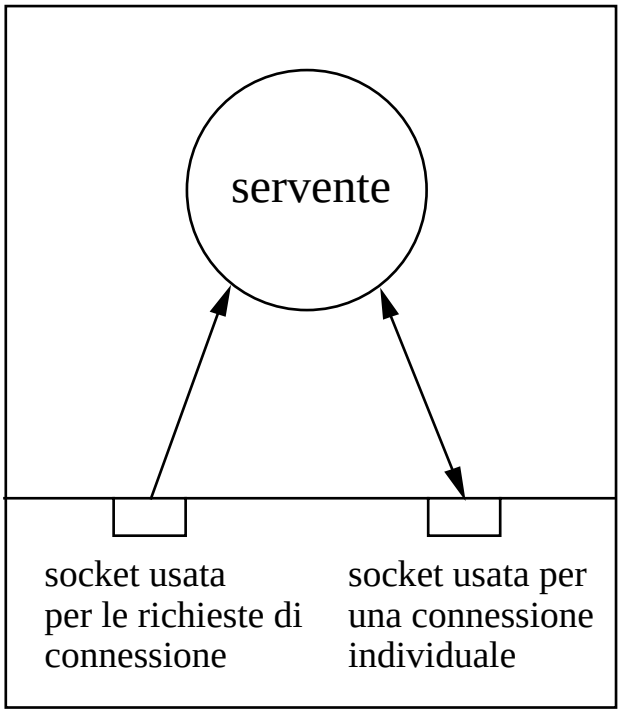
- Algoritmo del servente:

1. Creare e legare una socket di tipo SOCK_DGRAM (*socket*, *bind*)

2. In un ciclo infinito:
   ricevere una richiesta da un cliente (*recvfrom*)
   elaborare la richiesta e preparare la risposta
   inviare la risposta (*sendto*)

Struttura di un servente iterativo TCP

- Il servente è costituito da un solo processo che itera sulle connessioni: aspetta la connessione di un cliente, la accetta, interagisce, chiude la connessione con quel cliente e passa ad attendere il prossimo

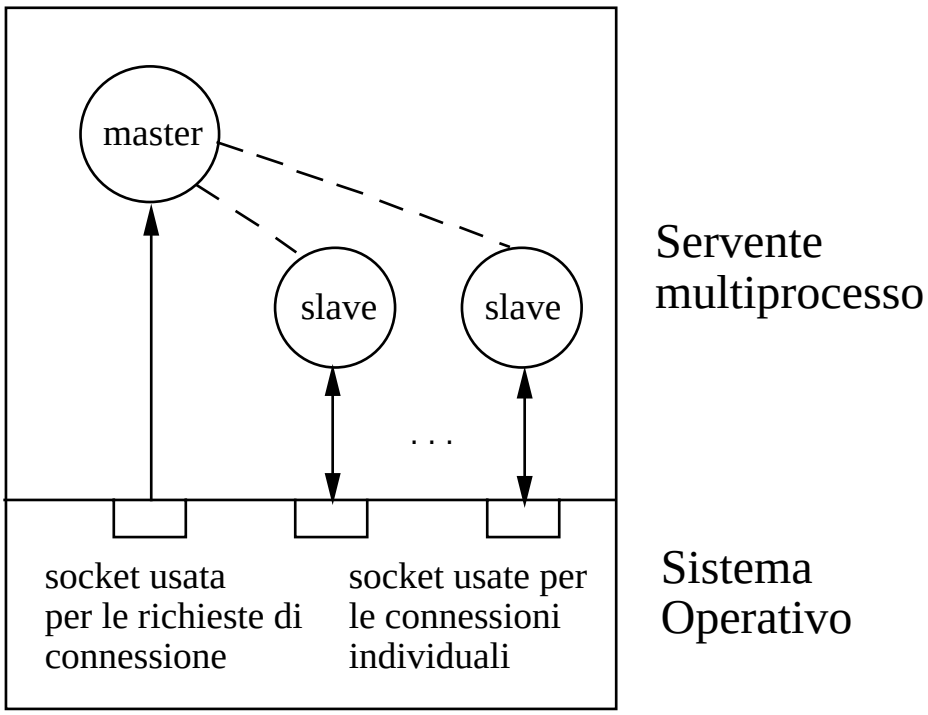


- Algoritmo del servente:

1. Creare, legare e rendere passiva una socket di tipo SOCK_STREAM (*socket*, *bind*, *listen*)
2. Accettare la connessione di un cliente (*accept*)
3 Comunicare con il cliente (*read*, *write*)
4. Quando si è finito con quel cliente, chiudere la connessione (*close*)
5. Ritornare al passo 3

Struttura di un servente concorrente TCP

- Il servente è costituito da più processi
- Un processo **master** accetta le richieste di connessione (apertura di una sessione), ma non comunica direttamente con i clienti
- Quando giunge una nuova connessione, il master crea un processo **slave,** cui demanda l'interazione col cliente
- Lo slave riceve le richieste del cliente ed invia le risposte
- Quando ha terminato con il suo cliente, uno slave chiude la socket e termina
- La concorrenza è tra le sessioni (non tra le singole richieste)



Struttura di un servente concorrente TCP
(segue)

- Algoritmo del servente:

Processo MASTER

1. Creare, legare e rendere passiva una socket di tipo SOCK_STREAM (*socket*, *bind*, *listen*)

2. In un ciclo infinito:
   2.1 attendere una richiesta di connessione di un cliente (*accept*)
   2.2 creare un processo slave per la gestione del cliente (*fork*)

Processo SLAVE

1. Ricevere la socket al momento della creazione

2. Interagire con il cliente (*read*, *write*)

3. Chiudere la connessione (*close*)

4. Terminare (*exit*)

- Tipicamente, un solo programma è adatto a contenere il codice del master e dello slave (che si differenziano per il valore di ritorno della fork)

- Qualora sia più conveniente far gestire l'interazione col cliente ad un programma scritto e compilato separatamente, il processo figlio può eseguire la primitiva UNIX execve

Struttura di un servente concorrente UDP

- Il servente è costituito da più processi
- Un processo **master** accetta le singole richieste di servizio, ma non invia personalmente le risposte
- Quando giunge una nuova richiesta, il master crea un processo **slave,** cui demanda la risposta al cliente
- Lo slave termina dopo aver gestito una sola richiesta

Processo MASTER

1. Creare e legare una socket di tipo SOCK_DGRAM (*socket*, *bind*)
2. In un ciclo infinito:
   2.1 ricevere una richiesta da un cliente (*recvfrom*)
   2.2 creare un processo slave per la gestione della risposta (*fork*)

Processo SLAVE

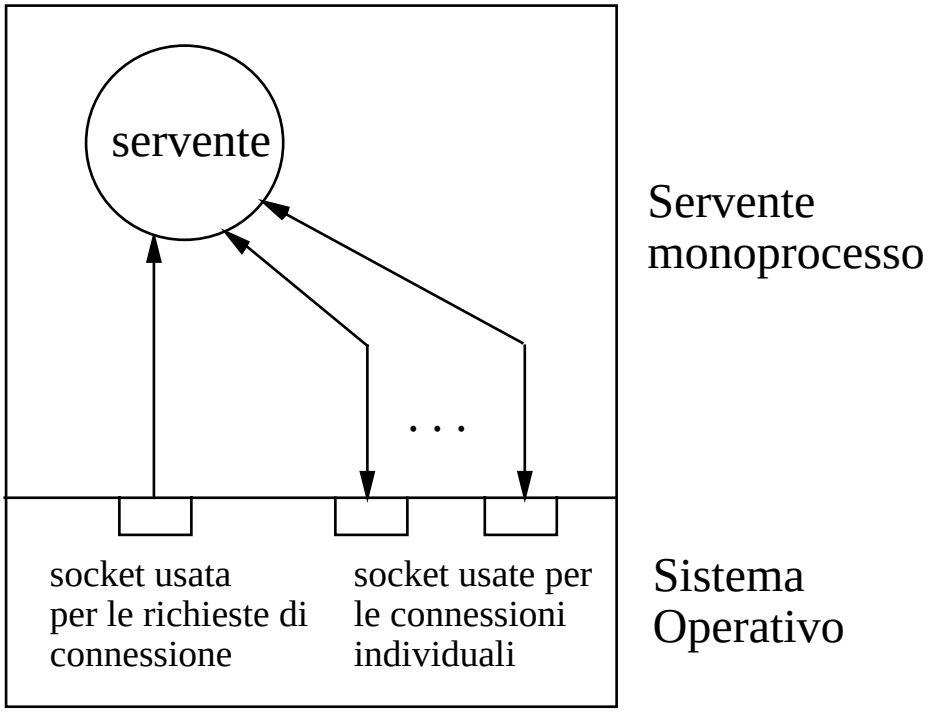
1. Ricevere contestualmente la socket e la richiesta del cliente, al momento della creazione
2. Preparare e spedire la replica al cliente (*sendto*)
3. Terminare (*exit*)

- In pratica questa soluzione si trova realizzata raramente, perchè il costo della creazione di un altro processo con fork

è troppo oneroso (rispetto al costo di preparazione ed invio delle singole repliche)

Struttura di un servente concorrente monoprocesso

- Il servente è costituito da un solo processo che usa la primitiva UNIX select per gestiore l'I/O concorrente
- Il processo gestisce più socket simultaneamente
- Quando una socket è pronta, ciò può voler dire
  - che una nuova richiesta di apertura di sessione è giunta (un nuovo cliente)
  - che un cliente esistente ha mandato un nuovo messaggio
- E' la socket originaria quella usata per accettare i nuovi clienti



Struttura di un servente concorrente monoprocesso (segue)

- Algoritmo del servente:

1. Creare, legare e rendere passiva una socket di tipo SOCK_STREAM (*socket*, *bind*, *listen*)

2. Inserire la socket nella lista di quelle si cui si può avere I/O concorrente

3. Attendere una comunicazione su una qualsiasi socket della lista (*select*)

4. Se la socket pronta è quella originaria:
   4.1 accettare la connessione di un cliente (*accept*)
   4.2 aggiungere la nuova socket alla lista

5. Se la socket pronta non è quella originaria:
   5.1 leggere una richiesta (*read*)
   5.2 preparare e spedire la risposta (*write*)

6. Ritornare al passo 3

Criteri di progetto

- Un servente iterativo UDP è adatto per servizi che richiedono una quantità di elaborazione trascurabile per ciascuna richiesta dei clienti

- Un servente iterativo TCP è adatto per servizi che richiedono una quantità di elaborazione trascurabile per ciascuna richiesta dei clienti, ma che necessitano di un protocollo di trasporto affidabile;
  Poichè il sovraccarico per stabilire e terminare ciascuna sessione può essere elevato, il tempo medio di risposta può non essere "piccolo"

- Un servente concorrente UDP è usato di rado;
  Poichè il sovraccarico per creare nuovi processi è elevato, tale scelta deve essere giustificata da un elevato tempo di calcolo per elaborare la risposta

- Un servente concorrente TCP è forse quello usato più di frequente, perchè si basa su trasporto dei messaggi affidabile ed è in grado di gestire più clienti, evitando di penalizzare con lunghe attese i clienti che eseguono sessioni brevi;
  Ne esistono due implementazioni: multiprocesso e monoprocesso

- La scelta tra le ultime due dipende dalla necessità o meno di condividere dati tramite memoria comune durante la gestione delle sessioni

Deadlock

- Un criterio importante per scegliere una implementazione concorrente invece di una iterativa è la possibilità di stallo o **blocco critico** (**deadlock**)

- Esempi di deadlock:
  Un cliente TCP non legge mai i messaggi di risposta
  => il buffer di trasmissione del servente si riempie ed una successiva write si blocca)

  Un cliente TCP non invia mai una richiesta
  => il servente si sospende sulla read dopo aver eseguito la accept

- Se un servente usa primitive bloccanti, un comportamento errato di un cliente può causare lo stallo

- In caso di attesa indefinita, un servente monoprocesso non potrà più soddisfare altre richieste

Domande ed esercizi

- Se una applicazione prevede fino a 20 clienti che inviano ciascuno 2 richieste al secondo ad un servente iterativo, qual è il tempo massimo che esso deve spendere per soddisfare ciascuna richiesta?

- Misurare quanto tempo impiega un servente concorrente *connection-oriented* per accettare una nuova connessione e generare un processo figlio per gestirla

Una libreria di procedure per i serventi

Una libreria di procedure per i serventi (segue)

Un servente iterativo UDP per il servizio TIME

Un servente iterativo TCP per il servizio DAYTIME

Un servente concorrente TCP per il servizio ECHO

Un servente concorrente TCP per il servizio ECHO (segue)

Un servente pseudoconcorrente TCP per il servizio ECHO

Un servente pseudoconcorrente TCP per il servizio ECHO (segue)