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 clienti

Algoritmo di un cliente TCP

1. Trovare il numero di porto del servizio desiderato (*getservbyname*)
2. Risolvere l'indirizzo IP della macchina dove si trova il servente (*gethostbyname* o *inet_addr*)
3. Allocare una socket di tipo SOCK_STREAM, specificando il protocollo TCP; la connessione può usare uno qualsiasi degli indirizzi IP del nodo ed un qualsiasi porto locale non ancora assegnato (a scelta di TCP) (*socket*)
4. Connettere la socket al servente (*connect*)
5. Comunicare col servente (invio richieste e ricezione risposte) (*read* e *write*)
6. Chiudere la connessione TCP (eventuale notifica al servente e rilascio della socket) (*shutdown*, *close*)

La primitiva getservbyname

- In una applicazione distribuita basata sulle socket, i serventi operano a numeri di porto predefiniti, che i clienti devono conoscere

- Per non "cablare" il numero di porto nel codice, è opportuno usare la funzione getservbyname

- getservbyname cerca in un *database* locale il numero del porto associato ad un dato servizio, dato il suo nome

- L'amministratore del sistema può modificare il numero di porto di un servizio senza che l'applicazione debba essere modificata, o anche definire nomi alternativi

- Ritorna l'indirizzo di una struttura di tipo servent, il cui campo s_port contiene il numero del porto associato al servizio desiderato

- La struttura servent è dichiarata nell'*include file* netdb.h:

```
structservent {
 char *   h_name;  /* official service name */
 char ** s_aliases; /* other aliases          */
 int        s_port      /* port for this service */
 char *   s_proto;   /* protocol to use       */
};
```

- Esempio: un cliente TCP ricerca il numero di porto ufficiale del servizio SMTP

```
#include <netdb.h>
struct  servent    *sptr;

if (sptr = getservbyname("smtp","tcp")) {
```

```
        /* ora sptr->s_port contiene il numero di porto
        del servizio smtp */
} else {  /* errore */  }
```

La primitiva getprotobyname

- Cerca in un *database* locale l'identificativo intero associato ad un protocollo, dato il suo nome
- Tale identificativo è richiesto come terzo parametro da socket (ricordare che l'interfaccia socket è generalizzata, non limitata a TCP/IP)
- Ritorna l'indirizzo di una struttura di tipo protoent, il cui campo p_proto contiene la costante intera associata
- La struttura protoent è dichiarata nell'*include file* netdb.h:

```
structprotoent {
 char *   p_name;  /* official protocol name */
 char ** p_aliases;/* other aliases          */
 int        p_proto   /* protocol port number */
};
```

- Esempio d'uso

```
#include <netdb.h>
struct protoent *pptr;

if (pptr = getprotobyname("tcp")) {
  /* ora pptr->p_proto contiene il numero
     del protocollo TCP */
} else {
        /* errore */
}
```

Identificazione del servente

- Ci sono diversi modi con cui un cliente può identificare il servente.
  Un cliente può:
  - contenere nel codice sorgente l'indirizzo IP o il nome del server
  - richiedere all'utente gli estremi del servente
  - leggere gli estremi del servente da un file
  - usare un protocollo separato (es.: inviare un *broadcast* al quale rispondono tutti i serventi)

- Scrivere il programma cliente in modo che sia l'utente finale a specificare il servente:
  - rende il cliente più generale
  - consente di cambiare la locazione del servente (uso di nomi simbolici)

- Si ha una notevole flessibilità se il cliente riceve l'indirizzo del servente come parametro sulla riga di comando.

- Il servente deve in generale poter essere specificato o col nome mnemonico o col suo indirizzo numerico IP nel formato *dotted decimal*.
  Es.:

  telnet nadis.dis.unina.it

  telnet 192.55.101.131

Parametrizzazione dei clienti

- Quando si progetta un cliente, è opportuno prevedere tra i parametri sulla riga di comando, oltre all'indirizzo del servente, il porto associato al servizio
  Es.:
  telnet *sistema_remoto porto*

- Un servizio può essere identificato per nome, o mediante il numero di porto standard ad esso associato
  Es.: il numero di porto prefissato per il servizio ECHO UDP è 7

- Ciò consente di eseguire il test di una nuova versione di una applicazione, mentre la precedente versione è in esercizio

- Le sintassi possibili per la riga di comando sono varie
  Es.:
  nadis.dis.unina.it smtp
  nadis.dis.unina.it:smtp

- Nel linguaggio C si accede ai parametri sulla riga di comando tramite argc e argv

La primitiva inet_addr

- Le primitive socket, connect, ecc. richiedono che l'indirizzo del servente sia specificato usando una struttura di tipo sockaddr_in, il cui campo sin_addr.s_addr contiene i 32 bit dell'indirizzo IP in formato binario

- Se l'indirizzo IP è specificato dall'utente nella notazione *dotted decimal*, il cliente deve eseguire una conversione

- La funzione di libreria **inet_addr** esegue la conversione da una stringa ASCII contenente l'indirizzo IP in notazione *dotted decimal* nel formato binario su 32 bit

- In caso d'errore **inet_addr** ritorna la costante INADDR_NONE

- Esempio d'uso di **inet_addr:**

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

#ifndef   INADDR_NONE
#define   INADDR_NONE      0xffffffff
#endif
char       *esempio = "192.55.101.131";
struct       sockaddr_in sin;

if (sin.sin_addr.s_addr = inet_addr(esempio) == INADDR_NONE)
{
    /* conversione impossibile */
}
```

La primitiva gethostbyname

- Se l'indirizzo è specificato in forma simbolica, si usa la primitiva **gethostbyname**

- **gethostbyname** prende una stringa ASCII contenente un indirizzo IP simbolico e ritorna l'indirizzo di una struttura hostent, il cui campo h_addr contiene l'indirizzo nel formato binario su 32 bit

- La struttura hostent è dichiarata nell'*include file* netdb.h:

```
structhostent {
  char*   h_name;         /* official host name  */
  char*   h_aliases;      /* other aliases       */
  int     h_addrtype              /* address type        */
  int   *   h_length;     /* address length      */
  char** h_addr_list;     /* list of addresses   */
};
#define h_addr     h_addr_list[0]
```

- **gethostbyname** ritorna zero se non riesce ad eseguire la traduzione del nome simbolico

La primitiva gethostbyname (segue)

- Esempio d'uso di **gethostbyname**

```
#include <netdb.h>
struct  hostent    *hptr;
char   *nomeesempio = "nadis.dis.unina.it";

if (hptr = gethostbyname(nomeesempio)) {
  /* ora hptr->h_addr contiene l'indirizzo binario
      e hptr->h_length la sua lunghezza                */
} else {
  /* hptr vale 0; indirizzo errato o traduzione impossibile */
}
```

- Una volta tradotto l'indirizzo (disponibile in h_addr), occorre copiarlo nel campo sin_addr della struttura di tipo sockaddr_in

- A tal scopo si raccomanda di usare la funzione di libreria standard BSD UNIX **bcopy**

- Poichè un indirizzo IP può contenere dei byte che valgono zero non si può usare la funzione di libreria standard strcpy

- bcopy copia un blocco di memoria da una locazione ad un'altra, indipendentemente dal contenuto

- E' disponibile la procedura **bzero** per inizializzare a zero un blocco di memoria

Scelta del porto locale

- Oltre all'*endpoint* remoto, una applicazione deve specificare un *endpoint* locale prima di poter usare una socket

- I serventi operano a numeri di porto predefiniti, che i clienti devono conoscere (o che ottengono per nome con getservbyname)

- Un cliente TCP però non opera su di un porto predefinito

- Il numero di porto locale di un cliente è indifferente, purchè
  - non coincida con un porto predefinito assegnato ad un servente
  - non coincida con quello scelto da un altro cliente

- Per un cliente, la scelta di un porto locale viene fatta in maniera trasparente da TCP

- Ciò avviene come effetto collaterale dell'invocazione della primitiva socket

- E' socket che riempie il campo della struttura dati associata alla socket, contenente il numero di porto locale lasciato non specificato dal cliente

Scelta dell’indirizzo IP locale

- Ogni host ha un indirizzo IP per ciascuna interfaccia di rete installata
- Uno stesso host può quindi avere più indirizzi IP
- In uscita, i pacchetti devono essere inviati sull'interfaccia corretta, in funzione del destinatario
- Questa scelta viene tipicamente eseguita dal software IP, che implementa il protocollo di *routing*
- Se l'indirizzo del nodo è unico, la scelta è forzata
- Se ci sono più indirizzi, è bene che un cliente non specifichi l'indirizzo IP locale prima di usare una socket
- La scelta può essere lasciata a TCP, che si fa carico di interagire con IP
- In questo modo si evita di dover codificare l'interazione con IP, il protocollo di instradamento (routing) dei messaggi sottostante TCP e UDP, e si rende il cliente più generale
- Il cliente lascia non specificato il campo della struttura dati associata alla socket, contenente l'indirizzo IP locale
- connect() riempie l'apposito campo della struttura dati associata alla socket

Algoritmo di un cliente UDP

1. Trovare il numero di porto del servizio desiderato (*getservbyname*)
2. Risolvere l'indirizzo IP della macchina dove si trova il servente (*gethostbyname* o *inet_addr*)
3. Allocare una socket di tipo SOCK_DGRAM, specificando il protocollo UDP; la connessione può usare uno qualsiasi degli indirizzi IP del nodo ed un qualsiasi porto locale non ancora assegnato (a scelta di UDP) (*socket*)
4. Nel caso di socket connessa, specificare il servente al quale devono essere inviati i messaggi
5. Comunicare col servente (invio richieste e ricezione risposte)
6. Chiudere la connessione

Clienti UDP e socket connesse

- Un cliente può usare una socket UDP in due modi: *connected* e *unconnected*

- Nella modalità **connessa**, il cliente invoca connect per specificare una tantum, prima di comunicare, il corrispondente remoto

- connect crea una associazione stabile col servente, il cui indirizzo viene registrato in modo che ...

- ... le successive comunicazioni sulla socket faranno riferimento sempre allo stesso servente (similmente al caso TCP)

- Nella modalità **non connessa**, il cliente specifica la destinazione remota ogni volta che invia un messaggio

- Un cliente adopera una socket connessa se comunica sempre con lo stesso servente

- Un cliente adopera una socket non connessa se aspetta di avere una richiesta da fare prima di decidere a quale servente inviarla

Comunicazione col servente tramite UDP

- Un cliente può usare read e write per comunicare col servente UDP
- Per ogni write UDP invia un singolo messaggio al servente
- Il messaggio inoltrato contiene tutti i dati da spedire passati alla write
- Se il cliente specifica un *buffer* sufficientemente grande, una read copia nel buffer il prossimo messaggio per intero
- A differenza del caso TCP, un cliente UDP non deve inserire l'invocazione di read in un ciclo per poter leggere un singolo messaggio

Sommario

- Si ha una notevole flessibilità in un cliente se esso riceve l'indirizzo del servente come parametro sulla riga di comando

- Il cliente usa *gethostbyname* (*inet_addr*) per convertire l'indirizzo del servente, specificato in forma simbolica (risp.: numerica)

- Al fine di poter eseguire il test di una nuova versione di una applicazione, mentre la precedente versione è ancora in esercizio, è utile prevedere tra i parametri sulla riga di comando del cliente anche il porto associato al servizio

- Un cliente lascia scegliere indirizzo e porto locali a TCP/IP, per evitare problemi di instradamento e di duplicazione di porti

- Per comunicare al servente la chiusura di una connessione TCP (in una o in entrambe le direzioni), un cliente invoca *shutdown*

- Un cliente UDP adopera una socket connessa se comunica sempre con lo stesso servente

Domande ed esercizi

- Quando viene risolto il nome di un nodo di rete, si può ottenere più di un indirizzo Internet. Perchè?

- Determinare se la rappresentazione degli interi sulla propria macchina coincide con quella standard di TCP/IP, nota come *network byte order*.
  (Suggerimento: scrivere un cliente che stampa (senza effettuare conversioni) il numero di porto standard del servizio ECHO basato su UDP; se il valore stampato è 7, allora la rappresentazione locale e quella standard TCP/IP coincidono)

- In quali circostanze un cliente TCP può chiudere una connessione con close anzichè con shutdown?

Una libreria per i clienti

Un cliente TCP per il servizio DAYTIME

Un cliente UDP per il servizio DAYTIME

Un cliente TCP per il servizio ECHO

Un cliente UDP per il servizio ECHO

Domande ed esercizi

- Implementare i programmi cliente descritti
- C'è bisogno di particolari privilegi per eseguirli?
- Modificare i clienti TCP presentati, introducendo la chiusura parziale della connessione
- Modificare il cliente UDP per il servizio ECHO, in modo che controlli inizialmente la raggiungibilità del servente, inviandogli un messaggio, e dichiarandolo irraggiungibile se la replica non previene entro 10 secondi
- Modificare il cliente UDP per il servizio ECHO, in modo che gestisca il caso in cui la rete duplichi una risposta del servente