Capitolo 34

Socket e UCSPI

`tcpclient` 3684 `tcpserver` 3684 `unixclient` 3681 `unixserver` 3681

All'interno di un sistema Unix, i metodi a disposizione dei programmi per comunicare tra loro sono diversi; quelli più evoluti fanno uso di «prese», ovvero di socket, a cui «attaccarsi».

Questi socket possono distinguersi generalmente in due categorie, in base alla modalità attraverso la quale avviene in pratica il collegamento tra i programmi: può trattarsi di una connessione locale, con qualcosa che assomiglia a un file FIFO (*pipe* con nome), oppure una connessione attraverso la rete. Nel primo caso si parla di socket di dominio Unix (*Unix domain socket*); nel secondo, si tratta di solito di socket di dominio Internet.

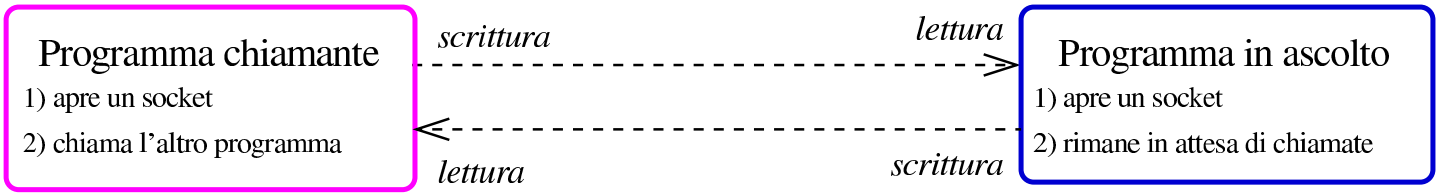
34.1 Principio di funzionamento

«

Quando si utilizzano i socket si distingue tra un programma che apre un socket, restando in ascolto di questo, e un programma che apre un socket per poi chiamare un altro programma che è in ascolto presso un certo indirizzo (definito in base al tipo di connessione). Si intende che per ottenere la connessione tra i due programmi, uno deve rimanere in ascolto, mentre l'altro deve attivare il suo sistema di ascolto prima di cercare di contattare il primo.

Una volta instaurata la comunicazione, i due programmi possono usare la connessione quasi come fosse l'accesso a un file, dove la lettura da una parte corrisponde alla scrittura dall'altra.

Figura 34.1. Comunicazione tra due programmi che usano i socket.



Ognuna delle due parti può chiudere la comunicazione quando vuole, chiudendo il descrittore che vi fa riferimento, come avviene con i file; dall'altro lato, si manifesta così un errore di lettura o di scrittura, a seconda del tipo di operazione che si stava svolgendo. Inoltre, va considerato il fatto che il programma che apre un socket per poi rimanere in attesa di chiamate, può mettere in coda le chiamate a cui non può rispondere perché già impegnato in una comunicazione. Naturalmente, si fissa un tetto massimo oltre il quale le chiamate vengono rifiutate.

Quando si definisce un socket, oltre a stabilirne il tipo di indirizzamento, deve essere specificato il modo in cui i dati vengono tra-

sferiti. Le tipologie più comuni sono il flusso continuo (*stream*) e la trasmissione a pacchetti senza controllo (datagrammi): nel primo caso il sistema verifica e garantisce che quanto viene trasmesso arrivi effettivamente (rispettando anche l'ordine di trasmissione), mentre nel secondo caso le trasmissioni sono a blocchi e non c'è un sistema di verifica.

34.2 Socket di dominio Unix

La comunicazione tra programmi, attraverso socket di dominio Unix, sfrutta un file speciale, di tipo socket. Utilizzando '**ls**', con l'opzione '**-l**', il file viene evidenziato da una lettera '**s**' all'inizio della stringa che descrive i permessi:

```
srwxrwxr-x 1 tizio tizio 0 set 20 14:44 orb-8085020182096096758
```

Questo file rappresenta generalmente il riferimento «visibile» usato dal programma che rimane in ascolto, mentre il programma chiamante si può limitare ad aprire un inode, senza che a questo sia abbinato un nome. Il file visibile diventa l'indirizzo a cui il programma chiamante fa riferimento per contattare la sua controparte.

Si può fare un controllo dello stato dei socket di dominio Unix con l'aiuto di Netstat, come nell'esempio seguente:

```
$ netstat --unix -p -a[Invio]

Active UNIX domain sockets (servers and established)
Proto RefCnt Flags    Type    State      I-Node PID/Program name  Path
unix  2      [ ACC ]  STREAM  LISTENING  3234   -                 /tmp/.X11-unix/X0
unix  7      [ ]      DGRAM              1573   -                 /dev/log
unix  2      [ ACC ]  STREAM  LISTENING  2421   -                 /dev/printer
unix  2      [ ACC ]  STREAM  LISTENING  2424   -                 /dev/gpmctl
unix  3      [ ]      STREAM  CONNECTED  3254   -                 /tmp/.X11-unix/X0
unix  3      [ ]      STREAM  CONNECTED  3253   750/xfig
unix  3      [ ]      STREAM  CONNECTED  3252   -                 /tmp/.X11-unix/X0
```

```
unix  3      [ ]      STREAM  CONNECTED  3251   746/twm
unix  3      [ ]      STREAM  CONNECTED  3250   -                  /tmp/.X11-unix/X0
unix  3      [ ]      STREAM  CONNECTED  3249   748/xclock
unix  3      [ ]      STREAM  CONNECTED  3244   -                  /tmp/.X11-unix/X0
unix  3      [ ]      STREAM  CONNECTED  3236   740/xinit
unix  3      [ ]      STREAM  CONNECTED  3160   -                  /dev/gpmctl
unix  3      [ ]      STREAM  CONNECTED  3159   656/mc
unix  2      [ ]      DGRAM              2909   -
unix  2      [ ]      DGRAM              2803   -
unix  2      [ ]      DGRAM              2702   -
unix  2      [ ]      DGRAM              2352   -
unix  2      [ ]      DGRAM              1667   -
```

Da un listato come questo si può intuire, per quanto possibile, il legame tra i processi. Per esempio, il programma '**mc**', in funzione con il numero PID 656, ha aperto un inode (3159) che risulta connesso; nella riga precedente, appare un altro inode (3160), anche questo connesso e associato al nome '`/dev/gpmctl`'. Conoscendo a cosa può riferirsi il file '`/dev/gpmctl`', si intende che si tratti del collegamento che c'è tra '**mc**' (Midnight Commander) e il demone che si occupa di controllare il movimento del mouse ('**gpm**').

Come si può osservare dalla colonna '**Type**' del listato, anche nei socket di dominio Unix si può distinguere tra connessioni continue, evidenziate dalla parola chiave '**STREAM**', e connessioni a datagramma, come suggerisce la parola chiave '**DGRAM**'.

34.3 Socket di dominio Internet

«

Le connessioni attraverso socket di dominio Internet si differenziano perché, invece di usare il riferimento a file speciali, utilizzano un indirizzo IP assieme a una porta (TCP o UDP). In tal modo si possono realizzare connessioni che vanno anche al di fuori dell'elaboratore locale.

Si può fare un controllo dello stato dei socket di dominio Internet con l'aiuto di Netstat, come nell'esempio seguente:

```
$ netstat --inet -p -a -n[Invio]

Active Internet connections (servers and established)
Proto Recv-Q Send-Q Local Address     Foreign Address State       PID/Program name
tcp        0      0 0.0.0.0:32768     0.0.0.0:*       LISTEN      357/rpc.statd
tcp        0      0 0.0.0.0:32769     0.0.0.0:*       LISTEN      558/rpc.mountd
tcp        0      0 0.0.0.0:515       0.0.0.0:*       LISTEN      519/lpd
tcp        0      0 0.0.0.0:847       0.0.0.0:*       LISTEN      245/rpc.ugidd
tcp        0      0 0.0.0.0:111       0.0.0.0:*       LISTEN      240/portmap
tcp        0      0 0.0.0.0:80        0.0.0.0:*       LISTEN      505/boa
tcp        0      0 127.0.0.1:953     0.0.0.0:*       LISTEN      349/named
tcp        0      0 192.168.1.1:32773 192.168.1.2:22  ESTABLISHED 938/ssh
udp        0      0 0.0.0.0:32768     0.0.0.0:*                   357/rpc.statd
udp        0      0 0.0.0.0:2049      0.0.0.0:*                   -
udp        0      0 0.0.0.0:32769     0.0.0.0:*                   349/named
udp        0      0 0.0.0.0:32771     0.0.0.0:*                   -
udp        0      0 0.0.0.0:32772     0.0.0.0:*                   558/rpc.mountd
udp        0      0 192.168.1.1:53    0.0.0.0:*                   349/named
udp        0      0 127.0.0.1:53      0.0.0.0:*                   349/named
udp        0      0 0.0.0.0:111       0.0.0.0:*                   240/portmap
```

Il listato di esempio è ridotto rispetto a quanto potrebbe essere riportato realmente. In questo caso si può osservare la presenza di una sola connessione attiva, la quale utilizza presso l'elaboratore remoto la porta 22 (protocollo SSH). Dal momento che si tratta di connessioni TCP/IP, invece si indicare una colonna con il tipo di flusso di dati, appare il protocollo, TCP o UDP, dove il primo costituisce in pratica una connessione continua e controllata, mentre il secondo consente solo l'invio di datagrammi.

34.4 Unix client-server program interface

UCSPI,[1] ovvero *Unix client-server program interface*, è un'interfaccia a riga di comando che consente la comunicazione, attraverso i socket, a programmi che sono sprovvisti di questa funzionalità. In

altri termini, consente di realizzare programmi che si avvalgono di questa interfaccia a riga di comando, senza bisogno di approfondire il problema della comunicazione con i socket.

Per la realizzazione di un'interfaccia UCSPI serve una coppia di programmi: uno per il servente UCSPI e l'altro per il cliente. Il primo dei due è il programma che si mette in ascolto, in attesa di chiamate, l'altro è il programma chiamante. Entrambi questi programmi hanno una sintassi uniforme per la riga di comando:

nome_eseguibile [***opzioni***] ***indirizzo*** ***applicazione*** [***argomenti_applicazione***]

L'indirizzo è ciò che serve a raggiungere il socket del servente; per esempio potrebbe essere il nome di un file socket, oppure un indirizzo IP completo di porta.

Pertanto, l'indirizzo indicato in fase di avvio del servente serve a creare il socket, mentre quello che riguarda il cliente, serve a raggiungere il servente.

Questo servente o cliente UCSPI, quando una connessione si instaura, avvia un altro programma, ovvero l'applicazione, come indicato alla fine della riga di comando (assieme alle opzioni e agli altri argomenti che possano essere necessari all'applicazione stessa); il programma ottiene poi le informazioni necessarie riferite alla connessione da alcune variabili di ambiente particolari. La comunicazione tra l'interfaccia UCSPI e l'applicazione avviene attraverso alcuni descrittori di file particolari.

Le opzioni comuni che deve avere un'interfaccia UCSPI sono quelle seguenti, a cui se ne possono aggiungere altre.

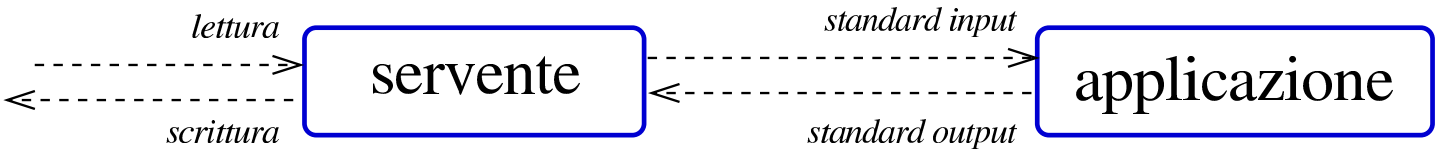
Opzione	Descrizione
`-v`	Mostra informazioni dettagliate.
`-Q`	Mostra informazioni solo sugli errori.
`-q`	Non emette alcuna informazione.

Le variabili di ambiente che vengono passate all'applicazione sono descritte nell'elenco seguente.

Variabile	Descrizione
`PROTO`	Contiene il nome del protocollo utilizzato.
protocollo*`LOCAL`	Si tratta di una serie di variabili che iniziano per il nome del protocollo, continuano con la stringa '**`LOCAL`**' e terminano in vario modo, descrivono le caratteristiche specifiche del protocollo dal lato locale.
protocollo*`REMOTE`	Si tratta di una serie di variabili che iniziano per il nome del protocollo, continuano con la stringa '**`REMOTE`**' e terminano in vario modo, descrivono le caratteristiche specifiche del protocollo dal lato remoto.

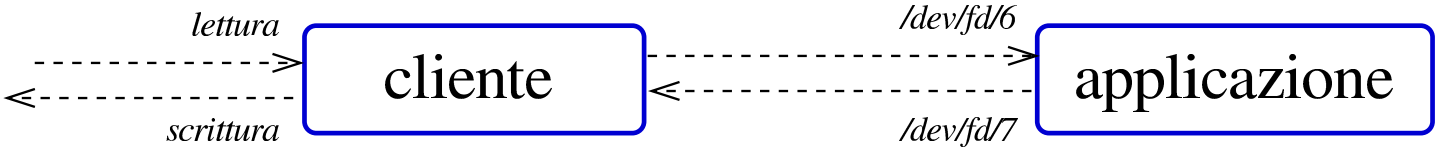
Come accennato, la comunicazione tra l'interfaccia e l'applicazione avviene attraverso dei descrittori standard. Nel caso del servente, l'applicazione riceve dati leggendo lo standard input, mentre trasmette emettendo dati attraverso lo standard output.

Figura 34.7. Comunicazione tra il servente e l'applicazione.



La comunicazione tra applicazione e il cliente UCSPI è più difficile, perché è necessario lasciare liberi i descrittori dei flussi standard comuni (standard input, standard output e standard error), a disposizione dell'applicazione, per i propri fini. Pertanto, si usano i descrittori sei e sette, rispettivamente per la lettura e la scrittura.

Figura 34.8. Comunicazione tra il cliente e l'applicazione.



A titolo di esempio, viene mostrato qualcosa di molto semplice: da una parte, un servente che, a ogni connessione, trasmette il contenuto del file '`/etc/passwd`'; dall'altra, un cliente che scorre questo risultato sullo schermo. Per cominciare, il servente viene definito in modo molto semplice:

```
$ servente indirizzo cat /etc/passwd[Invio]
```

Infatti, '**`cat`**' emette attraverso il suo standard output il contenuto del file '`/etc/passwd`', che viene prelevato dal programma che costituisce l'interfaccia UCSPI, mentre lo standard input che conterrebbe il flusso di dati in ingresso dalla connessione, viene ignorato da '**`cat`**'.

La predisposizione dal lato cliente diventa invece un po' più difficile: serve almeno uno script:

```
#!/bin/sh
cat <&6- | less
```

In questo modo, si usa ancora '**cat**', che attraverso lo standard input riceve invece quanto proveniente dal descrittore sei; quindi, quanto emesso da '**cat**' viene controllato da '**less**' (il descrittore sette non viene usato e questo significa che nulla viene inviato all'applicazione remota). Supponendo che lo script si chiami '**visualizza**' e sia collocato nella directory corrente:

```
$ cliente indirizzo ./visualizza [Invio]
```

Ciò dovrebbe essere sufficiente per poter visualizzare a ogni collegamento il contenuto del file '/etc/passwd' dell'elaboratore in cui si trova il servente.

Nell'esempio è stata mostrata una ridirezione particolare: '**<&6-**'. Il trattino finale serve a chiudere lo standard input e non è strettamente indispensabile. Nel caso la shell non consenta di usare questa combinazione, va bene anche soltanto '**<&6**'.

34.5 UCSPI-unix

UCSPI-unix[2] è un pacchetto che realizza l'interfaccia UCSPI per le comunicazioni attraverso socket di dominio Unix. Si compone principalmente di due programmi, la cui sintassi specifica si descrive nel modo seguente:

```
unixserver [opzioni] file_socket applicazione [argomenti_applicazione]
```

```
unixclient [opzioni] file_socket applicazione [argomenti_applicazione]
```

Come si può intendere, '**unixserver**' apre un socket di dominio Unix (un file) e attende una connessione, mentre '**unixclient**' contatta la controparte attraverso l'indicazione dello stesso file.

La comunicazione con l'applicazione rispettiva avviene secondo le modalità delle interfacce UCSPI e le opzioni sono quelle comuni, con l'aggiunta di altre specifiche per il tipo di socket (si consulti eventualmente *unixserver(1)*).

Volendo adattare l'esempio già mostrato in forma generalizzata a questo tipo di interfaccia, i comandi potrebbero essere quelli seguenti. Dal lato del servente:

```
$ unixserver /tmp/socket-prova cat /etc/passwd[Invio]
```

Dal lato del cliente serve uno script e il comando che avvia lo script:

```
#!/bin/sh
# ./visualizza
cat <&6- | less
```

```
$ unixclient /tmp/socket-prova ./visualizza[Invio]
```

Naturalmente, la scelta del file '`/tmp/socket-prova`' è arbitraria e dipende da come si avvia il servente.

Utilizzando uno script differente, è possibile controllare lo stato delle variabili di ambiente:

```
#!/bin/sh
set
```

Avviando '**unixclient**' con questo script, si possono notare, tra

le altre, le variabili seguenti, che riguardano precisamente UCSPI-unix:

```
PROTO=UNIX
UNIXLOCALGID=1001
UNIXLOCALPATH=/tmp/socket-prova
UNIXLOCALPID=2145
UNIXLOCALUID=1001
UNIXREMOTEEGID=1001
UNIXREMOTEEUID=1001
UNIXREMOTEPID=2112
```

Le informazioni che derivano da queste variabili dovrebbero essere comprensibili già dal nome di queste, comunque vengono descritte brevemente nell'elenco seguente.

Variabile	Descrizione
`PROTO`	Contiene la stringa '**`UNIX`**' a indicare che si tratta di socket di dominio Unix.

`UNIXLOCALUID`
 	
`UNIXLOCALGID`
	Si tratta rispettivamente del numero UID e GID del processo avviato localmente.

`UNIXLOCALPID`	Si tratta nel numero abbinato al processo locale.
`UNIXLOCALPATH`	Si tratta nel file socket a cui si fa riferimento per la connessione (lo stesso nome da entrambi i lati).

`UNIXREMOTEEUID`
 	
`UNIXREMOTEEGID`
	Si tratta rispettivamente del numero UID e GID del processo avviato dall'altra parte.

`UNIXREMOTEPID`	Si tratta nel numero abbinato al processo remoto.

34.6 UCSPI-tcp

«

UCSPI-tcp[3] è un pacchetto che realizza l'interfaccia UCSPI per le comunicazioni attraverso socket di dominio Internet, precisamente il protocollo TCP. Si compone principalmente di due programmi, la cui sintassi specifica si descrive nel modo seguente:

```
tcpserver [opzioni] nodo porta applicazione [argomenti_applicazione]
```

```
tcpclient [opzioni] nodo porta applicazione [argomenti_applicazione]
```

Anche in questo caso, '**tcpserver**' è il programma che si mette in ascolto (aprendo un socket di dominio Internet, con il protocollo TCP), mentre '**tcpclient**' contatta la controparte. Dal momento che si utilizza il protocollo TCP, il riferimento usato per comunicare è formato dall'indirizzo IP e dalla porta TCP del servente.

La comunicazione con l'applicazione rispettiva avviene secondo le modalità delle interfacce UCSPI e le opzioni sono quelle comuni, con l'aggiunta di altre specifiche per il tipo di socket (si consulti eventualmente *tcpserver(1)* e *tcpclient(1)*). In particolare, nel servente è possibile stabilire il numero massimo di connessioni in coda; inoltre, entrambe le parti possono fissare un tempo massimo di scadenza per i tentativi di connessione.

Volendo adattare l'esempio già mostrato in forma generalizzata a questo tipo di interfaccia, i comandi potrebbero essere quelli seguenti. Dal lato del servente:

```
$ tcpserver dinkel.brot.dg 1234 cat /etc/passwd[Invio]
```

Dal lato del cliente serve uno script e il comando che avvia lo script:

```
#!/bin/sh
# ./visualizza
cat <&6- | less
```

```
$ tcpclient dinkel.brot.dg 1234 ./visualizza [Invio]
```

Naturalmente, la scelta della porta 1234 è arbitraria, salvo il fatto che deve essere una porta libera e non privilegiata, dal momento che, nell'esempio, il servente viene avviato da un utente comune.

La comunicazione può risultare un po' in ritardo rispetto alle aspettative, nel caso venga fatta prima una verifica dell'identità delle parti attraverso il protocollo IDENT.

Utilizzando uno script differente, è possibile controllare lo stato delle variabili di ambiente:

```
#!/bin/sh
set
```

Avviando '**`tcpclient`**' con questo script, si possono notare, tra le altre, le variabili seguenti, che riguardano precisamente UCSPI-tcp:

```
PROTO=TCP
TCPLOCALHOST=roggen.brot.dg
TCPLOCALIP=192.168.1.2
TCPLOCALPORT=32993
TCPREMOTEHOST=dinkel.brot.dg
TCPREMOTEINFO=
TCPREMOTEIP=192.168.1.1
TCPREMOTEPORT=1234
```

Le informazioni che derivano da queste variabili dovrebbero essere comprensibili già dal nome di queste, comunque vengono descritte

brevemente nell'elenco seguente.

Variabile	Descrizione
`PROTO`	Contiene la stringa '**`TCP`**' a indicare che si tratta di socket di dominio Internet con protocollo TCP.

`TCPLOCALHOST`
`TCPLOCALIP`
`TCPLOCALPORT`
	Si tratta rispettivamente del nome, dell'indirizzo IP e della porta nell'ambito locale.

`TCPREMOTELHOST`
`TCPREMOTEIP`
`TCPREMOTEPORT`
	Si tratta rispettivamente del nome, dell'indirizzo IP e della porta nell'elaboratore remoto.

`TCPREMOTEINFO`	Informazioni particolari sulla controparte remota, ammesso che siano disponibili.

È da tenere in considerazione il fatto che '**`tcpserver`**' può essere controllato per evitare gli accessi indesiderati. Per questo si deve usare l'opzione '**`-x`**', abbinando un file costruito con '**`tcprules`**', il quale fa parte dello stesso pacchetto UCSPI-tcp (si veda *tcprules(1)*).

34.7 Riferimenti

«

- Jim Frost, *BSD sockets: a quick and dirty primer*, *http://www.google.com/search?q=Jim+Frost+BSD+sockets+a+quick+and+dirty+primer*
- D. J. Bernstein, *UNIX Client-Server Program Interface, UCSPI-1996*, 1996, *http://cr.yp.to/proto/ucspi.txt*

[1] In lingua inglese, UCSPI si pronuncia praticamente come se venisse letto nella lingua italiana: «u-c-s-p-i».

[2] **UCSPI-unix** GNU GPL

[3] **UCSPI-tcp** software libero per il quale non è consentita la diffusione in forma binaria, salvo approvazione esplicita da parte dell'autore