Interfaccia Socket

Le applicazioni di rete sono insiemi di processi su host diversi che si scambiano messaggi attraverso la rete. Esistono degli schemi base che regolano lo scambio di messaggi:

1. Client/server;
2. Publisher/Subscriber (Pub/Sub).

Client/Server:

E' la situazione più frequente. Il client fa sempre il primo passo con una richiesta, mentre il server fa il secondo passo e manda la risposta, rimettendosi poi in attesa di altre richieste. La richiesta del client può essere una richiesta di un dato oppure la trasmissione di un dato (cioè la richiesta di prendere in consegna un certo dato).

Quello che determina il ruolo di client e server è l'ordine dei messaggi e non il contenuto. In un'applicazione client/server, il client è quindi colui che fa il primo passo indipendentemente dal fatto che chieda o trasmetta un dato.

Dobbiamo anche fare attenzione ai nomi: client e server sono processi e non host; l'insieme di almeno un client e un server costituisce l'applicazione di rete. In particolare si può dire che un certo processo gioca/interpreta il ruolo di client o server all'interno dell'applicazione.

Durante l'esercitazione non useremo direttamente l'interfaccia socket, bensì una libreria sviluppata da un ex studente durante la sua tesi che incapsula l'utilizzo diretto della socket per non andare eccessivamente a basso livello (function wrapped).

Il programma, prima di usare la rete, deve creare un oggetto di tipo socket. Il socket è l'insieme di chiamate a funzione (punto di comunicazione) identificato da 3 parametri, ossia indirizzo IP locale, porta locale, modalità di trasmissione UDP oppure TCP. Funziona come end-point, ossia in una connessione c'è una socket sia in un host che nell'altro. Il programma server deve decidere esplicitamente il numero di porta locale affinché i client possano saperlo (la porta serve a indicare quale processo, l'IP quale host); i numeri da 0 a 1023 sono riservati a protocolli applicativi noti. Il programma che tuttavia crea un oggetto socket su queste porte deve avere i privilegi di root. Il programma client può deciderlo esplicitamente oppure lasciarlo decidere al proprio sistema operativo.

UDP:

Ogni pacchetto scambiato tra gli host è logicamente indipendente dai precedenti e successivi. Le perdite di pacchetti non vengono compensate in automatico né dalla rete né dal sistema operativo. Per questo motivo l'UDP è più leggero del TCP come uso di risorse di calcolo e di rete, ma il tipo di applicazione deve essere compatibile con questo tipo di comportamento (esempio, richiesta DNS). Se il server non ha nome occorre chiedere ad un utente sul server di scoprire e comunicarci il suo indirizzo IP, mentre lato server, siccome una risposta parte sempre dopo la relativa richiesta, l'indirizzo IP del client, così come la porta (notare che nel file infatti non viene specificato 20.000 nel server) è contenuto nelle informazioni di mittente della richiesta.

TCP:

E' byte oriented ossia non c'è correlazione a livello di programmazione tra il taglio di lettura e scrittura dei byte e l'effettivo pacchetto trasmesso. In altre parole, il sender e la sua specifica di quanti byte mandare il pacchetto, è scorrelato dal come il receiver decide di bufferizzarli e riceverli. E' utilizzata quando tra i pacchetti trasmessi c'è una relazione: sono parte di un "messaggio più grande" (ad es. un'immagine o un PDF). L'oggetto socket si preoccupa di numerare i pacchetti appartenenti alla stessa connessione per rilevare eventuali pacchetti persi e poterli ritrasmettere. Come vantaggio, l'utente scrive/legge su un archivio remoto con la stessa naturalezza di quando scrive/legge su un archivio locale come se la rete in mezzo non ci fosse. Come svantaggio invece, gli host trasmettitore e ricevitore devono "lavorare" di più dentro il sistema operativo, e i pacchetti persi e ritrasmessi arrivano in ritardo e questo può essere compatibile oppure no con il tipo di applicazione.