ADPnetwork: Low-level autoconfig

di Andrea de Prisco

Ohibò. Ciuchini e Suatoni parevano proprio averci preso la mano. E non mi lasciano più parlare....

Bando agli scherzi (e sottolineo scherzi!) questo mese torneremo a parlare di ADPnetwork (vera e propria) ovvero del software di rete che permette la comunicazione tra processi in esecuzione su macchine diverse, fisicamente collegate tra loro: attraverso una architettura di interconnessione circolare, formata da tanti collegamenti dedicati quante sono le macchine in rete. Come più volte ribadito, non si tratta di un semplice bus seriale condiviso (come può essere ad esempio quello dell'architettura Ethernet) ma di una struttura di interconnessione distribuita deterministica in cui ogni macchina può iniziare il suo accesso alla rete indipendentemente dall'attività delle altre macchine sulla rete stessa. Può anche succedere (e questo Ethernet... semplicemente se lo scorda) che nello stesso istante più macchine eseguano scambio di file o, più in generale, scambio di messaggi. E quando dico "stesso istante" mi riferisco alla sua accezione "fisica" del termine e non semplicemente "logica". Eppure, come per Ethernet, si tratta di un solo filo che "visita" tutte le macchine in rete. Dov'è il trucco? Nessun trucco: solo illusionismo...

ADPnetwork vs Ethernet

Avevo detto che non scherzavo più e invece ci sono cascato di nuovo. Ma quale illusionismo??? Si tratta solo ed esclusivamente di pura realtà (dei fatti). Spieghiamoci meglio. La differenza sostanziale che intercorre tra ADPnetwork ed Ethernet sta nel fatto che la prima è una architettura di rete deterministica, la seconda no. Per quanto vi possa sembrare strano (posto che chi mi legge non sia esperto di reti a basso livello) un nodo Ethernet quando accede alla rete lo fa per tentativi. Nel senso che prima di accedervi controlla che nessun altro stia utilizzando la struttura fisica (infatti una sola macchina per volta può accedere al bus seriale in scrittura), ma nell'attimo in cui decide di utilizzarla non può essere certa di ottenerne l'uso esclusivo. Ci prova... se gli va bene può iniziare la sua trasmissione, altrimenti abortisce l'operazione e ritenuta dopo un intervallo di tempo calcolato da un algoritmo pseudocausale. Infatti, nel medesimo istante due o più macchine possono avere la voglia di trasmettere su Ethernet e quindi simultaneamente testare la disponibilità del supporto fisico ottenendo tutte esito positivo (supporto disponibile) ma prima di iniziare la vera e propria trasmissione è necessario un secondo test atto ad identificare questo genere di collisioni. Il meccanismo in sé è abbastanza semplice: tra l'altro ne abbiamo già parlato un bel po' di mesi fa (forse anni!) in Appunti di Informatica sotto il "capitolo" Arbitraggio distribuito non deterministico. Il modo più semplice per ottenere l'arbitraggio è di disporre di una linea aggiuntiva "di servizio" mantenuta bassa quando il supporto di trasmissione non è utilizzato e alta quando lo è, da parte della macchina utilizzatrice. Ogni macchina che intende utilizzare la connessione testa la linea di servizio per vedere lo stato. Se è "basso", la pone "alto" e inizia a trasmettere, se è "alto" attende la liberazione del supporto. Può succedere, come detto, che simultaneamente più macchine testino lo stato della linea di servizio e, sempre contemporaneamente, queste macchine, trovato lo stato "basso", si approprino del bus alzando tale linea di servizio e cominciando a trasmettere. È chiaro che si ottiene una bella collisione che deve essere avvertita dalle stesse macchine trasmettitrici che, man mano che mandano il loro messaggio, devono testarne la "leggibilità" rileggendolo contemporaneamente. Ora, se rileggendo via via il messaggio trasmesso non vi sono discrepanze tra quanto inviato e quanto letto, vuol dire che la trasmissione procede bene ed effettivamente il bus è stato acquisito dalla macchina. Se invece una macchina scrive "pippo" e legge "pluto" vuol dire che qualcun altro (credendo come quest'ultima di avere uso esclusivo della connessione) sta trasmettendo simultaneamente: in questo caso occorre abortire la trasmissione da parte di tutte le macchine coinvolte nella collisione ritentando la comunicazione dopo un intervallo di tempo calcolato grazie ad una funzione pseudo casuale, possibilmente calcolata con algoritmi diversi sulle varie macchine (onde evitare, come i più attenti avranno intuito, continui rimbalzi).

Nel riquadro a fronte è illustrato un metodo per ottenere la medesima funzionalità senza linee aggiuntive ma tramite il solo cavo coassiale (come avviene per Ethernet) che quindi funge sia da bus seriale che da linea di controllo.

ADPnetwork, invece, è una architettura distribuita, deterministica: ogni macchina che intende accedere alla rete, può farlo in qualsiasi momento, senza assolutamente curarsi del fatto se nel medesimo istante altre macchine accedono alla rete. E in caso di «grosso carico» (ovvero di molte macchine che accedono contemporaneamente alla rete) non c'è assoluto rischio che qualche macchina non riesca ad effettuare la sua trasmissione, ma solo che il suo messaggio sia recapitato con un ritardo maggiore rispetto al caso di «carico leggero». Come detto più volte, infatti, ogni macchina è «proprietaria» del collegamento verso la macchina successiva e quindi può disporne (a meno di riempimenti di buffer... ma questi, si sa, sono problemi secondari!) della rete in ogni momento, semplicemente inoltrando al «next node» il suo messaggio. Contemporaneamente anche altre macchine possono fare la stessa cosa e,

ripeto, il maggiore o minore carico del sistema è evidenziato semplicemente da un «naturale» degrado delle prestazioni, ma non c'è assoluto rischio di attesa infinita (starvation) da parte di qualche nodo. Di contro, ogni macchina sarà interessata non solo ai pacchetti da essa spediti o ricevuti, ma anche da tutti i pacchetti in transito da una macchina «precedente» ad una macchina «successiva». Considerato poi che ADPnetwork per garantire una tolleranza ai guasti dovuti a caduta del supporto fisico necessita di pacchetti di Ack che confermano l'avvenuta ricezione del pacchetto trasmesso (e il senso di circolazione è sempre il medesimo) succede che qualsiasi macchina trasferisca un messaggio su qualsiasi altro nodo, vengono coinvolte tutte le macchine collegate in rete: o per recapitare il pacchetto al destinatario, o per inoltrare il pacchetto di Ack (la conferma) al mittente. Quindi potenza sì, ma anche ampio dispiego di forze.

L'autoconfigurazione a basso livello

Perché a basso livello? Semplice: per distinguerla da quella ad alto livello. Ma non credo affatto di essermi spiegato a sufficienza. Infatti, per configurazione ad alto livello intendo la capacità delle singole macchine (già in rete, ovvero configurate a basso livello) di conoscere la conformazione dei singoli nodi: nomi simbolici delle macchine e device accessibili via rete su queste. Ma l'argomento non riguarda me, ma i «soliti» Ciuchini e Suatoni che si stanno occupando del Net-Handler e Net-Server della rete.

Ma torniamo ad ADPnetwork «allo stato puro», ovvero quale mezzo «logico» di comunicazione interprocess-internodes.

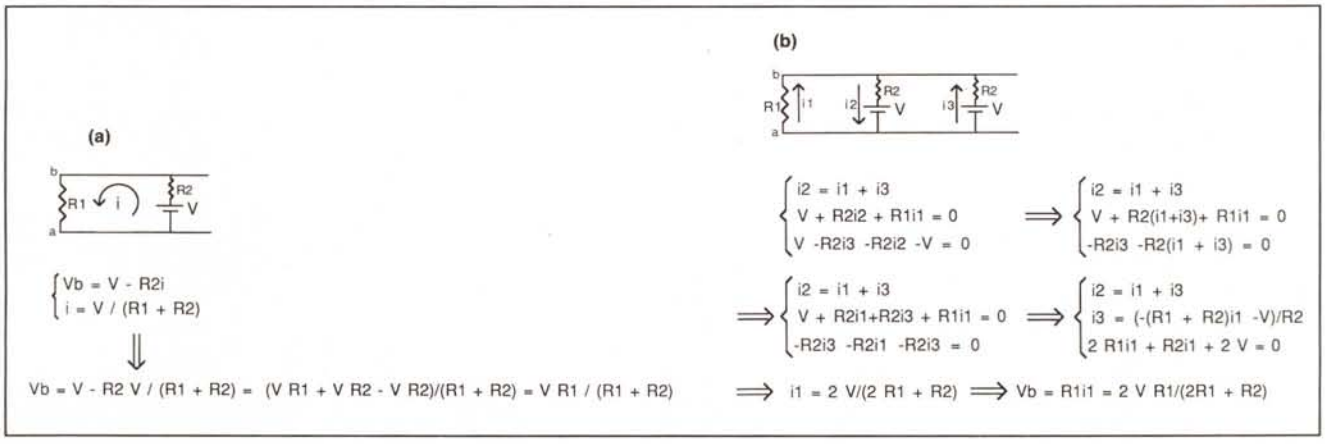
Sono passati diversi mesi dall'ultimo appuntamento riguardante il software di rete propriamente detto, e molte modifiche sono state apportate per aumenta-

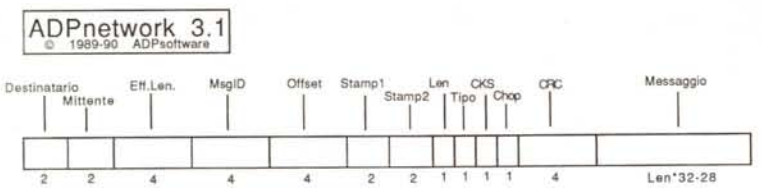
Un cavo coassiale arbitro

Nell'articolo abbiamo parlato dell'arbitraggio distribuito non deterministico (che nulla ha a che vedere con ADPnetwork ma con Ethernet) di un bus (seriale o parallelo) effettuato grazie ad una linea aggiuntiva di controllo. In questo riquadro mostreremo come sia possibile effettuare la medesima funzionalità senza ricorrere a strutture ausiliare ma al solo cavo seriale coassiale sul quale normalmente viaggiano i dati un bit dopo l'altro. Come detto è necessario implementare un meccanismo che permetta alle macchine in rete di capire «al volo» se il supporto fisico è occupato da altre trasmissioni o è disponibile e, in questo caso, prenderne possesso esclusivo.

Il tutto si basa su una opportuna terminazione del cavo seriale effettuata da una coppia di resistenze, ad esempio da 25 ohm l'una (per un totale quindi di 50) da sistemare alle due estremità del cavo coassiale, tra calza e nucleo. Ogni macchina attaccata al supporto è collegata tanto alla calza quando al nucleo in «parallelo» a tutte le macchine in rete. Se una macchina intende utilizzare la rete applica una tensione continua V tra nucleo e calza del cavo e immediatamente dopo verifica la differenza di potenziale tra questi due punti. Nella figura (a) contenuta in questo riquadro, R1 è la resistenza totale delle due terminazioni e R2 la resistenza interna del generatore di F.E.M. Se la macchina che tenta l'accesso alla rete è, in quell'istante, unica, tra calza e nucleo sarà presente una differenza di potenziale pari a VR1/(R1+R2). Se nello stesso istante anche una seconda macchina applica la sua DDP ed esegue il test, le due macchine rileveranno una DDP tra calza e maglia pari a 2VR1/(2R1+R2) che è, ovviamente, maggiore del caso precedente e quindi rileveranno automaticamente la collisione. Naturalmente durante l'utilizzo del supporto la DDP rimane applicata in modo da segnalare agli altri utilizzatori che la rete è occupata. Da notare, come più volte ribadito nell'articolo, che una macchina intenzionata ad utilizzare la rete non disponibile (perché usata in quel momento da un altro nodo) non può fare altro che aspettare un tempo «t» e ritentare sperando di essere più fortunata. Nel caso limite di molte macchine che tentano accessi sulla rete, può addirittura verificarsi che nessuna (nel senso di «nemmeno una») macchina riesca ad utilizzare il supporto sebbene non ancora impegnato.

Fortunatamente, nel concreto, l'utilizzo delle reti tipo Ethernet è ben più basso del livello di saturazione, grazie soprattutto all'alta velocità di trasmissione che permette a tutti i nodi di impegnare sempre per tempi brevi il supporto fisico.





StandardFrame di Rete

```
struct Frame
       {
       UWORD Dest;
       UWORD Mitt;
       ULONG EffLen;
       ULONG MsgID;
       ULONG Offset;
       UWORD Stamp1;
       UWORD Stamp2;
       UBYTE Len;
       TEXT  Type;
       UBYTE Chop;
       UBYTE CkSum;
       ULONG CRC;
       TEXT  Body[2];
       };
```

Figura 1

re le prestazioni generali del sistema. In figura 1 è mostrato il nuovo formato dei pacchetti di rete (in seguito denominati frame per non confonderli coi dos packet di AmigaDos). La principale differenza rispetto alla (pre)release 3.0 è che ogni macchina è identificata a basso livello ad un NetID di 16 bit e a livello del Net-Handler dal consueto nome simbolico di massimo 9 caratteri. E a differenza, sempre, della versione precedente, ogni macchina sceglie automaticamente un NetID (attraverso un algoritmo pseudo casuale) e nessuna operazione è quindi richiesta da parte dell'operatore per inizializzare il singolo nodo. È chiaro, inoltre, che ogni nodo non può semplicemente scegliere «a caso» il suo NetID, ma, in un certo senso, deve sincerarsi che nessun'altra macchina si chiama (o intende chiamarsi) nello stesso modo.

Se state pensando che ciò, in un sistema completamente distribuito come ADPnetwork, è assolutamente impossibile avete proprio ragione. E non sto dando (ancora) i numeri. Si sa, infatti, che in mancanza di un «distributore centralizzato» di NetID le singole macchine non possono mettersi tra loro d'accordo in modo da avere ognuna un NetID differente in quanto nessuna potrebbe mai essere certa che l'ID scelto anche temporaneamente da ognuna non sia stato scelto contemporaneamente anche da altre. E non c'è niente da fare a riguardo, se non scendere ad opportuni compromessi.

Poniamoci nel caso minimale di due macchine collegate in rete che stanno autoconfigurandosi. Nessuna delle due macchine ha un ID di rete e devono ottenerne uno al termine della fase di autoconfigurazione. Diciamo che la prima delle due macchine decide di chiamarsi «3123» e la seconda «15678». Prima però di dare per valido il proprio NetID è necessario mandare un messaggio circolare alle altre macchine contenente la propria proposta di identificatore per vedere se nessuna delle macchine in rete ha da obiettare qualcosa. Allora, la macchina 1 manderà un messaggio del tipo: «avete qualcosa in contrario al fatto che mi chiami 3123?» e messaggio analogo (ma con ID = 15678) lo manderà la macchina 2. Pochi istanti dopo i due messaggi giungono alle macchine «successive» (essendo sole due macchine, sono sempre le stesse) e nessuna delle due macchine obietterà, dato che l'ID proposto è diverso dal loro. All'istante successivo i due messaggi giungono nuovamente ai rispettivi mittenti avendo completato il giro ed ognuna (riconoscendolo proprio grazie all'ID proposto) «capisce» che nessuno ha obiettato e che quindi quell'ID va bene. Una roba simile, naturalmente, non funziona. È chiaro, del resto. Infatti, nel caso che tutt'e due le macchine avessero scelto (pseudo-casualmente) lo stesso ID, ad esempio 15678 e... 15678, non appena la proposta «dell'altro nodo» arriva, il nodo in questione crede che sia la sua stessa proposta che ha fatto il giro dell'intera rete e lo prende per buono. Da questo momento in poi due macchine si chiamerebbero nella stessa maniera con evidenti malfunzionamenti dell'intero sistema dato che non sarebbe più distinguibile la prima macchina dalla seconda.

È necessario, allora, istituire una forma di riconoscimento «propria proposta» ben più affidabile del solo NetID che potrebbe essere uguale a quello di un altro.

Ovviamente non esiste un metodo assolutamente certo per riconoscere il proprio messaggio proposta come tale, ma possiamo spingerci al punto da rendere particolarmente improbabile una collisione di questo genere.

In figura 2 è mostrato il formato del pacchetto di autoconfig. Il campo destinatario è, ovviamente, non significativo, dal momento che tale messaggio è diretto a tutte le macchine in rete. Segue il campo mittente: qui la macchina che «vuole configurarsi» mette la sua proposta di NetID. Nei tre campi successivi la macchina mittente mette alcune informazioni aggiuntive, come detto, per abbassare notevolmente la probabilità che un'altra macchina in rete prenda per suo il pacchetto in questione. Le informazioni riguardano l'ammontare della memoria libera in quell'istante, l'identificatore del processo che sta effettuando l'operazione e l'indirizzo della porta sulla quale il processo di autoconfigurazione aspetta l'esito. Oltre a questo il NetID è funzione sia di un algoritmo pseudocasuale che dell'orologio interno delle singole macchine. In questo modo, per avere ancora una collisione, è necessario che in due macchine si verifichino contemporaneamente i seguenti eventi: le due macchine hanno esattamente la stessa architettura, stessa espansione di memoria e schede installate; hanno effettuato i loro rispettivi boot esattamente nello stesso istante e i loro orologi interni «tamponati» non sono sfasati temporalmente nemmeno di un solo tick di sistema; le due macchine eseguono al boot esattamente la stessa procedura, lanciando i medesimi programmi contemporaneamente. Come dire che per verificarsi una collisione «di vedute» ce ne vuole davvero parecchia (di sfortuna). Il campo Stamp (implementato anche nel frame standard) permette a qualsiasi macchina in rete di uccidere eventuali «zombi» in circolazione. Se si verifica ad esempio che una macchina

immette un frame sulla rete e poi muore (va in crash, ad esempio), tale frame (o al più il relativo Ack) potrebbero finire per circolare «all'infinito» sulla rete dato che ogni macchina rimasta continuerebbe a reinoltrarlo non riconoscendolo come proprio. Per venire meno a questo inconveniente, quanto un frame qualsiasi passa per una qualsiasi macchina, prima di reinoltrarlo (e posto che nessun altro nodo abbia già provveduto) il nodo «reinoltrante» lo marchia col suo NetID nel campo Stamp in modo tale che se ripassa di nuovo da quel nodo, vuol dire che la macchina destinataria non esiste più e quindi lo si può tranquillamente uccidere (lett.: «non reinoltrare»).

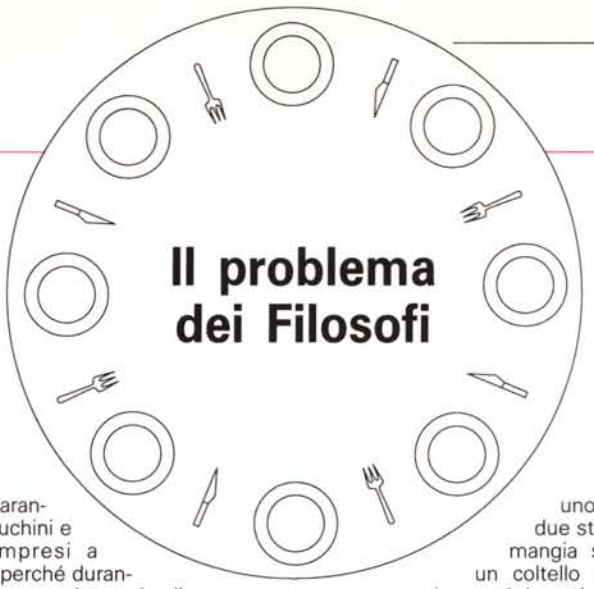
È chiaro che se oltre a morire la macchina destinataria (o mittente) muore anche la macchina che ha settato lo Stamp il frame resterà comunque in vita e quindi in circolazione. È per questo motivo (fig. 1) che nella struttura del frame standard ho previsto addirittura due Stamp distinti da far settare a due macchine diverse: il «casino» è ancora possibile, ma le probabilità che ciò si verifichi scendono ancora....

Concludendo

Come avrete notato, per i soliti motivi di spazio, non è stato possibile effettuare una trattazione più particolareggiata del meccanismo di autoconfigurazione a basso livello. Del resto, però, ad essere onesti al 100%, non era nemmeno nostro intento un livello di dettaglio molto maggiore, dato che ADPnetwork non sarà un prodotto di pubblico dominio (non cercatelo su MC-Link: discorso analogo per l'ADPmttb) e non possiamo certo svelarvi proprio-proprio tutto. Questi articoli, comunque, hanno valore didattico e chi volesse cimentarsi in imprese simili (magari per altre macchine, così non ci fa concorrenza...) troverà sicuramente (almeno speriamo) spunti validi per risolvere facilmente un po' di problemi nemmeno tanto banali. Se poi qualcuno intendesse interfacciarsi con la nostra rete con macchine non Amiga, inutile dirlo, avrebbe tutto il supporto necessario all'operazione. Se qualcuno è interessato: buon lavoro!

MC

Il problema dei Filosofi



Molti si saranno chiesti, Ciuchini e Suatoni compresi a quanto pare, perché durante tutta la fase sperimentale di ADPnetwork identificavo ogni macchina in rete con il nome di un filosofo.

Tutto cominciò nel 1984, a lezione di Sistemi di Elaborazione dell'Informazione II del corso di laurea in Scienze dell'Informazione presso l'università di Pisa. In questo corso si insegnano le tecniche di programmazione multitak, la gestione di risorse condivise, monitor, semafori, time sharing, ottimizzazione dei dispositivi di I/O, teniche anti deadlock e anti starvation, fairness e tante altre cosette davvero interessanti. E se in quel periodo sognavo un computer vero (non certo un Apple II o MS-DOS) e mi consolavo col VIC-20, poi divenuto 64, a lezione mi divertì molto il cosidetto problema dei filosofi che racchiude in sé interessanti problematiche proprie di quel corso.

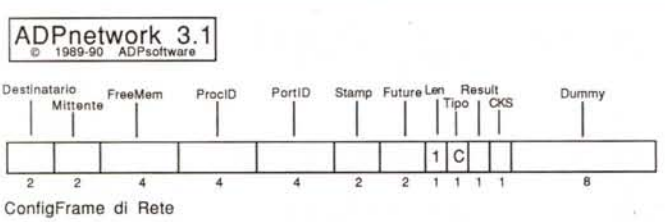
Immaginate una bella tavola imbandita. Un certo numero di piatti (e quindi posti) e un pari numero di posate: metà forchette e metà coltelli. La disposizione è quella mostrata nella figura contenuta in questo stesso riquadro.

I commensali sono tutti filosofi con l'ovvia caratteristica di stare seduti a tavola in uno solo di questi due stati: un filosofo mangia se dispone di un coltello e di una forchetta altrimenti pensa (filosofeggia). E, ovviamente, se pensa solo e non mangia mai muore di fame. In pratica i filosofi sono processi utilizzatori e le posate risorse condivise.

Ogni posata è una risorsa condivisa da due utilizzatori. Il problema sta nello scrivere gli «n» processi «filosofi», gli «n/2» processi forchetta e gli «n/2» processi coltelli e lanciare il tutto senza provocare (neanche a lungo andare) deadlock o eccessivi «pensamenti» da parte di più del 50% dei filosofi.

Simpatico poi verificare la percentuale media di utilizzo delle risorse e del filosofeggiamento dei filosofi al variare delle possibili tecniche di condivisione delle risorse adoperate.

È chiaro, infine, che cercare di accaparrare al sopraggiungere della fame una delle due posate e aspettare la liberazione dell'altra è il modo più banale... per ottenere il blocco totale del sistema dopo pochissimi cicli alimentazione-pensiero. Non aggiungo altro: il mese prossimo una possibile soluzione utilizzando, guardacaso, l'ADPmttb. Della serie: ogni scarrafone...




```
struct ConfigFrame
       {
       UWORD Dest;
       UWORD Mitt;
       ULONG FreeMem;
       ULONG ProcID;
       ULONG PortID;
       UWORD Stamp;
       UWORD Future;
       UBYTE Len;
       TEXT  Type;
       UBYTE Result;
       UBYTE CkSum;
       };
```

Figura 2