# ADPnetwork: una rete per Amiga

*di Andrea de Prisco*

seconda puntata

*Dopo la propedeutica introduzione su ADPnetwork fatta lo scorso mese, in questa seconda puntata analizzeremo più dettagliatamente il funzionamento dei processi più importanti e mostreremo lo schema di collegamento delle macchine, attualmente effettuato per mezzo della porta seriale*

Bisogna dire una cosa: se Amiga non fosse stato un computer multitasking, realizzare una rete come ADPnetwork sarebbe stato davvero difficile. Proprio ieri mattina, ad esempio, ho iniziato a studiare la struttura di un potenziale software di gestione di un microcontroller capace di gestire in parallelismo simulato un certo numero di eventi assolutamente asincroni (quindi indipendenti) tra loro. Nonostante alla fine abbia avuto io la meglio, la mia mente s'era così abituata a ragionare in termini di processi paralleli (e indipendenti anch'essi) che le maledizioni inoltrate a quel povero microcontroller ormai nessuno riusciva più a contarle. Fortunatamente non dovremo aspettare ancora molto (almeno me lo auguro) prima di vedere i primi processori dotati di linguaggio macchina multitasking, anche se realizzato a semplici colpi di time sharing. Provate ad immaginare, ad esempio, cosa significa per un singolo programma controllare eventi differenti senza mai effettuare attese attive a scapito di altre operazioni da portare a termine: ad esempio un bel programma che riceve flussi di dati da due canali, elaborando gli eventuali dati provenienti dal canale 1 e, simultaneamente, redirigere l'input del canale 2 su un terzo canale d'uscita. Può succedere ad esempio che mentre dal canale 1 arrivano dati da elaborare sul canale 2 arrivino dati da ridirigere sul canale 3 e tutte le operazioni debbano essere compiute molto velocemente onde evitare perdite di dati da riempimenti di buffer.

Certo, in informatica, tutto quello che si può ben dire si può ben fare, ma volete mettere una soluzione al problema realizzata con due o più processi cooperanti quanto è più elegante e raffinata di una soluzione monotask capace di andare avanti solo ed esclusivamente a colpi di interrupt e variabili globali?

Fine dello sfogo.

## Riassunto della puntata precedente

ADPnetwork, lo ripetiamo per chi si fosse sintonizzato solo ora, adotta uno schema di funzionamento «circolare» in cui ogni macchina ha un nodo precedente, dal quale riceve il flusso dei dati, e uno successivo al quale trasmette, o ritrasmette, dati. Ogni macchina analizzerà i dati in arrivo dalla rete e dovrà essere in grado di riconoscere messaggi per sé da inoltrare agli opportuni server, oppure da rimettere in circolo non riconoscendosi come destinatario. In questo modo è sia possibile che qualsiasi mac-

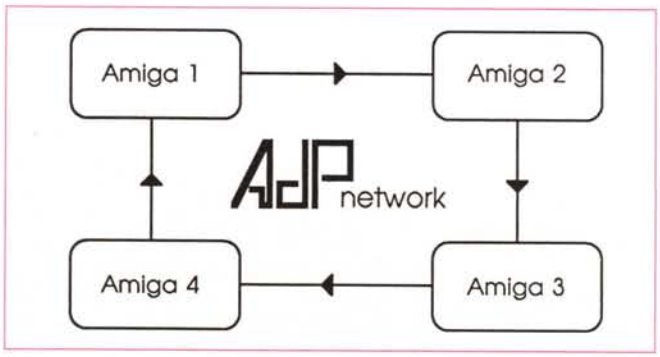


*Figura 1*
*Quattro Amiga collegati in rete via ADPnetwork.*

china dialoghi con qualsiasi altra macchina della rete, sia che in ogni istante più macchine effettuino operazioni sulla rete. Facendo un rapido esempio, se colleghiamo (figura 1) quattro macchine in rete attraverso ADPnetwork (la 1 con la 2, la 2 con la 3, la 3 con la 4 e quest'ultima con la 1) la macchina 2 per dialogare con la 4 passerà attraverso la 3 così come quest'ultima per «parlare» con la 2 passerà attraverso la 4 e la 1. Analogamente è possibile che CONTEMPORANEAMENTE la macchina 1 esegua un'operazione sulla 2 e che la 3 esegua una qualsiasi altra operazione sulla macchina 4. Questo grazie al fatto che la struttura di comunicazione è solo apparentemente condivisa da tutte le macchine: in realtà ogni nodo è banalmente proprietario del collegamento fisico con la macchina successiva, tutto qui.

L'attuale release funzionante di ADPnetwork, la 3.0, permette a qualsiasi processo in esecuzione su qualunque macchina di inviare messaggi a qualsiasi altro processo in esecuzione su qualsivoglia altra macchina collegata alla rete. Ogni messaggio può essere di lunghezza arbitraria e per inoltrarlo via rete il processo mittente dovrà naturalmente specificare in nodo destinatario, la porta mttb esistente su quel nodo (creata cioè dal processo destinatario) e consegnare il messaggio al software di rete. Sarà poi questo che, impacchettandolo opportunamente in frame di rete ed utilizzando l'interfaccia d'uscita verso la macchina «successiva» farà in modo (naturalmente con la complicità di tutti i processi di rete di tutte le macchine «attraversate») che il messaggio giunga a destinazione e sulla giusta porta. Se una macchina decide di uscire dalla rete, basta che sconnetta il suo ingresso e la sua uscita e li colleghi tra di loro: in questo caso si ripristina automaticamente il collegamento circolare e tutte le rimanenti macchine possono continuare ad adoperare la rete.

### *Packer e Spacker*

Sempre lo scorso mese vi ho anticipato che i due processi Packer e Spacker del Software di Rete (SDR), front-end verso AmigaDOS, si preoccupano rispettivamente di formare i frame di rete da spedire e di ricostruire i messaggi in arrivo man mano che giungono i vari frame da altre macchine. È ovvio inoltre che impacchettamento e spacchettamento dei messaggi deve essere una coppia di funzioni non visibili ai processi AmigaDOS, ai quali importa solo di spedire un messaggio ad una determinata macchina e riceverne da altre. Oltre a questo, il processo Spacker deve essere in grado di mantenere più liste degli arrivi, dal momento che possono arrivare più richieste da più macchine i cui frame, come detto, non sono regolamentati da un ordine di arrivo identico a quello delle rispettive partenze. I frame di rete viaggiano infatti in maniera indipendente l'uno dall'altro e, dato che ogni macchina è autorizzata ad annullare frame di transito contenenti errori di trasmissione e ad inserire tra un frame ed un altro in transito anche propri frame per altre macchine, capirete bene che la conquista dell'arrivo per questi sarà quantomeno faticata. Pensate ad esempio ad una rete di trasporto merci basata su ferrovia e navigazione (ad esempio un bel collegamento con le isole). Dalle stazioni di partenza vengono formati vari convogli per le relative destinazioni, spezzati però in più parti quando si tratta di attraversare tragitti marini (un intero treno, per lungo, non entra in una nave...). Immaginate inoltre (per rendere l'esempio più vicino al traffico su rete) che per motivi di ottimizzazione ogni volta che c'è da caricare una nave di carri ferroviari si cerchi sempre di riempire al massimo ogni nave, prelevando carri anche da convogli diversi. Ovviamente, però, a destinazione i convogli dovranno arrivare sempre e comunque con tutti i vagoni al loro posto e nel medesimo ordine della partenza, dunque le stazioni d'arrivo dovranno raccogliere man mano i vagoni che arrivano e ricomporre i convogli prima di strillare «in arrivo sul terzo binario...».

Eh, già! un solo binario non basta proprio: può succedere che arrivino prima un po' di pezzi del treno 208, poi tre vagoni del treno 665, poi ancora del treno 208, poi un carro del 256...

Spacker funziona proprio allo stesso modo. Arrivato un qualsiasi frame, la prima operazione che compie è vedere se già ha inizializzato una lista d'arrivo relativa a quel determinato messaggio. L'identificazione unica del messaggio è riportata all'interno di ogni frame: basta guardare il campo mittente e il campo MsgID, incrementato dal mittente stesso ogni nuova spedizione. Tale informazione è presente anche nelle liste d'attesa dello Spacker. O, meglio, è presente se la lista era già stata istanziata prece-


*ADPnetwork allo SMAU. Due Amiga collegati in rete visualizzavano le immagini di AMIGAllery memorizzate sull'hard disk della macchina a sinistra.*