di Andrea de Prisco

Cooperazione ad ambiente locale

Siamo finalmente giunti alla puntata «centrale» della rubrica Multitasking. Da questo numero in poi la nostra attenzione sarà focalizzata sui meccanismi di comunicazione a «scambio messaggi», propria della cooperazione ad ambiente locale, ma spesso presente anche in linguaggi di programmazione paralleli più propriamente ad ambiente globale

La netta distinzione

Nelle puntate precedenti abbiamo mostrato alcuni meccanismi di cooperazione-sincronizzazione basati sull'utilizzo (arbitrato) di strutture dati condivise. La condivisione di dati da parte di più processi paralleli è tipica della cooperazione ad ambiente globale. È proprio all'interno di questo che sono presenti le strutture comuni, e un certo numero di meccanismi messi a disposizione dal sistema permette di accedervi in maniera mutuamente esclusiva. Abbiamo così parlato di sospensione del multitasking, operazioni di Lock-UnLock, semafori, monitor: tutte con l'unico scopo di scongiurare disastrose collisioni tra processi paralleli.

A partire da questo numero, se volete, potete anche dimenticare tutto quanto narratovi (o quasi). Infatti nella cooperazione ad ambiente locale non esiste alcuna risorsa (o struttura) condivisa, ma tutto è regolato dai processi stessi. Processi, naturalmente, in grado di comunicare, ma sicuramente meno inclini a fare danni «in giro per il sistema». Ciò significa, tra l'altro, che non esistendo più strutture dati e risorse condivise, queste sono sempre e comunque inglobate all'interno di processi. Tanto per fare subito un esempio, se due o più processi hanno bisogno di un buffer per effettuare operazioni di inserimento ed estrazione di oggetti, nel caso «ambiente globale» tale buffer sarebbe stata una risorsa condivisa il cui accesso viene regolato dalle solite P e V o da un monitor; nel caso «ambiente locale» è necessario creare un processo che al suo interno ha come struttura privata il buffer e tramite scambio messaggi con gli altri processi effettua inserimenti ed estrazioni di oggetti (figura 1).

Comunicazioni inter process

In figura 2 è mostrata schematicamente una coppia di processi in grado di comunicare tra loro. È il caso più semplice: abbiamo un processo A che produce dati da inviare a B che li utilizza. Sempre in figura 2 la freccia che collega i due processi è il canale di comunicazione tra questi. E non è, come potrebbe sembrare, una struttura condivisa dai due processi in quanto a livello di questi il canale non è visibile. È sì un oggetto condiviso, ma ad un livello più basso: al livello del nucleo del sistema operativo. Infatti tanto A quanto B non hanno visione del canale ma della comunicazione in una forma più astratta. A manda messaggi a B, B riceve messaggi da A: che ci sia di mezzo un canale lo sappiamo noi e il sistema operativo, ma non i processi. Questo, naturalmente, quando la cooperazione è realmente ad ambiente locale: in realtà il più delle volte i linguaggi di programmazione parallela non sono così «puliti» fino in fondo e spesso lasciano visione all'interno dei processi anche di oggetti non meglio identificati ma comunque, almeno per certi versi, condivisi. Ma non scendiamo troppo nei dettagli subito: avremo modo di approfondire meglio l'argomento in seguito.

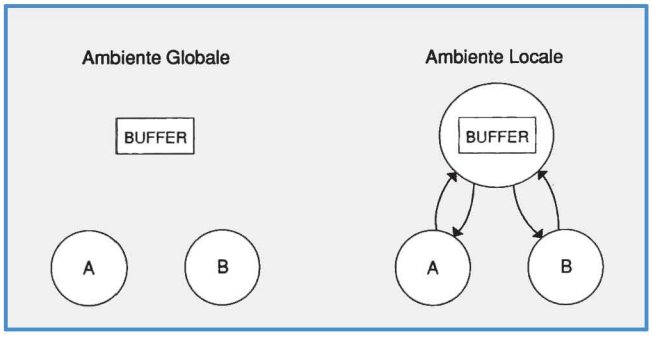

Figura 1 - Nella cooperazione ad ambiente globale le risorse condivise vengono arbitrate da meccanismi di mutua esclusione, in quella ad ambiente locale le risorse sono inglobate nei processi.

Dicevamo che il canale è un oggetto che permette la comunicazione tra processi ma non è direttamente visibile da questi. Un po' come il Program Counter nella programmazione in Assembler (quando questo non è allocato in un normale registro di macchina) o i vari puntatori alle aree dati nei linguaggi di programmazione ad alto livello.

Il canale è dunque direttamente (nonché esclusivamente) utilizzato dal nucleo del sistema operativo per effettuare la comunicazione.

Ciò che avviene all'interno dei processi (come in figura 2) è che il processo mittente a un certo punto esegue una operazione di «send» per spedire un messaggio (ad esempio un dato) al processo B. Questo, per ricevere l'informazione in arrivo da A, analogamente eseguirà la sua operazione di comunicazione che nel caso del processo destinatario sarà una «receive». La «send» avrà tra i suoi parametri l'oggetto da spedire, la «receive» una variabile dello stesso tipo nella quale riceverà l'oggetto trasmesso. In pratica a trasmissione avvenuta l'effetto finale sarà quello di assegnare alla variabile indicata nella funzione «receive» del processo destinatario il valore dell'oggetto trasmesso dal mittente e indicato nella funzione «send».

Il canale, conseguentemente, eredita il tipo dell'oggetto trasmesso che come abbiamo detto è anche uguale a quello della variabile targa del processo destinatario.

Ma per definire un canale non basta indicare il suo tipo ma occorre in qualche modo definire anche mittente e destinatario della comunicazione. Esistono fondamentalmente due strade: canali con nomi espliciti dei processi (figura

3a) e canali con porte (figura 3b). Ricordando che tale definizione non spetta né al processo mittente né al processo destinatario di una comunicazione ma semplicemente al sistema operativo che deve avere un «quadro chiaro» di tutta la situazione, è al momento della compilazione dei vari processi che sono definiti implicitamente anche i vari canali.

Nel caso di canali con nomi espliciti dei moduli il canale è definito dalla tripla:

```
(Mittente, Destinatario, Tipo)
```

e viene dedotta dal sistema in base alle operazioni di «send» presenti nei processi mittenti e alle operazioni di «receive» nei processi destinatari che contengono come primo parametro il nome del processo partner. Se ad esempio nel processo A troviamo:

```
send(B, messaggio)
```

nel processo B:

```
receive(A, variabile)
```

e tanto «messaggio» quanto «variabile» sono dello stesso tipo T, il canale è definito dalla tripla:

```
(A,B,T)
```

Nel caso di canali con porte il canale è definito dalla coppia ordinata:

```
(PortaMittente, PortaDestinatario)
```

Le porte sono oggetti privati dei processi e rappresentano delle interfacce logiche tra i processi e i canali di comunicazione. Hanno lo stesso tipo di messaggio da trasmettere e quindi di canale che interfacciano.

Nelle operazioni di «send» presenti nei processi mittenti e nelle operazioni di «receive» nei processi destinatari si fa esplicito riferimento non ai processi partner ma alla propria porta di ingresso o di uscita precedentemente definita.

Ciò che è ora necessario è «linkare», prima dell'utilizzo, una porta con un processo, sia nel caso del mittente (che linka la sua porta al nome del processo destinatario) che nel caso del destinatario (che linka la sua porta al nome del processo mittente). Se ad esempio nel processo A troviamo:

```
out port PA: T
var messaggio: T
bind(PA, B)
send(PA, messaggio)
```

nel processo B:

```
in port PB: T
var variabile: T
bind(PB, A)
receive(PB, variabile)
```

il canale è definito dalla coppia:

```
(PA, PB)
```

e non è più presente il tipo in quanto già definito nella dichiarazione delle porte.

Comunicazioni sincrone e asincrone

C'è un particolare che abbiamo volutamente sorvolato fino ad ora: la sincro-

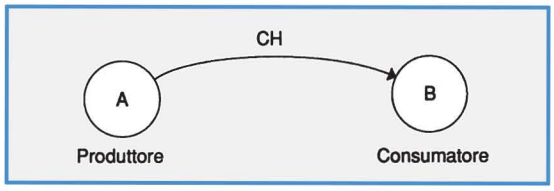

Figura 2 - Nella cooperazione ad ambiente locale le comunicazioni inter process avvengono esclusivamente attraverso messaggi trasmessi, ricevuti e canali di comunicazione.

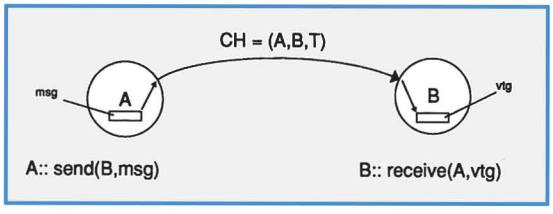

Figura 3a - Canali con nomi espliciti dei processi. In questo caso il canale è definito dalla tripla ordinata (Mittente,Destinatario,TipoDelMessaggio).

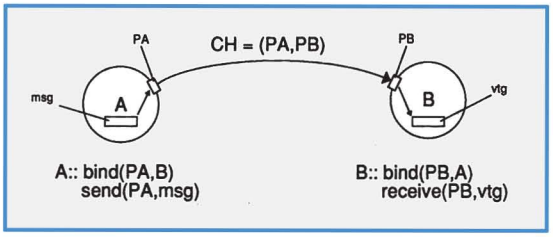

Figura 3b - Canali con porte. Qui il canale è identificato dalla coppia di porte (PA,PB) rigorosamente dello stesso tipo e connesse mediante l'esecuzione di «bind» corrispondenti nei due processi.

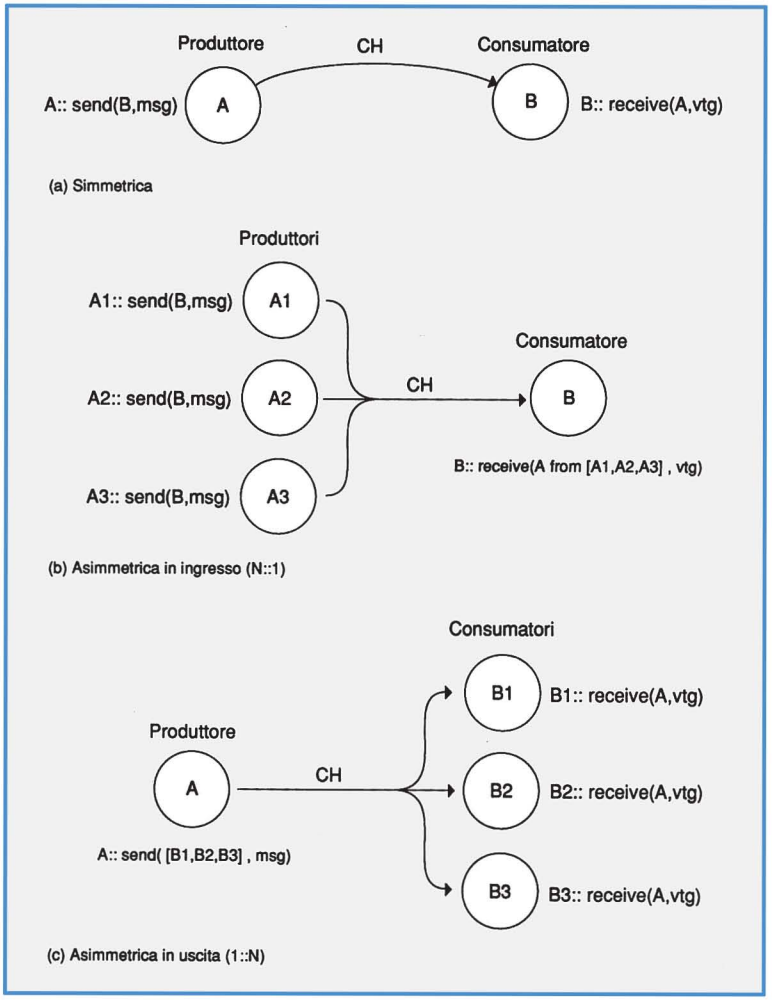

Figura 4 - Forme di comunicazione.

nia o asincronia delle comunicazioni. Cosa succede, per dirla in breve, se, quando il mittente effettua una «send», il destinatario non effettua la corrispondente «receive» perché momentaneamente indaffarato in altri lavori? Il mittente deposita il messaggio e continua oppure attende pazientemente il destinatario per consegnare personalmente il messaggio? Esistono in genere tutt'e due le possibilità: e, naturalmente, non esiste la soluzione migliore, ma possono essere utili l'una o l'altra possibilità a seconda dei casi. È chiaro, inoltre, che la comunicazione asincrona è possibile solo associando al canale un'area di memoria dove immagazzinare i messaggi spediti e non ancora letti.

Discorso del tutto analogo per le operazioni di «receive». Cosa deve fare il destinatario se un messaggio richiesto non è stato ancora spedito? Resta lì impalato ad aspettare, oppure continua per la sua strada senza il messaggio richiesto? Come nel caso precedente può essere utile disporre di entrambe le soluzioni a seconda dei casi. Nei prossimi numeri, quando cominceremo a mostrare alcuni programmi multitask vedremo come utilizzare a seconda dei casi le varie chance disponibili.

Forme di comunicazione

In figura 4 sono mostrate le forme di comunicazione base. La prima, la più semplice, prevede semplicemente che vi sia nella comunicazione un solo mittente e un solo destinatario (comunicazione simmetrica). È il caso che abbiamo già utilizzato negli esempi precedenti in cui un processo invia e un processo riceve attraverso un canale. Naturalmente è possibile instaurare una nuova comunicazione anche dal processo destinatario verso il mittente (invertendo i ruoli) attraverso un nuovo canale diverso dal primo. Infatti, posto che il tipo dei dati che i due processi si scambiano vicendevolmente è lo stesso, i due canali sono diversi essendo differenti il mittente e il destinatario nei due casi. Il primo canale sarà ad esempio identificato dalla terna:

(A, B, T)

il secondo dalla terna:

(B, A, T)

Le forme di comunicazione asimmetriche (rappresentate in figura 4b e 4c) consentono rispettivamente l'esistenza di più mittenti o più destinatari sul medesimo canale. La comunicazione asimmetrica in uscita (figura 4c) è detta anche comunicazione per diffusione in quanto il medesimo messaggio è inviato simultaneamente a più destinatari.

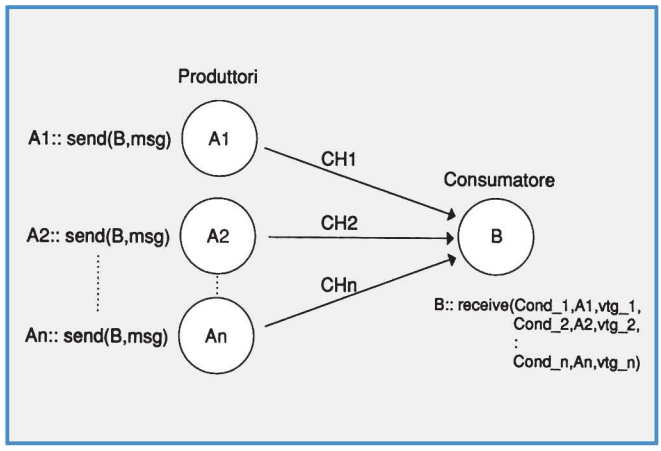

Figura 5
Controllo del nondeterminismo.

Nel primo caso della comunicazione asimmetrica in ingresso, si tratta di messaggi diversi (provenienti da processi diversi) che si accodano sul canale e che verranno letti in istanti successivi dal processo destinatario. Tanto i mittenti di figura 4b quanto destinatari di figura 4c non sono a conoscenza dell'asimmetria del canale: per loro la comunicazione è sempre e comunque simmetrica (il loro partner nella comunicazione è unico). Viceversa il destinatario di una comunicazione asimmetrica in ingresso e il mittente di una comunicazione asimmetrica in uscita sono coscienti della asimmetria in quanto il primo dovrà esplicitamente indicare da quali processi (più d'uno) riceve, il secondo a quali processi spedire.

Il nondeterminismo

Se fino ad oggi pensavate che gli attuali calcolatori fossero delle macchine deterministiche, avete una visione rigidamente sequenziale dei calcolatori.

Oggi, fermo restando la determinicità di un singolo processo (che ad ogni sequenza di input fornisce sempre un'unica sequenza di output), le cose stanno ben diversamente nelle macchine multitasking. Infatti a livello di processi nessuna ipotesi è fatta sull'avanzamento parallelo di questi, né sull'indeterminatezza di alcune situazioni che possono verificarsi. L'esempio tipico è mostrato proprio in figura 4b: i processi A1, A2, A3 sono assolutamente indipendenti; utilizzano (senza saperlo) lo stesso canale verso B il quale riceve sequenzialmente i messaggi dai tre processi mittenti. Se nello stesso istante logico più processi inviano messaggi a B non è definito a priori in quale sequenza questi saranno recapitati al destinatario. Né B potrà fare nulla per forzare una sua preferenza o priorità.

E se in alcuni casi possiamo procedere incuranti del problema, in altri potrebbe essere necessario pilotare in qualche modo la scelta del messaggio da ricevere per primo.

Naturalmente in questo caso non avremo più un unico canale asimmetrico ma tanti canali simmetrici quanti sono i mittenti (figura 5). All'interno del destinatario la receive avrà come parametri tante triple quanti sono i mittenti.

Ogni tripla sarà formata da una variabile targa (nella quale, eventualmente, ricevere il messaggio), il nome del mittente e una variabile di condizionamento (ad esempio un'espressione logica) che maschera o meno la lettura da quel canale. Così se abbiamo 5 canali e altrettanti processi che spediscono e vogliamo ricevere solo dai primi due processi sarà sufficiente far valere TRUE i primi due parametri booleani e FALSE i rimanenti tre.

Niente paura

Certo, per chi ha utilizzato per proprie risorse di calcolo sempre in maniera sequenziale questo linguaggio potrà sembrare anche un tantino ostico, ma per fortuna una volta presa la mano tutto torna ad essere semplice e magicamente affascinante.

Tra gli utenti dei piccoli sistemi i più fortunati sono gli utenti Amiga che dispongono di un vero e proprio computer multitask con tanto di processi, messaggi, porte, risorse, anche se non sempre gestiti o gestibili nel più pulito dei modi.

Gli utenti MS-DOS per giocare «al multitasking» hanno invece una possibilità diversa, molto più interessante, anche se poco proponibile dal punto di vista economico: acquistare una scheda per il loro PC dotata di almeno un transputer e due mega di ram e utilizzare su questa il linguaggio OCCAM.

Così come faremo noi a partire dal prossimo numero: metteremo un po' il naso nel multitask che si circonda per spulciare insieme le cose più interessanti dal punto di vista didattico e applicativo. Inutile ricordarvi che sono, come sempre, ben accette tutte le collaborazioni dei lettori che perverranno in redazione. Per il momento buon lavoro e appuntamento al prossimo mese.

MC