di Andrea de Prisco

Dalle stelle allo stallo

In questa puntata di Multitasking, parleremo del problema dello stallo di due o più processi. Situazione in cui si ha praticamente un blocco del sistema non a causa di errore di programmazione dei singoli processi, ma solo per la concorrenza di questi e per il fatto che il sistema sul quale ci muoviamo concede con troppa «leggerezza» l'uso esclusivo di risorse condivise

Problema di feeling

Lo scorso mese abbiamo visto come proteggerci da incauti utilizzi delle risorse condivise regolando l'accesso a queste tramite funzioni di sistema come la «Lock-UnLock» o i meno rudimentali semafori.

Non ci siamo però, volutamente, posti il problema di come proteggerci dallo stallo ossia da quelle situazioni in cui più processi non possono proseguire perché in attesa di risorse in quel momento utilizzate da altri processi che attendono a loro volta altre risorse utilizzate da noi.

È chiaro che se il processo «A» assume il controllo esclusivo della risorsa «a» e tenta di accaparrarsi anche la risorsa «b» attualmente in uso presso il processo «B» che a sua volta cerca di guadagnare la risorsa «a», i due processi rimarranno bloccati (in stallo) per sempre. A meno di non prendere i soliti, necessari, provvedimenti.

Facciamo subito un primo esempio: immaginiamo di dover accedere in maniera esclusiva ad una struttura dati condivisa sulla quale effettuare operazioni. Fatto questo procediamo all'acquisizione di una seconda struttura dati per aggiornarla con i nuovi valori della prima per poi terminare ancora le operazioni su quest'ultima prima di rilasciarla completamente.

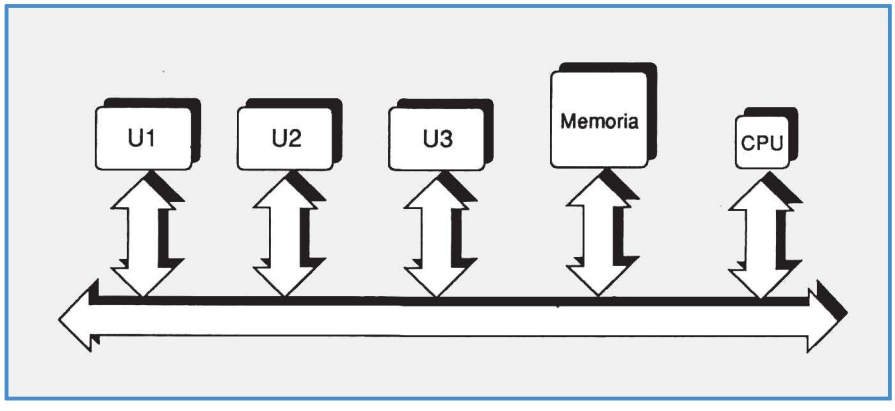

Figura 1 - Ipotesi di sistema composto da una CPU, una memoria e tre risorse condivise U1, U2, U3.

Potrebbe essere l'esempio della lista posti e lista attesa di un qualsiasi volo aereo: chiusa l'accettazione dei prenotati è necessario accedere alla lista dei posti per conteggiare quelli ancora liberi, accedere alla lista d'attesa per prelevare un pari numero di viaggiatori «speranzosi», e accedere nuovamente alla lista posti per aggiornare l'elenco dei viaggiatori.

Utilizzando le «P» e «V» viste lo scorso mese, il processo potrebbe avere una struttura di questo tipo:

```
P(Sem1)
Utilizzo struttura 1
P(Sem2)
Aggiornamento struttura 2
V(Sem2)
Utilizzo struttura 1
V(Sem1)
```

Un'altra procedura, differente dalla prima potrebbe avere la necessità di accedere alle due strutture dati condivise in ordine inverso:

```
P(Sem2)
Utilizzo struttura 2
P(Sem1)
Aggiornamento struttura 1
V(Sem1)
Utilizzo struttura 2
V(Sem2)
```

Cosa succede se le due procedure appena mostrate sono eseguite contemporaneamente ad esempio da due terminali diversi?

Molto probabilmente si ha uno stallo in quanto il primo processo acquisisce l'uso esclusivo della struttura 1 e non la rilascia fino a quando non ha eseguito tutte le sue operazioni.

Contemporaneamente potrebbe fare lo stesso la seconda procedura, acquisendo l'uso esclusivo della struttura 2 prima che lo faccia anche il primo processo. In pratica potremmo facilmente trovarci nella situazione in cui il processo 1 ha acquisito la struttura 1, il processo 2 ha acquisito la struttura 2, il processo 1 attende la liberazione della

struttura 2 (in mano al processo 2 che, di certo, non la rilascia) e il processo 2 attende la struttura 1. Attesa infinita...

Una prima soluzione

Un primo metodo drastico per risolvere il problema è quello di vedere le varie risorse cui siamo interessati nel corso della nostra elaborazione come un'unica risorsa indivisibile. In altre parole avremo un solo semaforo che controlla l'accesso simultaneo a tutte le risorse (in gioco) contemporaneamente. In pratica ogni processo che accede ad una qualsiasi risorsa condivisa blocca l'accesso a tutte le risorse disponibili e dunque, banalmente, il problema è risolto utilizzando erroneamente la forza.

Erroneamente proprio perché si avrebbe un costoso degrado delle prestazioni globali a causa del fatto che non è giusto bloccare tutto il bloccabile solo perché un singolo processo deve accedere ad una singola risorsa. E in più sarebbe come fare un vergognoso passo indietro rispetto alla gestione risorse «a semafori» che ha proprio come vantaggio principale il fatto di suddividere le risorse condivise in classi distinte e proprio per questo singolarmente arbitrabili.

Ciò che bisogna fare è prevenire i possibili stalli: agire prima che sia troppo tardi con metodi non cruenti (escludendo, cioè, la possibilità di uccidere i processi coinvolti in uno stallo).

Grafi e prenotazioni

Un metodo per prevenire gli stalli c'è e consiste nell'istituire, oltre alla mutua esclusione implementata attraverso semafori, un meccanismo di prenotazione delle risorse.

In pratica ogni processo quando entra in una sezione critica che a sua volta contiene altre sezioni critiche chiede l'uso esclusivo della prima e in più si prenota per quelle che, eventualmente, utilizzerà al suo interno.

Il sistema concederà in questo modo l'uso esclusivo della prima sezione critica solo se le prenotazioni indicate non generano la possibilità di stallo con gli altri processi (che a loro volta hanno fornito le loro prenotazioni).

Nell'esempio sopra visto, la prima P sul Sem1 diventerà una P su Sem1 con prenotazione Sem2: in pratica si chiede che venga dato l'uso esclusivo della risorsa controllata da Sem1 e si avvisa il sistema che prima di rilasciare tale risorsa potrebbe essere utilizzata anche la risorsa controllata da Sem2.

Il codice del primo processo diventa grosso modo così:

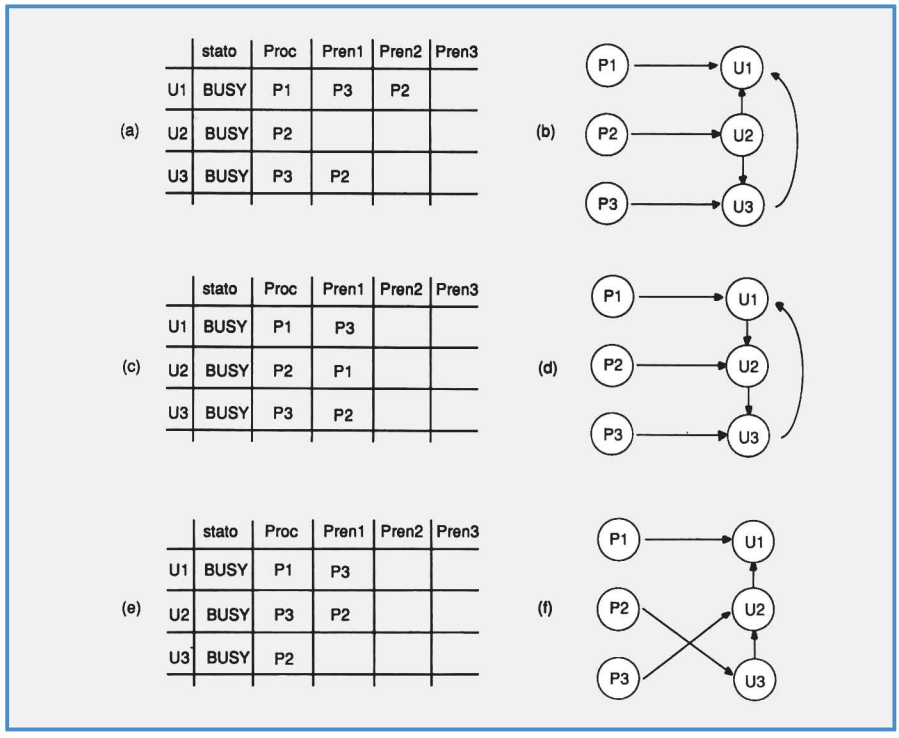

Figura 2 - Tabelle e grafi delle prenotazioni risorse condivise.

```
P(Sem1, Sem2)
Utilizzo struttura 1
P(Sem2)
Aggiornamento struttura 2
V(Sem2)
Utilizzo struttura 1
V(Sem1)
```

quello del secondo:

```
P(Sem2, Sem1)
Utilizzo struttura 2
P(Sem1)
Aggiornamento struttura 1
V(Sem1)
Utilizzo struttura 2
V(Sem2)
```

A questo punto il sistema rifiuterà la P(Sem2, Sem1) del secondo processo se verrà eseguita dopo la complementare richiesta del processo 1, evitando in questo modo lo stallo. Ma come fa il sistema per stabilire che una certa sequenza di prenotazioni può indurre uno stallo?

L'algoritmo esiste, ma per descriverlo nel più semplice dei modi ci avvarremo di un grafo.

In pratica il sistema mantiene una tabella delle prenotazioni che si modifica man mano che vengono eseguite le varie P e V sui semafori. Prima di concedere o meno l'uso esclusivo di una risorsa condivisa viene generato all'interno del sistema il grafo delle prenotazioni ed esplorato per stabilire se esiste o meno la possibilità di stallo.

La tabella delle prenotazioni è formata da una riga per ogni risorsa condivisa e contiene i seguenti campi:

1) stato della risorsa
2) processo utilizzatore
3) lista prenotazioni.

Il primo campo è generalmente impiegato per indicare lo stato della risorsa in questione. Il secondo indica il processo che, in quel momento, sta utilizzando in modo esclusivo la risorsa. Il terzo campo contiene la lista dei processi che hanno prenotato la risorsa ma che ancora non ne hanno chiesto l'uso esclusivo. A partire dalla tabella delle prenotazioni è generato il grafo delle prenotazioni come mostrato nei tre esempi di figura 2.

Nel primo dei tre esempi l'unità U1 è attualmente utilizzata dal processo P1 e ha una prenotazione da parte di P3 e P2, l'unità U2 è utilizzata da P2, l'unità U3 è utilizzata da P3 e ha una sola prenotazione da parte di P2. Nel grafo corrispondente (figura 2b) gli archi uscenti dai nodi «processo» indicano l'utilizzo in esclusiva da parte di questi, gli archi uscenti dai nodi «unità» indicano le dipendenze dovute alle prenotazioni. Così U3 (sempre in figura 2b) è

utilizzata in maniera esclusiva da P3 ed avendo quest'ultimo una prenotazione anche per U1 (P3 compare infatti anche nella lista prenotazioni di questa unità) è con questa collegata da un arco unità-unità.

L'assenza di stallo è graficamente rappresentata dall'assenza di cicli di dipendenza tra le unità: non esiste infatti un percorso che partendo dall'unità UX attraverso le frecce dirette torni nuovamente sull'unità di partenza.

Diverso è il caso della tabella di figura 2c e relativo grafo di figura 2d. Lì il percorso circolare esiste, infatti partendo da un qualsiasi nodo unità seguendo le frecce si torna allo stesso nodo di partenza: stallo immediato.

Del resto dando uno sguardo più attento alla relativa tabella possiamo dedurre lo stallo anche senza bisogno di grafo. L'unità U1 è utilizzata da P1; l'unità U2 è utilizzata da P2; l'unità U3 è utilizzata da P3. Inoltre P3 ha una prenotazione per U1, P2 per U3 e P1 per U1. P1 «mollerà» U1 dopo aver utilizzato anche U2 che è utilizzata da P2 che la lascerà dopo aver utilizzato U3. Questa, a sua volta, è utilizzata da P3 che la lascerà dopo aver usato anche U1 che è utilizzata attualmente da P1. Classico gatto che si morde la coda...

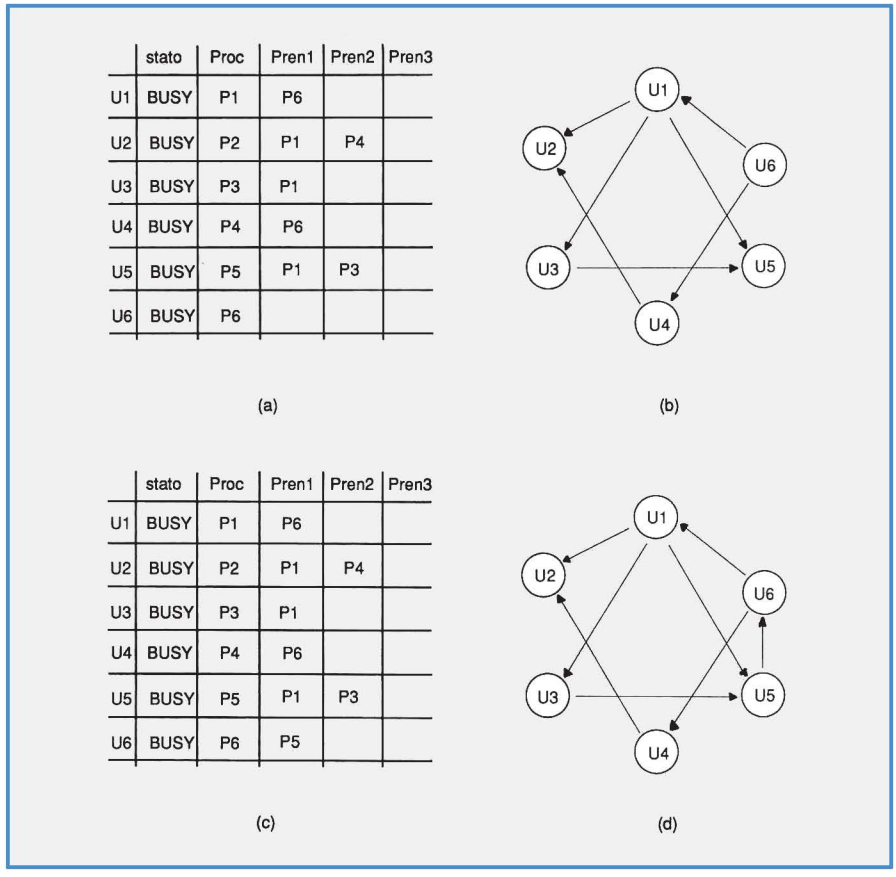

Figura 3 - Tabella prenotazioni e relativo grafo per sei processi che si contendono sei unità. Nella seconda delle due ipotesi si ha uno stallo.

Costruiamo un grafo

Come esempio finale di quest'articolo proviamo a giocare con sei processi che si contendono l'assegnazione esclusiva di altrettante risorse condivise. Possiamo immaginare che al sistema operativo arrivino le seguenti sei richieste da altrettanti processi:

```
P1:: P(Sem1, Sem2, Sem3, Sem5)
P2:: P(Sem2)
P3:: P(Sem3, Sem5)
P4:: P(Sem4, Sem2)
P5:: P(Sem5)
P6:: P(Sem6, Sem1, Sem4)
```

Come negli esempi precedenti il semaforo «SemX» controlla l'unità UX e nelle P con più parametri il primo identifica il semaforo sul quale si richiede l'accesso. I rimanenti parametri indicano le prenotazioni su altrettante risorse condivise. Ad occhio, sapreste giudicare se la sequenza di P sopra indicata mette i processi (non necessariamente tutti) in stato di stallo?

No, non c'è stallo: possono essere soddisfatte tutte le P. Vediamo perché: ci aiuteremo con un grafo, mostrato in figura 3b. La costruzione è assai semplice. Si tracciano tanti nodi quante sono le unità condivise e da ognuno di questi tante frecce dirette verso i nodi segnalati nelle liste di prenotazione. Così da quanto evidenziato dalla prima delle sei P da U1 tracceremo tre frecce verso U2, U3, U5. Con la seconda P, non essendoci prenotazioni non aggiungeremo alcuna freccia. Con la terza P tracceremo semplicemente un arco da U3 a U5 e così via fino alla sesta P ottenendo il grafo mostrato in figura 3b. Come facilmente visibile «a occhio» non esiste un percorso circolare che partendo da uno qualsiasi dei nodi riporti al nodo di partenza: non c'è infatti pericolo di stallo.

Certo la cosa cambia se poco dopo P5 esegue una V(Sem5) (liberando U5) e prima che altri processi blocchino la risorsa esegue un bel P(Sem5, Sem6). La tabella prenotazioni è mostrata in figura 3c, il nuovo grafo in figura 3d. Qui lo stallo c'è in quanto esiste un percorso circolare che partendo da U1, U5 o U6 porta allo stesso nodo di partenza: il sistema non deve permettere questo: P5 verrà sospeso come se avesse trovato la sezione critica occupata da un altro processo mentre sarà concessa la medesima sezione critica ad esempio a P1 o P3 che avendola già precedentemente prenotata non aggiungono nuovi archi al grafo e quindi probabilità di «introdurre» cicli e quindi stalli.

Dal punto di vista del computer

Certo un conto è prendere carta e penna e disegnare un bel grafo, ben altro è la visione dal punto di vista del computer che ha schemi di funzionamento diversi dalla nostra visione-intuizione. Un grafo può essere memorizzato sotto forma di lista multipla in cui i nodi sono gli elementi della lista e gli archi i puntatori tra questi. E la visita di un grafo non è certo un problema irrisolvibile, specialmente quando si hanno a disposizione linguaggi di programmazione ricorsivi. In altre parole il grafo viene costruito e aggiornato in memoria ad ogni chiamata P contenente prenotazioni. L'aggiornamento in questo modo è semplicissimo: basta riportare nei vari campi i puntatori agli elementi indicati nella lista di prenotazione. Per la visita i problemi non sono poi così tanti. Si parte infatti dal nodo che abbiamo in quel momento aggiornato (relativo alla unità che si intende utilizzare, indicata nel primo parametro della P) e da quello si scorre il grafo come fosse un albero. La visita ha esito positivo solo se termina senza mai ripassare per il nodo in partenza. Da notare che non è possibile entrare in cicli in cui non è compreso l'ultimo nodo aggiornato in quanto ciò indicherebbe la preesistenza di cicli che avremmo dovuto scartare precedentemente.

MC