SISTEMI

programmazione concorrente

Prof. Viglietti Francesco

Classe 4 B info

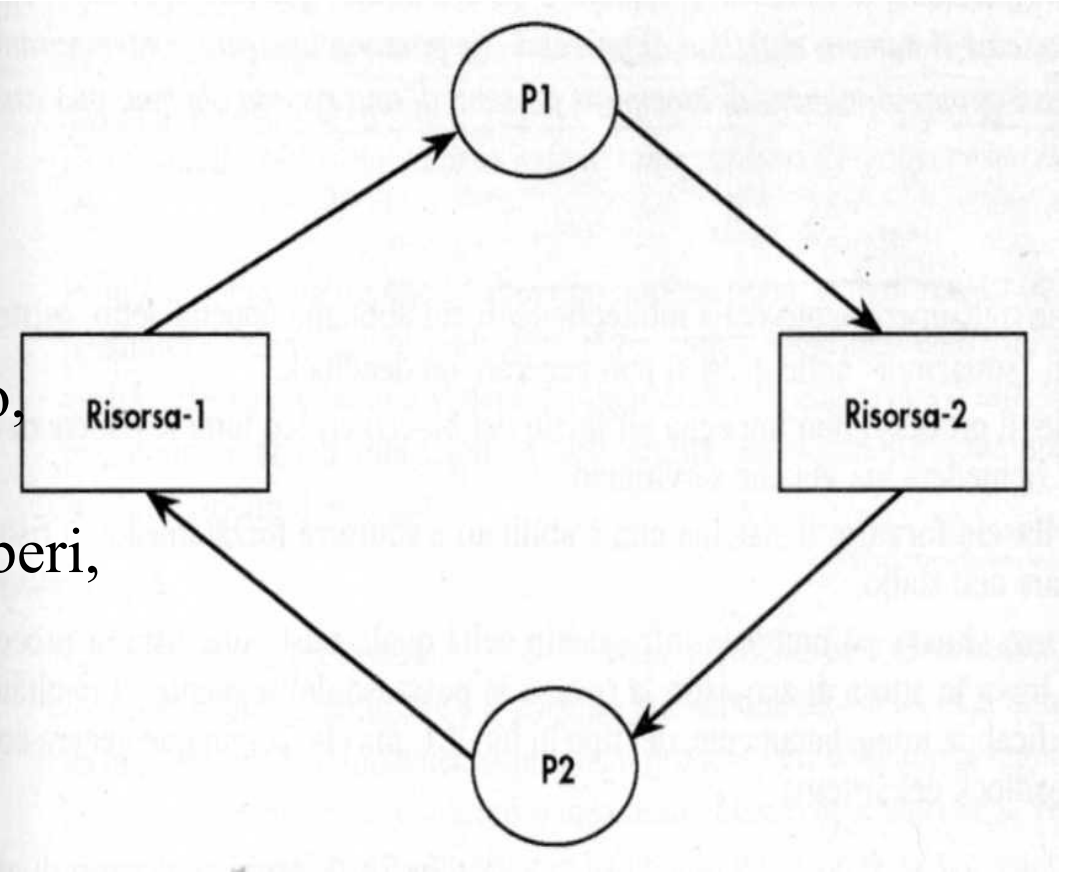
A.S. 2010-11

Introduzione - 1

Un sistema di elaborazione si dice in condizione di deadlock (stallo) se nessuno dei processi in corso riesce a portare a termine il suo lavoro. Il punto di vista è quello dell'utente. Dal punto di vista del OS, invece, i processi non riescono a muoversi regolarmente tra gli stati del diagramma del ciclo di vita dei processi: il motivo principale può essere che non funziona più come dovrebbe l'avvicendamento dei processi nell'uso delle risorse.

Supponiamo di avere i processi P1 e P2 che stanno lavorando rispettivamente con le risorse Risorsa-1 e Risorsa-2.

In un certo istante il processo P1 ha bisogno, per continuare regolarmente, di dati prodotti dalla risorsa Risorsa-2: in attesa di avere queste informazioni si pone in stato di Attesa. Nel frattempo, P2 chiede di lavorare con la risorsa Risorsa-1 e, aspettando che questa si liberi, si pone anch'esso in stato di Attesa. Risultato: i due processi sono in fase di attesa a tempo indeterminato!



Il deadlock - 1

Nella vita di un sistema di elaborazione, i moduli del OS si trovano a dover fronteggiare numerose problemi legati alla sincronizzazione, ma il più delicato si verifica quando il sistema si trova a dover controllare le zone critiche, ovvero SC che contengono istruzioni indivisibili e quindi le risorse richieste non verranno rilasciate che al termine dell'esecuzione. Questo è l'ambiente principale nel quale si può generare uno stallo o deadlock.

Più in generale, si può avere un deadlock non solo quando una risorsa richiede un uso esclusivo a un solo processo, ma anche (semaforo generalizzato) quando viene superata la molteplicità della risorsa.

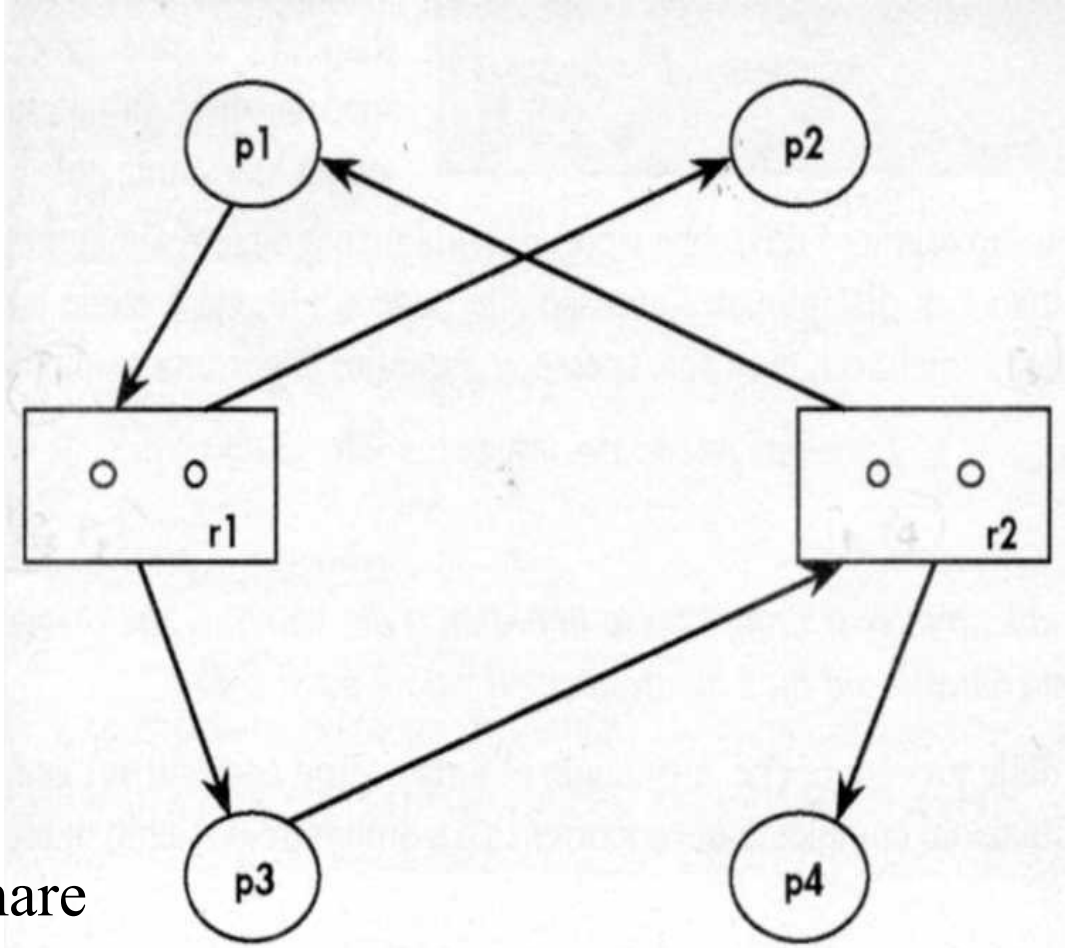
Oltre la mutua esclusione o il superamento della molteplicità, esistono le seguenti altre classi di "situazioni" nelle quali si può generare un deadlock:

allocazione parziale: il processo non impegna all'inizio del blocco critico tutte le risorse di cui avrà bisogno, ma le richiederà via via che serviranno;

negazione del prerilascio forzato: il sistema non è abilitato a sottrarre forzatamente la risorsa al processo per evitare uno stallo;

attesa circolare: è una situazione piuttosto infrequente nella quale esiste una lista di processi ognuno dei quali si trova in attesa di acquisire la risorsa in possesso del seguente. Il risultato è uno stallo non identificabile subito, ma che comunque genera complessivamente un deadlock del sistema.

Il deadlock - 2



L'esempio prevede due risorse r1 e r2 con molteplicità 2, 4 processi p1, p2, p3 e p4: apparentemente i processi p1 e p3 sembrano essere bloccati.

In realtà, anche se i due processi citati non possono proseguire il loro lavoro, il deadlock non si verifica: i processi p2 e p4 hanno già le risorse di cui hanno bisogno per continuare e terminare il proprio lavoro. Al termine di questi due, i processi p1 e p3 possono acquisire le risorse richieste (p1 richiede r1, p3 richiede r2) e quindi proseguire nella loro opera.

L'importante è che, in un certo istante, esista almeno un processo in grado di continuare il suo lavoro, poiché al termine esso rilascerà le risorse in suo possesso rendendole disponibili per gli altri processi.

In questo istante bisogna sperare che ci sia almeno un altro processo in grado di lavorare: se questo non accade siamo di fronte a un deadlock.

Gestione stallo

1) ***prevenzione***: il OS deve intuire quando sussistono le condizioni per lo stallo; è sufficiente fare in modo che tali eventi non si verifichino tutti simultaneamente. Il problema è che un livello di controllo eccessivo riduce in modo rilevante le prestazioni del sistema di elaborazione: negare tutti gli accessi e le allocazioni che potrebbero generare situazioni anche lontanamente pericolose può causare un notevole rallentamento considerato che il deadlock è, un evento piuttosto raro;

2) ***riconoscimento***: il sistema deve riconoscere quando il sistema di elaborazione si trova in situazione di stallo, una volta effettuata la rilevazione del problema dovrà effettuare un recupero (recovery) dello stato del sistema appena prima dello stallo.

Come fa realmente il sistema a capire quando si è verificato un deadlock?

Se un processo è in attesa di un dispositivo molto lento, l'attesa potrebbe essere giustificata e corretta. Il pericolo, in pratica, è quello che si mettano in azione i meccanismi di recovery anche quando non c 'è alcun problema.

Prevenzione

La prevenzione propone principalmente: ***tecniche di allocazione globale delle risorse, di allocazione gerarchica e l'algoritmo del banchiere***.

La tecnica di **allocazione globale delle risorse** prevede che le risorse necessarie a un processo gli vengano assegnate tutte insieme, evidentemente solo quando sono tutte contemporaneamente libere; in attesa che si verifichi questa situazione, il processo resta in attesa. Questa tecnica ha un alto margine di sicurezza, tuttavia può capitare che un processo impegni un risorsa che utilizzerà dopo molto tempo costringendo altri processi che ne hanno urgenza ad attendere a lungo. Insomma, in questo modo si possono rilevare notevoli rallentamenti nel tempo medio di esecuzione di un processo.

La tecnica di **allocazione gerarchica delle risorse** impone un ordine gerarchico tra le risorse che vengono distribuite in livelli di priorità. Se un processo possiede una risorsa di livello *j*, allora non può acquisire altre risorse di livello inferiore, ma solo di livello superiore. Se il processo ha necessità di allocare risorse di livello inferiore al *j* attuale, allora dovrà rilasciare quella in suo possesso, allocare quella a livello inferiore e richiedere nuovamente la risorsa a livello *j*. Questa tecnica non sempre è indicata poiché non ha molto senso considerare a livello diverso delle risorse tra loro equivalenti; in questo caso può essere più efficiente il metodo precedente dell'allocazione globale.

Algoritmo del banchiere - 1

Presuppone che per ogni processo sia noto il numero massimo di unità di ogni risorsa di cui esso avrà bisogno. Il nome di questo algoritmo viene dalla necessità, in un sistema bancario, di non impegnare completamente la propria cassa così da non poter soddisfare nessuno degli altri clienti. Nel nostro àmbito, il significato è che il OS, tra tutte le richieste di allocazione effettuate dai processi, dovrà scegliere quella che consente ad almeno un processo di poter lavorare e concludere. In questo modo non avremo una situazione di stallo neanche se un processo resta, in fase di attesa poiché esiste almeno un altro processo che può continuare il proprio lavoro.

Esempio: (N: molteplicità della risorsa contesa; Mi: quantità massima di unità della risorsa richiesta dal processo i; Ai: unità della risorsa attualmente allocate al processo i; Ri: unità richieste nell'istante attuale dal processo i (ovviamente $Ai + Ri < Mi$); L: quantità delle unità attualmente libere). Immaginiamo di avere 3 processi che si contendono 10 unità della stessa risorsa in modo esclusivo ($N = 10$): la quantità massima di risorse richieste dai processi è rispettivamente 6, 4 e 7.

Attualmente il sistema ha allocato ai tre processi, in ordine, 3, 1 e 2 unità e sono in sospeso le richieste di 1, 2 e 3 unità: il sistema deve ora decidere quale delle tre richieste soddisfare per prima.

Algoritmo del banchiere - 2

Task	A	R	M
1	3	1	6
2	1	2	4
3	2	3	7

Il problema può schematizzarsi come indicato in tabella.
Scegliamo quali richieste R soddisfare (al momento attuale restano disponibili L=4 unità).
Se scegliamo di allocare un'unità al primo processo, avremo altre 3 unità ancora disponibili, quindi saremo in grado di soddisfare sia le attuali richieste del secondo processo sia una sua ulteriore richiesta di 1 unità che porta il lavoro al termine, oppure potremo assegnarle in blocco al terzo processo che però non potrà terminare il suo lavoro visto che le sue richieste massime arrivano a 7.
Questa scelta soddisfa l'algoritmo del banchiere poiché l'assegnazione consente ad un altro processo, il secondo, di terminare il proprio lavoro.
Se scegliamo di allocare 2 unità al secondo task, avremo 2 unità residue che potranno soddisfare temporaneamente le richieste del primo task (con il resto di 1 unità) ma non gli consentirà di terminare; il terzo task, invece, risulta immediatamente in attesa. L'unità rimanente potrà essere utilizzata ancora dal secondo task per condurre a termine le proprie operazioni: complessivamente, questa soluzione non è desiderabile ma non genera necessariamente uno stallo.
Se scegliamo il terzo task, avremo 1 sola unità residua: la richiesta del primo processo potrà essere soddisfatta ma non lo porterà alla sua conclusione. In definitiva, la scelta migliore è di privilegiare il primo dei nostri 3 processi.

Riconoscimento - 1

Per quanto riguarda il **riconoscimento** del deadlock, non esistono soluzioni generali. Il problema presenta due aspetti: in primis si deve riuscire a identificare il deadlock, poi si deve decidere come effettuare il recovery del sistema nello stato precedente allo stallo. Il primo aspetto trova una soluzione di tipo statistico, nel senso che il sistema deve essere progettato per effettuare delle stime dei tempi massimi accettabili di possesso o di attesa di una risorsa; se un processo supera i tempi massimi stimati potrà essere considerato in stato di stallo e quindi soggetto a recupero. Il problema principale risiede nell'aleatorietà dell'approccio, per cui può capitare che si intervenga in situazioni nelle quali non si è verificato uno stallo: tra poco vedremo come utilizzare uno strumento pratico come il grafico di allocazione delle risorse per individuare le situazioni a rischio.

Il secondo aspetto riguarda il recupero. Le soluzioni sono diverse:

1)interruzione di *tutti* i processi in fase di deadlock;

2)interruzione di un processo per volta finché non viene eliminato il deadlock generale;

3)rilascio forzato di una risorsa per volta da parte di un processo prescelto finché non viene elimi-
nato il deadlock generale.

Riconoscimento - 2

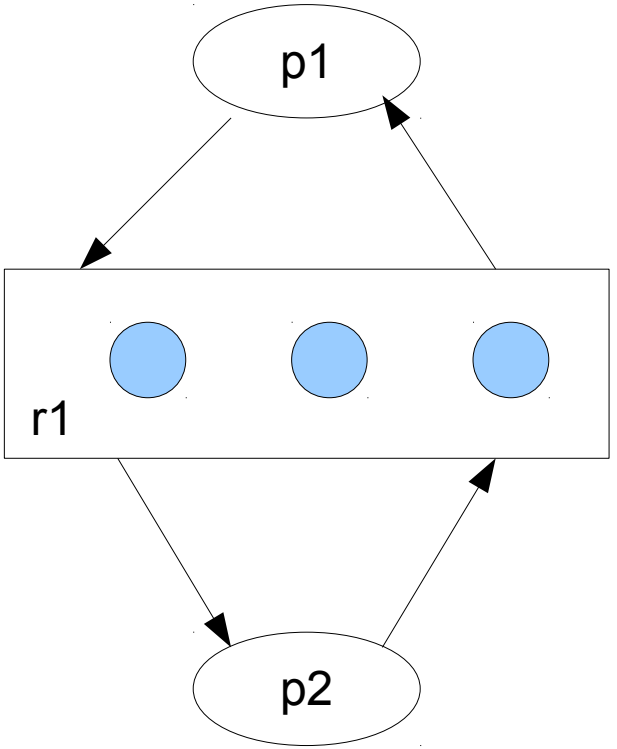
Il I metodo è sicuramente infallibile. Il II metodo è molto più complesso da realizzare: si procede con il recupero dello stato della macchina antecedente lo stallo. Si prevede una procedura che seleziona un processo da eliminare e poi controlla se si è ancora in stato di deadlkock: se sì, si elimina un altro processo, altrimenti si prosegue per i processi superstiti mentre si passa al ripristino di quelli eliminati. Questo metodo è il più consigliato dei tre anche se causa un notevole sovraccarico al sistema. Il III metodo è applicabile solo in situazioni di deadlock non grave. Anche in questo caso si fa un'operazione selettiva, tuttavia si prevede di lavorare ancora più di precisione del caso precedente: l'analisi è svolta più precisamente dal punto di vista della risorsa e non del processo ricercando quale singola allocazione di una risorsa può essere interrotta in modo da sbloccare la situazione. Spesso, non il processo nel suo complesso causa lo stallo del sistema, ma un semplice gruppo di unità particolarmente richieste e che sono state assegnate in un ordine poco felice. Questo metodo diventa inapplicabile nelle condizioni di stallo più contorte: può risultare più veloce interrompere il processo piuttosto che rintracciare l'allocazione responsabile del guaio. Una volta forzato il rilascio della risorsa da parte del processo si potrà pensare di trasformare l'allocazione in nuova richiesta della stessa.

Riconoscimento - 3

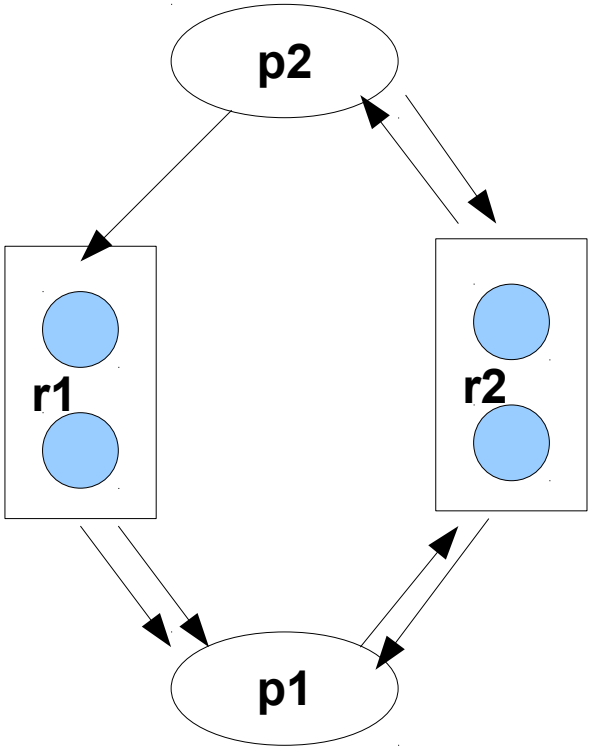
Ma come scegliere il processo da interrompere? Ecco i criteri più significativi:
1)la priorità del processo: si sceglie il processo da interrompere tra quelli a priorità più bassa;
2)quanto a lungo ha lavorato il processo e quanto ancora resta da fare: se il processo ha già lavorato a lungo costerebbe molto farlo ricominciare e se ha quasi terminato non conviene interromperlo perché potrebbe velocizzare lo sblocco dell'ingorgo;
3)quante e quali risorse sono state utilizzate e se è facile riottenerne l'allocazione. Se blocchiamo un processo che richiede molte risorse particolarmente intasate o difficili da acquisire, sì rischia che questo venga ad essere nuovamente sospeso in tempi brevi;
4)quante risorse servono al processo per terminare il suo lavoro.
Tra queste elencate, il sistema definirà una strategia che riterrà opportuna.

Esempio - 1

Consideriamo il grafo che presenta due processi e tre unità di una risorsa.
Possiamo notare che il processo ***p2*** ha già allocato le due risorse di cui ha bisogno, mentre il primo processo ha allocato una sola unità ed è in attesa di allocare la seconda e, nell'istante a cui si riferisce il grafo, esso si trova in attesa.
Non appena il secondo processo termina il suo lavoro, rilascia le risorse e quindi consente al primo processo di riattivarsi, acquisire la risorsa che gli serve e terminare. Se, prima che sia stato effettuato un rilascio, arriva un'altra richiesta dì unità da parte del processo ***p2,*** allora otteniamo un deadlock!



Esempio - 2



Si hanno 2 task e 2 risorse, composte da 2 unità.
Nell'istante a cui si riferisce il grafo, p1ha le 2 unità di r1, 1 unità di r2 e richiede l'altra; p2 ha 1 unità di r2, e richiede l'altra e 1 unità di r1. In queste condizioni il sistema si trova in deadlock, in quanto nessuno dei due processi è in possesso di tutte le risorse di cui ha bisogno per poter proseguire. Dobbiamo provvedere a un riconoscimento e quindi dobbiamo sostanzialmente decidere "come" e "su quale" processo agire: p2 possiede solo 1 unità di r2, mentre p1 possiede in tutto 3 unità delle nostre risorse, per cui è lecito pen-sare che sia ben più avanti con il lavoro. Inoltre, se si riesce a liberare p1 dallo stallo e gli si consente di terminare, allora alla fine esso deallocherà una notevole quantità di risorse che torneranno disponibili per altre operazioni. Il nostro consiglio è di agire su p2 che sembra aver compiuto meno operazioni; ci sembra, inoltre, sufficiente togliere al processo il possesso della sua unità della r2 così che la richiesta di p1 possa essere esaudita e possa continuare il suo lavoro sino alla conclusione. A questo punto p2 può riallocare l'unità che ha perso insieme con tutte le altre di cui ha bisogno.

Starvation

Un problema legato al deadlock è il **blocco individuale (starvation),** che si verifica quando solo alcuni dei processi del sistema restano bloccati mentre, complessivamente, il sistema lavora.

Il suo verificarsi è per lo più casuale e dipendente da una sfortunata sequenza di richieste degli altri processi: se abbiamo una serie di processi a priorità alta che continuano ad avvicendarsi nel sistema non dando spazio a un processo a priorità molto bassa, allora otterremo che quest'ultimo processo sarà a tutti gli effetti bloccato, mentre globalmente il sistema continua a lavorare. Data l'aleatorietà del fenomeno, risulta molto difficile il riconoscimento così come la prevenzione; l'unica possibilità è legata alla dinamicità degli algoritmi di assegnazione delle risorse che devono cercare di non penalizzare oltre un certo limite le richieste meno importanti.