8 - Deadlock

domenica 24 aprile 2022 18:32

Un insieme di processi è in deadlock quando tutti i processi sono in attesa di un evento che può accadere solo grazie a un processo dello stesso insieme

4 CONDIZIONI PER L'AVVENIMENTO DI UN DEADLOCK

1)Mutua esclusione: almeno una risorsa non è condivisibile
-es: un processo richiede una risorsa ma è già utilizzata da un altro processo, quindi deve aspettare che si libera
2)Hold and Wait: un processo ha già una risorsa ma ne aspetta un'altra (tenuta da un altro processo)
3)No pre-emption: le risorse possono essere lasciate solo dopo il loro utilizzo, non possono essere tolte prima
4)Attesa circolare: un insieme di processi attendono ciclicamente il liberarsi di una risorsa

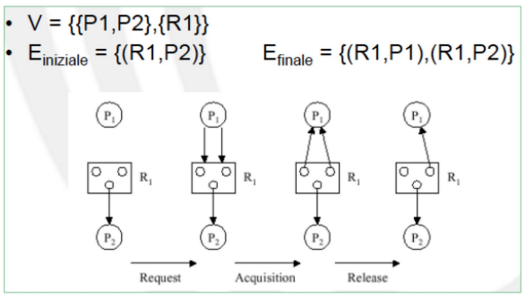
•Devono verificarsi tutte e 4 le condizioni per avere un deadlock

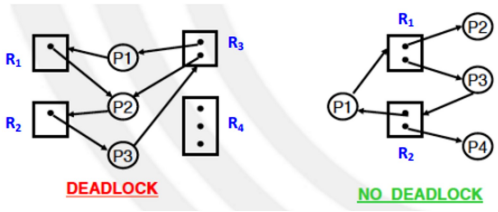
MODELLO ASTRATTO (RAG)

•RAG: Resource Allocation Graph
•Un modo di visualizzare l'utilizzo delle risorse che ci permette di verificare se abbiamo un deadlock

•Se il RAG non contiene cicli, allora non ci sono deadlock
•Se il RAG contiene cicli:
-se si ha una sola istanza per risorsa si ha deadlock
-se ci sono più istanze, dipende dall'allocazione

•**V = nodi**
-Cerchi: processi
-Rettangoli = risorse (CPU, I/O, memoria)
-pallini nei rettangoli: istanze di una risorsa
•**E = archi**
-da processi a risorse: processo richiede risorsa
-da risorse a processi: processo detiene risorsa





4 MODI PER GESTIRE UN DEADLOCK

1)Prevenzione statica: evitare che si verifichi una delle 4 condizioni
2)Prevenzione dinamica: si valuta ogni allocazione per vedere se questa può portare al deadlock
3)Rilevamento e ripristino (detection and recovery): uso di algoritmi per il rilevamento di deadlock senza conoscenza a priori delle risorse e delle richieste
4)Algoritmo dello struzzo: non si fa nulla

PREVENZIONE STATICA

Dobbiamo impedire che si verifichi una delle 4 condizioni

1)Mutua esclusione: non si può togliere
2)Hold and wait: il processo alloca fin da subito tutte le risorse che gli servono anche se non deve usarle subito
-può portare a starvation o a diminuire l'uso delle risorse
3)No pre-emption: il processo che richiede una risorsa rilascia tutte quelle in suo possesso
-applicabile solo per risorse il cui stato può essere ristabilito (CPU, registri, semafori, file)
4)Attesa circolare: assegno una priorità ad ogni risorsa
-un processo può richiedere risorse SOLO in ordine crescente di priorità

PREVENZIONE DINAMICA

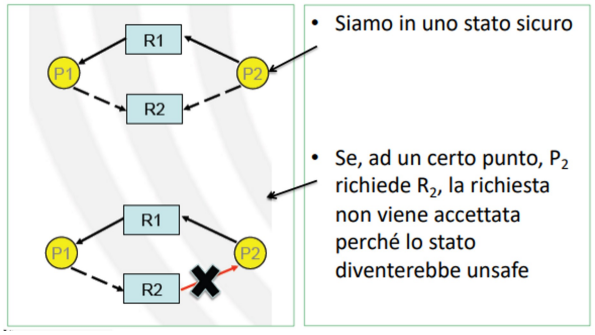
•Valuto ogni allocazione delle risorse per vedere se possono portare al deadlock
 -se l'allocazione di una risorsa da parte di un processo può portare al deadlock, gliela vieto
•Il sistema si trova in uno stato SAFE se **tutti i processi ottengono le risorse di cui hanno bisogno e riescono a terminare**
 -non necessariamente le risorse vengono date nell'ordine in cui vengono richieste
 -**in stato SAFE non ho deadlock**
•Se non tutti i processi ottengono le risorse che richiedono siamo in stato UNSAFE
 -può portare al deadlock, ma non necessariamente

•La prevenzione dinamica consiste nell'utilizzare algoritmi che lasciano il sistema sempre in stato SAFE
 -**Algoritmo con RAG**
 -**Algoritmo del banchiere**

ALGORITMO CON RAG

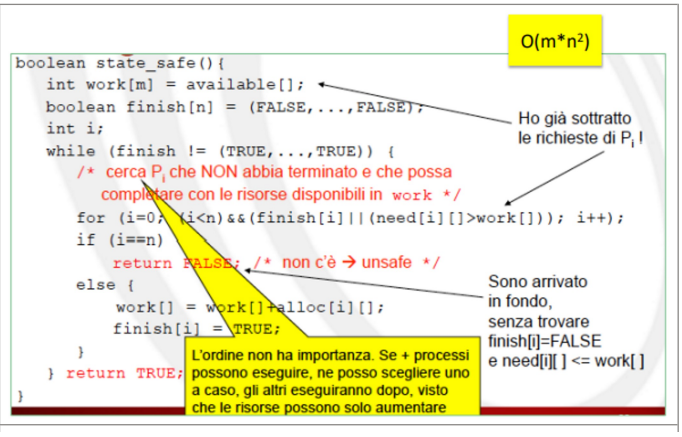
•Funziona solo se ho **un'istanza per ogni risorsa**
•Ogni processo deve dire quali risorse vorrebbe usare durante la sua esecuzione
•Le richieste vengono soddisfatte solo se non creano un ciclo nel RAG (perché creerebbero uno stato UNSAFE)

NOTA: la freccia tratteggiata significa che il processo richiederà quella risorsa in futuro



ALGORITMO DEL BANCHIERE

•Funziona con **più istanze ma risulta meno efficiente**
•Ad ogni richiesta, si verifica se è possibile soddisfarla rimanendo in uno state SAFE
•Costituito da 2 algoritmi:
 -algoritmo di allocazione
 -algoritmo di verifica dello stato



- 5 processi: P_0,P_1,P_2,P_3,P_4
- 3 risorse:
  - A (10 istanze) B (5 istanze) C (7 istanze)
- Fotografia al tempo T_0:

	Allocation	Max	Available	Need
	A B C	A B C	A B C	A B C
P_0	0 1 0	7 5 3	3 3 2	7 4 3
P_1	2 0 0	3 2 2		1 2 2
P_2	3 0 2	9 0 2		6 0 0
P_3	2 1 1	2 2 2		0 1 1
P_4	0 0 2	4 3 3		4 3 1

- Siamo ancora in uno stato safe? Verifichiamolo:
  - work = (2,1,0)
  - i=0 finish=FALSE, need[0]=(7,2,3) > work
  - i=1 finish=FALSE, need[1]=(0,2,0) > work
  - i=2 finish=FALSE, need[2]=(6,0,0) > work
  - i=3 finish=FALSE, need[3]=(0,1,1) > work
  - i=4 finish=FALSE, need[4]=(4,3,1) > work
- Stato unsafe!

RILEVAMENTO DEADLOCK E RIPRISTINO

- Consiste nell'utilizzare algoritmi che non richiedono conoscenze a priori sulle risorse e sulle richieste
  - **algoritmo di rilevazione e ripristino con RAG**
  - **algoritmo di rilevamento del banchiere**
- L'unico svantaggio di entrambi gli algoritmi è il costo di ripristino

ALGORITMO DI RILEVAZIONE E RIPRISTINO CON RAG

- Funziona solo se ho un'istanza per ogni risorsa
- Analizza il grafo di attesa, verifica se ci sono deadlock (**detenction**) e inizia il ripristino (**recovery**)

ALGORITMO DI RILEVAMENTO DEL BANCHIERE

- Funziona con più istanze
- Verifica che una serie di richieste non causi deadlock

O($m*n^2$)

```
int work[m] = available[m];
bool finish[] = (FALSE,...,FALSE), found = TRUE;
while (found) {
        found = FALSE;
        for (i=0; i<n && !found; i++) {
        /* cerca un Pi con richiesta soddisfacibile */
            if (!finish[i] && req_vec[i][] <= work[]){
            /* assume ottimisticamente che Pi esegua fino al termine
            e che quindi restituisca le risorse */
                work[] = work[] + alloc[i][];
                finish[i]=TRUE;
                found=TRUE;
            }
        }
} /* se finish[i]=FALSE per un qualsiasi i, Pi è in deadlock */
```

Se non è così il possibile deadlock verrà evidenziato alla prossima esecuzione dell'algoritmo

- 5 processi: P_0,P_1,P_2,P_3,P_4
- 3 tipi di risorsa:
  - A (7 istanze), B (2 istanze), C (6 istanze)
- Fotografia al tempo T_0:

	Allocation	Request	Available
	A B C	A B C	A B C
P_0	0 1 0	0 0 0	0 0 0
P_1	2 0 0	2 0 2	
P_2	3 0 3	0 0 0	
P_3	2 1 1	1 0 0	
P_4	0 0 2	0 0 2	

- Siamo in una situazione di deadlock? Verifichiamolo:
  - work = (0,0,0)
  - i=0 req[0] = (0,0,0) <= work OK
    work = work + (0,1,0) = (0,1,0) finish[0] = true P_0 ✓
  - i=1 req[1]=(2,0,2) <= work NO ✗
  - i=2 req [2] = (0,0,0) <= work OK
    work = work + (3,0,3) = (3,1,3) finish[2] = true P_2 ✓
  - i=3 req[3] = (1,0,0) <= work OK
    work = work + (2,1,1) = (5,2,4) finish[3] = true P_3 ✓
  - ...
- La sequenza <P_0, P_2, P_3, P_4, P_1> dà finish[i] = true per ogni i

- Supponiamo invece che P_2 richieda un'ulteriore istanza della risorsa C

	Allocation	Request	Available
	A B C	A B C	A B C
P_0	0 1 0	0 0 0	0 0 0
P_1	2 0 0	2 0 2	
P_2	3 0 3	0 0 1	
P_3	2 1 1	1 0 0	
P_4	0 0 2	0 0 2	

- Siamo in una situazione di deadlock? Verifichiamolo
  - work = (0,0,0)
  - i=0 req[0]=(0,0,0) <= work OK
    work = work + (0,1,0) = (0,1,0) finish[0] = true P_0 ✓
  - i=1 req[1]=(2,0,2) <= work NO
  - i=2 req[2]=(0,0,1) <= work NO
  - i=3 req[3]=(1,0,0) <= work NO
  - i=4 req[4]=(0,0,2) <= work NO
- DEADLOCK formato da P_1, P_2, P_3, P_4

RIPRISTINO

•L'algoritmo di rilevamento può essere chiamato:
-dopo ogni richiesta
-ogni N secondi
-quando l'utilizzo della CPU scende sotto una certa soglia

•Il ripristino può agire in 2 modi:
-**uccidere i processi coinvolti** (si possono uccidere tutti o ucciderli selettivamente fino alla scomparsa del deadlock
-**riprendere le risorse contese** (viene stabilita una politica per decidere in quale ordine vengono tolte le risorse)
•Entrambe le soluzioni risultano costose

CONCLUSIONI

•Normalmente si opta per **non fare nulla** in quanto i deadlock si verificano con una frequenza bassissima e la loro gestione risulta troppo costosa
•Oppure le risorse vengono partizionate in classi
-per ogni classe viene scelto l'algoritmo più appropriato
-le classi possono dividersi in: risorse interne, memoria, risorse di processo e spazio di swap