Deadlock

Andrea Comar

20 November 2024

1 Deadlock

Problema del deadlock nasce basandosi su un sistema con piu' processi che condividono risorse. Il problema è l'accesso alle risorse condivise che però possono essere usate in mutua esclusione. Se ogni processo necessita di un'unica risorsa il deadlock non si verica. Il problema è quando più processi necessitano di più risorse differenti, alle quali si accede in mutua esclusione. Le attese circolari che si verificano si chiamao per l'appunto **deadlock**. Può essere gestito a diversi livelli.

- tramite prevensione
- tecniche per riconoscimento e risoluzione dei deadlock

La risorsa Può essere

- pre-rilasciabili: possono tolte senza effetti dannosi, per esempio memoria centrale
- non pre-rilasciabili: non possono essere tolte al processo, pena fallimento esecuzione (es. stampante)

Ovviamente deadlock su risorse **non pre-rilasciabili**.
protoccollo prevede: richiesta, uso e rilascio della risorsa.
Se risorsa non disponibile alla richiesta, ci sono diverse opzioni (attesa, attesa limitata, ecc..)

Esempio mediante uso di semafori:

```
METODO CLASSICO PER UNA RISORSA
void process_A(void){
    down(&resource_1);
    use_resource
    up(&resource_1);
}
```

SLIDE: doppio down (risorsa 1 e 2) e doppio up (risorsa 1 e 2), può portare a deadlock.

Se tutti i processi richiedono le risorse nello stesso ordine, non si verifica il deadlock.

```
void process_A(void){
    down(&resource_1); // se interrotto qua...(1)
    down(&resource_2); // quindi prende la risorsa 2(3)
    use_resource
    up(&resource_2);
    up(&resource_1);
}

void process_B(void){
    down(&resource_1);  // ... B non procede (2)
    down(&resource_2);
    use_resource
    up(&resource_2);
    up(&resource_1);
}
```

Al contrario, se i processi richiedono le risorse in ordine diverso, si verifica il deadlock.

```
void process_A(void){
    down(&resource_1); //prende risorsa 1 ...(1)
    down(&resource_2); // risorsa 2 bloccata (3)
    use_resource
    up(&resource_1);
    up(&resource_2);
}

void process_B(void){
    down(&resource_2); //prende risorsa 2 ...(2)
    down(&resource_1); // risorsa 1 bloccata !!! (4)
    use_resource
    up(&resource_2);
    up(&resource_1);
}
```

Ovviamente difficile che il codice sia sicuro, diversi programmatori operano. Necessari metodi per **riconoscere** se c'è il rischio di deadlock oppure se è in corso.

definizione di deadlock: insieme di processi in attesa di risorse che non possono essere rilasciate.

4 condizioni necessarie ma non sufficienti per il deadlock:

- **Mutua esclusione**: ogni risorsa è assegnata a un solo processo, oppure disponibile

- **Hold and wait**: processo in possesso di risorse può chiederne altre
- **No preemption**: risorse possono essere rilasciate solo volontariamente dal processo
- **Catena di attesa circolare**: insieme di processi in attesa di risorse che non possono essere rilasciate

Basta negare una delle 4 condizioni per evitare il deadlock. Possibile approccio per prevenirlo

1.1 Grafo di allocazione delle risorse

- **Nodi**: processi (cerchi) e risorse (quadrato)
- **Archi**: assegnazione di risorse

Nel grafo due tipi di archi:

- **Archi di richiesta**: processo in attesa di risorsa, da processo a risorsa
- **Archi di assegnazione**: processo in possesso di risorsa, da risorsa a processo

Un **ciclo** nel grafo indica un possibile deadlock.
Se il grafo non contiene cicli, non c'è deadlock.Invece se il grafo possiede un ciclo:
(..)
Si potrebbe testare sul grafo la presenza di cicli, compito immane, non viene fatto,
CI sono politiche che ci possono portare a deadlock mentre altre no. Ad esempio first come first served, non porta a deadlock.
Invece round robin, può portare a deadlock. Tendenzialmente tutte le potiche che prevedono prelazione rischiano di essere non safe

1.2 gestione del deadlock

Vari approcci dal meno costoso al più costoso:

- ignoro il problema: è costoso, non è una pratica così evitata
- sistema di risoluzione di deadlock
- evitare dinamicamente le situazioni di stallo, di fronte a richiesta il SO prima controlla che non ci siano rischi di situazione critica di stallo (algoritmo del banchiere). C'è una possibilità di deadlock, sistema tuttavia passa attraverso configurazioni sicuri, se non garantito non alloco manco a risorse disponibili

- assicurare che il sistema non possa mai entrare in deadlock negando una delle 4 condizioni.

L'approccio varia ovviamente in base alle situazioni e alle categorie di risorse.

1.3 Ignorare il problema

Soluzione fattibile in determinati contesti, ad esempio nel caso della sistem call fork. Processo che fa fork potrebbe avere dei semafori, perché il processo figlio eredita i semafori del padre.
Altro esempio in cui non viene gestito a priori è la rete ethernet.
Adottato in realtà dalla maggior parte dei sistemi, meglio qualche stallo occasionale che può essere risolto magari con un riavvio

1.4 permettere il deadlock

LAscio che il sistemi entri in deadlock, riconosco e risolvo il deadlock. In questo caso si mantiene un grafo di allocazione delle risorse. Aggiorno ogni volta il grafo man mano che viene aggiornato. Ogni tanto viene lanciato un algoritmo per vedere se si sono cicli, il costo dell'algoritmo è elevato ovvero $O(n^2)$.
Delle volte invece del grafo si preferisce usare delle matrici. Che tipo di strutture dati?
Array E delle ..., array A delle risorse disponibili, matrice C delle risorse allocate, matrice R delle risorse richieste.
(slide)
Siamo in una situazione di deadlock se nello stato corrente non esiste un interleaving di esecuzione che mi porti a terminazione. Algoritmo va a cercare possibile ordine di esecuzione che mi porti a terminazione. Se lo trova non siamo in situazione stallo.
vettore Finish, inizializzato a false, se Finish[i] = false, processo i non può terminare.
confronto riga matrice con vettore A risorse disponibili, sperando trovare processo per soddisfare risorse.
Se trovo processo che soddisfa le richieste, porto finish true, alloco e mi torna risorse.
Se non trovo processo che soddisfa le richieste, non posso terminare.
Problema di questi algoritmi è che sono costosi, ad esempio $O(m * n^2)$.
Inoltre quante volte lanciare l'algoritmo?
Dipende dal sistema.

- ad ogni richiesta da parte di un processo: minimizzo il numero di processi coinvolti nei deadlock e impatto deadlock, ma terribilmente costoso
- ad intervalli regolari
- ad una soglia di utilizzo di cpu, se cpu sotto utilizzata possibile deadlock

Come risolvere? mediante **prerilascio**: cerco risorsa più facilmente interrompibile, difficile da capire e raramente praticabile
rollback:uso di checkpoint nei programmi, in caso di deadlock alcuni programmi coinvolti vengono riportati con dei checkpoint, con conseguente rilascio delle risorse. Difficile scegliere il quantitativo minimo di processi per risolvere lo stallo. Via assai complicata, se blocco tutti i processi può aver senso? quali processi, quanti? (slide presenta dei dubbi)

1.5 Evitare dinamicamente il deadlock

Evitare dinamicamente il deadlock, è possibile decidere se assegnare o meno una risorsa ad un processo?
Stato sicuro: esiste un ordine di esecuzione che mi porta a terminazione.
Partendo da uno stato sicuro, posso verificare se nel caso di allocazione lo stato successivo è ancora sicuro?
Per valutare necessito di alcune informazioni aggiuntive: (slide)

- ogni processo dichiara fin da subito il numero massimo di risorse di ogni tipo di cui avrà bisogno

Nel modello uno si utilizza la matrice C, la matrice R, il vettore A disponibili, matrice M che rappresenta il numero massimo di risorse di ogni tipo che un processo avrà bisogno. Con queste informazioni posso calcolare le traiettorie di risorse.
Stato sicuro: $< P_1, P_2, ..., P_n >$ è una sequenza sicura se posso eseguire il processo nell'ordine indicato con le mie risorse soddisfacendo sempre le richieste senza deadlock. Se uno stato possiede una sequenza sicura allora è uno stato sicuro
slide