Deadlock (Stallo)

Nella programmazione concorrente

Se la spooling area si esaurisce, i processi si bloccano in stallo: ciascuno aspetta che lo spazio di spool aumenti, cosa che puó verificarsi solo se almeno uno finisce di stampare sul disco.
Si supponga che due thread richiedano memoria secondo questo schema:

```
 P1                              P2
.... istruzioni                 .... istruzioni
richiesta di 50 Kbyte           richiesta di 100 Kbyte
....                            ....
richiesta di 70 Kbyte           richiesta di 60 Kbyte
....                            ....
```

Si supponga che siano disponibili 200 Kbyte di memoria.
Primitiva di ricezione messaggi bloccante: blocca il canale.

```
 P1                              P2
.... istruzioni                 .... istruzioni
receive(daP2)                   receive(daP1)
....                            ....
send(aP2,msg1)                  send(aP1,msg2)
....                            ....
```

Nei processi concorrenti

- protezione di una sezione critica

```
     P1
while(true){
    B1=0;
    while(B2==0){};
    SezioneCritica1;
    B1=1;
}
```

```
     P2
while(true)
    B2=0:
    while(B1==0){};
    SezioneCritica2;
    B2=1;
}
```

- problema dei 5 filosofi Tutti i filosofi prendono la forchetta di sinistra

Processi concorrenti

```
class A extends Thread{
     B b;    void init(B x){b=x;}
     public void run(){ funcA(); }
     synchronized void funcA(){
       System.out.println("sono entrato in A"); try{ Thread.sleep(1000);} catch(InterruptedException e){}
       System.out.println("A cerca di chiamare B");   b.s();
     }
     synchronized void s() { System.out.println("s in A"); }
}
class B extends Thread{
     A a;    void init(A x){a=x;}
     public void run(){ funcB(); }
     synchronized void funcB(){
       System.out.println("sono entrato in B"); try{ Thread.sleep(1000);} catch(InterruptedException e){}
       System.out.println("B cerca di chiamare A");   a.s();
     }
     synchronized void s() { System.out.println("s in B"); }
}
class ab2{
     public static void main(String[] a){
        A t1=new A(); B t2=new B(); t1.init(t2); t2.init(t1); t1.start(); t2.start();
        try{t2.join();} catch(InterruptedException e){}  try{t1.join();} catch(InterruptedException e){}
        System.out.println(" ");
     }
}
```

Definizione dello stallo

Attenzione: la seguente definizione assume che i thread richiedano risorse prima della loro esecuzione e le rilasciano quando terminano, non durante!

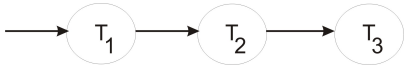
- Sistema fisico: specificato da m tipi di risorse $R_1, R_2, \ldots, R_m$ presenti in $w_1, w_2, \ldots, w_m$ istanze per ogni tipo. La Capacitá del sistema é espressa con il vettore $W = (w_1, w_2, \ldots, w_m)$.
- Vettore delle risorse allocate: $P(k) = (P_1(k), \ldots, P_m(k))$ rappresenta le risorse allocate nel sistema C dopo l'evento a_k in α.
- Vettore delle risorse richieste: $Q(k) = (Q_1(k), \ldots, Q_m(k))$ rappresenta le risorse richieste dal sistema C dopo l'evento a_k in α.
- Vettore delle risorse disponibili: $V(k) = (v_1(k), v_2(k), \ldots, v_m(k))$, dove $v_j(k) = w_j - \sum_i^n P_{ij}(k)$, j=1..m.

Consideriamo n sistemi di Thread $C_1, C_2, \ldots, C_n$, e una combinazione parallela dei thread $C = C_1 \| C_2 \| \ldots \| C_n$.
Lo **Stato s_k del sistema C** é descritto dalle matrici $[P(k), Q(k)]$, dove $P(k) = [P_1(k) \ldots P_n(k)]$ e $Q(k) = [Q_1(k) \ldots Q_n(k)]$.
La creazione o terminazione di sequenze di esecuzione si ottiene aggiungendo o cancellando righe.

Definizione formale di Deadlock Sia $C = C_1 \| C_2 \| \ldots \| C_n$ una combinazione parallela di thread e $\alpha = a_1 \ldots a_k$, $\sigma = s_0 s_1 \ldots s_k$ siano sequenze parziali di esecuzione e di stato. Se $\exists$ un insieme di indici D, tale che per ogni i $\in D$, $Q_i(k)$ non é minore o uguale a $V(k) + \sum_{j \notin D} P_j(k)$ allora s_k contiene un deadlock.

Esempio

R=(R1 R2 R3 R4) W=(3, 3, 3, 3)



Esecuzione sequenziale di tre thread

	q	r
T1	(0 1 2 1)	(0 1 1 0)
T2	(1 2 0 0)	(0 1 1 0
T3	(0 1 1 0)	(1 2 1 1))

Risorse richieste

	$\overline{T_1}$	T_1	$\overline{T_2}$	T_2	$\overline{T_3}$	T_3
Q	(0 1 2 1)	(0 0 0 0)	(1 2 1 0)	(0 0 0 0)	(0 1 1 0)	(0 0 0 0)	(0 0 0 0)
P	(0 0 0 0)	(0 1 2 1)	(0 0 1 1)	(1 2 1 1)	(1 1 0 1)	(1 2 1 1)	(0 0 0 0)
V	(3 3 3 3)	(3 2 1 2)	(3 3 2 2)	(2 1 2 2)	(2 2 3 2)	(2 1 2 2)	(3 3 3 3)

Sequenza di allocazione risorse

Condizioni necessarie

Mutua esclusione: la risorsa non condivisibile. Se occupata, un processo richiedente deve attendere fino a che essa non sia rilasciata

Possesso e attesa: questa cndizione viene anche chiamata **hold&wait**. Un processo puó richiedere (attendere) risorse dopo averne acquisite giá altre

Impossibilitá di prelazione: una risorsa assegnata ad un processo pu essere rilasciata dal processo solo spontaneamente

Attesa circolare: deve esistere un gruppo di processi $(P1, ..., Pn)$ tali che P_1 é in attesa di una risorsa di P_2, P_2 é in attesa di una risorsa di P_3, ... , P_{n-1} é in attesa di una risorsa di P_n, P_n in attesa di una risorsa di P_1.

Grafi di allocazione risorse

I vertici del grafo sono di due tipi:

- $P = P_1, P_2, \ldots P_n$, insieme di tutti i processi nel sistema.
- $R = R_1, R_2, \ldots R_m$, insieme di tutte le risorse esistenti nel sistema.

Gli archi del grafo sono anche di due tipi:

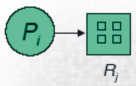
- archi di richiesta risorse: archi orientati Pi $\rightarrow$ Rj
- archi di allocazione risorse: archi orientati Rj $\rightarrow$ Pi

Simboli usati nei grafi allocazione risorse

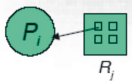
Processo

Risorsa presente in 4 istanze

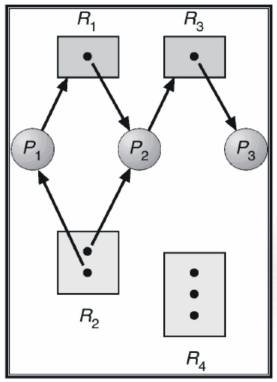
Il processo Pi richiede
una istanza della risorsa Rj



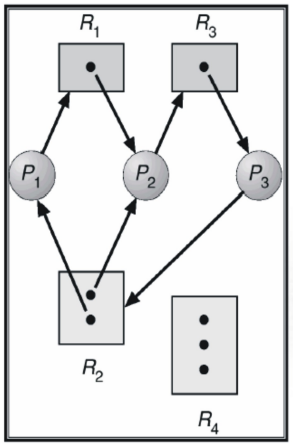
Il processo Pi alloca
una istanza della risorsa Rj



Grafi di allocazione risorse

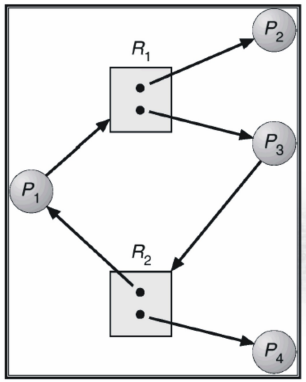


Questo grafo di allocazione risorse non contiene cicli e non si trova in stallo



Questa situazione rappresenta uno stallo: il grafo contiene cicli

Grafi di allocazione risorse



Contiene cicli senza stallo

Riassumendo:

- Se il grafo non contiene cicli $\rightarrow$ non c'é deadlock
- Se il grafo contiene un ciclo:
  - se c' solo una istanza per tipo di risorsa, c'é deadlock
  - se ci sono diverse istanze per tipo di risorsa, c'é la possibilitá di avere deadlock.

Come affrontare il deadlock

- Primo approccio: consentire che il sistema entri in deadlock , rilevare il problema e recuperare la situazione. Rilevare e recuperare.
- Secondo approccio: Evitare i deadlock. Richiede informazioni aggiuntive sulla richiesta delle risorse Decide se soddisfare la richiesta o meno.
- Terzo approccio: prevenire le condizionei di deadlock. Prevenire i deadlock vuol dire Decidere di volta in volta se soddisfare le richieste, Assicurare che almeno una delle condizioni necessarie non sia verificata.
- Quarto approccio: ignorare il problema e recupero manuale

Rilevazione dello stallo

```
Q:risorse richieste; P:risorse allocate; V:risorse disponibili, Deadlock: insie
I={1 2 ... n};
L := {};
repeat
  L' := L;
  for i:=1 to n do
    if i not in L and Qi <= V then
      V := V + Pi;
      L := L U {i};
    end if
  end for
until L = L'; Deadlock := I - L;
```

Una volta che lo stallo é rilevato:

1. La selezione dei processi vittima secondo i seguenti *Criteri di selezione*: Prioritá, Tempo di calcolo, Tipo di risorse occupate
2. terminazione dei processi: totale o incrementale

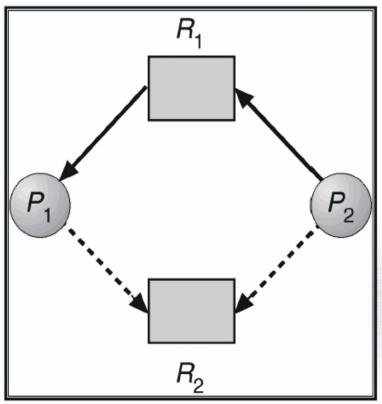
Prevenzione

Negazione di almeno una delle quattro condizioni necessarie

- **Mutua esclusione** Richiesto per risorse non condivisibili mediante utilizzo dello spooling.
- **Hold and Wait**
  - Si richiede che i processi richiedano e allocano le risorse di cui hanno bisogno durante la loro esecuzione prima di cominciare l'esecuzione.
  - Si richiede che un processo richieda risorse solo quando il processo non ha nessuna risorsa allocata.
  - Si ammette che un processo che possiede risorse possa chiederne altre, a patto di rilasciare le prime
- **Non sottrazione** Se un processo che ha allocato qualche risorsa richiede un'altra risorsa che non puó essere immediatamente allocata, allora la risorsa allocata viene rilasciata.
- **Attesa circolare** Si impone un ordinamento totale su tutti i tipi di risorse, e si richiede che ogni processo richieda risorse in ordine crescente di numerazione.

Evitare lo stallo

Ogni processo → massimo numero di risorse per ogni tipo di cui lui puó avere bisogno
Il sistema → massimo numero di risorse per ogni tipo che pu allocare al singolo processo
Sequenza sicura (P1,...,Pn) una sequenza di processi sicura se le risorse di cui necessita Pi sono libere o in possesso di processi Pj con j¡i.
Stato sicuro il sistema in uno stato sicuro di assegnazione di risorse se, quando assegna risorse ai processi, allora essi appartengono ad una sequenza sicura.
Algoritmo basato sul grafo di allocazione risorse per evitare i deadlock
Utilizzabile quando ogni tipo di risorsa ha una sola istanza.
Arco di reclamo Pi → Rj indica che il processo Pj puó richiedere la risorsa Rj.
Una richiesta puó essere soddisfatta se l'allocazione non genera un ciclo nel grafo di allocazione risorse



Grafo per evitare lo stallo

Stati sicuri

Def. Un sistema in uno stato sicuro se esiste una sequenza di schedulazione che pu soddisfare tutte le richieste.
Se un sistema si trova in uno stato sicuro $\rightarrow$ Non ci sono deadlocks. Se un sistema si trova in uno stato non sicuro $\rightarrow$ Possibilitá di deadlock.
Evitare i deadlock vuol dire assicurare che un sistema non entrerá mai in uno stato non sicuro

Algoritmo del banchiere: esamina le richieste massime per vedere se possono essere soddisfatte

Algoritmo del banchiere

Sia n = numero di processi, e m = numero di tipi di risorse.

- **Risorse disponibili V**: Vettore di lunghezza m. Se V[j] = k, ci sono k istanze del tipo di risorsa Rj disponibili.
- **Risorse massime richieste Max**: matrice n x m . Se Max [i,j] = k, il processo Pi puó richiedere al piú k istanze del tipo di risorsa Rj.
- **Risorse allocate P**: matrice n x m. Se P[i,j] = k allora il processo i-esimo ha correntemente allocate k istanze di Rj.
- **Risorse richieste Q**: matrice n x m: **Q[i,j]=Max[i,j]-P[i,j]**

```
I={1 2 ... n};
L := {};
repeat
  L' := L;
  for i:=1 to n do
     if i not in L and Qi <= V then
        V := V + Pi;
        L := L U {i};
     end if
  end for
until L = L'; if I - L = {0} lo stato e' sicuro;
```

Discussione

- Possiamo sapere se un sistema di processi é in stallo o meno con l'algoritmo di rilevazione dello stallo oppure analizzando il grafo di allocazione risorse.
- Con gli stessi metodi possiamo sapere se fornendo una risorsa ad un processo saremo o meno in una situazione di stallo.
- In linea di principio, non sappiamo peró se una situazione nella quale non c'e' stallo é anche sicura, cioé non andrá in stallo in seguito $\rightarrow$ richiesta massima di risorse da parte dei processi, cosa che in generale non é dato a sapere.
- Soluzione di uno stallo: o selezionando uno o piú processi 'vittima' e interrompendo la loro esecuzione oppure, se si conoscono le richieste massime dei processi, utilizzando l'algoritmo del banchiere.
- L'interruzione della esecuzione dei processi vittima puó essere troppo dannosa per il sistema, perché richiede di eseguire l'applicazione dall'inizio $\rightarrow$ CHECKPOINTS. In questo caso l'esecuzione puó ripartire dall'ultimo stato salvato. Il costo é peró molto alto.

In ogni modo, quindi, la gestione dello stallo é una operazione estremamente complessa. Tipicamente la soluzione é demandata al programmatore.