Deadlock

Deadlock

- Descrizione delle situazioni di stallo che impediscono il completamento del lavoro a gruppi di processi concorrenti
- Presentazione di metodi per prevenire/evitare/gestire situazioni di stallo

Il problema del deadlock

- I processi che sono in attesa possono permanere indefinitamente in tale stato se le risorse che hanno richiesto sono in possesso di altri processi a loro volta in attesa.
- Esempio:
  - Il sistema ha 2 nastri.
  - P_1 e P_2 utilizzano ciascuno un nastro e ciascuno di loro richiede anche l'altro.
- Esempio:
  - semafori *A* e *B*, inizializzati a 1.

P_0	P_1
wait (A);	*wait(B)*
wait (B);	*wait(A)*

Deadlock

Un gruppo di processi entra in stallo quando ciascun processo del gruppo attende un evento che può essere causato solo da un altro processo del gruppo

Caratterizzazione del deadlock

Un deadlock si verifica solo se sono soddisfatte simultaneamente quattro condizioni:

- **Mutua esclusione**: soltanto un processo alla volta può usare la risorsa.
- **Possesso e attesa**: un processo che ha almeno una risorsa deve attendere per acquisire ulteriori risorse possedute da altri.
- **Impossibilità di prelazione**: una risorsa può essere liberata soltanto volontariamente dal processo che la possiede.
- **Attesa circolare**: esiste un gruppo di processi $\{P_0, P_1, \ldots, P_n\}$ in attesa di risorse, tali che P_0 sta aspettando una risorsa che è posseduta da P_1, P_1 sta aspettando una risorsa che è posseduta da P_2, …, P_{n-1} sta aspettando una risorsa che è posseduta da P_n, e P_n sta aspettando una risorsa che è posseduta da P_0.

Risorse di sistema e processi

- Tipi di risorse $R_1, R_2, \ldots, R_m$

  cicli di CPU, spazio di memoria, periferiche di I/O, file...

- Ogni risorsa di tipo R_i ha W_i istanze .
- Ogni processo utilizza una risorsa come segue:
  - richiesta
  - uso
  - rilascio

Grafo di allocazione delle risorse

Un insieme di nodi V ed un insieme di archi E.

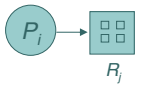
- V è diviso in due tipi di nodi:
  - $P = \{P_1, P_2, \ldots, P_n\}$ che è l'insieme di tutti i processi del sistema.
  - $R = \{R_1, R_2, \ldots, R_m\}$, che è l'insieme di tutti i tipi di risorse del sistema.
- arco di richiesta – arco orientato $P_i \rightarrow R_j$
- arco di assegnazione – arco orientato $R_j \rightarrow P_i$

Grafo di allocazione delle risorse

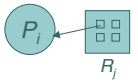
- Processo:

- Tipo di risorsa con 4 istanze:

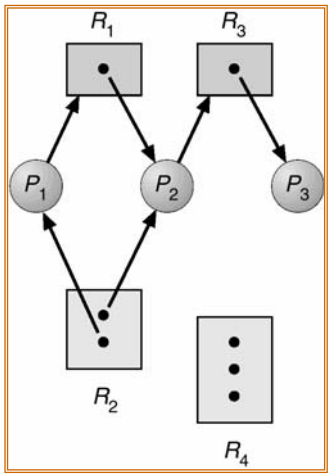
- P_i chiede un'istanza di risorsa di tipo R_j:



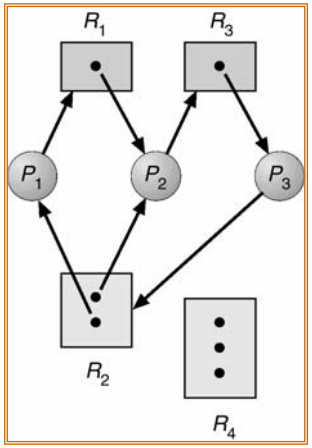
- P_i possiede un'istanza di risorsa di tipo R_j:



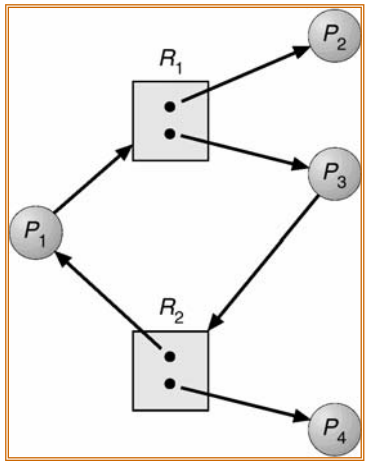
Esempio di grafo di allocazione delle risorse



Grafo di allocazione delle risorse con un deadlock



Grafo di allocazione delle risorse con un ciclo ma nessun deadlock



Punti chiave

- Se il grafo non contiene cicli ⇒ nessun deadlock.
- Se il grafo contiene cicli ⇒
  - se ogni tipo di risorsa ha una sola istanza, allora deadlock.
  - se ogni tipo di risorsa ha più istanze, non è detto che ci sia deadlock.

Metodi di gestione dei deadlock

- Garantire che il sistema non entri mai in uno stato di deadlock.
- Permettere che il sistema entri in uno stato di stallo, ma lo rilevi e recuperi uno stato corretto.
- Ignorare del tutto il problema e fingere che i deadlock non si presentino mai nel sistema; frequente in molti sistemi operativi, compreso UNIX.

Prevenire il deadlock

Restringendo i modi in cui le richieste di risorse possono essere effettuate.

- **Mutua esclusione** – non necessaria per le risorse condivise; deve essere soddisfatta per le risorse non condivisibili.
- **Possesso ed attesa** – bisogna garantire che ogni volta che un processo chiede una risorsa, non possegga già qualche altra risorsa.
  - Richiede che ogni processo chieda e ottenga tutte le risorse prima di iniziare l'esecuzione
  - Permette che un processo chieda le risorse soltanto quando non ne ha altre.

  Basso utilizzo delle risorse; blocco possibile.

Prevenire il deadlock

- **Impossibilità di prelazione**
  1. Se un processo possiede alcune risorse e chiede un'altra risorsa che non può essergli assegnata immediatamente, allora tutte le sue risorse vengono prelazionate.
  2. In alternativa, se il processo che detiene la risorsa è in attesa, la risorsa viene prelazionata e assegnata al richiedente. Altrimenti, il processo richiedente deve attendere (e le sue risorse possono esser prese da altri durante l'attesa)
- **Attesa circolare** – imporre un ordinamento globale di tutti i tipi di risorsa. Ogni processo chiede le risorse in ordine incrementale di numerazione.

Ordinamento globale delle risorse

Associamo ad ogni tipo di risorsa $R = \{R_1, R_2, \ldots, R_m\}$ un intero tramite la funzione F: R →N

F(tape)=1, F(disk drive)=5, F(printer)=9, …

Un processo che detiene R_i, può richiedere un'istanza di R_j se e solo se $F(R_j)>F(R_i)$

Risultato: non può esserci attesa circolare!

Dim. (per assurdo): $P = \{P_0, P_1, \ldots, P_n\}$ tali che $P_i \rightarrow P_{i+1}$

$P_i \rightarrow P_{i+1}$ implica che $F(R_i) < F(R_{i+1})$, per tutti gli i.

$F(R_0) < F(R_1) < \ldots < F(R_n) < F(R_0)$

Evitare il deadlock

Il sistema deve avere informazioni aggiuntive su come verranno richieste le risorse.

- Il modo più semplice richiede che ogni processo dichiari il numero massimo di risorse di cui può averebisogno.
- Un algoritmo per evitare i deadlock esamina dinamicamente lo stato di allocazione delle risorse per garantire che non possa esistere una condizione di attesa circolare.
- Lo stato di allocazione delle risorse è definito dal numero di risorse disponibili e assegnate e dal numero massimo di richieste dei processi.

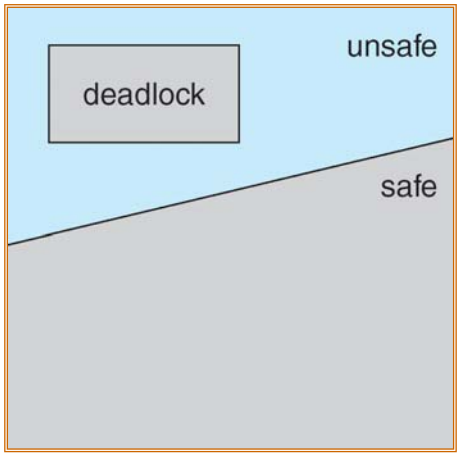
Lo stato sicuro

- Uno stato è sicuro se il sistema può assegnare le risorse per ogni processo in un certo ordine ed evitare il deadlock.
- Un sistema è in uno stato sicuro soltanto se esiste una **sequenza sicura**.
- La sequenza $<P_1, P_2, \ldots, P_n>$ è sicura se per ogni P_i, le richieste di risorse che P_i può attualmente fare possono essere soddisfatte dalle risorse attualmente disponibili + le risorse tenute da tutti i processi P_j, con $j<i$.
  - Se le risorse di cui il processo P_i ha bisogno non sono immediatamente disponibili, allora P_i aspetta finchè tutti i P_j hanno finito.
  - Quando hanno finito, P_i può ottenere tutte le risorse necessarie, completare il suo compito, restituire le risorse assegnate e terminare.

Punti chiave

- Se il sistema è in uno stato sicuro ⇒ no deadlock.
- Se il sistema è in uno stato non sicuro ⇒ possibilità di deadlock.
- Per evitare (deadlock) ⇒ assicurarsi che il sistema non entri mai in uno stato non sicuro.

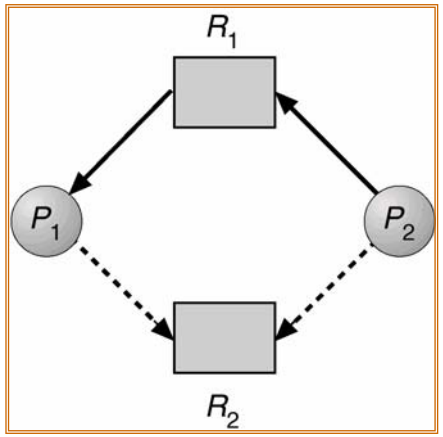
Spazi di stato sicuro, non sicuro e di deadlock



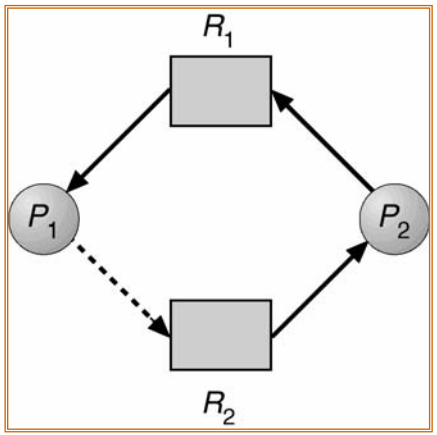
Algoritmo del grafo di allocazione delle risorse

- Arco di prenotazione P_i ---> R_j indica che il processo P_j può chiedere la risorsa R_j; rappresentato da una linea tratteggiata.
- Quando un processo chiede una risorsa, l'arco di prenotazione è convertito in un arco di richiesta.
- Quando una risorsa è liberata dal processo, l'arco di assegnazione è convertito in un arco di prenotazione.
- Le risorse devono essere reclamate a priori nel sistema.

Grafo di allocazione delle risorse per evitare i deadlock



Stato non sicuro in un grafo di allocazione delle risorse



L'algoritmo del banchiere

- Istanze multiple.
- Ogni processo deve dichiarare il numero massimo di istanze per ogni tipo di risorsa di cui può avere bisogno.
- Quando un processo richiede delle risorse potrebbe dover attendere.
- Quando un processo ottiene tutte le sue risorse deve restituirle in un periodo di tempo finito.

Strutture dati dell'algoritmo del banchiere

n = numero dei processi m = numero di tipi di risorse.

- ***Available:*** vettore di lunghezza m. Se Available $[j] = k$, ci sono k istanze di risorse di tipo R_j disponibili.
- ***Max:*** una matrice $n \times m$. Se $Max\ [i,j] = k$, allora il processo P_i può chiedere al più k istanze di risorse di tipo R_j.
- ***Allocation:*** una matrice n x m. Se $Allocation[i,j] = k$ allora al processo P_i sono attualmente assegnate k istanze di risorsa del tipo R_j.
- ***Need:*** matrice n x m. Se $Need[i,j] = k$, allora il processo P_i può avere bisogno di altre k istanze di risorse del tipo R_j per completare il suo compito.

$$Need\ [i,j] = Max[i,j] - Allocation\ [i,j].$$

Stato sicuro – algoritmo di verifica

1. Siano *Work e Finish* vettori rispettivamente di lunghezza m e n. Inizializziamo:
   $Work = Available$
   $Finish\ [i] = false\ per\ i = 1,2,3, \ldots, n.$
2. Si cerca una i tale che:
   (a) $Finish\ [i] == false$
   (b) $Need_i \leq Work$
   Se non esiste una tale i, passare al punto 4.
3. $Work = Work + Allocation_i$
   $Finish[i] = true$
   passare al punto 2.
4. Se, per ogni i, $Finish\ [i] ==$ true, allora il sistema è in uno stato sicuro.

L'algoritmo di richiesta delle risorse per il processo P_i

***Request* = vettore di richiesta per il processo P_i.**
Se $Request_i[j] = k$ allora il processo P_i vuole k istanze della risorsa di tipo R_j.

1. **Se $Request_i \leq Need_i$ *passare al punto* 2. Altrimenti sollevare una condizione di errore.**
2. **Se $Request_i \leq Available$, passare al punto 3. Altrimenti P_i deve attendere.**
3. **Il sistema assegna al processo P_i le risorse richieste modificando lo stato nel modo seguente:**

   $Available = Available - Request_i;$
   $Allocation_i = Allocation_i + Request_i;$
   $Need_i = Need_i - Request_{i;;}$

   - ***Se sicuro $\Rightarrow$ le risorse vengono assegnate a P_i.***
   - ***Se non sicuro $\Rightarrow$ P_i deve aspettare, e viene ristabilito il vecchio stato di allocazione delle risorse.***

Esempio dell'algoritmo del banchiere

- Si consideri un sistema con 5 processi da P_0 a P_4 e 3 tipi di risorse:
  - A – 10 istanze,
  - B – 5 istanze,
  - C – 7 istanze.
- Supponiamo che all'istante T_0 ci sia la seguente situazione del sistema:

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

Esempio dell'algoritmo del banchiere

- Il contenuto della matrice *Need* è definito come Max – Allocation ed è:

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

- Il sistema è in uno stato sicuro poichè la sequenza < P_1, P_3, P_4, P_2, P_0> soddisfa i criteri di sicurezza.

Esempio di richiesta di P_1 (1,0,2)

- Controlliamo che (1,0,2) ≤ (1,1,2) e che (1,0,2) ≤ (3,3,2)

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

- L'esecuzione dell'algoritmo di sicurezza mostra che la sequenza <P_1, P_3, P_4, P_0, P_2> soddisfa i criteri di sicurezza.
- La richiesta per (3,3,0) da parte di P_4 può essere soddisfatta?
- La richiesta per (0,2,0) da parte di P_0 può essere soddisfatta?

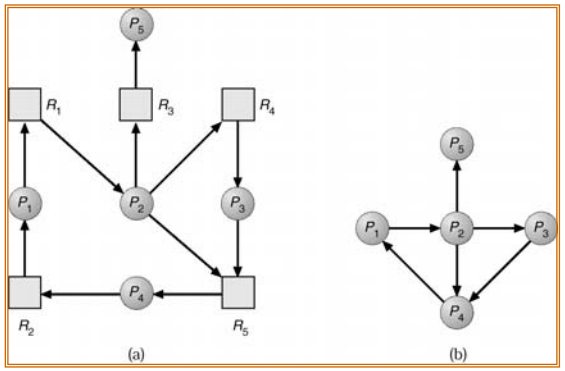
Rilevazione del deadlock

- Permettere al sistema di entrare in uno stato di deadlock
- Algoritmo di rilevazione
- Schema di recupero

Singola istanza di ogni tipo di risorsa

- Mantiene un *grafo di attesa* (wait-for graph).
  - I nodi risorse sono rimossi.
  - $P_i \rightarrow P_j$ se P_i sta aspettando P_j.
- Il sistema deve invocare periodicamente un algoritmo che cerca un ciclo nel grafo.
- Un algoritmo per rilevare un ciclo in un grafo richiede n^2 operazioni, dove n è il numero di nodi del grafo.

Grafo di allocazione delle risorse e corrispondente grafo di attesa



Grafo di allocazione delle risorse Corrispondente grafo di attesa

Istanze multiple di un tipo di risorsa

- ***Available:*** un vettore di lunghezza *m* indica il numero di risorse disponibili per ogni tipo.
- ***Allocation:*** una matrice *n* x *m* definisce il numero di risorse di ogni tipo attualmente assegnate a ogni processo.
- ***Request:*** una matrice *n* x *m* matrix indica la richiesta corrente di ogni processo. Se *Request* [i,j] = *k*, allora il processo P_i richiede altre *k* istanze della risorsa di tipo R_j.

Algoritmo di rilevamento

1. Supponiamo che *Work* e *Finish* siano rispettivamente vettori di lunghezza *m* e *n*. Inizializziamo:
   (a) *Work* = *Available*
   (b) for *i* = 1,2, …, *n*, if $Allocation_i \neq 0$, then *Finish*[i] = false; else *Finish*[i] = *true*
2. Sia *i* un indice *tale* che :
   (a) *Finish*[*i*] == *false*
   (b) $Request_i \leq Work$

   Se un tale *i* non esiste, passare al punto 4

Algoritmo di rilevamento

3. *Work* = *Work* + $Allocation_i$
   Finish[*i*] = *true*
   passare al punto 2
4. If *Finish*[*i*] == false, per qualche *i*, $1 \leq i \leq n$, allora il sistema è in uno stato di deadlock. Inoltre, se *Finish*[*i*] == *false*, allora il processo P_i è in stallo

Questo algoritmo richiede *m* x n^2 operazioni per rilevare se il sistema è finito in un deadlock.

Esempio di algoritmo di rilevazione

- Consideriamo un sistema con 5 processi da P_0 a P_4 e 3 tipi di risorse:
  - A (7 istanze),
  - *B* (2 istanze),
  - *C* (6 istanze).
- Supponiamo che all'istante T_0 si abbia il seguente stato di allocazione delle risorse:

	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	

- La sequenza <P_0, P_2, P_3, P_1, P_4> provoca *Finish*[*i*] = true per ogni *i*.

Algoritmo di rilevamento

- Supponiamo che P_2 faccia una richiesta supplementare per un'istanza di tipo C:

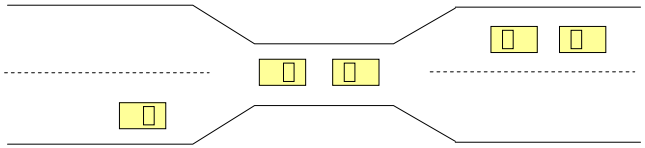
	Request	Available
	A B C	A B C
P_0	0 0 0	0 0 0
P_1	2 0 2	
P_2	0 0 1	
P_3	1 0 0	
P_4	0 0 2	

- Stato del sistema?
  - Anche se possiamo riprendere le risorse del processo P_0, il numero di risorse disponibili non è sufficiente per soddisfare le richieste degli altri processi.
  - C'è uno stallo, formato dai processi P_1, P_2, P_3, e P_4.

Uso dell'algoritmo di rilevazione

- Quando, e con quale frequenza, bisognerebbe invocare l'algoritmo di rilevazione dipende da:
  - Con quale frequenza può accadere un deadlock?
  - Quanti processi saranno coinvolti nel deadlock quando questo si verificherà?
    - uno per ogni ciclo disgiunto.
- Se l'algoritmo di rilevazione è invocato in momenti casuali possono essere trovati molti cicli nel grafo di risorsa. In generale potremmo non essere in grado di dire quale dei molti processi in deadlock ha causato il deadlock.

Esempio dell'incrocio sul ponte



- Se si verifica un deadlock, può essere risolto se una macchina torna indietro.
- Molte macchine potrebbero dover tornare indietro.
- Il blocco (starvation) è possibile.

Ripristino del deadlock: termine del processo

- Abortire tutti i processi in deadlock.
- Abortire un processo alla volta fino ad eliminare il ciclo di deadlock.
- In quale ordine devono essere i processi da abortire?
  - Priorità del processo.
  - Per quanto tempo il processo ha elaborato e per quanto tempo ancora il processo proseguirà prima di completare l'operazione pianificata.
  - Quante e quali tipi di risorse sono state usate dal processo.
  - Quante altre risorse il processo necessita per completare l'elaborazione.
  - Quanti processi devono essere terminati.
  - Sapere se il processo è interattivo o batch.

Ripristino del deadlock: rilascio anticipato delle risorse

- Selezione della vittima – minimizzarne il costo.
- Rollback – ritornare ad uno stato sicuro e far ripartire il processo da quello stato.
- Starvation – in un sistema in cui la selezione della vittima è basata soprattutto sui fattori costo, alcuni processi potrebbero essere sempre scelti come vittime.

Approccio combinato alla gestione dei deadlock

- Combinare i tre approcci di base:
  - prevenire
  - evitare
  - rilevare

  Questi approcci permettono l'uso ottimale di ogni risorsa nel sistema.
- Ripartizione delle risorse in una classe ordinata gerarchicamente.
- Uso della tecnica più appropriata per la gestione dei deadlock.