Stallo di processi

Tecniche per evitare uno stallo

Stefano Quer
Dipartimento di Automatica e Informatica
Politecnico di Torino

Evitare il deadlock

- Le tecniche per **evitare** condizioni di stallo
  - Forzano i P a fornire (a priori) informazioni aggiuntive sulle richieste che effettueranno nel corso della loro esistenza
    - Ogni P deve indicare quali e quante risorse di ogni tipo saranno necessarie per terminare il suo compito
    - Tali informazioni permetteranno di schedulare i processi in modo che non si verifichino deadlock
      - Se l'esecuzione di un processo può causare deadlock essa viene ritardata opportunamente

Evitare il deadlock

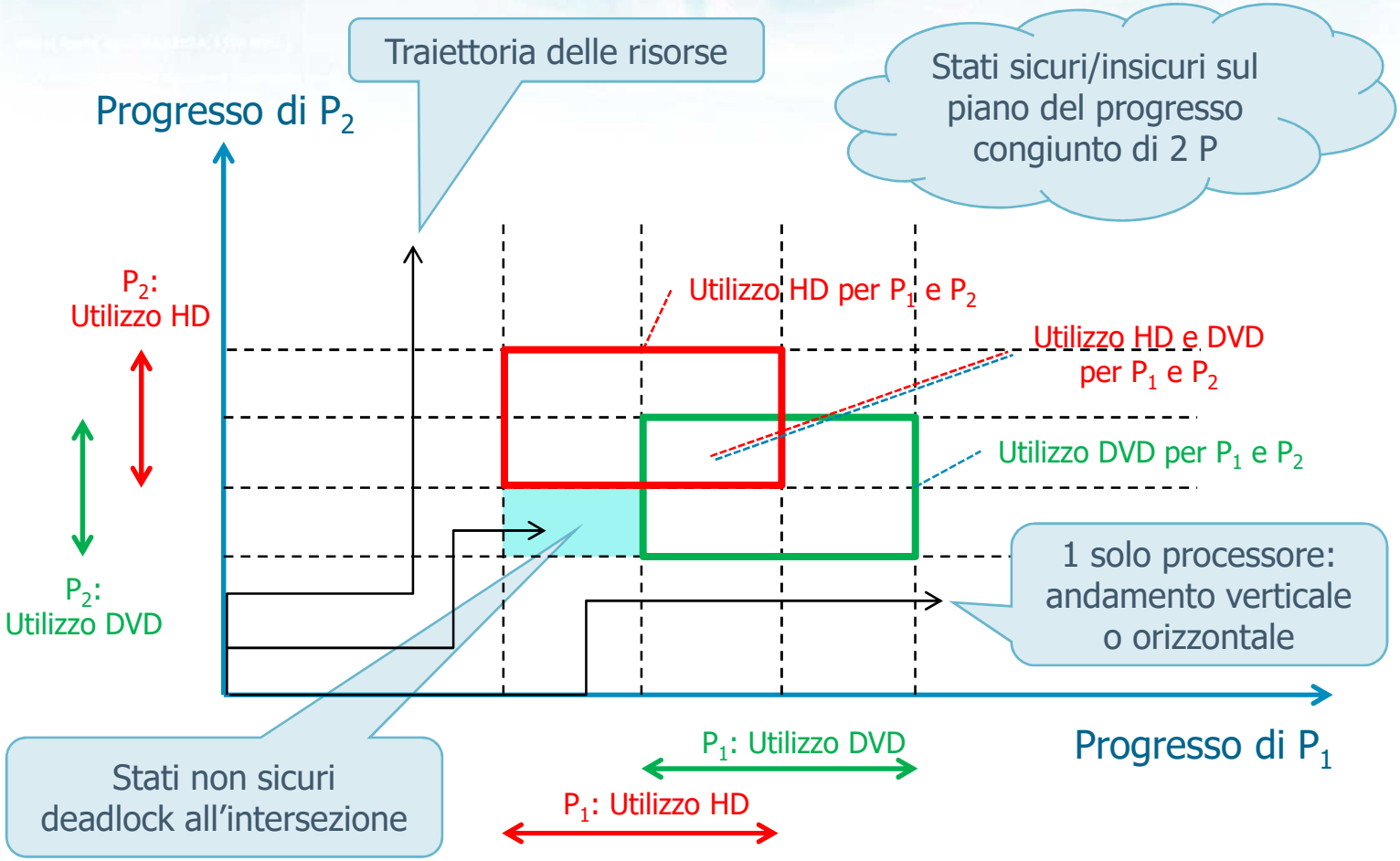
- I principali algoritmi
  - Si differenziano per la quantità e il tipo di informazioni richieste
    - Il modello più semplice si basa sul forzare tutti i processi a dichiarare il massimo numero di risorse di ciascun tipo di cui il processo avrà bisogno
  - In genere provocano una riduzione nell'utilizzo delle risorse e una minore efficienza del sistema
  - Si basano sul concetto di **stato sicuro** e di **sequenza sicura**

Stato sicuro

Stato sicuro	Il sistema è in grado di • Allocare le risorse richieste a tutti i processi in esecuzione • Impedire il verificarsi di uno stallo • Trovare una sequenza sicura
Sequenza sicura	Una sequenza di schedulazione dei processi $\{P_1, P_2, \ldots, P_n\}$ tale che per ogni P_i le richieste che esso può ancora effettuare possono essere soddisfatte impiegando le risorse attualmente disponibili più le risorse liberate dai processi P_j con $j<i$

Uno stato si dice **non sicuro** in caso contrario.
Non necessariamente uno stato non sicuro è uno stato di stallo: condurrà in uno stato di stallo in caso di comportamento standard.

Progresso congiunto di due processi



Strategie

- Per evitare uno stallo ci si assicura che il sistema rimanga sempre in uno stato sicuro
  - All'inizio il sistema è in uno stato sicuro
  - Ogni richiesta di nuova risorsa
    - Sarà soddisfatta se lascerà il sistema in uno stato sicuro
    - Sarà ritardata in caso contrario; il processo che ha effettuato la richiesta sarà posto in attesa
- Esistono due classi di strategie
  - Con risorse aventi istanze unitarie
  - Con risorse aventi istanze multiple

Algoritmo per istanze unitarie

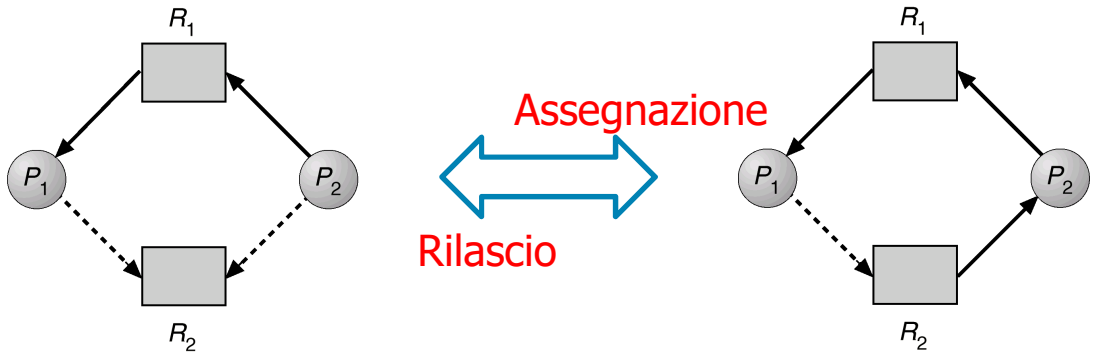
- Sono basate sulla determinazione di cicli, gestendo il grafo di rivendicazione
  - Tutte le richieste che saranno effettuate **devono** essere dichiarate a priori, ovvero all'inizio dell'esecuzione
    - Esse sono indicate come archi di reclamo sul grafo di rivendicazione
  - Nel momento in cui una richiesta viene effettuata
    - Il corrispondente arco di reclamo **dovrebbe** essere trasformato in un arco di assegnazione
    - Prima di effettuare tale trasformazione si verifica però se soddisfacendola si generano dei cicli

Algoritmo per istanze unitarie

- Se il grafo non presenterà comunque alcun ciclo, la conversione viene effettuata e la risorsa assegnata
- In caso negativo, l'assegnazione della risorsa richiesta porterebbe il sistema in uno stato non sicuro e quindi viene rimandata

➢ Quando una risorsa viene rilasciata

- L'arco di assegnazione ritorna a essere un arco di reclamo (per gestire eventuali richieste successive)



Algoritmo per istanze multiple

- Valutano lo stato del sistema per comprendere se le risorse disponibili sono sufficienti per portare a termine tutti i P
  - Si basano sul numero di risorse disponibili, assegnate e massimo richiesto
- Ogni processo
  - Deve dichiarare a priori il massimo uso di risorse
  - Quando richiede una risorsa può essere bloccato
  - Quando ottiene le risorse che gli servono deve garantire che le restituirà in un tempo finito

Algoritmo per istanze multiple

- Algoritmo del Banchiere (Dijkstra, [1965])
  - È costituito da **due** sezioni
    - La prima verifica che uno stato sia sicuro
    - La seconda verifica che una nuova richiesta possa essere soddisfatta permettendo al sistema di rimanere in uno stato sicuro
- L'algoritmo utilizza le strutture dati elencate di seguito

Algoritmo per istanze multiple

Siano dati:
- Un insieme di processi P_r con cardinalità n
- Un insieme di risorse R_c con cardinalità m

Nome	Dim.	Contenuto e significato
Fine	[n]	Fine[r]=false indica che P_r non ha terminato
Assegnate	[n][m]	Assegnate[r][c]=k P_r possiede k istanze di R_c
Massimo	[n][m]	Massimo[r][c]=k P_r può richiedere al massimo k istanze di R_c
Necessità	[n][m]	Necessità[r][c]=k P_r ha bisogno di altre k istanze di R_c $\forall i \forall j$ Necessità[i][j]=Massimo[i][j]-Assegnate[i][j]
Disponibili	[m]	Disponibili[c]=k disponibilità pari a k per R_c

Esempio

- Applicando l'algoritmo del banchiere il sistema sottostante si trova in uno stato sicuro?
  - Sequenza sicura: P_1, P_3, P_0, P_2, P_4

P	Fine	Assegnate	Massimo	Necessità	Disponibili
		R_0 R_1 R_2	R_0 R_1 R_2	R_0 R_1 R_2	R_0 R_1 R_2
P_0	F	0 1 0	7 5 3		3 3 2
P_1	F	2 0 0	3 2 2		
P_2	F	3 0 2	9 0 2		
P_3	F	2 1 1	2 2 2		
P_4	F	0 0 2	4 3 3		

Esempio

- La richiesta di P_1 (1, 0, 2) può essere soddisfatta?
  - Si …
  - Nuovo stato del sistema …

P	Fine	Assegnate	Massimo	Necessità	Disponibili
		R_0 R_1 R_2	R_0 R_1 R_2	R_0 R_1 R_2	R_0 R_1 R_2
P_0	F	0 1 0	7 5 3	7 4 3	3 3 2
P_1	F	2 0 0	3 2 2	1 2 2	
P_2	F	3 0 2	9 0 2	6 0 0	
P_3	F	2 1 1	2 2 2	0 1 1	
P_4	F	0 0 2	4 3 3	4 3 1	

Esempio

- Il nuovo stato è sicuro?
  - Sequenza sicura: P_1, P_3, P_0, P_4, P_2

P	Fine	Assegnate	Massimo	Necessità	Disponibili
		R_0 R_1 R_2	R_0 R_1 R_2	R_0 R_1 R_2	R_0 R_1 R_2
P_0	F	0 1 0	7 5 3	7 4 3	2 3 0
P_1	F	3 0 2	3 2 2	0 2 0	
P_2	F	3 0 2	9 0 2	6 0 0	
P_3	F	2 1 1	2 2 2	0 1 1	
P_4	F	0 0 2	4 3 3	4 3 1	

Esercizio

Stesso stato iniziale

- La richiesta P_4 (3, 3, 0) può essere soddisfatta?
  - No ... non c'è disponibilità
- La richiesta P_0 (0, 3, 0) può essere soddisfatta?
  - No ... lo stato risultante non è sicuro

P	Fine	Assegnate	Massimo	Necessità	Disponibili
		R_0 R_1 R_2	R_0 R_1 R_2	R_0 R_1 R_2	R_0 R_1 R_2
P_0	F	0 1 0	7 5 3	7 4 3	2 3 0
P_1	F	3 0 2	3 2 2	0 2 0	
P_2	F	3 0 2	9 0 2	6 0 0	
P_3	F	2 1 1	2 2 2	0 1 1	
P_4	F	0 0 2	4 3 3	4 3 1	

Algoritmo del banchiere

- Verifica di una richiesta da parte di P_i

```
se
 ∀j Richieste[i][j]≤Necessità[i][j]
 AND
 ∀j Richieste[i][j]≤Disponibili[j]
 ALLORA
   ∀j Disponibili[j]=Disponibili[j]-Richieste[i][j]
   ∀j Assegnate[i][j]=Assegnate[i][j]+Richieste[i][j]
   ∀j Necessità[i][j]=Necessità[i][j]-Richieste[i][j]

se lo stato risultante è sicuro
  si conferma tale assegnazione
  altrimenti si ripristina lo stato precedente
```

Algoritmo del banchiere

- Verifica di uno stato

```
1.
∀i∀j Necessita[i][j]=Massimo[i][j]-Assegnate[i][j]
∀i Fine[i]=false

2.
Trova i per cui
Fine[i]=falso AND ∀j Necessità[i][j]≤Disponibili[j]
Se tale i non esiste goto step 4

3.
∀j Disponibili[j]=Disponibili[j]+Assegnate[i][j]
Fine[i]=true
goto step 2

4.
Se ∀i Fine[i]=true il sistema è in uno stato sicuro
```

Esercizio

- La richiesta P_1 (1, 0, 1) può essere soddisfatta?
  - Si …
- La richiesta P_2 (1, 0, 1) può essere soddisfatta?
  - No … lo stato risultante non è sicuro

P	Fine	Assegnate	Massimo	Necessità	Disponibili
		R_0 R_1 R_2	R_0 R_1 R_2	R_0 R_1 R_2	R_0 R_1 R_2
P_0	F	1 0 0	3 2 2		1 1 2
P_1	F	5 1 1	6 1 3		
P_2	F	2 1 1	3 1 4		
P_3	F	0 0 2	4 2 2		

Esercizio

- I seguenti due stati sono sicuri o non sicuri?

(problemi risorsa unica)

P	F	A	M	N	D
P_0	F	3	9		3
P_1	F	2	4		
P_2	F	2	7		

... stato sicuro

P	F	A	M	N	D
P_0	F	4	9		2
P_1	F	2	4		
P_2	F	2	7		

... stato non sicuro

Algoritmo del banchiere

- La complessità dell'algoritmo del banchiere è
  - $O(m \cdot n^2) = O(|R| \cdot |P|^2)$
- Si basa inoltre su ipotesi poco realistiche
  - I processi devono indicare le richieste in anticipo
    - Le risorse necessarie non sempre sono note
    - Inoltre non è noto quando saranno necessarie
  - Suppone le risorse siano in numero costante
    - Le risorse possono aumentare o ridursi a causa di guasti temporanei o duraturi
  - Richiede una popolazione fissa di processi
    - I processi attivi nel sistema aumentano e si riduco in maniera dinamica