
Parte III

Deadlock

Deadlock (stallo)

- Su di un tavolo ci sono un piatto ed una forchetta
- **A** e **B** sono seduti al tavolo, per mangiare ciascuno dei due ha bisogno sia del piatto che della forchetta
- Sciagurata sequenza di eventi:
 1. **A** prende la forchetta
 2. **B** prende il piatto
 3. Essendo il piatto occupato, **A** *si pone in attesa*
 4. Essendo la forchetta occupata, **B** *si pone in attesa*
 5. **A** e **B** attendono per sempre, e muoiono di fame
- Si è verificata una situazione di *deadlock*

Risorse

- Il deadlock si verifica quando i processi possono acquisire e detenere in maniera esclusiva le risorse
- Sono risorse, la CPU la memoria (centrale e di massa), e tutti i dispositivi
- Risorse di due tipi:
 - **Rilasciabili** (*preemptable*): possono essere sottratte al processo senza pregiudicarne l'esecuzione
 - **Non Rilasciabili** (*nonpreemptable*): se sottratte causano il fallimento dell'esecuzione del processo

Uso delle risorse

- Sequenza di azioni nell'uso di una risorsa:
 1. *Richiesta della risorsa*
 2. *Uso della risorsa*
 3. *Rilascio della risorsa*
- Se al passo 1. la risorsa non è disponibile, il processo può essere messo in attesa, o la sua esecuzione può essere terminata (a seconda dei casi)
- Si suppone ciascun processo detenga la risorsa per un tempo *finito*

Definizione

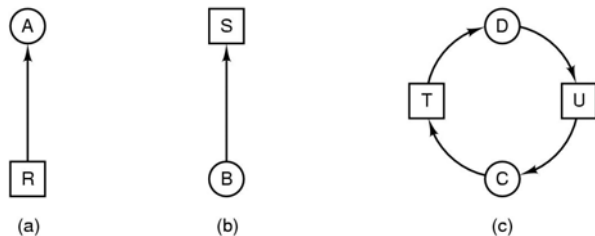
Un insieme di processi si dice in *deadlock* se ciascuno dei processi è in attesa di un evento che solo qualcuno degli altri può causare

- Tipicamente l'*evento* è il rilascio di una risorsa
- La situazione è di stallo: nessuno dei processi può:
 - Continuare l'esecuzione
 - Rilasciare una risorsa
 - Essere risvegliato

Condizioni necessarie

- È stato dimostrato che per il *deadlock* sussistono le seguenti condizioni necessarie:
 1. Mutual exclusion: alcune risorse vengono assegnate in uso esclusivo
 2. Hold and wait: processi che detengono risorse possono richiederne altre
 3. No preemption: non è possibile sottrarre risorse già assegnate
 4. Circular wait: si deve essere creata una situazione di attesa circolare
- Se c'è un *deadlock* tutte queste condizioni sono sempre soddisfatte

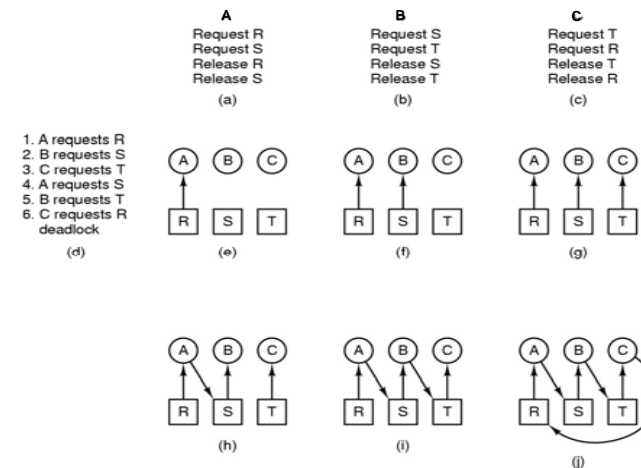
Modelli per il deadlock



Modello basato su grafi orientati:

- (a) Il processo **A** detiene la risorsa **R**
- (b) Il processo **B** richiede la risorsa **S**
- (c) I processi **C** e **D** sono in *deadlock* (attesa circolare)

Esempio di deadlock



Gestione del deadlock

Diversi approcci possibili

1. Ignorare: fare finta che il problema non esista
2. Rilevare e risolvere: rilevare il deadlock e rimuovere una delle cause
3. Evitare: adottare una politica accorta di allocazione delle risorse
4. Prevenire: fare in modo che una delle condizioni necessarie non si verifichi mai

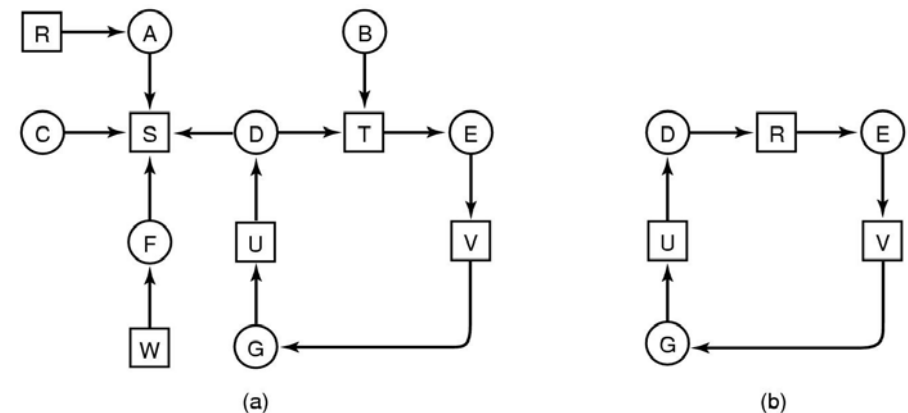
La filosofia dello struzzo

- Si fa finta che il problema non esista
- Se il deadlock si verifica il sistema va in crash
- È un modo serio e ragionevole di comportarsi?
- Sì se il costo della risoluzione del deadlock supera statisticamente quello dei danni causati
- Comportamento adottato da alcuni SO per quelle risorse che possono essere causa di deadlock con probabilità molto bassa
- Usato da molti sistemi per basi dati (DBMS)

Rilevazione (risorse singole)

- Si mantiene il grafo di allocazione, un nodo per ogni processo e per ogni risorsa
- Ogni volta che un processo **P** richiede una risorsa **R**
 - a) se disponibile, si aggiunge un arco da **R** a **P**
 - b) se occupata, si aggiunge un arco da **P** a **R**
 - c) Quando **R** si libera si inverte l'arco
- Ogni volta che un processo **P** rilascia una risorsa **R**
 - d) si rimuove l'arco da **R** a **P**
- A ciascun passo del tipo a), b) o c) si controlla se il grafo è ciclico: i cicli rappresentano deadlock

Rilevazione: esempio

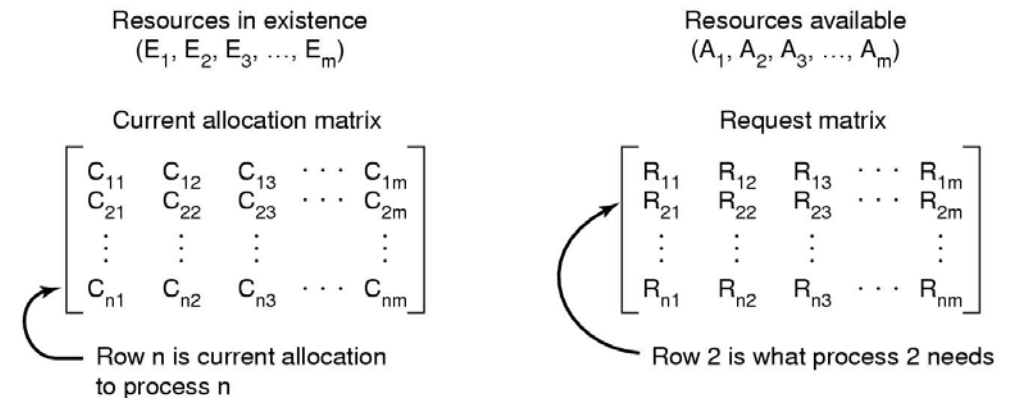


Rilevazione (risorse multiple)



- Sono presenti n processi e m tipi risorse
- Per ogni tipo di risorsa sono disponibili più copie
- Si definiscono le seguenti quantità:
 - E_r : numero di copie esistenti della risorsa r
 - A_r : numero di copie disponibili della risorsa r
 - $C_{p,r}$: copie della risorsa r allocate al processo p
 - $R_{p,r}$: numero di ulteriori copie della risorsa r necessarie al processo p per terminare

Matrici di allocazione



Algoritmo di rilevazione

- Si ripete il controllo ad ogni richiesta od allocazione
- Si *individuano* e si *marcano* i processi che sicuramente possono terminare, ripetendo i seguenti passi:
 1. Cerca un processo k non marcato per cui $R_{k,i} \leq A_i, \forall i$
 2. Marca il processo k e poni $A_i = A_i + C_{k,i}, \forall i$
- I processi per cui la condizione del passo 1. è soddisfatta possono (teoricamente) terminare perché le risorse necessarie sono ancora disponibili
- Se terminano rilasciano le risorse attualmente detenute
- I processi che restano non marcati alla fine dell'algoritmo sono in deadlock

Algoritmo di rilevazione : esempio

Tape drives
Plotters
Scanners
CD Roms

 $E = (4 \quad 2 \quad 3 \quad 1)$

Tape drives
Plotters
Scanners
CD Roms

 $A = (2 \quad 1 \quad 0 \quad 0)$

Current allocation matrix

$$C = \begin{bmatrix} 0 & 0 & 1 & 0 \\ 2 & 0 & 0 & 1 \\ 0 & 1 & 2 & 0 \end{bmatrix}$$

Request matrix

$$R = \begin{bmatrix} 2 & 0 & 0 & 1 \\ 1 & 0 & 1 & 0 \\ 2 & 1 & 0 & 0 \end{bmatrix}$$

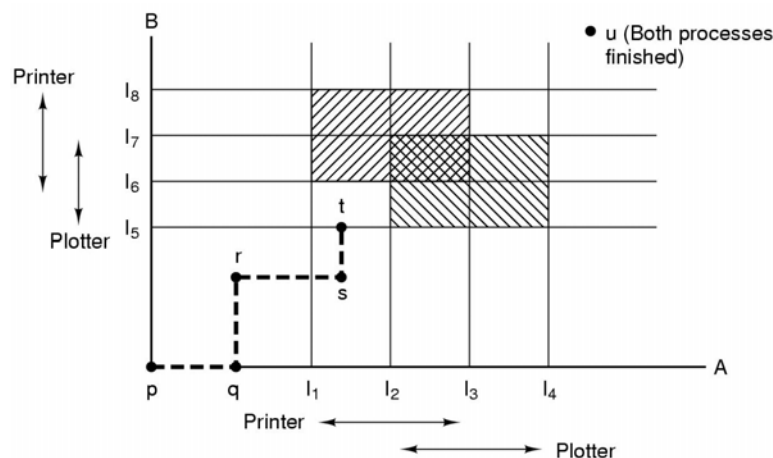
Risoluzione del deadlock

- Prerilascio
 - La risorsa viene sottratta ad un altro processo
 - Possibile solo per alcune risorse
- Rollback
 - Salvataggio periodico dello stato dei processi
 - I processi in deadlock vengono *riportati indietro*
- Eliminazione
 - Un processo viene eliminato in modo da rendere il grafo aciclico
 - Deve essere fatto ripartire dall'inizio (*restart*)

Evitare il deadlock

- Si basa sul concetto di *stato sicuro*
- *Stato sicuro* è uno stato di allocazione delle risorse dal quale è possibile *uscire*
- *Uscire* significa che esiste una sequenza di allocazione che permette a tutti i processi di terminare
- Il SO prima di allocare una risorsa si deve accertare che il nuovo stato, dopo l'allocazione, sia sicuro
- Ad esempio controllando che il grafo risultante sia aciclico

Traiettorie



Traiettorie

- I processi **A** e **B** richiedono entrambi sia la stampante che il plotter
- Ogni punto nel grafico rappresenta uno stato congiunto di **A** e **B**
- l_1, l_2, \dots Istruzioni dei processi **A** e **B**
- Le zone tratteggiate rappresentano *situazioni impossibili*: processi detengono la stessa risorsa
- Alcune zone bianche limitrofe rappresentano stati *non sicuri*, cioè senza via d'uscita
- Es: il rettangolo $l_1 - l_2, l_5 - l_6$

L'algoritmo del banchiere

- Il banchiere dispone di un certo capitale complessivo
- I clienti chiedono prestiti a più riprese
- Ciascun cliente ha limite massimo di credito
- Il cliente garantisce prima o poi la restituzione se gli viene concesso il suo credito massimo
- Quando un cliente chiede denaro, il banchiere controlla se dandoglielo continua a rimanere in uno stato sicuro
- Altrimenti pone il cliente in attesa
- L'aumento del prestito sarà concesso prima o poi, quando gli altri clienti avranno restituito i loro prestiti

Stati sicuri: esempio

	Has Max			Has Max			Has Max			Has Max			Has Max		
A	3	9		A	3	9	A	3	9	A	3	9	A	3	9
B	2	4		B	4	4	B	0	-	B	0	-	B	0	-
C	2	7		C	2	7	C	2	7	C	7	7	C	0	-
	Free: 3 (a)			Free: 1 (b)			Free: 5 (c)			Free: 0 (d)			Free: 7 (e)		

- Un solo tipo di risorsa presente in 10 esemplari
- Lo stato di partenza è sicuro perché esiste una sequenza di allocazione e rilasci che fa terminare tutti i processi
- La sequenza è **B, C, A**

Stati non sicuri: esempio

	Has Max			Has Max			Has Max			Has Max		
A	3	9		A	4	9	A	4	9	A	4	9
B	2	4		B	2	4	B	4	4	B	-	-
C	2	7		C	2	7	C	2	7	C	2	7
	Free: 3 (a)			Free: 2 (b)			Free: 0 (c)			Free: 4 (d)		

- Il primo stato è sicuro, ma il secondo non lo è
- L'allocazione di una ulteriore unità ad **A** fa sì che né **A** né **C** possano terminare
- **B** può terminare, ma le quattro unità che rilascia non bastano né per **A** né per **C**

L'algoritmo del banchiere: esempio

	Has Max			Has Max			Has Max		
A	0	6		A	1	6	A	1	6
B	0	5		B	1	5	B	2	5
C	0	4		C	2	4	C	2	4
D	0	7		D	4	7	D	4	7
	Free: 10 (a)			Free: 2 (b)			Free: 1 (c)		

- Lo stato di partenza **(a)** è sicuro
- Lo stato **(b)** è sicuro: sequenze (C,D,B,A; C,B,D,A; C,B,A,D)
- Se B chiede l'aumento di 1 il banchiere lo nega, poiché lo stato **(c)** non è sicuro: nessuna sequenza possibile

Banchiere con risorse multiple

	Process	Tape drives	Plotters	Scanners	CD ROMs
A	3	0	1	1	
B	0	1	0	0	
C	1	1	1	0	
D	1	1	0	1	
E	0	0	0	0	

Resources assigned

	Process	Tape drives	Plotters	Scanners	CD ROMs
A	1	1	0	0	
B	0	1	1	2	
C	3	1	0	0	
D	0	0	1	0	
E	2	1	1	0	

Resources still needed

E = (6342)
P = (5322)
A = (1020)

- Le matrici rappresentano le risorse assegnate e quelle ancora necessarie
- Quando viene chiesta l'allocazione di una risorsa si prova a costruire le matrici relative al nuovo stato
- Si applica lo stesso algoritmo visto per la rilevazione del deadlock per verificare se lo stato è sicuro

Prevenzione del deadlock

- Si fa in modo che non sia mai verificata almeno una delle quattro condizioni necessarie:
 1. Mutual exclusion: possibilità di assegnazione di risorse in uso esclusivo
 2. Hold and wait: possibilità di richiedere risorse quando già se ne detengono
 3. No preemption: impossibilità di sottrarre risorse già assegnate (non è ragionevole)
 4. Circular wait: possibilità che si verifichi una situazione di attesa circolare

Negazione della mutua esclusione

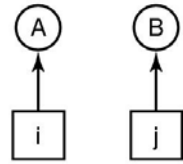
- Per molti tipi di risorse questa soluzione non è possibile
- Possibile indirettamente tramite virtualizzazione della risorsa
- Es: Stampante
 - Per sua natura deve essere assegnata in uso esclusivo
 - Tramite il meccanismo di spool la stampa viene effettuata in modo virtuale
 - Un processo che deve stampare non ha bisogno di farsi assegnare la stampante in uso esclusivo

Negazione della 'hold and wait'

- Possibile tramite l'*allocazione in blocco*
- Ciascun processo deve richiedere tutte le risorse di cui abbisogna in un'unica soluzione
- Se non disponibili viene bloccato fino a quando non sono tutte libere
- Soluzione sempre tecnicamente possibile, ma in genere molto inefficiente
- Limita molto la concorrenza dei processi e l'utilizzazione delle risorse

Negazione dell'attesa circolare

1. Imagesetter
2. Scanner
3. Plotter
4. Tape drive
5. CD Rom drive



- Le risorse vengono numerate con numeri progressivi
- Un processo può richiedere solo risorse con numeri più alti di quelle che detiene
- I processi si accodano e viene evitata l'attesa circolare
- Consente assegnazioni parziali: più efficiente dell'allocazione in blocco

Starvation

- Problema strettamente connesso al deadlock
- *To starve*: morire di fame
- Problema che si manifesta in tutte le situazioni in cui un processo può essere bloccato e messo in attesa
- Deve essere evitato che la prevenzione del deadlock provochi attese indefinite
- Vari tipi di soluzioni legati a priorità dinamiche e timeout
- Problema già discusso nel caso dello scheduling della CPU