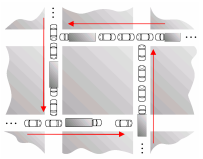
Lezione 8:

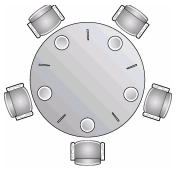
I DEADLOCK
(STALLO DEI PROCESSI)

- Caratterizzazione dei deadlock
- Metodi per la gestione dei deadlock
  - Prevenire i deadlock
  - Evitare i deadlock
  - Rilevare i deadlock
  - Ripristino da situazioni di deadlock
- Approccio combinato alla gestione dei deadlock

Cos'e' un deadlock?



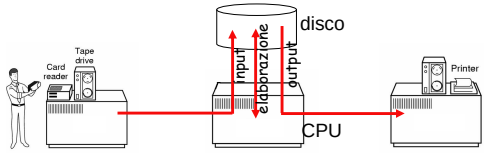
Traffico "a croce uncinata"



Il problema dei filosofi a cena

- In un ambiente multiprogrammato più processi possono entrare in competizione per ottenere un numero finito di risorse.
- ***Deadlock*** — insieme di processi bloccati: ciascun processo "possiede" una risorsa ed attende di acquisire una risorsa allocata ad un altro processo dell'insieme.

Altri esempi



- Esempio 1
  - Nei primi sistemi batch alcuni sistemi di spooling richiedevano il completamento dell'output prima di cominciare la stampa
  - Se l'output era maggiore della dimensione del disco il sistema si bloccava
  - Quali sono i processi bloccati e qual'e' la risorsa contesa?
- Esempio 2
  - Semafori *A* e *B*, inizializzati a 1:

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

Qual'e' la risorsa contesa?

Modello di sistema

- Insieme di processi $P_1, P_2, \ldots, P_n$
- Insieme di risorse $R_1, R_2, \ldots, R_m$
  - *Fisiche: cpu, memoria, stampanti, dispositivi di I/O*
  - *Logiche: file, semafori, sezioni critiche*
- Ciascun tipo di risorsa R_i puo' avere W_i istanze.
  - Ad es.: in un sistema biprocessore la "risorsa CPU" ha $W_i = 2$ istanze
- Una tabella nel kernel conserva lo stato di ogni risorsa (se e a chi e' assegnata)
- se un processo richiede una risorsa gia' assegnata, il processo richiedente e' inserito in una apposita coda di processi in attesa

Sequenza di uso della risorsa

- Il protocollo di accesso alle risorse da parte dei processi prevede…
  - **Richiesta** — se la richiesta non può essere soddisfatta immediatamente, il processo richiedente deve attendere fino all'acquisizione della risorsa.
  - **Utilizzo** – il processo opera sulla risorsa
  - **Rilascio** – il processo rilascia la risorsa
- Richiesta/rilascio realizzati con apposite system call: *request/release device*, *open/close file*, *allocate/free memory*.

Caratterizzazione dei deadlock

Teorema: Una situazione di deadlock può verificarsi **se e solo se** valgono simultaneamente le seguenti condizioni:

- **Mutua esclusione**: esiste almeno una risorsa non condivisibile, cioè un solo processo alla volta può usare quella risorsa.
- **Possesso ed attesa**: un processo, che possiede almeno una risorsa, attende di acquisire ulteriori risorse possedute da altri processi.
- **Impossibilità di prelazione**: una risorsa può essere rilasciata dal processo che la possiede solo volontariamente, al termine del suo compito.
- **Attesa circolare**: esiste un insieme $\{P_1, \ldots, P_n\}$ di processi in attesa, tali che P_1 è in attesa di una risorsa che è posseduta da P_2, P_2 è in attesa di una risorsa posseduta da P_3, …, P_{n-1} è in attesa di una risorsa posseduta da P_n, e P_n è in attesa di una risorsa posseduta da P_1.

Come si puo' rappresentare un deadlock

Un deadlock si puo' rappresentare mediante un

grafo di allocazione risorse:

Un grafo è costituito da un insieme di vertici (o nodi) V variamente connessi per mezzo di un insieme di archi E.

- L'insieme V è partizionato in due sottoinsiemi:
  - $P = \{P_1, P_2, \ldots, P_n\}$ è l'insieme costituito da tutti i processi del sistema.
  - $R = \{R_1, R_2, \ldots, R_m\}$ è l'insieme di tutti i tipi di risorse del sistema.
- L'insieme E consiste di
  - Arco di richiesta: $P_i \rightarrow R_j$
  - Arco di assegnazione: $R_j \rightarrow P_i$

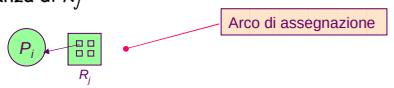
Grafo di allocazione risorse

- Processo

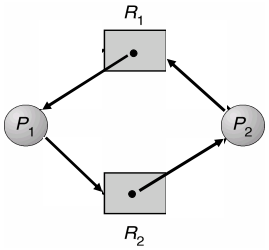
- Tipo di risorsa con 4 istanze

- P_i richiede un'istanza di R_j

- P_i possiede un'istanza di R_j



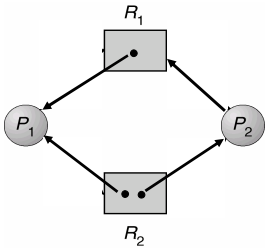
Esempio 1



- 2 processi
- 2 risorse
  - R1 → W1 = 1 istanza
  - R2 → W2 = 1 istanze
- P1 possiede un'istanza di R1 e attende un'istanza di R2
- P2 possiede un'istanza di R2 attende un'istanza di R1

Il sistema e' in deadlock?

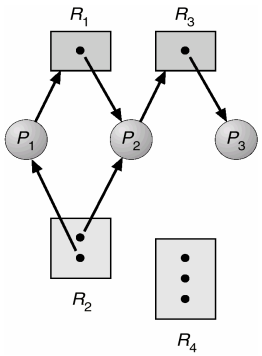
Esempio 2



- 2 processi
- 2 risorse
  - R1 → W1 = 1 istanza
  - R2 → W2 = 2 istanze
- P1 possiede un'istanza di R1 e un'istanza di R2
- P2 possiede un'istanza di R2 attende un'istanza di R1

Il sistema e' in deadlock?

Esempio 3

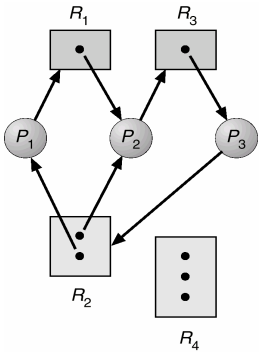


- 3 processi
- 4 risorse
  - R1 → W1 = 1 istanza
  - R2 → W2 = 2 istanze
  - R3 → W3 = 1 istanza
  - R4 → W4 = 3 istanze
- P1 possiede un'istanza di R2 e attende un'istanza di R1
- P2 possiede un'istanza di R1 e di R2 e attende un'istanza di R3
- P3 possiede un'istanza di R3

Il sistema e' in deadlock?

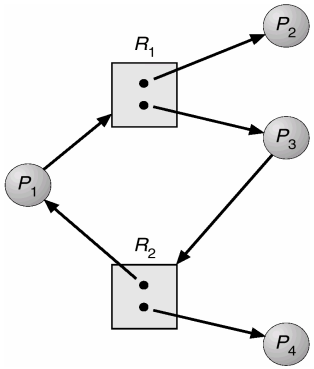
Esempio 4

- 3 processi
- 4 risorse
  - R1 → W1 = 1 istanza
  - R2 → W2 = 2 istanze
  - R3 → W3 = 1 istanza
  - R4 → W4 = 3 istanze
- P1 possiede un'istanza di R2 e attende un'istanza di R1
- P2 possiede un'istanza di R1 e di R2 e attende un'istanza di R3
- P3 attende un'istanza di R3 e attende un'istanza di R2



Il sistema e' in deadlock?

Esempio 5



- 4 processi
- 2 risorse
  - R1 → W1 = 2 istanza
  - R2 → W2 = 2 istanze
- P1 possiede un'istanza di R2 e attende un'istanza di R1
- P2 possiede un'istanza di R1
- P3 possiede un'istanza di R1 e attende un'istanza di R2
- P4 possiede un'istanza di R2

Il sistema e' in deadlock?

riassumendo

ogni risorsa ha solo un'istanza
→ la presenza di un ciclo e' condizione necessaria e sufficiente per un deadlock

- se c'e il ciclo nel grafo c'e' anche il deadlock (esempio 1)
- se non c'e' il ciclo non c'e' deadlock

qualche risorsa ha piu' istanze
→ la presenza di un ciclo e' condizione necessaria ma non sufficiente per un deadlock

- se non c'e' ciclo nel grafo non c'e' deadlock (esempio 3)
- se c'e' il ciclo nel grafo il deadlock
  - puo' esserci deadlock (esempio 4)
  - puo' non esserci deadlock (esempio 5)

Metodi per la gestione dei deadlock

- Assicurare che **il sistema non entri *mai* in uno stato di deadlock.**
  - Prevenzione dei deadlock: evitare che si verifichino contemporaneamente le condizioni per il deadlock
  - Evitare i deadlock: se sta per verificarsi un deadlock esso viene evitato.
- **Permettere al sistema di entrare in uno stato di deadlock,** quindi ripristinare il sistema.
  - Determinare la presenza **di un deadlock**.
  - Ripristinare il sistema **da un deadlock**.
- **Ignorare il problema** e fingere che i deadlock non avvengano mai nel sistema;

Prevenire i deadlock

- **Mutua esclusione** — in generale non è richiesta per risorse condivisibili (es. file in sola lettura), ma alcune risorse non possono essere condivise quindi in generale non e' possibile prevenire i deadlock evitando le risorse condivise.
- **Possesso e attesa** — evitare "possesso e attesa" significa "non possedere" oppure "non attendere". Quindi
  - Richiedere ad un processo di allocare tutte le risorse necessarie prima che inizi l'esecuzione (evitare l'attesa)
  - consentire la richiesta di risorse solo quando il processo non ne possiede alcuna (evitare il possesso)
  - Basso impiego delle risorse (risorse possedute e non utilizzate, rilasciare e riprendere piu' volte le risosrse)
  - È possibile che si verifichi l'attesa indefinita (vedi pbl. dei filosofi a cena).

Prevenire i deadlock

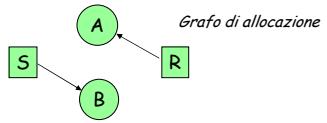
- **Impossibilità di prelazione** —
  - Se un processo, che possiede alcune risorse, richiede un'altra risorsa che non gli può essere allocata immmediatamente, allora rilascia tutte le risorse possedute.
  - Le risorse rilasciate (prelazionate al processo) vengono aggiunte alla lista delle risorse che il processo sta attendendo.
  - Il processo viene avviato nuovamente solo quando può ottenere sia le vecchie che le nuove risorse .
  - scarsa efficienza (le risorse vengono acquisite e rilasciate)

Prevenire i deadlock

- **Attesa circolare** — si impone un ordinamento totale su tutti i tipi di risorsa e si pretende che ciascun processo richieda le risorse in ordine crescente.

Esempio: 2 processi (A e B) e 2 risorse (R e S) con 1 istanza

- A possiede R
- B possiede S



Grafo di allocazione

Ordinamento delle risorse
1 : stampante
2: scanner
3: plotter
4: unita' nastro
5: unita' CD

- se R > S allora A non puo' richiedere S
- se R < S allora B non puo' richiedere R

Assenza di cicli

Evitare i deadlock

Presuppone che il sistema conosca a priori informazioni addizionali sulle richieste future dei processi.

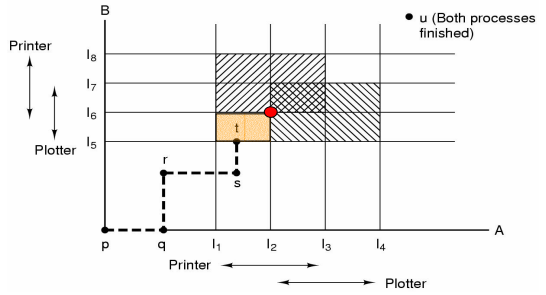
- Il modello più semplice e utile richiede che ciascun processo dichiari il numero massimo di risorse di ciascun tipo di cui può usufruire.
- Un algoritmo (alg. di *deadlock-avoidance*) esamina dinamicamente lo stato di allocazione delle risorse per assicurare che non si possa verificare mai una condizione di attesa circolare.
- Gli algoritmi di deadlock-avoidance si basano sul concetto di stato sicuro

Stato sicuro

- Uno stato e' sicuro se:
  - non e' in stallo
  - e' possibile soddisfare le richieste pendenti eseguendo i processi in un qualche ordine (sequenza sicura)
- Se un sistema è in stato sicuro ⇒ non si evolve verso il deadlock.
- Se un sistema è in stato non sicuro ⇒ si può evolvere in deadlock.
- Quando un processo richiede una risorsa disponibile, il sistema deve decidere se l'allocazione immediata lasci il sistema in **stato sicuro**.

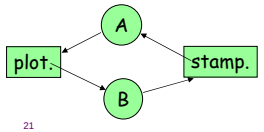
Esempio

Esempio:



Stato non sicuro

Stato di deadlock



Sequenza sicura

- Una sequenza $\langle P_1, P_2, \ldots, P_n \rangle$ è sicura se, per ogni P_i, le risorse che P_i può ancora richiedere possono essergli allocate sfruttando le risorse disponibili + le risorse possedute da tutti i P_j, con $j < i$.
  - Se le richieste di P_i non possono essere soddisfatte immediatamente, allora P_i può attendere finché P_j ha terminato.
  - Quando P_j ha terminato, P_i può ottenere le risorse richieste, eseguire i suoi compiti, restituire le risorse allocate e terminare.
  - Quando P_i termina, P_{i+1} può ottenere le risorse richieste, etc.

Esempio di stato sicuro

La matrice di allocazione di una risorsa riporta, per ogni processo, il numero di istanze possedute e il massimo numero di istanze che utilizzera'

Esempio: 3 processi (P_1, P_2 e P_3) e 10 istanze disponibili di una data risorsa

	Has	Max
P_1	3	9
P_2	2	4
P_3	2	7

Free: 3 (a)

	Has	Max
P_1	3	9
P_2	4	4
P_3	2	7

Free: 1 (b)

	Has	Max
P_1	3	9
P_2	0	–
P_3	2	7

Free: 5 (c)

	Has	Max
P_1	3	9
P_2	0	–
P_3	7	7

Free: 0 (d)

	Has	Max
P_1	3	9
P_2	0	–
P_3	0	–

Free: 7 (e)

Lo stato a) e' sicuro, infatti

- si assegnano 2 istanze a P_2 → 1 istanza libera (stato b)
- P_2 termina e rilascia le sue istanze → 5 istanze libere (stato c)
- si assegnano 5 istanze a P_3 → 0 istanze libere (stato d)
- P_3 termina e rilascia le sue istanze → 7 istanze libere (stato e)
- si possono assegnare 6 risorse a P_1 che termina regolarmente
- $\langle P_2, P_3, P_1 \rangle$ e' una sequenza sicura

Esempio di stato non sicuro

	Has	Max
P_1	3	9
P_2	2	4
P_3	2	7

Free: 3 (a)

	Has	Max
P_1	4	9
P_2	2	4
P_3	2	7

Free: 2 (b)

	Has	Max
P_1	4	9
P_2	4	4
P_3	2	7

Free: 0 (c)

	Has	Max
P_1	4	9
P_2	—	—
P_3	2	7

Free: 4 (d)

Se pero' dallo stato (a) si assegna una ulteriore istanza al processo P_1 si ottiene uno stato non sicuro (stato b) che conduce a un deadlock (stato d)

Osservazione: nello stato (b), anche assegnando le 2 risorse libere a P_1 o P_3 (invece che a P_2) si ottiene un deadlock

Algoritmo del banchiere

- IDEA : in base delle risorse disponibili soddisfare prima le richieste di un processo che, puo' completare la propria esecuzione, permettendo cosi' il rilascio di tutte le risorse in suo possesso
- Ciascun processo deve dichiarare a priori il massimo impiego di risorse.
- Quando un processo richiede una risorsa si verifica prima che il nuovo stato sia uno stato sicuro. In caso contrario il processo attende
- Il nome deriva dal fatto che esso puo' essere applicato in una banca per evitare che il cassiere consegni tutto il denaro disponibile (le risorse), e non possa piu' soddisfare le richieste di tutti i suoi clienti (i processi)

Algoritmo del banchiere

Dati necessari per l'algoritmo nel caso di
- 1 risorsa R con piu' istanze
- n processi P_i i=1,..,n

- ***Available***: numero di istanze disponibili della risorsa R
- ***Max***: *array di lunghezza n. Max[i]* e' il massimo numero di istanze che P_i può richiedere
- ***Allocation***: *array di lunghezza n. Allocation[i]* e' il numero di istanze attualmente allocate a P_i
- ***Need***: *array di lunghezza n. Need[i]* e' il numero di ulteriori istanze necessarie a P_i per completare il proprio task.

(Need[i] = Max[i] – Allocation[i])

Algoritmo di richiesta delle risorse per il processo P_i

Sia *Request[i]*, numero di istanze richieste dal processo P_i.

1. Se *Request[i]* ≤ *Need[i]* , si vada al passo 2. Altrimenti, si riporti una condizione di errore, poiché il processo ha ecceduto il massimo numero di richieste.
2. Se *Request[i]* ≤ *Available* , si vada al passo 3. Altrimenti P_i deve attendere, poiché le risorse non sono disponibili.
3. Il sistema simula l'allocazione al processo P_i delle risorse richieste, modificando lo stato di allocazione delle risorse come segue:

   Available = Available – Request[i]
   Allocation[i] = Allocation[i] + Request[i]
   Need[i] = Need[i] – Request[i]

   - Se lo stato è sicuro ⇒ le risorse vengono definitivamente allocate a P_i .
   - Se lo stato è non sicuro ⇒ P_i deve attendere, e viene ripristinato il vecchio stato di allocazione delle risorse.

Algoritmo di verifica della sicurezza

Come fare a vedere se uno stato e' sicuro?
Cerchiamo una sequenza sicura

1. Siano *Finish* vettore di lunghezza *n* e *Work* un intero. Si inizializzi:
   a) *Work = Available* *così' non perdiamo Available*
   b) *Finish[i] = false* per *i = 1,2, …, n*
2. Si cerchi *i* tale che valgano contemporaneamente:
   a) *Finish[i] = false*
   b) *Need[i] ≤ Work*

   Se tale *i* non esiste, si esegua il passo 4.
3. *Work = Work + Allocation[i]*
   Finish[i] = true
   torna al passo 2.
4. Se *Finish [i] == true* per ogni *i*, il sistema è in stato sicuro.

Esempio: n=3 processi

	Alloc	Max	Need
P_1	3	9	6
P_2	2	4	2
P_3	2	7	5
Avail	3		

Finish[1] = Finish[2] = Finish[3] = false
Work = Available = 3

i=1 *Need[1]* ≤ *Work* e *Finish[i]* = *false* ? no
i=2 *Need[2]* ≤ *Work* e *Finish[2]* = *false* ? **Si**
Work = 3+2 = 5 Finish[2] = true

i=1 *Need[1]* ≤ *Work* e *Finish[1]* = *false* ? no
i=2 *Need[2]* ≤ *Work* e *Finish[2]* = *false* ? no
i=3 *Need[3]* ≤ *Work* e *Finish[3]* = *false* ? **Si**
Work = 5+2 = 7 Finish[3] = true

i=1 *Need[1]* ≤ *Work* e *Finish[1]* = *false* ? **Si**
Work = 7+3 = 10 Finish[1] = true

Fine ➔ Stato sicuro
La sequenza <P_2, P_3, P_1 > soddisfa i criteri di sicurezza
(Finish[i] == true per ogni i)

Esempio: n=3 processi

	Alloc	Max	Need
P_1	4	9	5
P_2	2	4	2
P_3	2	7	5
Avail = 2			

Finish[1] = Finish[2] = Finish[3] = false
Work = Available = 2

i=1 *Need[1]* ≤ *Work* e *Finish[1]* = *false* ? no
i=2 *Need[2]* ≤ *Work* e *Finish[2]* = *false* ? **Si**
Work = 2+2 = 4 Finish[2] = true

i=1 *Need[1]* ≤ *Work* e *Finish[1]* = *false* ? no
i=2 *Need[2]* ≤ *Work* e *Finish[2]* = *false* ? no
i=3 *Need[3]* ≤ *Work* e *Finish[3]* = *false* ? no

Fine ➔ Stato non sicuro
dopo P_2 nessuno riesce a terminare
(Finish[1] = Finish[3] = false)

Algoritmo del banchiere

L'algoritmo puo' essere generalizzato al caso di piu' risorse con le dovute modifiche:

- ***Available***: Vettore di lunghezza *m*. Available[*j*] e' il numero di istanze disponibili della risorsa R_j.
- ***Max***: *matrice n x m*. *Max*[*i,j*] e' il massimo numero di istanze di R_j che il processo P_i può richiedere.
- ***Allocation***: matrice *n x m*. Allocation[*i,j*] e' il numero di istanze di R_j attualmente allocate a P_i
- ***Need***: matrice *n x m*. *Need*[*i,j*] e' il numero di ulteriori istanze di R_j necessarie a P_i per completare il proprio task.
  (Need[*i,j*] = *Max*[*i,j*] – *Allocation*[*i,j*])

Prima di soddisfare una richiesta,
bisogna verifica lo stato della sicurezza
per tutte le risorse R_j j=1,..,m

Esempio di applicazione dell'algoritmo del banchiere

- 5 processi, da P_0 a P_4;
- 3 tipi di risorse: *A* (10 istanze), *B* (5 istanze), e *C* (7 istanze).
- Istantanea al tempo T_0:

	Allocation	*Max*	*Work*	*Need*
	A B C	*A B C*	*A B C*	*A B C*
P_0	0 1 0	7 5 3	3 3 2	7 4 3
P_1	2 0 0	3 2 2		1 2 2
P_2	3 0 2	9 0 2		6 0 0
P_3	2 1 1	2 2 2		0 1 1
P_4	0 0 2	4 3 3		4 3 1

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

Passo 1

	Allocation	Max	Available	Need
	A B C	A B C	A B C	A B C
P_0	0 1 0	7 5 3	3 3 2	7 4 3
P_1	2 0 0	3 2 2		1 2 2
P_2	3 0 2	9 0 2		6 0 0
P_3	2 1 1	2 2 2		0 1 1
P_4	0 0 2	4 3 3		4 3 1

- e' possibile soddisfare le richieste di P_1
- Alla fine dell'esecuzione P_1 rilascia le risorse ➔ Available = (5 3 2)
- Sequenza sicura < P_1 >

Passo 2

		Allocation	Max	Available	Need
		A B C	A B C	A B C	A B C
	P_0	0 1 0	7 5 3	5 3 2	7 4 3
terminato	P_1	0 0 0	3 2 2		1 2 2
	P_2	3 0 2	9 0 2		6 0 0
	P_3	2 1 1	2 2 2		0 1 1
	P_4	0 0 2	4 3 3		4 3 1

- e' possibile soddisfare le richieste di P_3
- Alla fine dell'esecuzione P_3 rilascia le risorse ➔ Available = (7 4 3)
- Sequenza sicura < P_1 , P_3 >

Passo 3

		Allocation	Max	Available	Need
		A B C	A B C	A B C	A B C
	P_0	0 1 0	7 5 3	7 4 3	7 4 3
terminato	P_1	0 0 0	3 2 2		1 2 2
	P_2	3 0 2	9 0 2		6 0 0
terminato	P_3	0 0 0	2 2 2		0 1 1
	P_4	0 0 2	4 3 3		4 3 1

- e' possibile soddisfare le richieste di P_4
- Alla fine dell'esecuzione P_4 rilascia le risorse ➔ Available = (7 4 5)
- Sequenza sicura < P_1 , P_3 , P_4 >

Passo 4

		Allocation	Max	Available	Need
		A B C	A B C	A B C	A B C
	P_0	0 1 0	7 5 3	7 4 5	7 4 3
terminato	P_1	0 0 0	3 2 2		1 2 2
	P_2	3 0 2	9 0 2		6 0 0
terminato	P_3	0 0 0	2 2 2		0 1 1
terminato	P_4	0 0 0	4 3 3		4 3 1

- e' possibile soddisfare le richieste di P_0
- Alla fine dell'esecuzione P_0 rilascia le risorse ➔ Available = (7 5 5)
- Sequenza sicura < P_1 , P_3 , P_4, P_0 >

Passo 5

		Allocation	Max	Available	Need
		A B C	A B C	A B C	A B C
terminato	P_0	0 0 0	7 5 3	7 5 5	7 4 3
terminato	P_1	0 0 0	3 2 2		1 2 2
	P_2	3 0 2	9 0 2		6 0 0
terminato	P_3	0 0 0	2 2 2		0 1 1
terminato	P_4	0 0 0	4 3 3		4 3 1

- e' possibile soddisfare le richieste di P_2
- Alla fine dell'esecuzione P_2 rilascia le risorse ➔ Available = (10 5 7)
- Sequenza sicura < P_1 , P_3 , P_4 , P_0 , P_2 >

Debolezze dell'algoritmo del banchiere

- Richiede che il numero delle risorse rimanga costante, ma questa non e' un'ipotesi realistica
- Richiede che il numero di processi rimanga costante, ma cio' non e' vero
- Richiede che i processi restituiscano le risorse in un tempo finito (vero in alcuni sistemi, ad es. real time)
- Richiede che i processi dichiarino le proprie necessita' prima dell'esecuzione (quasi mai possibile)

Secondo approccio

- Prevenire i deadlock
  - Sottoutilizzo delle risorse
- Evitare i deadlock
  - Necessita' di informazioni sulle richieste future dei processi

In entrambi i casi : elevato overhead

- Approccio alternativo: si permette al sistema di entrare in uno stato di deadlock e poi si ripristna il sistema
- Sono necessari:
  - Algoritmo di rilevamento del deadlock
  - Algoritmo di ripristino dal deadlock

Algoritmo di rilevamento

Variante della verifica della sicurezza dell'algoritmo del banchiere per n processi e m risorse multiple

- *Available*: vettore di lunghezza *m*, indica il numero di istanze disponibili di ogni risorsa.
- *Allocation*: matrice *n x m*, definisce il numero di istanze di ciascuna risorsa attualmente allocate a ciascun processo.
- *Request*: matrice *n x m*, indica la richiesta corrente di ciascun processo. Se *Request* [*i,j*] = *k*, il processo P_i richiede *k* istanze supplementari della risorsa R_j.

Osservazione: manca la matrice Max e quindi anche Need

Algoritmo di rilevamento

1. Siano *Work* e *Finish* due vettori di lunghezza *m* e *n*, rispettivamente. Inizialmente si ponga:
   a) *Work* = *Available*
   b) Per *i* = 1,2, …, *n*, se $Allocation_i \neq 0$, *Finish*[*i*] = *false*; altrimenti, *Finish*[*i*] = *true*
2. Si trovi un indice *i* tale che valgano entrambe le condizioni:
   a) *Finish*[*i*] = *false*
   b) $Request_i \leq Work$
   Se tale *i* non esiste si vada al passo 4.
3. $Work = Work + Allocation_i$
   Finish[*i*] = *true*
   si vada al passo 2.
4. Se *Finish*[*i*] == *false*, per qualche *i*, $1 \leq i \leq n$, il sistema è in uno stato di deadlock. Inoltre, se *Finish*[*i*] == *false*, P_i è in deadlock.

L'algoritmo richiede un numero di operazioni dell'ordine di $O(m \times n^2)$ per determinare se il sistema è in uno stato di deadlock.

Esempio di applicazione dell'algoritmo di rilevamento

- Cinque processi, da P_0 a P_4;
- tre tipi di risorse: A (7 istanze), *B* (2 istanze), e *C* (6 istanze).
- Istantanea al tempo T_0:

	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	

In questo istante il sistema non e' in deadlock

- La sequenza

  $\langle P_0, P_2, P_3, P_1, P_4 \rangle$

  conduce a *Finish*[*i*] = *true* per tutti gli *i*.

Esempio di applicazione dell'algoritmo di rilevamento

- P_2 richiede un'istanza supplementare della risorsa *C*.

	Request
	A B C
P_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?
  - Il sistema puo' riprendere le risorse possedute dal processo P_0, ma il numero di risorse disponibili non è sufficiente a soddisfare gli altri processi
  - $\Rightarrow$ deadlock: processi P_1, P_2, P_3, e P_4.

Impiego dell'algoritmo di rilevamento

- Quando e quanto spesso richiamare l'algoritmo di rilevamento dipende da:
  - Frequenza (presunta) con la quale si verificano i deadlock;
  - Numero di processi che vengono eventualmente influenzati dal deadlock (e sui quali deve essere effettuato un *rollback*):
    - Almeno un processo per ciascun ciclo.
- Se l'algoritmo viene richiamato in momenti arbitrari, possono essere presenti molti cicli nel grafo delle risorse $\Rightarrow$ non si può stabilire quale dei processi coinvolti nel ciclo abbia "causato" il deadlock.
- Alternativamente, l'algoritmo può essere richiamato ogni volta che una richiesta non può essere soddisfatta immediatamente

Ripristino dal deadlock: Terminazione di processi

- Terminazione in abort di tutti i processi in deadlock.
- Terminazione in abort di un processo alla volta fino all'eliminazione del ciclo di deadlock.
- In quale ordine si decide di terminare i processi? I fattori significativi sono:
  - La priorità del processo.
  - Il tempo di computazione trascorso e il tempo ancora necessario al completamento del processo.
  - Quantità e tipo di risorse impiegate.
  - Risorse ulteriori necessarie al processo per terminare il proprio compito.
  - Numero di processi che devono essere terminati.
  - Tipo di processo: interattivo o batch.

Ripristino dal deadlock: Prelazione di risorse

- **Selezione di una vittima** — È necessario stabilire l'ordine di prelazione per minimizzare i costi.
- **Rollback** — Un processo a cui sia stata prelazionata una risorsa deve ritornare ad uno stato sicuro, da cui ripartire.
- **Starvation** — Alcuni processi possono essere sempre selezionati come vittime della prelazione: includere il numero di rollback nel fattore di costo.

Approccio combinato alla gestione del deadlock

- Si combinano i tre approcci di base:
  - prevenzione
  - evitare i deadlock
  - rilevamento

  permettendo l'uso dell'approccio ottimale per ciascuna risorsa del sistema.
- Si suddividono le risorse in classi gerarchicamente ordinate.
- Si impiegano le tecniche più appropriate per gestire i deadlock in ciascuna classe.

Terzo approccio

Gli approcci per
- Prevenire i deadlock
- Evitare i deadlock

Sono costosi, impongono restrizioni sull'uso delle risorse e possono richiedere informazioni sull'uso futuro delle risorse

Occorre tenere presente che un deadlock dipende dalla schedulazione dei processi, e non solo da come sono fatti i processi

➔ un deadlock puo' essere un evento relativamente raro

Siamo disposti a prevenire ed evitare sempre qualcosa che potrebbe accadere raramente?

Ignorare il problema

Molti S.O. (tra cui Windows e Unix) ignorano il problema. L'amministratore del sistema provvedera', a mano, a ripristinare il sistema

Ovviamente S.O. con funzioni particolari, potrebbero richiedere funzionalita' di prevenzione dei deadlock