SISTEMI OPERATIVI

(MODULO DI INFORMATICA II)

Lo stallo (deadlock)

Prof. Luca Gherardi

Prof.ssa Patrizia Scandurra (anni precedenti)

Università degli Studi di Bergamo

a.a. 2012-13

Sommario

- Uso di risorse condivise
- Caratterizzazione del deadlock
  - Condizioni per l'occorrenza del deadlock
  - Grafo di allocazione delle risorse
- Metodi di gestione del deadlock
  1. Ignorare il problema
  2. Prevenire il deadlock
  3. Evitare il deadlock
  4. Rilevazione e ripristino del deadlock

Uso di risorse condivise

- Un sistema offre un numero finito di risorse
  - Cicli di CPU, spazio di memoria, periferiche di I/O, ecc..
- Per ogni risorsa il processo può richiedere delle istanze
  - Le istanze sono identiche tra loro (e.g. due CPU)
  - Al processo non interessa quale specifica istanza ricevere
  - Le risorse vengono quindi classificate in classi
- Risorse condivise usabili:
  - In modo non esclusivo
  - Solo in modo mutuamente esclusivo

Uso di risorse condivise

- Un processo segue il seguente schema per utilizzare una risorsa:
  - Richiesta di uso della risorsa
  - Uso della risorsa
  - Rilascio della risorsa

- La richiesta e il rilascio delle risorse avvengono attraverso chiamate di sistema (e.g. `acquire()`, `release()`)
  - Il sistema operativo mantiene una tabella che memorizza lo stato delle risorse e l'eventuale processo utilizzatore

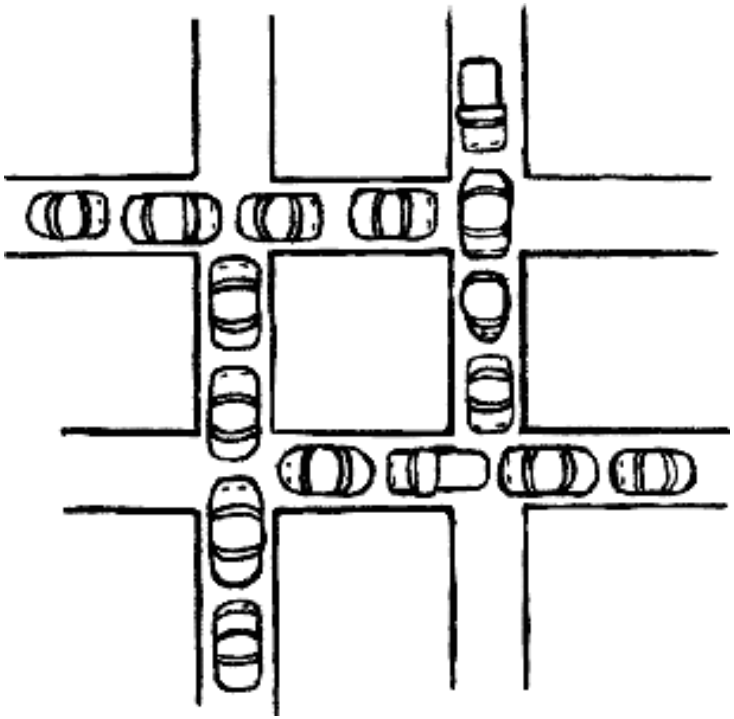
Deadlock ("stallo")

- **Definizione**: un gruppo di processi entra in uno stallo quando tutti i processi del gruppo attendono il rilascio di una risorsa che può essere rilasciata solo da uno dei processi in attesa
  - Esempio con due processi (P_a e P_b) e due risorse (R_1 e R_2)
  - Il processo P_a ottiene il possesso della risorsa R_1
  - Il processo P_b ottiene il possesso della risorsa R_2
  - Il processo P_a richiede la risorsa R_2
  - Il processo P_b richiede la risorsa R_1

Propagazione del deadlock

- Nuovi processi possono via via entrare indefinitamente in tale stato se le risorse richieste sono in possesso di altri processi a loro volta in stallo

Esempio "Incrocio stradale":



Caratterizzazione del deadlock

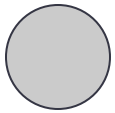
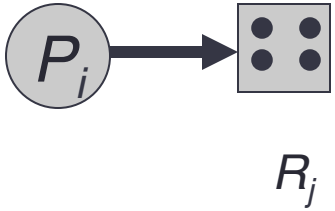
Si ha deadlock se si verificano **simultaneamente** le seguenti condizioni (*condizioni necessarie*):

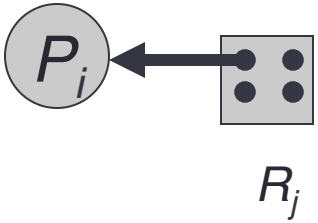
- **Mutua esclusione**
  - Almeno una risorsa deve poter essere acceduta da un solo processo alla volta (gli altri vengono messi in attesa)
- **Possesso e attesa**
  - Un processo possiede una risorsa ed è in attesa per un'altra risorsa
- **Non-preemptive**
  - Una risorsa posseduta da un processo non può essere rilasciata se non per spontanea volontà del processo stesso
- **Attesa circolare**
  - $\{P_0, P_1, \ldots, P_N\}$, P_0 attende una risorsa posseduta da P_1, P_1 attende una risorsa posseduta da P_2, …, P_N attende una risorsa posseduta da P_0

Grafo di allocazione delle risorse (1)

- Grafo di allocazione delle risorse G (V,E):
  - Insieme di nodi V
  - Insieme di archi E
- Nodi:
  - processi del sistema $P = \{P_1, P_2, \ldots, P_n\}$
  - risorse del sistema $R = \{R_1, R_2, \ldots, R_m\}$
    eventualmente con più istanze identiche
- Archi:
  - arco di richiesta:
    da processo a risorsa $P_i \rightarrow R_j$
  - arco di assegnazione:
    da risorsa a processo $R_j \rightarrow P_i$

Grafo di allocazione delle risorse (2)
Notazione

- Processo 
- Tipo di risorsa con 4 istanze 
- Arco di richiesta
  (si noti che raggiunge la risorsa) 

- Arco di assegnazione
  (si noti che parte dall'istanza) 

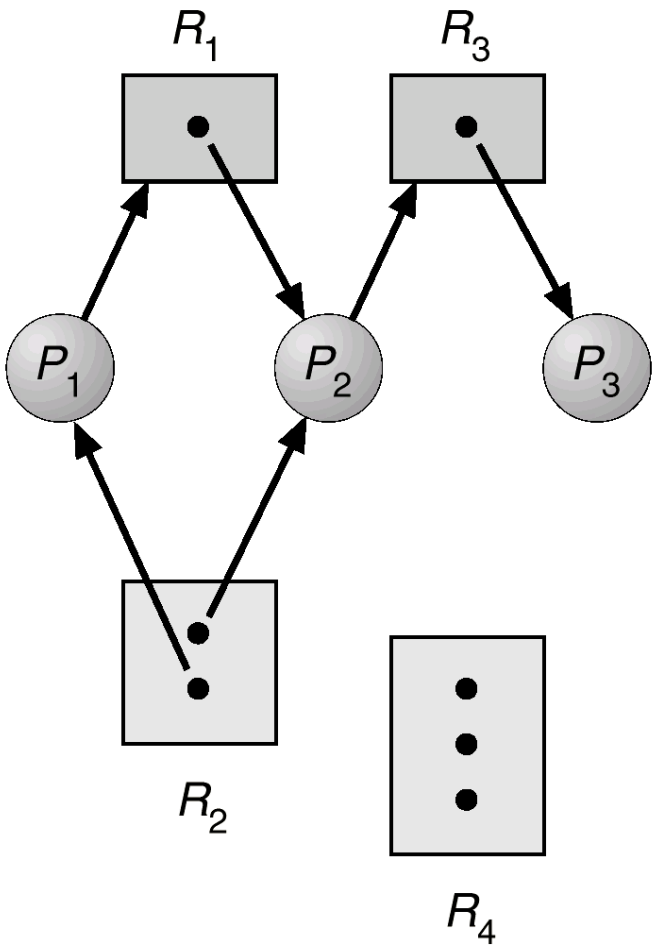
Punti chiave

- Se il grafo non contiene cicli ⇒ nessun deadlock
- Se il grafo contiene cicli ⇒
  - se ogni tipo di risorsa inclusa nel ciclo ha una sola istanza, allora si ha un deadlock
    - Condizione necessaria e sufficiente
  - se ogni tipo di risorsa ha più di un'istanza, allora si ha una possibilità di deadlock
    - Condizione necessaria ma non sufficiente

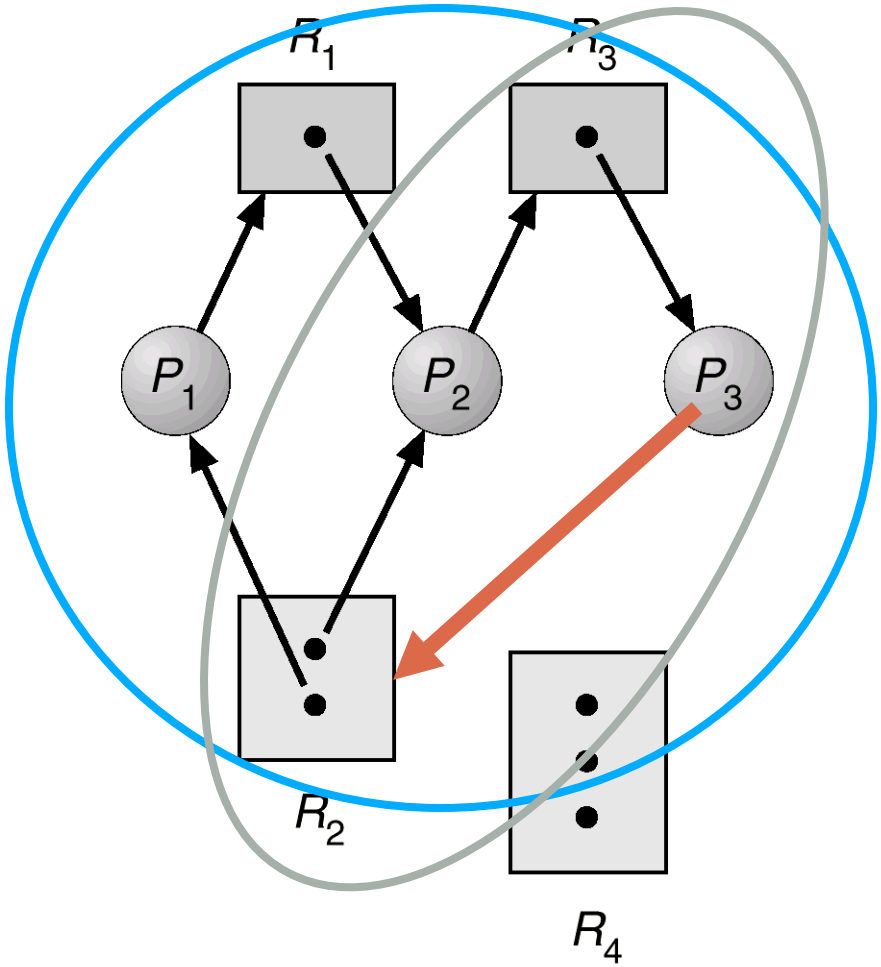
Grafo di allocazione delle risorse (3)

Ciclo con deadlock

a) Senza deadlock



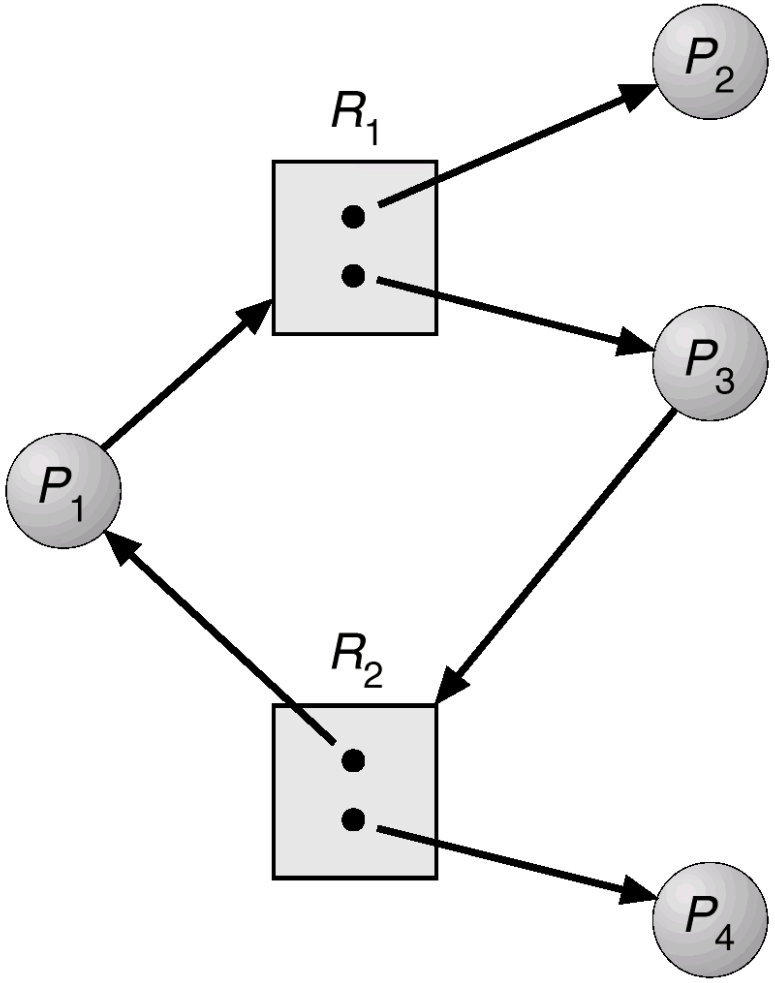
b) Con deadlock (due cicli)



Grafo di allocazione delle risorse (3)

Ciclo senza deadlock

- P_4 prima o poi cederà il possesso di R_2
- Che potrà quindi essere acquisita da P_3
- Non siamo in una condizione di deadlock



Metodi di gestione dei deadlock

1. **Ignorare** il deadlock
2. **Prevenire** il deadlock (deadlock prevention)
   - Puntano ad evitare che si verifichino tutte e quattro le condizioni necessarie
   - Descrivono come le risorse devono essere richieste
3. **Evitare** il deadlock (deadlock avoidance)
   - Il sistema operativo conosce in anticipo quali risorse un processo utilizzerà
   - In base a queste informazioni decide se tutte le richieste possono essere accettate senza causare stalli
4. **Rilevare e recuperare** il deadlock (deadlock detection & recovery)

Nessun metodo è ottimale, meglio una loro combinazione a seconda della "classe di risorsa"

1. Ignorare il deadlock

Algoritmo dello struzzo

- Applicato dalla maggior parte dei SO odierni (e dalla JVM)
  - È più economico dei metodi che prevedono di prevenire, evitare o individuare degli stalli
- Ma le cose stanno cambiando…per via delle applicazioni multi-thread

2. Prevenzione del deadlock

- **Principio**: prevenire il deadlock facendo sì che almeno una delle condizioni "necessarie" non sia soddisfatta:
  - Mutua esclusione
  - Possesso ed attesa
  - Nessun rilascio anticipato
  - Attesa circolare

Mutua esclusione

- La condizione può essere invalidata rimuovendola per le risorse intrinsecamente condivisibili
- La condizione non può mai essere invalidata per le risorse intrinsecamente non condivisibili
  - Per questo motivo tipicamente non si cerca di prevenire i deadlock negando la condizione di mutua esclusione

Possesso ed attesa (1)

- La condizione può essere invalidata garantendo che ogni volta che un processo chiede risorse, non possegga già qualche altra risorsa

- Due possibili tecniche:

1. Un processo chiede e ottiene tutte le risorse prima di iniziare l'esecuzione
2. Un processo che possiede alcune risorse e vuole chiederne altre deve:
   - rilasciare tutte le risorse che possiede
   - chiedere tutte quelle che servono (incluse eventualmente anche alcune di quelle che già possedeva

Possesso ed attesa (2)

- Esempio: un processo deve leggere dei file da DVD, copiarli in un file e successivamente stamparli

- Tecnica 1: DVD, file e stampante vengono acquisiti prima di iniziare l'elaborazione
- Tecnica 2: il processo acquisisce DVD e file, finita la copia rilascia le risorse, dopodiché acquisisce file e stampante.

- Problemi:
  - Scarso utilizzo delle risorse (in quanto molte risorse vengono bloccate ma non usate)
  - Possibile starvation (un processo potrebbe attendere indefinitivamente delle risorse)

Nessun rilascio anticipato (1)

- Si impone ad un processo che è in attesa di alcune risorse di rilasciarne alcune tra quelle che già possiede
  - In questo modo altri processi riescono ad acquisire tutte le risorse di cui necessitano

- Applicabilità: va bene per risorse il cui stato può essere facilmente salvato e ricaricato
  - Registri CPU e memoria centrale
  - Non va bene ad esempio per nastri o stampanti

Nessun rilascio anticipato (2)

Protocollo A:

Se un processo P detiene alcune risorse e ne chiede altre:

1. Se tutte le risorse richieste sono **disponibili**, gli vengono assegnate
2. Se non tutte le risorse richieste sono **disponibili**,
   1. il processo richiedente rilascia tutte le risorse in possesso
   2. le aggiunge alla lista delle risorse che attende
3. Il processo viene riavviato solo quando può riottenere tutte le risorse

Nessun rilascio anticipato (3)

Protocollo B:

Se un processo P detiene alcune risorse e ne chiede altre:

1. Se tutte le risorse richieste sono **disponibili**, gli vengono assegnate
2. Se alcune delle risorse richieste non sono disponibili e sono **assegnate ad un processo Q in attesa** di ulteriori risorse, le risorse richieste e detenute dal processo in attesa (Q) vengono
   - rilasciate anticipatamente e assegnate al processo richiedente P
   - inserite tra quelle per cui il processo Q è in attesa
3. Se alcune risorse richieste **non sono disponibili e non sono assegnate a processi in attesa** di altre risorse, il processo richiedente P deve
   - attendere che si liberino
   - ripartire quando ottiene tutte le risorse necessarie

Attesa circolare (1)

- La condizione può essere invalidata impedendo che si creino attese circolari

Un modo consiste nell'imporre che ogni processo chieda le risorse in un *ordine incrementale*

Attesa circolare (2)

- Un ordinamento globale univoco viene imposto su tutti i tipi di risorsa R_i
  - Si implementa tramite una funzione $f(R_i) = n$
- Se un processo chiede k istanze della risorsa R_j e detiene solo risorse R_i con **$i<j$**,
  - se le k istanze della risorsa R_j sono disponibili gli vengono assegnate
  - altrimenti, il processo deve attendere
- Un processo non potrà mai chiedere istanze della risorsa R_j se detiene risorse R_i con **$i \geq j$**

Attesa circolare (3)

- In questi casi, se un processo chiede k istanze della risorsa R_j e detiene risorse R_i con $i \geq j$, il processo deve
  - rilasciare tutte le istanze delle risorse R_i
  - chiedere le k istanze della risorsa R_j
  - chiedere le istanze delle risorse R_i ($i>j$) che deteneva precedentemente
- Ordinare le risorse non è sufficiente per evitare l'attesa circolare
  - I programmatori devono scrivere i programmi in modo che l'ordinamento sia rispettato
- Tipicamente le risorse si ordinano in base all'utilizzo
  - E.g. f(disco) > f(stampante)
  - Poiché tipicamente si usa prima il disco che la stampante
- Esistono alcuni programmi che verificano l'ordine di acquisizione dei lock sulle risorse

3. Evitare il deadlock

- Verificare **a priori** se la sequenza di richieste e rilasci di risorse effettuate da un processo porta al deadlock
  - tenendo conto delle risorse già assegnate ai processi già accettati nel sistema

- Obiettivi:
  - Alto sfruttamento delle risorse
  - Alta efficienza del sistema
  - Semplicità di gestione

Informazioni per evitare il deadlock

- Necessità di informazioni **a priori** sullo stato di allocazione delle risorse e sul comportamento dei processi:
  - il *numero massimo* di risorse per ogni processo
  - risorse disponibili
  - risorse assegnate
  - richieste e rilasci futuri di risorse

Stato di allocazione
(MAX,disponibili,allocate)

Algoritmi per evitare il deadlock

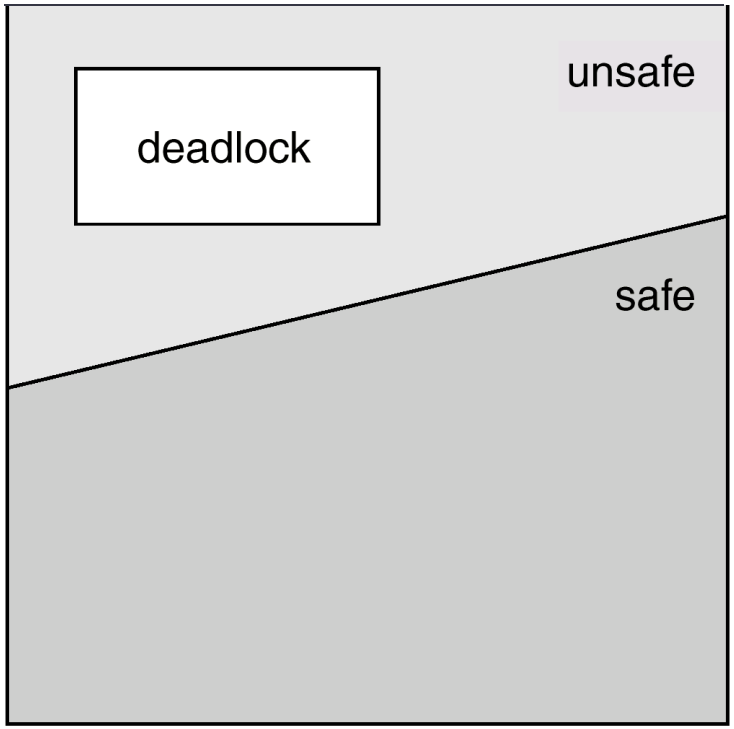
- Un algoritmo generico "per evitare i deadlock" esamina dinamicamente lo stato di allocazione delle risorse per accertarsi che la condizione di attesa circolare non possa mai verificarsi
  - Il modello più semplice consiste nel chiedere ad ogni processo di dichiarare il numero massimo di risorse che userà per ogni tipo
  - E iniziare ad assegnare le risorse solo se la richiesta complessiva non porterà allo stallo

- Due algoritmi più fini:
  - **Algoritmo del grafo di allocazione delle risorse** (per un singolo tipo di risorsa) [Dijkstra, 1965]
  - **Algoritmo del banchiere** [Habermann, 1969] (per risorse multiple)
- Entrambi basati sul concetto di **"stato sicuro"**

Stato sicuro (1)

- Def. 1: Uno stato si dice sicuro se il sistema può allocare le risorse richieste da ogni processo in un certo ordine garantendo che non si verifichi deadlock

  - stato sicuro
    - ➔ no deadlock
  - stato non sicuro
    - ➔ deadlock possibile
    - ➔ il SO non lo può impedire



Stato sicuro (2)

- Def. 2: Uno stato è sicuro se esiste una sequenza di processi sicura

- Sequenza sicura: Una sequenza di processi $<P_1, P_2, \ldots, P_n>$ è una sequenza sicura per l'allocazione corrente se le richieste che ogni processo P_i può fare possono essere soddisfatte dalle risorse attualmente disponibili più tutte le risorse detenute dai processi P_j con $j<i$

Stato sicuro (3)

- Si considerino i seguenti processi all'istante t_0 in un sistema che può fornire 12 istanze di una risorsa

t_o	Richieste max.	Unità possedute
P_0	10	5
P_1	4	2
P_2	9	2

- All'istante t_0 il sistema è in uno stato sicuro (3 ist. libere)
  - Sequenza P_1, P_0, P_2
    - P_1 riceve tutte le istanze necessarie (2) dopodiché le rilascia (5 istanze libere)
    - P_0 riceve tutte le istanze necessarie (5) dopodiché le rilascia (10 istanze libere)
    - P_2 riceve tutte le istanze necessarie (7) dopodiché le rilascia (12 istanze libere)

Stato sicuro (4)

- Un sistema può passare da uno stato sicuro ad uno insicuro
- Si consideri il sistema precedente nel quale all'istante t_1 (P1 ha tutte le istanze necessarie) P2 richiede e riceve un'ulteriore istanza

t_1	Richieste max.	Unità possedute
P_0	10	5
P_1	4	4
P_2	9	3

- Quanto P_1 rilascerà le istanze acquisite ci saranno 4 istanze libere e nessuno dei due processi potrà ricevere il massimo di istanze necessarie

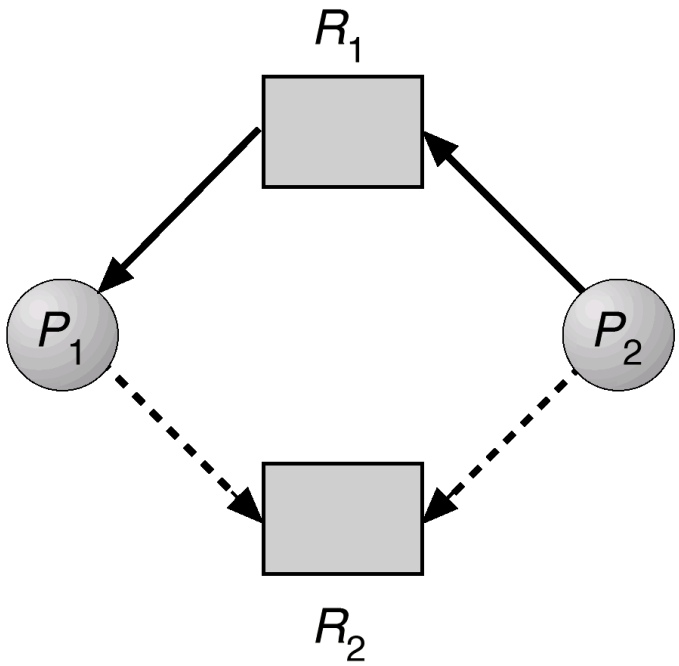
Come evitare il deadlock?

- **IDEA**: Garantire che il sistema passi da uno stato sicuro ad un altro stato sicuro quando un processo chiede una nuova risorsa

- **Schema**:
  - Si parte da uno stato iniziale sicuro
  - Una richiesta di risorsa viene soddisfatta se la risorsa è disponibile **e** se il sistema va in uno stato sicuro
  - Se la risorsa non è disponibile, il processo deve attendere

Algoritmo del grafo di allocazione delle risorse (1)

- Può essere applicato ai casi in cui ogni tipo di risorsa ha una sola istanza
- È un'estensione del grafo di assegnazione delle risorse
- Si introduce un nuovo **arco di reclamo** (da processo a risorsa)
  - Indica che in futuro il processo richiederà la risorsa



Algoritmo del grafo di allocazione delle risorse (2)

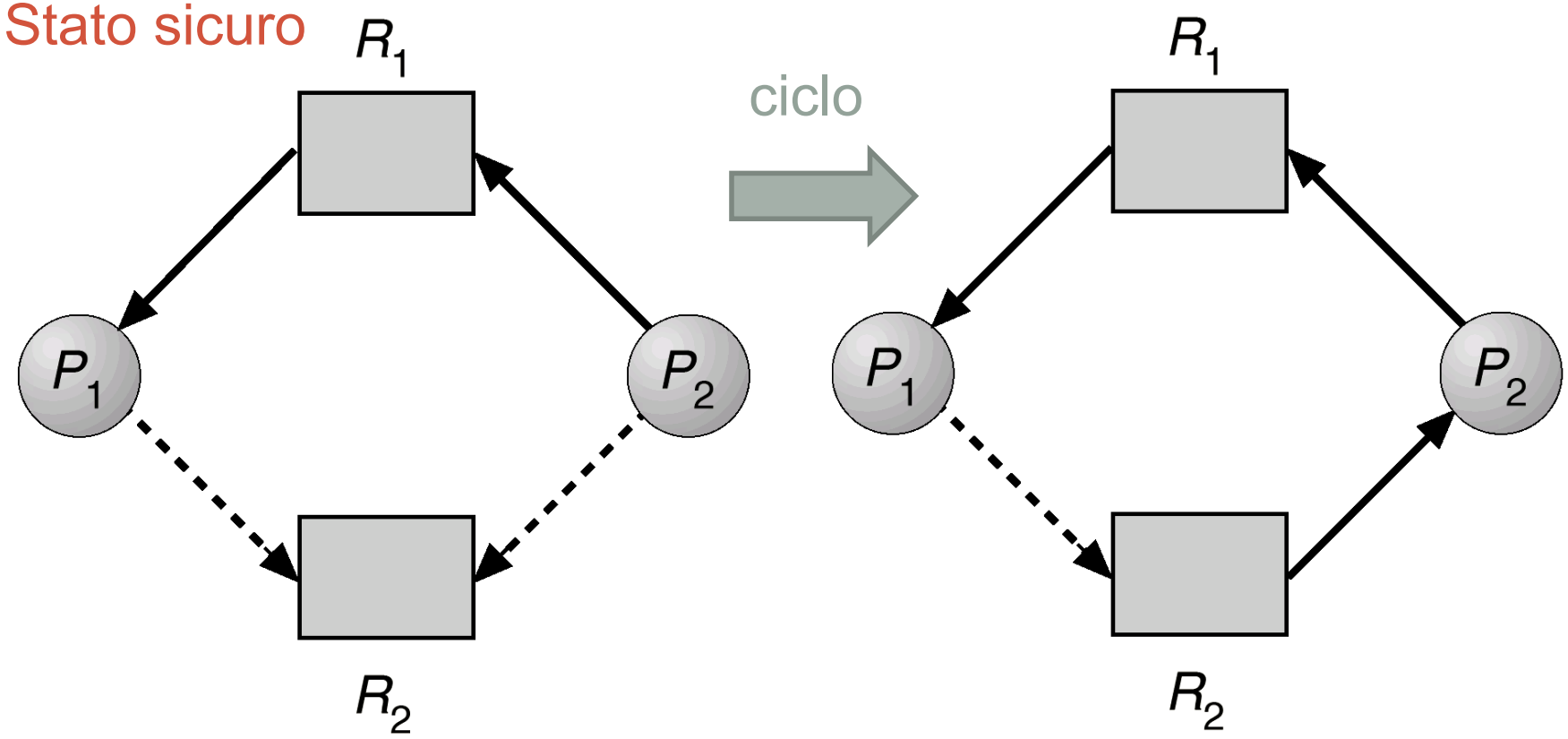
- Inizialmente si costruisce il grafo di allocazione con archi di reclamo (ci si basa su informazioni a priori sulle richieste future)
- Poi, applichiamo le *trasformazioni di archi* :
  - Quando un processo chiede una risorsa, l'arco di reclamo è convertito in un arco di richiesta
  - Quando una risorsa è assegnata, l'arco di richiesta è convertito in uno di assegnazione
  - Quando una risorsa è liberata dal processo, l'arco di assegnazione è convertito in un arco di reclamo
- Se si evidenziano **cicli** nel grafo -- verifica in $O(n^2)$ --, **lo stato non è sicuro** e quindi non si può accettare la richiesta di risorse dell'ultimo processo inserito

Algoritmo del grafo di allocazione delle risorse (3)

- Esempio: supponiamo che P2 chieda R2

Stato non sicuro

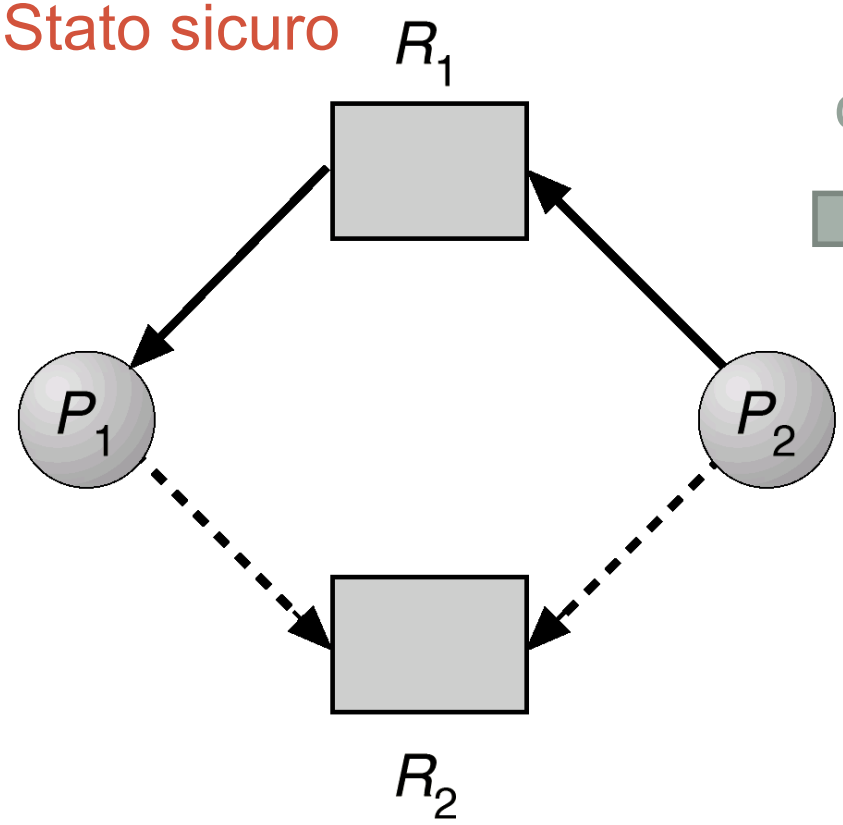
Stato sicuro



Algoritmo del grafo di allocazione delle risorse (4)

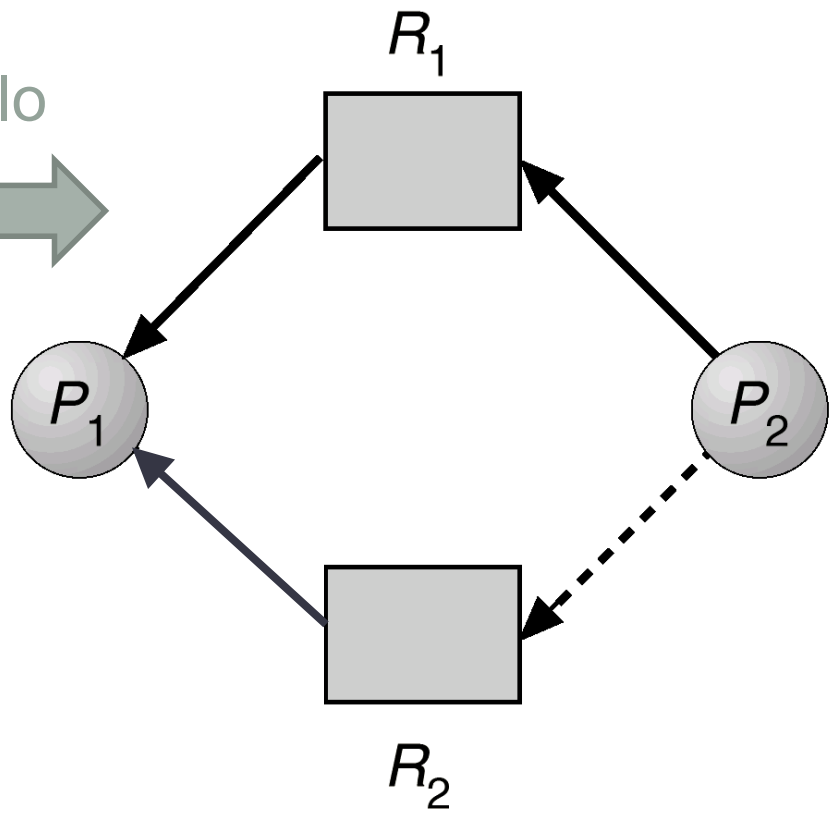
- Esempio: supponiamo che P1 chieda R2

Stato sicuro



ciclo

Stato sicuro



Algoritmo del banchiere (1)

- Gestisce istanze multiple delle risorse
- È tuttavia meno efficiente dell'algoritmo del grafo di allocazione delle risorse
- Il numero massimo di istanze richieste deve essere dichiarato a priori
- Un processo deve restituire in un tempo finito le risorse utilizzate

Algoritmo del banchiere (2)

Definisce le seguenti strutture dati

- m: numero risorse, n: numero processi

`Available[m]`	Numero istanze disponibili
`Max[n][m]`	Numero istanze massime richieste per processo
`Allocation[n][m]`	Numero istanze associate a ciasun processo
`Need[n][m]`	Numero risorse ancora necessarie a ciascun processo

- $\texttt{Allocation}_i$: vettore risorse assegnate al processo i (stessa notazione per `Max` e `Need`)
- `Need[i,j] = Max[i,j] – Allocation [i,j]`

Algoritmo del banchiere (3)

Procedura di verifica dello stato sicuro

Verifica la sicurezza di uno stato, complessità $O(mn^2)$

1. Si dichiarano i vettori `Work[m]` e `Finish[n]`
   - `Work=Available`; `Finish[i]=false` per `i=0,1,..,n-1`
2. Si cerca `i` tale che:
   - `Finish[i]==false && Need`$_i$ $\leq$ `Work`
   - N.B. La disuguaglianza sul vettore è vera se per ogni `j`
     - `Need`$_i$`[j]` $\leq$ `Work[j]`
3. Se tale `i` non esiste, si prosegue col punto al passo 5
4. `Work = Work+Allocation`$_i$; `Finish[i]=true;`
   - Vai al passo 2
5. Se, per ogni `i`, `Finish[i]==true`, allora lo stato è sicuro

Algoritmo del banchiere (4)
Procedura di richiesta delle risorse

Determina se una richiesta può garantire uno stato sicuro

- Sia $\texttt{Request}_i$ il vettore di richieste del processo P_i
  - riga i-esima della matrice `Request[n][m]`
- Se `Requesti[j] = k` allora P_i richiede k istanze di risorse di tipo R_j

1. Se $\texttt{Request}_i \leq \texttt{Need}_i$, si passa al punto 2
   - In caso contrario si solleva un errore: P_i ha ecceduto il numero max di richieste
2. Se $\texttt{Request}_i \leq \texttt{Available}$, si passa al punto 3
   - In caso contrario P_i deve attendere: risorse non disponibili

Algoritmo del banchiere (5)

Procedura di richiesta delle risorse

3. Si simula un assegnazione delle risorse richieste aggiornando lo stato:
   - $Available = Available - Request_i$
   - $Allocation_i = Allocation_i + Request_i$
   - $Need_i = Need_i - Request_i$
4. Si valuta attraverso l'algoritmo precedente la sicurezza dello stato risultante
   - Stato sicuro ➔ risorse concesse a Pi
   - Stato non sicuro ➔ Pi deve aspettare e viene ristabilito il vecchio stato di allocazione delle risorse

Algoritmo del banchiere (6)
Esempio

- Si consideri un sistema con 5 processi (da P_0 a P_4) e3 tipi di risorse
  - A (10 istanze), B (5 istanze) e C (7 istanze)
- Supponiamo che all'istante T_0 lo stato sia:

Available		
A	B	C
3	3	2

Allocation			
	A	B	C
P_0	0	1	0
P_1	2	0	0
P_2	3	0	2
P_3	2	1	1
P_4	0	0	2

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

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

Algoritmo del banchiere (7)

Esempio

- Il sistema si trova inizialmente in uno stato sicuro
- La sequenza $<P_1, P_3, P_4, P_0, P_2>$ soddisfa i criteri di sicurezza

Available		
A	B	C
3	3	2

Allocation			
	A	B	C
P_0	0	1	0
P_1	2	0	0
P_2	3	0	2
P_3	2	1	1
P_4	0	0	2

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

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

Algoritmo del banchiere (7)
Esempio

- Supponiamo ora che P_1 richieda un'istanza di A e due istanze di C
  - $Request_1$ = [1, 0, 2]
- Applicando il secondo algoritmo si passano con successo i punti 1 e 2
  - $Request_i \leq Need_i$ [1, 2, 3]
  - $Request_i \leq Available$ [3, 3, 2]
- Aggiorniamo lo stato

Allocation			
	A	B	C
P_0	0	1	0
P_1	3	0	2
P_2	3	0	2
P_3	2	1	1
P_4	0	0	2

Available		
A	B	C
2	3	0

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

Algoritmo del banchiere (8)
Esempio

- Verifichiamo ora che il nuovo stato sia sicuro
  - La sequenza $<P_1, P_3, P_4, P_0, P_2>$ soddisfa i criteri di sicurezza
  - Le risorse vengono quindi definitivamente assegnate

Allocation			
	A	B	C
P_0	0	1	0
P_1	3	0	2
P_2	3	0	2
P_3	2	1	1
P_4	0	0	2

Available		
A	B	C
2	3	0

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

Algoritmo del banchiere (9)
Esempio

- Supponiamo ora che P_0 richieda e due istanze di B
  - $Request_0$ = [0, 2, 0]
- Applicando il secondo algoritmo si passano con successo i punti 1 e 2
- Aggiorniamo lo stato
- Si può verificare che non è sicuro ($need_i > work_i$ per ogni i), le risorse non sono quindi assegnate

Allocation			
	A	B	C
P_0	0	3	0
P_1	3	0	2
P_2	3	0	2
P_3	2	1	1
P_4	0	0	2

Available		
A	B	C
2	1	0

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

4. Rilevazione e recupero

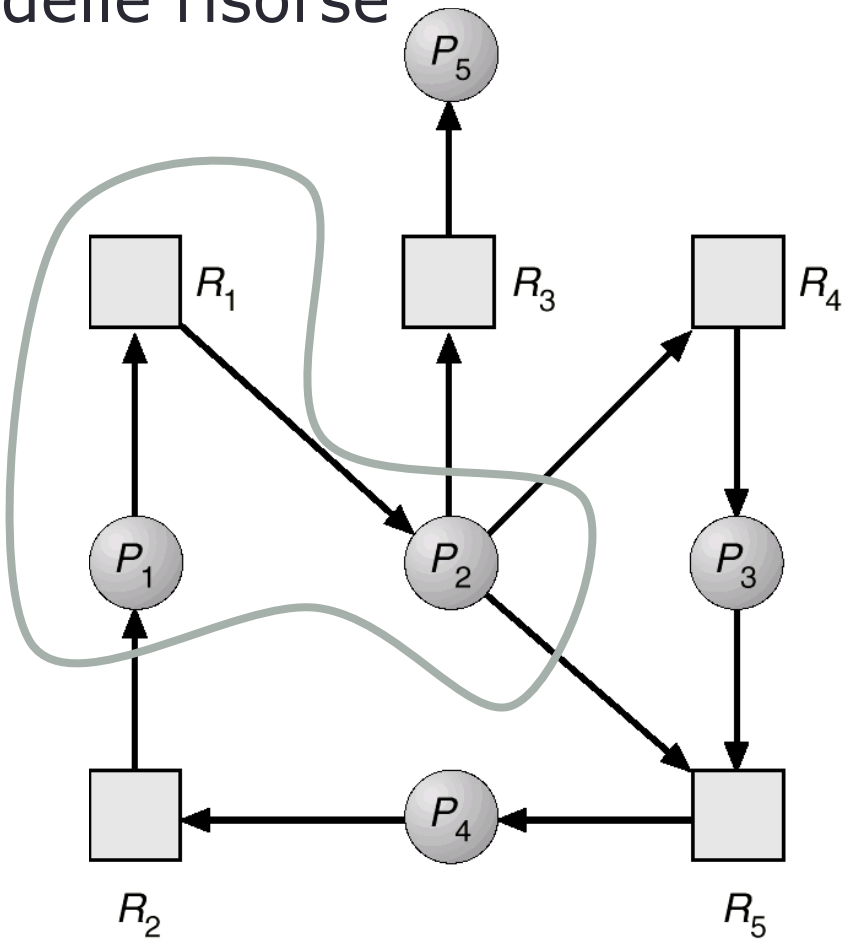
- Senza algoritmi di prevenzione o per evitare il deadlock, tale situazione può verificarsi il sistema deve essere in grado di:
  - **rilevare** la presenza di situazioni di deadlock **dopo** che sono avvenute
  - **ripristinare** una situazione di corretto funzionamento eliminando il deadlock

Rilevazione del deadlock

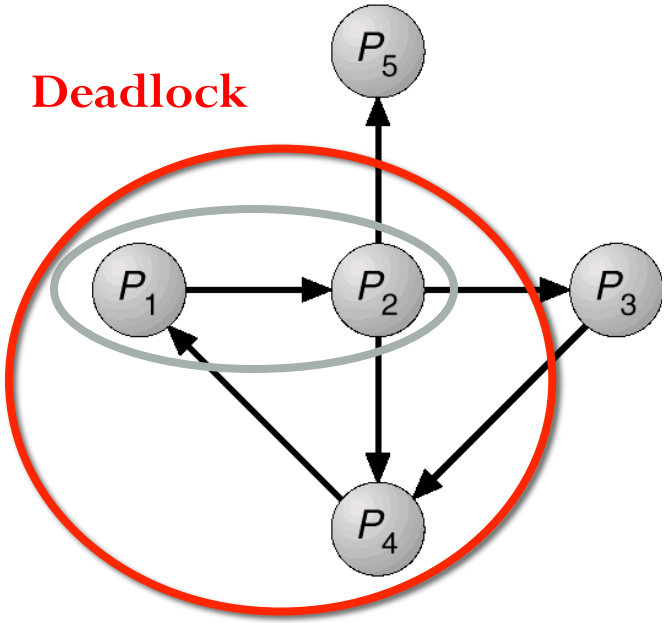
- Algoritmi di rilevamento :
  - Singole istanze per ogni risorsa
    - Si introduce il concetto di *grafo di attesa*
    - Evoluzione del grafo di assegnazione delle risorse
      - Vengono tolti i nodi delle risorse e si congiungono gli archi
      - $P_i \rightarrow R_k \rightarrow P_j$ diventa $P_i \rightarrow P_j$
    - Un ciclo rileva la presenza di un deadlock
  - Istanze multiple: algoritmo completo [Coffman *et al.*]

Rilevazione con istanze singole delle risorse (1)

- Grafo di allocazione delle risorse



Grafo di attesa (wait-for)



Rilevazione con istanze singole delle risorse (2)

- Mantenimento e analisi "periodica" del grafo di attesa
- Verifica in tempo $O(n^2)$
- I processi in deadlock sono quelli coinvolti in ciascun ciclo presente nel grafo

Rilevazione con istanze multiple delle risorse (1)

Definisce le seguenti strutture dati

- m: numero risorse, n: numero processi

`Available[m]`	Numero istanze disponibili di ciascuna risorsa
`Allocation[n][m]`	Numero istanze associate a ciasun processo
`Request[n][m]`	Numero istante attualmente richieste da ciascun processo

- $\texttt{Allocation}_i$: vettore risorse assegnate al processo i (stessa notazione per `Request`)

Rilevazione con istanze multiple delle risorse (2)
Procedura di rilevazione dei deadlock

Verifica la presenza di un deadlock, complessità $O(mn^2)$

1. Si dichiarano i vettori `Work[m]` e `Finish[n]`
   - `Work=Available`;
   - `Finish[i]=false` se $assegnate_i \neq 0$, `true` altrimenti
     - N.B $assegnate_i \neq 0$ se `assegnate[i][k] ≠0 per ogni k`
2. Si cerca `i` tale che:
   - `Finish[i]==false && Request`$_i \leq$ `Work`
     - N.B. La disuguaglianza sul vettore è vera se $Request_i[j] \leq Work[j]$ per ogni `j`
3. Se tale `i` non esiste, si prosegue col punto al passo 5

Rilevazione con istanze multiple delle risorse (3)
Procedura di rilevazione dei deadlock

Verifica la presenza di un deadlock, complessità $O(mn^2)$

4. `Work = Work+Allocation`$_i$; `Finish[i]=true`
   - Vai al passo 2
5. Se, per qualche `i`, `Finish[i]==false`, allora il sistema è in stallo
   - Più precisamente i processi caratterizzati da tale indice sono in stallo

Rilevazione con istanze multiple delle risorse (3) Esempio

- Si consideri un sistema con 5 processi (da P_0 a P_4) e 3 tipi di risorse
  - A (7 istanze), B (2 istanze) e C (6 istanze)
- Supponiamo che all'istante T_0 lo stato sia:

Available		
A	B	C
0	0	0

Tutti i processi hanno almeno una risorsa assegnata

Allocation			
	A	B	C
P_0	0	1	0
P_1	2	0	0
P_2	3	0	3
P_3	2	1	1
P_4	0	0	2

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

Finish	
P_0	False
P_1	False
P_2	False
P_3	False
P_4	False

Rilevazione con istanze multiple delle risorse (4) Esempio

- Per P_0 vale che `Finish[0]==false && Request`$_0$ $\leq$ `Work`
- Aggiorniamo `Work = Work + Allocation`$_0$ e `finish[0]=true`

Work		
A	B	C
0	0	0

Work		
A	B	C
0	1	0

Allocation			
	A	B	C
P_0	0	1	0
P_1	2	0	0
P_2	3	0	3
P_3	2	1	1
P_4	0	0	2

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

Finish	
P_0	True
P_1	False
P_2	False
P_3	False
P_4	False

Rilevazione con istanze multiple delle risorse (5) Esempio

- Per P_2 vale che `Finish[2]==false && Request`$_2$ $\leq$ `Work`
- Aggiorniamo `Work = Work + Allocation`$_2$ e `finish[2]=true`

Work		
A	B	C
0	1	0

Work		
A	B	C
3	1	3

Allocation			
	A	B	C
P_0	0	1	0
P_1	2	0	0
P_2	3	0	3
P_3	2	1	1
P_4	0	0	2

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

Finish	
P_0	True
P_1	False
P_2	True
P_3	False
P_4	False

Rilevazione con istanze multiple delle risorse (6) Esempio

- Per P_3 vale che `Finish[3]==false && Request`$_3$ $\leq$ `Work`
- Aggiorniamo `Work = Work + Allocation`$_3$ e `finish[3]=true`

Work		
A	B	C
3	1	3

Work		
A	B	C
5	2	4

Allocation			
	A	B	C
P_0	0	1	0
P_1	2	0	0
P_2	3	0	3
P_3	2	1	1
P_4	0	0	2

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

Finish	
P_0	True
P_1	False
P_2	True
P_3	True
P_4	False

Rilevazione con istanze multiple delle risorse (7) Esempio

- Per P_4 vale che `Finish[4]==false && Request`$_4$ $\leq$ `Work`
- Aggiorniamo `Work = Work + Allocation`$_4$ e `finish[4]=true`

Work		
A	B	C
5	2	4

Work		
A	B	C
5	2	6

Allocation			
	A	B	C
P_0	0	1	0
P_1	2	0	0
P_2	3	0	3
P_3	2	1	1
P_4	0	0	2

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

Finish	
P_0	True
P_1	False
P_2	True
P_3	True
P_4	True

Rilevazione con istanze multiple delle risorse (8) Esempio

- Per P_1 vale che `Finish[1]==false && Request`$_1$ $\leq$ `Work`
- Aggiorniamo `Work = Work + Allocation`$_1$ e `finish[1]=true`

Work		
A	B	C
5	2	6

Work		
A	B	C
7	2	6

Allocation			
	A	B	C
P_0	0	1	0
P_1	2	0	0
P_2	3	0	3
P_3	2	1	1
P_4	0	0	2

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

Finish	
P_0	True
P_1	True
P_2	True
P_3	True
P_4	True

Rilevazione con istanze multiple delle risorse (9) Esempio

- La sequenza <P_0, P_2, P_3, P_4, P_1> porta ad avere `Finish[i]==true` per ogni `i`
- Il sistema non è in stallo

Finish	
P_0	True
P_1	True
P_2	True
P_3	True
P_4	True

Rilevazione con istanze multiple delle risorse (10) Esempio

- Supponiamo ora che il processo P_2 faccia richiesta di un'istanza di C
- Eseguiamo di nuovo l'algoritmo di rilevazione

Available		
A	B	C
0	0	0

Tutti i processi hanno almeno una risorsa assegnata

Allocation			
	A	B	C
P_0	0	1	0
P_1	2	0	0
P_2	3	0	3
P_3	2	1	1
P_4	0	0	2

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

Finish	
P_0	False
P_1	False
P_2	False
P_3	False
P_4	False

Rilevazione con istanze multiple delle risorse (11) Esempio

- Per P_0 vale che `Finish[0]==false && Request`$_0$ $\leq$ `Work`
- Aggiorniamo `Work = Work + Allocation`$_0$ e `finish[0]=true`

Work		
A	B	C
0	0	0

Work		
A	B	C
0	1	0

Allocation			
	A	B	C
P_0	0	1	0
P_1	2	0	0
P_2	3	0	3
P_3	2	1	1
P_4	0	0	2

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

Finish	
P_0	True
P_1	False
P_2	False
P_3	False
P_4	False

Rilevazione con istanze multiple delle risorse (12) Esempio

- A questo punto non è possibile trovare P_i tale che `Finish[i]==false && Request`$_i \leq$ `Work`
- I processi P_1, P_2, P_3 e P_4 sono in stallo

Work		
A	B	C
0	1	0

Allocation			
	A	B	C
P_0	0	1	0
P_1	2	0	0
P_2	3	0	3
P_3	2	1	1
P_4	0	0	2

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

Finish	
P_0	True
P_1	False
P_2	False
P_3	False
P_4	False

Rilevazione con istanze multiple delle risorse (12) Esempio

- Quando invocare l'algoritmo di rilevazione?

- Ogni volta che una richiesta di allocazione non può essere immediatamente soddisfatta
  - Rilevazione immediata del processo causante lo stallo
  - Pochi risorse e processi bloccati
  - Considerevole sovraccarico computazionale
- A intervalli di tempo prestabiliti
  - Minor sovraccarico computazionale
  - Rilevazione del causante più complessa
  - Molte risorse e processi possono essere bloccati da tempo
- Quando l'utilizzo della CPU scende sotto una determinata soglia
  - Le situazioni di stallo fanno calare drasticamente l'efficienza della CPU

Ripristino del deadlock (1)

- Un deadlock può essere ripristinato secondo due modalità
  - Terminare uno o più dei processi convolti nel deadlock
  - Esercitare la prelazione delle risorse

Ripristino del deadlock (2)

Terminazione dei processi in deadlock

- Il sistema recupera immediatamente le risorse assegnate ai processi che vengono terminati
- Due modalità di terminazione:
  - Abortire tutti i processi in deadlock
    - Semplice e poco oneroso in termini computazionali
    - Ma costoso: i processi potrebbero aver compiuto molto lavoro che andrebbe perso
  - Abortire un processo alla volta fino al ripristino
    - Meno costoso in termini di calcolo
    - Ma oneroso in termini computazionali
      - Algoritmo di rilevazione eseguito dopo ogni terminazione

Ripristino del deadlock (3)

Terminazione dei processi in deadlock

- Terminare un processo può non essere sufficiente:
  - Le risorse possono rimanere in uno stato inconsistente
  - Devono essere ripristinate
    - Esempio: se un processo viene terminato mentre sta leggendo un file, il file risulterà aperto in lettura dopo la terminazione. È necessario aggiornare la lista dei file aperti
- La scelta del processo da terminare viene fatta in base a criteri di costo:
  - Priorità dei processi
  - Tempo trascorso dall'avvio e tempo restante
  - Quantità e tipo di risorse impiegate
  - Quantità di risorse ancora da richiedere

Ripristino del deadlock (4)

Rilascio anticipato delle risorse

- Le risorse vengono sottratte ad alcuni processi ad assegnate ad altri fino a che non si risolve il deadlock

- Bisogna tenere in considerazione tre fattori:
  - È necessario **selezionare quali risorse e quali processi** sottoporre a prelazione
    - Criteri di minimizzazione del costo (simili ai precedenti)
  - Occorre **ripristinare** il processo a cui è stata sottratta una risorsa
    - Ideale: ricondurlo in uno stato sicuro
    - Costoso in quanto è necessario salvare informazioni sugli stati precedenti
    - Pratica: tipicamente il processo viene terminato e riavviato

Ripristino del deadlock (5)

Rilascio anticipato delle risorse

- Occorre evitare l'**attesa indefinita**
  - Cioè non terminare sempre lo stesso processo
  - Si utilizza anche il numero di terminazioni come fattore di costo