Sistemi Operativi
Esercizi Sincronizzazione

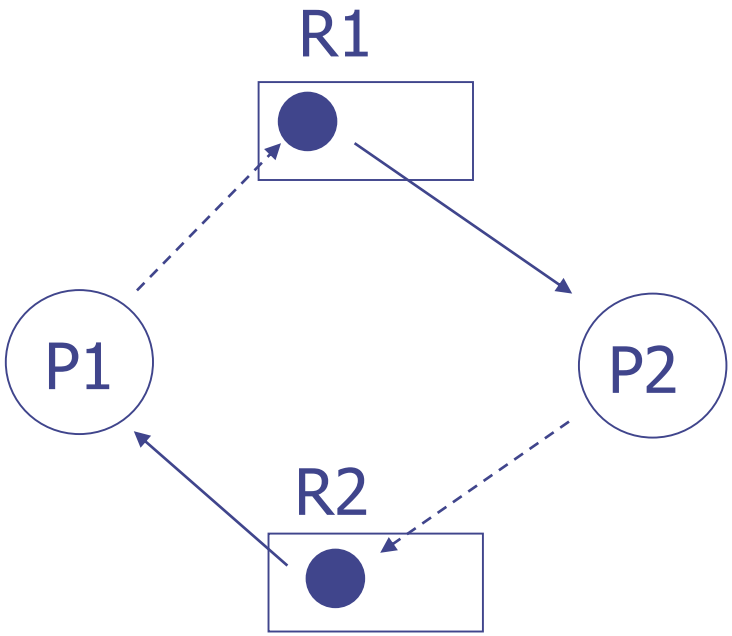
Docente: Claudio E. Palazzi
cpalazzi@math.unipd.it

Grafo di Assegnazione delle Risorse

- Si consideri un sistema con 2 processi (P1, P2), e 2 tipologie di risorse (R1, R2) con disponibilità: 1 risorsa di tipo R1 e 1 risorsa di tipo R2.
- Si consideri la seguente cronologia di richieste:
  1) P2 richiede R1
  2) P1 richiede R2
  3) P2 richiede R2
  4) P1 richiede R1
- Verificare se alla fine il sistema si trovi in condizioni di stallo

Grafo di Assegnazione delle Risorse

- Il grafo di allocazione delle risorse sarà il seguente:



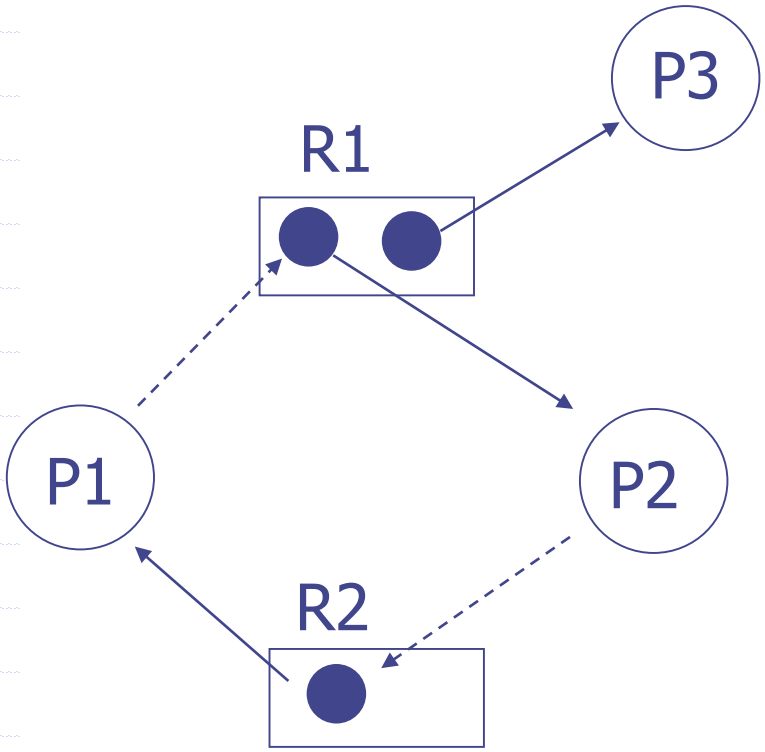
Si forma un ciclo: dunque il sistema è in **stallo**

Grafo di Assegnazione delle Risorse

- Si consideri un sistema con 3 processi (P1, P2, P3), e 2 tipologie di risorse (R1, R2) con disponibilità: 2 risorse di tipo R1 e 1 risorsa di tipo R2.
- Si consideri la seguente cronologia di richieste:
  1) P2 richiede R1
  2) P1 richiede R2
  3) P2 richiede R2
  4) P3 richiede R1
  5) P1 richiede R1
- Verificare se alla fine il sistema si trovi in condizioni di stallo

Grafo di Assegnazione delle Risorse

- Il grafo di allocazione delle risorse sarà il seguente:



Si forma un ciclo ma il sistema NON è in **stallo**

Infatti un elemento del ciclo (R1) ha molteplicità 2 e una delle due istanze è fuori da cicli: P3 può terminare e rilasciare una istanza di R1 che viene così assegnata a P1

Verifica Deadlock

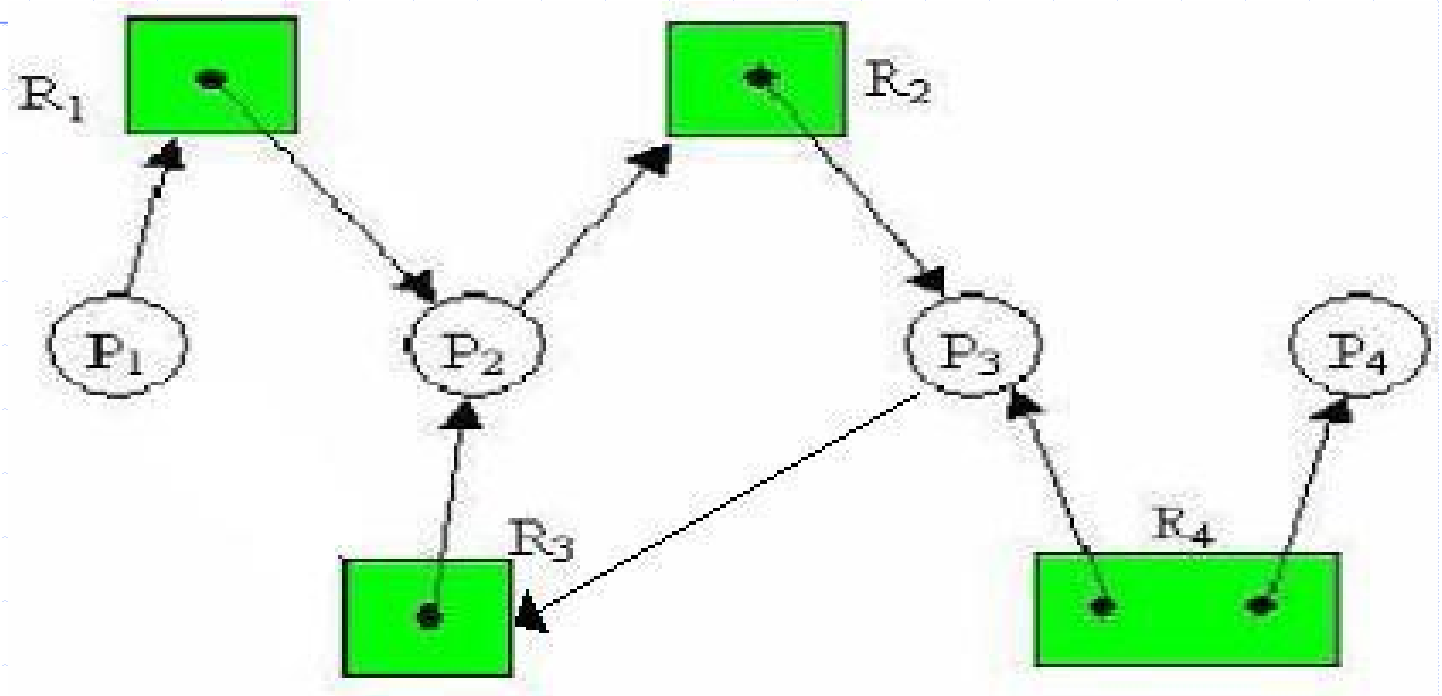
- Si consideri un sistema con 4 processi (P1, P2, P3, P4), e 4 tipologie di risorse (R1,R2,R3,R4) con disponibilità: 1 risorsa di tipo R1, 1 risorsa di tipo R2, 1 risorsa di tipo R3, 2 risorse di tipo R4.
- Si assuma che:
  - ogni volta che un processo richieda una risorsa libera, questa venga assegnata al processo richiedente;
  - ogni volta che un processo richieda una risorsa già occupata, il richiedente deve attendere che la risorsa si liberi prima di impossessarsene (coda FIFO di attesa)
- Si consideri la seguente successione cronologica di richieste e rilasci di risorse:

  1) P2 richiede R1,R2,R3
  2) P3 richiede R2,R4
  3) P2 rilascia R2
  4) P4 richiede R4
  5) P1 richiede R1
  6) P2 richiede R2
  7) P3 richiede R3
- Verificare se alla fine il sistema si trovi in condizioni di stallo

Verifica Deadlock

- Si consideri un sistema con 4 processi (P1, P2, P3, P4), e 4 tipologie di risorse (R1,R2,R3,R4) con disponibilità: 1 risorsa di tipo R1, 1 risorsa di tipo R2, 1 risorsa di tipo R3, 2 risorse di tipo R4.
- Si assuma che:
  - ogni volta che un processo richieda una risorsa libera, questa venga assegnata al processo richiedente;
  - ogni volta che un processo richieda una risorsa già occupata, il richiedente deve attendere che la risorsa si liberi prima di impossessarsene (coda FIFO di attesa)
- Si consideri la seguente successione cronologica di richieste e rilasci di risorse:
  1) P2 richiede R1,R2,R3 ➡ Le risorse sono libere, quindi vengono assegnate, compresa R2
  2) P3 richiede R2,R4 ➡ P3 ottiene R4, ma non R2 che è assegnata a P2
  3) P2 rilascia R2 ➡ R2, rilasciata, viene assegnata subito a P3 che l'aveva richiesta
  4) P4 richiede R4 ➡ Assegnata, poiché vi sono 2 istanze di R4
  5) P1 richiede R1 ➡ R1 è già assegnata a P2 e la richiesta non può essere soddisfatta
  6) P2 richiede R2 ➡ R2 è già assegnata a P3 e la richiesta non può essere soddisfatta
  7) P3 richiede R3 ➡ R3 è già assegnata a P2 e la richiesta non può essere soddisfatta
- Verificare se alla fine il sistema si trovi in condizioni di stallo

Sol – Grafo Allocazione Risorse



- Alla fine delle operazioni descritte, il grafo di allocazione delle risorse appare come in figura. In tale grafo esiste un ciclo di richieste/assegnazioni che coinvolge P2, R2, P3, R3: pertanto, il sistema **è in stallo**.

Verifica Deadlock

- Dato il sistema descritto dalla seguente rappresentazione insiemistica di assegnazione delle risorse:

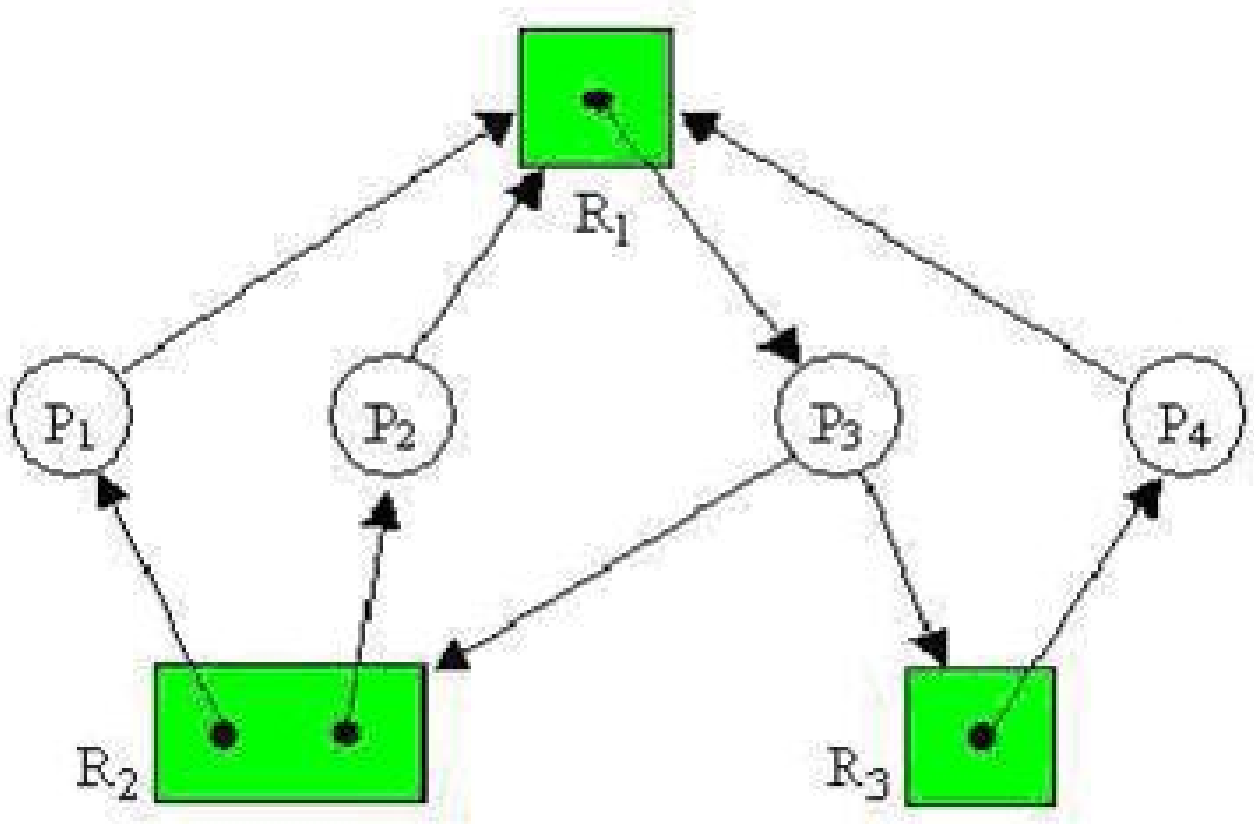
  P = P1, P2, P3, P4

  R = R1, R2, R2, R3

$$\mathrm{E} = P_1 \rightarrow R_1, P_2 \rightarrow R_1, P_3 \rightarrow R_2, P_3 \rightarrow R_3, P_4 \rightarrow R_1$$
$$R_1 \rightarrow P_3, R_2 \rightarrow P_1, R_2 \rightarrow P_2, R_3 \rightarrow P_4$$

- si verifichi, anche tramite analisi del relativo grafo di allocazione delle risorse, se il sistema si trovi in condizione di stallo.
- Successivamente, si discuta per ogni processo se la sua rimozione forzata (con conseguente liberazione delle risorse eventualmente in suo possesso ed annullamento delle sue richieste di risorse) porti ad un cambiamento della situazione precedentemente rilevata.

Sol – Grafo Allocazione Risorse



Sol – Grafo Allocazione Risorse

- La figura riporta la versione grafica della rappresentazione insiemistica data, al cui interno si distinguono 3 percorsi chiusi:

$$\text{percorso 1: } R_2 \rightarrow P_1 \rightarrow R_1 \rightarrow P_3 \rightarrow R_2$$
$$\text{percorso 2: } R_2 \rightarrow P_2 \rightarrow R_1 \rightarrow P_3 \rightarrow R_2$$
$$\text{percorso 3: } R_3 \rightarrow P_4 \rightarrow R_1 \rightarrow P_3 \rightarrow R_3$$

- I percorsi 1 e 2 contengono risorse a molteplicità > 1, e quindi per ciascuno di essi occorre una specifica analisi di dettaglio. Il percorso 3 invece contiene solo risorse unarie, per cui possiamo affermare che i processi P3 e P4 sono sicuramente in stallo. Essendo il processo P3, che è in stallo, presente anche nei percorsi 1 e 2, possiamo concludere che anche i processi P1 e P2 si trovano in situazione di stallo. Pertanto, in definitiva, l'intero sistema è in **stato di stallo**.

Sol – Grafo Allocazione Risorse

- Analizziamo ora la situazione che seguirebbe all'eliminazione di singoli processi dal sistema:
  - eliminiamo P1: la situazione non cambierebbe in quanto l'anello di sicuro stallo (percorso 3) non verrebbe intaccato; inoltre anche il processo P2 rimarrebbe bloccato dato che la risorsa R1 da esso richiesta è impegnata nel percorso 3.
  - eliminiamo P2: la situazione è analoga alla precedente: gli altri processi rimarrebbero bloccati.
  - eliminiamo P3: la situazione cambierebbe, dato che R1, posseduta dal processo P3 eliminato, diverrebbe libera; i processi P1, P2 e P4 hanno tutti una richiesta pendente per tale risorsa, ed indipendentemente dall'ordine con il quale possano venir soddisfatte tali richieste, i processi riprenderanno sequenzialmente ad avanzare, conseguentemente liberando il sistema dal precedente stato di stallo.
  - eliminiamo P4: la situazione non cambierebbe in quanto la risorsa R3, rilasciata dal processo eliminato, verrebbe assegnata al processo P3; che però resterebbe ancora bloccato stante la perdurante indisponibilità della risorsa R2, con il conseguente blocco dei percorsi 1 e 2.

Verifica Stato Sicuro

- Un sistema si compone di 4 processi e 5 risorse condivise a diversa molteplicità.
- All'istante considerato, lo stato di allocazione delle risorse a processi e i loro bisogni massimi previsti sono riportati in Tabella.

Processo	Risorse $(R_1, R_2, R_3, R_4, R_5)$	
	Allocazione attuale	Richiesta massima
A	1 0 2 1 1	1 1 2 1 4
B	2 0 1 1 1	2 2 3 2 1
C	1 1 0 1 0	2 1 4 1 0
D	1 1 1 1 0	1 1 3 2 1

- Assumendo che il vettore di disponibità delle risorse allo stato corrente sia uguale a [0 0 2 1 2], si discuta se il sistema sia in uno **stato sicuro** (che eviti *deadlock* anche nel caso in cui tutti i processi richiedano simultaneamente il massimo delle risorse per loro richiedibili)

Verifica Deadlock – Sol

- Vettore disponibilità [0 0 2 1 2]

Processo	Risorse $(R_1, R_2, R_3, R_4, R_5)$	
	Allocazione attuale	Richiesta massima
A	1 0 2 1 1	1 1 2 1 4
B	2 0 1 1 1	2 2 3 2 1
C	1 1 0 1 0	2 1 4 1 0
D	1 1 1 1 0	1 1 3 2 1

- il processo D può essere eseguito fino alla fine
- Quando ha finito, il vettore delle risorse disponibili è [1 1 3 2 2]
- Sfortunatamente però ora il sistema potrebbe andare incontro a ***deadlock*** qualora i processi rimanenti chiedessero effettuassero la loro "Richiesta massima" di risorse
- Dunque la situazione iniziale non rappresentava uno **stato sicuro**

Algoritmo del banchiere

- Evita le situazioni di stallo
  - Come anche l'uso dei grafi di allocazione
- Detto così perché, simile a una banca (virtuosa), non versa mai tutte le risorse disponibili al fine di poter sempre soddisfare i propri clienti
- Richiede che i processi dichiarino il massimo quantitativo di risorse che useranno
- Ad ogni nuova richiesta verifica se l'assegnazione lascerebbe il sistema in uno stato sicuro
  - Se così è le risorse vengono assegnate
  - Viceversa il processo deve attendere

Realizzaz. algoritmo del banchiere

- N = num processi nel sistema; M = num risorse
- Servono alcune strutture dati:
  - **Disponibili:** matrice M; *Disponibili [j] = k* significa che sono disponibili k risorse del tipo *Rj*.
  - **Massimo:** matrice N x M; *Massimo [i,j] = k* significa che il processo *Pi* può richiedere un massimo di *k* istanze della risorsa di tipo *Rj*.
  - **Assegnate:** matrice N x M; *Assegnate [i,j] = k* significa che al processo *Pi* sono state attualmente assegnate *k* istanze della risorsa di tipo *Rj*.
  - **Necessità:** matrice N x M; *Necessità [i,j] = k* significa che il processo *Pi* per completare il suo compito può avere bisogno di altre *k* istanze della risorsa di tipo *Rj*.
    - Si noti che *Necessità [i,j] = Massimo [i,j] - Assegnate [i,j]*

Realizzaz. algoritmo del banchiere

- Algoritmo di verifica della sicurezza:
  1. Sia *Lavoro [j]= Disponibili [j]* per j = 0, 1, …, M-1
     e *Fine [i] = falso* per *i = 0, 1, …, N-1*
  2. Si cerchi indice *i* tale che valgano entrambe:
     a) *Fine [i] == falso*
     b) *Necessità [i, j]<= Lavoro [j]* per j = 0, 1, …, M-1

     Se tale *i* non esiste, esegue passo 4.
  3. *Lavoro [j] = Lavoro [j] + Assegnate [i, j]*

     Fine [i] = vero

     torna al passo 2
  4. Se *Fine [i] == vero* per ogni *i*, allora il sistema è in stato sicuro
- O (M x N x N)

Realizzaz. algoritmo del banchiere

- Algoritmo di richiesta delle risorse
  - Se *Richieste [i] <= Necessità [i]*, allora esegue passo 2, altrimenti ERRORE per superato num max di richieste
  - Se *Richieste [i] <= Disponibili* allora esegue passo 3, altrimenti *Pi* deve attendere che si liberino delle risorse
  - Simula l'assegnazione delle risorse richieste al processo *Pi*, modificando lo stato di assegnazione delle risorse come segue:
    - *Disponibili = Disponibili – Richieste [i]*
    - *Assegnate [i] = Assegnate [i] + Richieste [i]*
    - *Necessità [i] = Necessità [i] - Richieste [i]*
  - Se lo stato di assegnazione delle risorse risultante è sicuro, la transazione è completata e al processo *Pi* si assegnano le risorse richieste; altrimenti *Pi* deve attendere.

Esempio algoritmo del banchiere

- Un sistema ha 4 processi (A, B, C, D) e 5 risorse (R1, R2, R3, R4, R5) da ripartire. L'attuale allocazione e i bisogni massimi sono i seguenti:

Processo	*Allocate*	*Massimo*
A	1 0 2 1 1	3 1 2 1 3
B	2 0 1 1 1	3 3 4 2 1
C	1 1 0 1 0	2 1 4 1 0
D	1 1 1 1 0	1 1 3 2 1

- Considerando il vettore delle risorse disponibili uguale a [0 1 3 1 2], e utilizzando l'Algoritmo del Banchiere, si discuta se il sistema sia in uno stato sicuro.

Esempio algoritmo del banchiere

- Calcoliamo la matrice delle *Necessità*:

Processo	*Allocate*	*Massimo*	*Necessità*
A	1 0 2 1 1	3 1 2 1 3	2 1 0 0 2
B	2 0 1 1 1	3 3 4 2 1	1 3 3 1 0
C	1 1 0 1 0	2 1 4 1 0	1 0 4 0 0
D	1 1 1 1 0	1 1 3 2 1	0 0 2 1 1

- Inizialmente il vettore delle risorse disponibili è uguale a [0 1 3 1 2], quindi *Lavoro* è [0 1 3 1 2].
- Inoltre *Fine* è [0 0 0 0]

Esempio algoritmo del banchiere

Processo	*Allocate*	*Massimo*	*Necessità*
A	1 0 2 1 1	3 1 2 1 3	2 1 0 0 2
B	2 0 1 1 1	3 3 4 2 1	1 3 3 1 0
C	1 1 0 1 0	2 1 4 1 0	1 0 4 0 0
D	1 1 1 1 0	1 1 3 2 1	0 0 2 1 1

- Il proc. D potrebbe essere eseguito fino alla fine poiché ciascun elemento di Necessità è minore o uguale al corrispondente elemento di *Lavoro* [0 1 3 1 2]
  Al termine *Lavoro* diventa [1 2 4 2 2], in quanto somma del vettore precedente e delle risorse allocate a D precedentemente [1 1 1 1 0].
  Inoltre *Fine* diventa [0 0 0 1].

Esempio algoritmo del banchiere

Processo	*Allocate*	*Massimo*	*Necessità*
A	1 0 2 1 1	3 1 2 1 3	2 1 0 0 2
B	2 0 1 1 1	3 3 4 2 1	1 3 3 1 0
C	1 1 0 1 0	2 1 4 1 0	1 0 4 0 0
D	1 1 1 1 0	1 1 3 2 1	0 0 2 1 1

- Dopo, il proc. C potrebbe essere eseguito e al suo completamento, il vettore delle risorse disponibili (vettore *Lavoro* nell'algoritmo) diventa [2 3 4 3 2] mentre *Fine* diventa [0 0 1 1].
- Questo permette di eseguire e terminare il processo A ottenendo [3 3 6 4 3] come *Lavoro* e [1 0 1 1] come *Fine*.
- Così si può eseguire e terminare anche il processo B.
- **Il sistema è quindi in uno stato sicuro**.