Scuola universitaria professionale della Svizzera italiana
Dipartimento tecnologie innovative
Istituto sistemi informativi e networking

SUPSI

Deadlock e Starvation

Amos Brocco, Ricercatore, DTI / ISIN

3 aprile 2012

Condivisione di risorse

- In un sistema si trovano delle risorse che possono essere utilizzate (correttamente) solo da un processo/thread alla volta
  - stampanti
  - dischi
  - variabili in memoria

- Per gestire gli accessi concorrenti si usano dei meccanismi di mutua esclusione che garantiscono a un processo/thread l'accesso esclusivo a una risorsa

Utilizzo di una risorsa

(1) Richiesta della risorsa
- ...attesa se la risorsa è occupata
  - il processo è bloccato
  - o può terminare con un errore

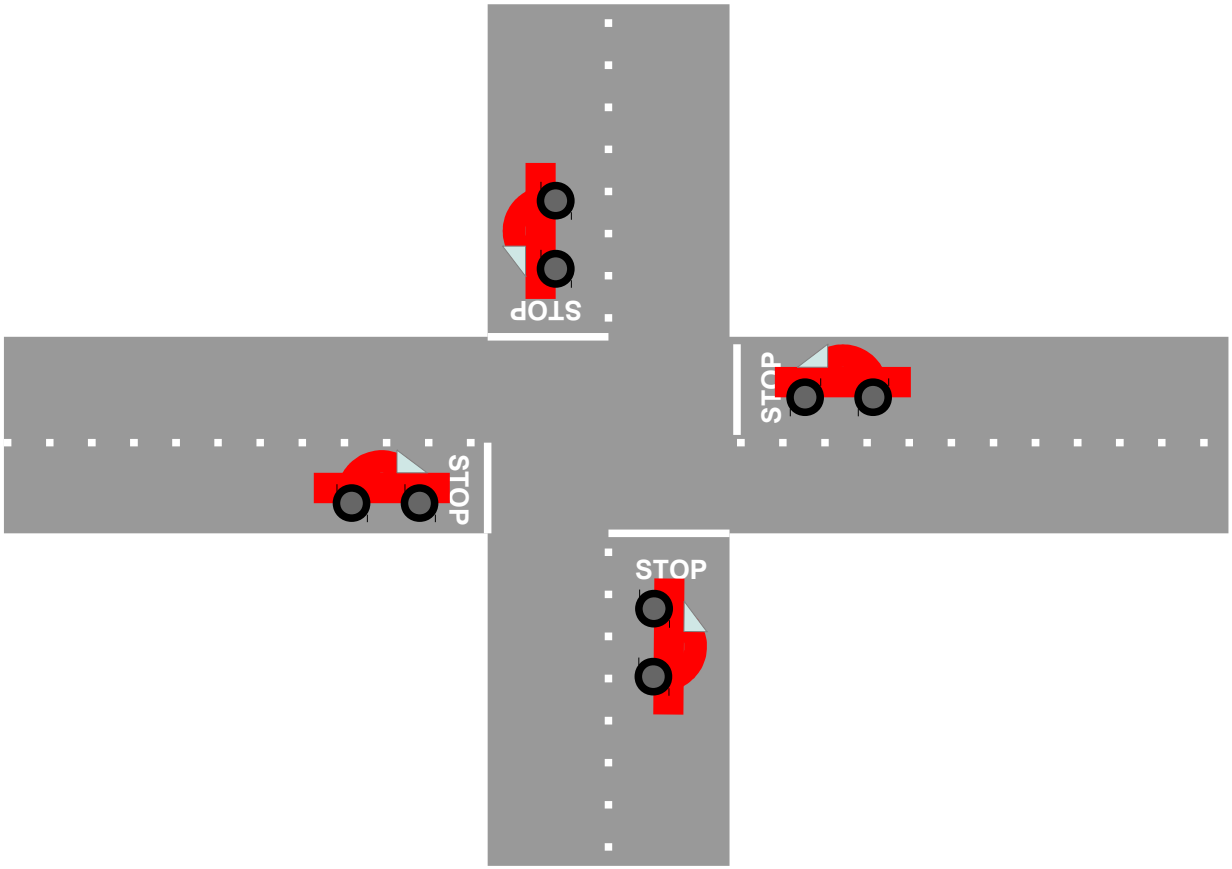
(2) Utilizzo della risorsa

(3) Rilascio della risorsa

Condivisione di risorse

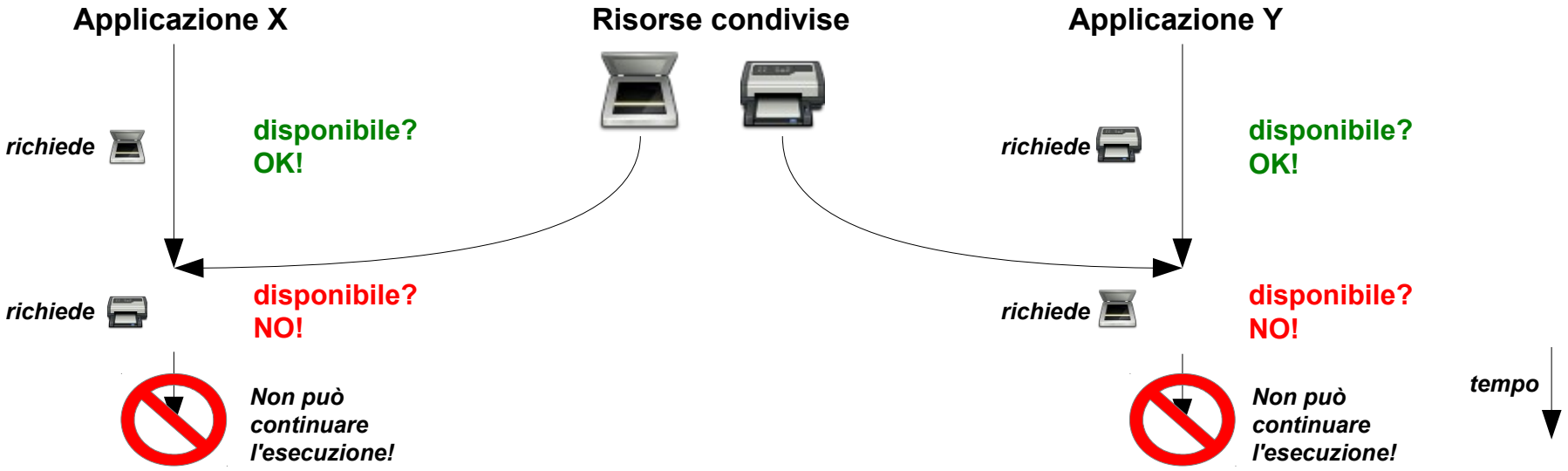
- Cosa succede se più risorse cercano di accedere contemporaneamente alla stessa risorsa?

Un esempio



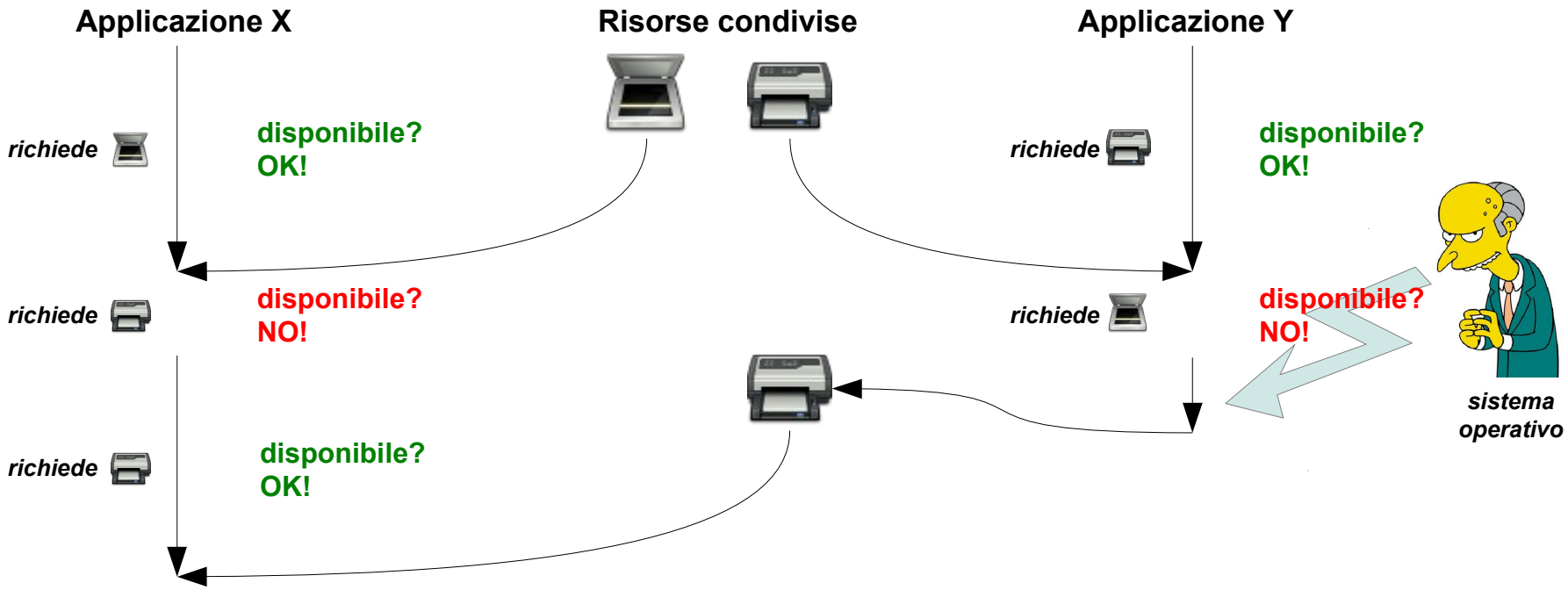
Un altro esempio

- Supponiamo che due applicazioni, X e Y, vogliano fare una fotocopia...
  - Entrambe le applicazioni hanno bisogno di avere accesso esclusivo alla stampante e allo scanner
  - Il programmatore di X ha deciso di richiedere prima lo scanner e poi la stampante
  - Il programmatore di Y ha deciso di richiedere prima la stampante e poi lo scanner



E il sistema operativo?

- Ci si potrebbe chiedere perché il sistema operativo non regoli semplicemente la situazione togliendo la risorsa a un processo che la detiene...



Risorse non pre-rilasciabili

- Distinguiamo due tipi di risorse:
  - **Risorse pre-rilasciabili**: (*preemptable*) possono essere tolte a un processo senza causare errori
  - **Risorse non pre-rilasciabili**: (*nonpreemptable*) il processo da errore se le risorse vengono tolte

Deadlock

- Un **deadlock** si può verificare quando più processi/thread ottengono accesso esclusivo a una risorsa
  - Un insieme di processi/thread è in **deadlock** se ogni processo dell'insieme è in attesa di un evento (il rilascio di una risorsa) che solo un altro processo dell'insieme può causare
- I processi dell'insieme non possono continuare la loro esecuzione, né rilasciare le risorse che detengono in quel momento

Esempio: senza deadlock

```c
#include <pthread.h>
#include <stdio.h>

pthread_mutex_t alpha;
pthread_mutex_t beta;

void* pallino() {
  pthread_mutex_lock(&alpha);
  printf("Pallino: Ho alpha!\n");
  pthread_mutex_lock(&beta);
  printf("Pallino: Ho beta!\n");
  pthread_mutex_unlock(&beta);
  printf("Pallino: Ho rilasciato beta!\n");
  pthread_mutex_unlock(&alpha);
  printf("Pallino: Ho rilasciato alpha!\n");
}
```

```c
void* pinco() {
  pthread_mutex_lock(&alpha);
  printf("Pinco: Ho alpha!\n");
  pthread_mutex_lock(&beta);
  printf("Pinco: Ho beta!\n");
  pthread_mutex_unlock(&beta);
  printf("Pinco: Ho rilasciato beta!\n");
  pthread_mutex_unlock(&alpha);
  printf("Pinco: Ho rilasciato alpha!\n");
}
void main() {
  pthread_t t1, t2;
  pthread_mutex_init (&alpha, NULL);
  pthread_mutex_init (&beta, NULL);
  pthread_create(&t1, NULL, &pinco, NULL);
  pthread_create(&t2, NULL, &pallino, NULL);
  pthread_join(t1, NULL);
  pthread_join(t2, NULL);
  pthread_mutex_destroy (&alpha);
  pthread_mutex_destroy (&beta);
}
```

Esempio: possibile deadlock

```
#include <pthread.h>
#include <stdio.h>

pthread_mutex_t alpha;
pthread_mutex_t beta;

void* pallino() {
  pthread_mutex_lock(&beta);
  printf("Pallino: Ho beta!\n");
  pthread_mutex_lock(&alpha);
  printf("Pallino: Ho alpha!\n");
  pthread_mutex_unlock(&alpha);
  printf("Pallino: Ho rilasciato alpha!\n");
  pthread_mutex_unlock(&beta);
  printf("Pallino: Ho rilasciato beta!\n");
}
```

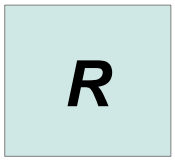
```
void* pinco() {
  pthread_mutex_lock(&alpha);
  printf("Pinco: Ho alpha!\n");
  pthread_mutex_lock(&beta);
  printf("Pinco: Ho beta!\n");
  pthread_mutex_unlock(&beta);
  printf("Pinco: Ho rilasciato beta!\n");
  pthread_mutex_unlock(&alpha);
  printf("Pinco: Ho rilasciato alpha!\n");
}
void main() {
  pthread_t t1, t2;
  pthread_mutex_init (&alpha, NULL);
  pthread_mutex_init (&beta, NULL);
  pthread_create(&t1, NULL, &pinco, NULL);
  pthread_create(&t2, NULL, &pallino, NULL);
  pthread_join(t1, NULL);
  pthread_join(t2, NULL);
  pthread_mutex_destroy (&alpha);
  pthread_mutex_destroy (&beta);
}
```

Condizioni necessarie per avere un deadlock

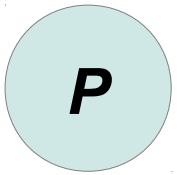
- **Mutua esclusione**
  - ogni risorsa è assegnata al massimo a un processo / thread
- **Richiesta incrementale**
  - i processi / thread che detengono una risorsa possono richiedere altre risorse
- **Risorse non pre-rilasciabili**
  - Le risorse assegnate non possono essere tolte dal sistema operativo. Sono i processi / thread stessi che devono rilasciarle
- **Attesa circolare**
  - Deve esistere una catena circolare di due o più processi in cui ogni processo attende una risorsa posseduta dal processo successivo nella catena

Modellizzazione dei deadlock

- Con un **grafo di allocazione** è possibile modellizzare i processi e le risorse, in modo da rilevare i possibili deadlock



Risorsa



Processo

- Una freccia da una risorsa a un processo indica che la risorsa è in possesso del processo

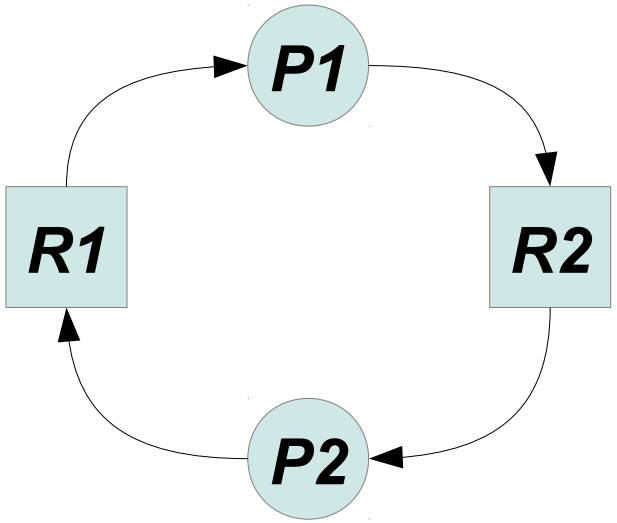
P detiene la risorsa R

- Una freccia da un processo a una risorsa indica che il processo è in attesa di quella risorsa

P è in attesa della risorsa R (attualmente detenuta da un altro processo)

Deadlock

- Se nel grafo di allocazione sono presenti dei cicli significa che c'è un deadlock



Come gestire un deadlock

- Esistono diverse strategie
  - ignorare il problema
  - riconoscere la situazione e risolverla
  - gestire in modo adeguato le allocazioni di risorse
  - evitare una delle quattro condizioni necessarie per un deadlock

Ignorare il problema

- ... algoritmo dello struzzo (*metti la testa sotto la sabbia e fai finta che il problema non esiste*)

- Se il problema non si presenta troppo spesso (o con meno frequenza rispetto ad altri problemi, es. hardware), e se la soluzione risulta troppo complessa, è meglio non pagare il prezzo
  - soluzione “implementata” dai sistemi operativi attuali

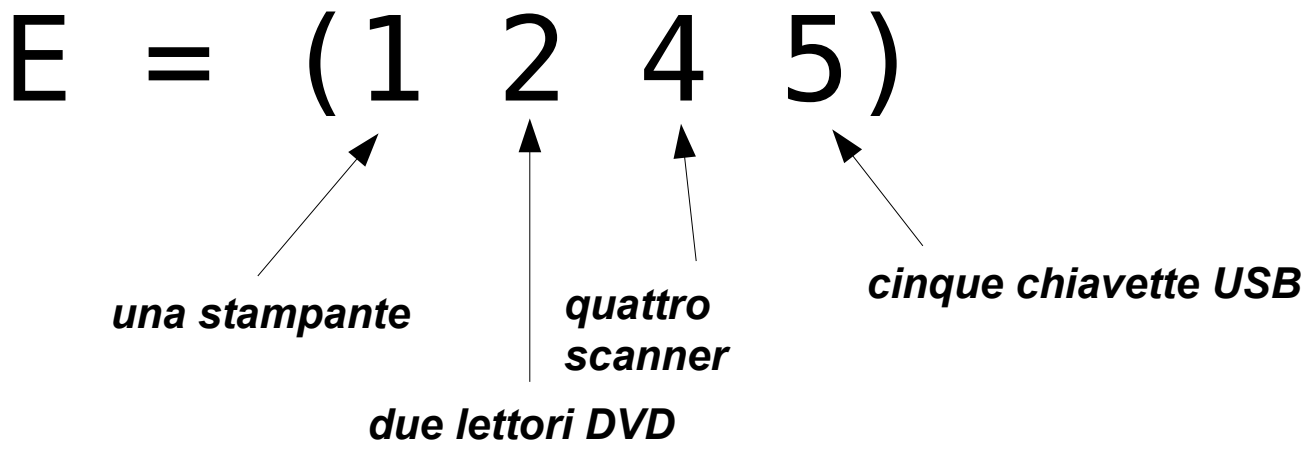
Riconoscere la situazione di deadlock e risolverla

- Possiamo costruire un grafo di allocazione e controllare costantemente se ci sono deadlock
  - se abbiamo più di una risorsa per ogni tipo diventa troppo complesso

Vettori e matrici di allocazione

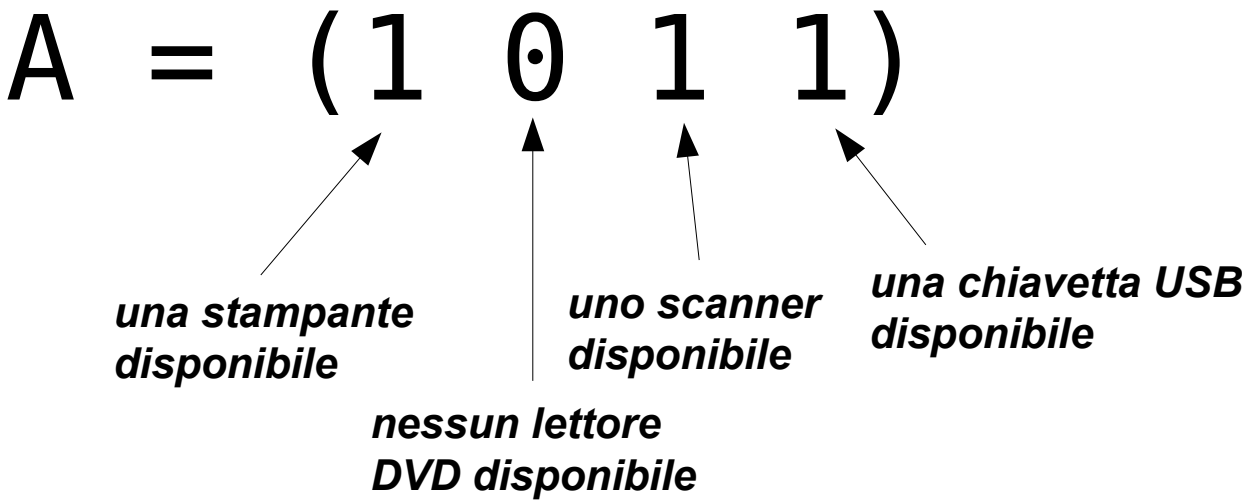
- Questo metodo permette di riconoscere situazioni di deadlock in sistemi in cui esistono più risorse per ogni tipo (es. due stampanti)

- Indichiamo, in un **vettore delle risorse esistenti E**, il numero di risorse per ogni classe che abbiamo sul sistema:



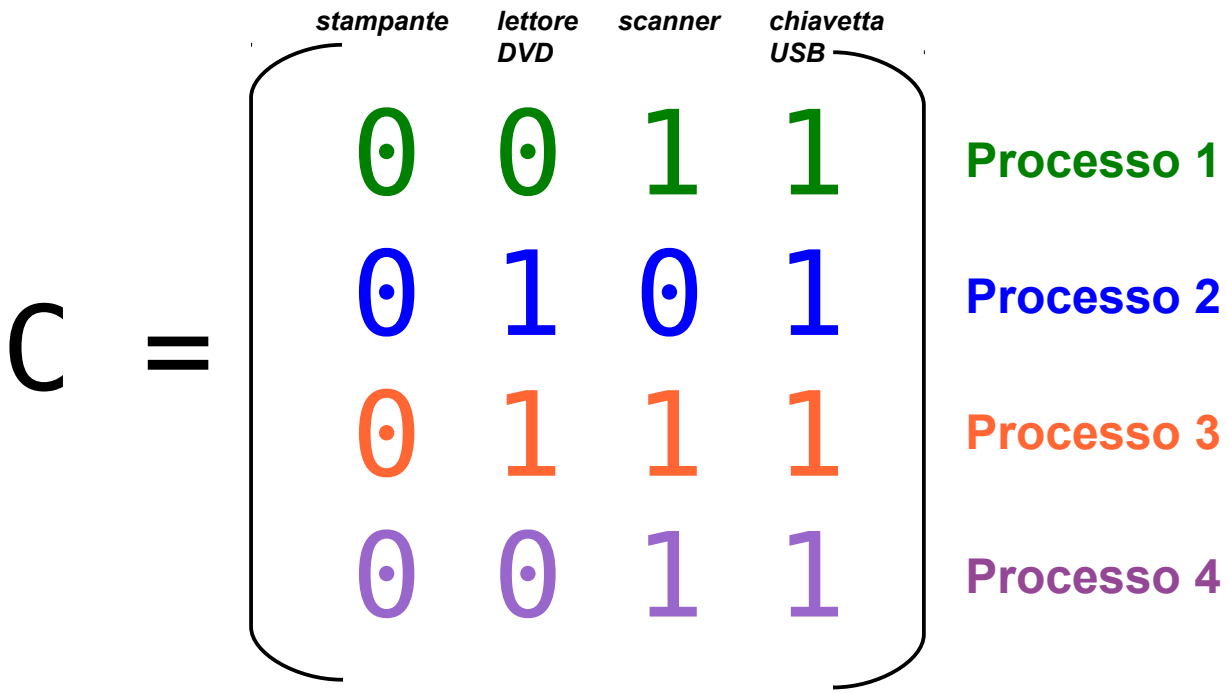
Vettori e matrici di allocazione

- In un altro **vettore delle risorse disponibili A**, il numero di risorse per ogni classe che sono attualmente disponibili (non utilizzate da processi) sul sistema:



Vettori e matrici di allocazione

- Per specificare come sono attualmente assegnate le risorse utilizziamo una **matrice di allocazione C** :
  - ogni riga indica le risorse allocate attualmente ad un processo
  - le colonne indicano come le risorse sono allocate attualmente



Vettori e matrici di allocazione

- Le richieste future di ogni processo sono indicate in una **matrice di richiesta R** :
  - ogni riga indica le risorse necessarie al processo per continuare l'esecuzione e poi terminare

$$R = \begin{pmatrix} \textit{stampante} & \textit{lettore DVD} & \textit{scanner} & \textit{chiavetta USB} \\ 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 1 \\ 0 & 0 & 0 & 1 \\ 0 & 1 & 1 & 1 \end{pmatrix} \begin{matrix} \\ \textbf{Processo 1} \\ \textbf{Processo 2} \\ \textbf{Processo 3} \\ \textbf{Processo 4} \end{matrix}$$

Vincoli

- La somma delle risorse allocate e di quelle disponibili deve essere uguale al numero delle risorse esistenti

- Un processo non avere accesso a più risorse di quelle esistenti (allocazione corrente + richieste future ≤ risorse esistenti)

Algoritmo di verifica (algoritmo del banchiere)

- Dato il vettore delle risorse disponibili A, cerco nella matrice delle richiesta R un processo P che è possibile soddisfare:
  - se trovo un processo valido, aggiungo i valori nella riga di C corrispondente a P al vettore A, cancello le righe corrispondenti a P dalle matrici R e C e ricomincio
  - se non trovo niente l'algoritmo termina

- Se quando l'algoritmo termina ci sono ancora processi nella matrice il sistema è in deadlock
  - un sistema è in uno **stato sicuro** se c'è un ordine di schedulazione per cui ogni processo riesce a terminarsi

Esempio

A = (1 0 1 1)

E = (1 2 4 5)

C =

0	0	1	1
0	1	0	1
0	1	1	1
0	0	1	1

R =

R1	*R2*	*R3*	*R4*	
1	0	0	0	*P1*
0	0	1	1	*P2*
0	0	0	1	*P3*
0	1	1	1	*P4*

Cerco nella matrice delle richiesta R un processo P che è possibile soddisfare...

- Posso scegliere P1, P2, o P3: scelgo **P2**

A = (1 **1** 1 **2**)

C =

0	0	1	1
~~0~~	~~1~~	~~0~~	~~1~~
0	1	1	1
0	0	1	1

R =

1	0	0	0
~~0~~	~~0~~	~~1~~	~~1~~
0	0	0	1
0	1	1	1

Cerco nella matrice delle richiesta R un processo P che è possibile soddisfare...

- Posso scegliere P1, P3, o P4: scelgo **P4**

A = (1 1 **2** **3**)

C =

0	0	1	1
~~0~~	~~1~~	~~0~~	~~1~~
0	1	1	1
~~0~~	~~0~~	~~1~~	~~1~~

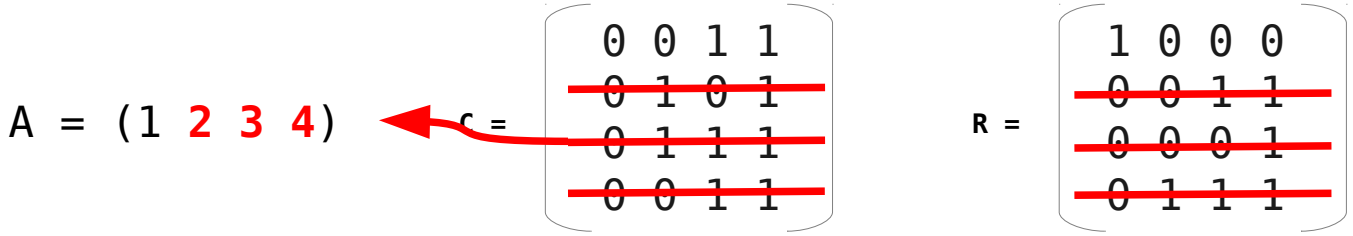
R =

1	0	0	0
~~0~~	~~0~~	~~1~~	~~1~~
0	0	0	1
~~0~~	~~1~~	~~1~~	~~1~~

Esempio

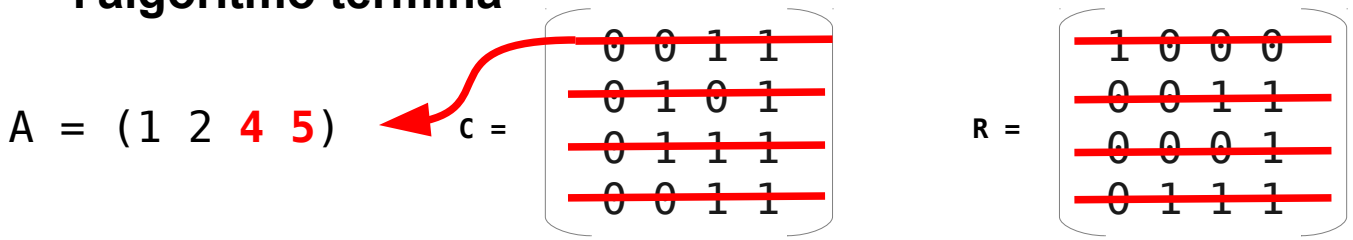
Cerco nella matrice delle richiesta R un processo P che è possibile soddisfare...

- Posso scegliere P1 o P3: scelgo **P3**



Cerco nella matrice delle richiesta R un processo P che è possibile soddisfare...

- scelgo **P1**
- **l'algoritmo termina**



Il sistema era in uno stato sicuro, non c'è rischio di deadlock!

Un altro esempio

A = (1 0 1 1)

E = (1 2 4 5)

C =

0	0	1	1
0	1	0	1
0	1	1	1
0	0	1	1

R =

R1	R2	R3	R4	
1	2	3	0	*P1*
0	0	1	1	*P2*
0	0	0	2	*P3*
0	2	3	1	*P4*

Cerco nella matrice delle richiesta R un processo P che è possibile soddisfare...

- Posso scegliere solo **P2**

A = (1 **1** 1 **2**)

C =

0	0	1	1
~~0~~	~~1~~	~~0~~	~~1~~
0	1	1	1
0	0	1	1

R =

1	2	3	0
~~0~~	~~0~~	~~1~~	~~1~~
0	0	0	2
0	2	3	1

Cerco nella matrice delle richiesta R un processo P che è possibile soddisfare...

- Posso scegliere solo **P3**

A = (1 **2** **2** **3**)

C =

0	0	1	1
~~0~~	~~1~~	~~0~~	~~1~~
~~0~~	~~1~~	~~1~~	~~1~~
0	0	1	1

R =

1	2	3	0
~~0~~	~~0~~	~~1~~	~~1~~
~~0~~	~~0~~	~~0~~	~~2~~
0	2	3	1

Un altro esempio

Cerco nella matrice delle richiesta R un processo P che è possibile soddisfare...

- Non posso scegliere nessun processo!
  - sia P1 che P4 richiedono 3 risorse di tipo R3, ma solo 2 sono disponibili

$$A = (1\ 2\ 2\ 3) \qquad C = \begin{pmatrix} 0 & 0 & 1 & 1 \\ 0 & 1 & 0 & 1 \\ 0 & 1 & 1 & 1 \\ 0 & 0 & 1 & 1 \end{pmatrix} \qquad R = \begin{pmatrix} 1 & 2 & 3 & 0 \\ 0 & 0 & 1 & 1 \\ 0 & 0 & 0 & 2 \\ 0 & 2 & 3 & 1 \end{pmatrix}$$

stato non sicuro: rischio un deadlock!

Recupero da un deadlock

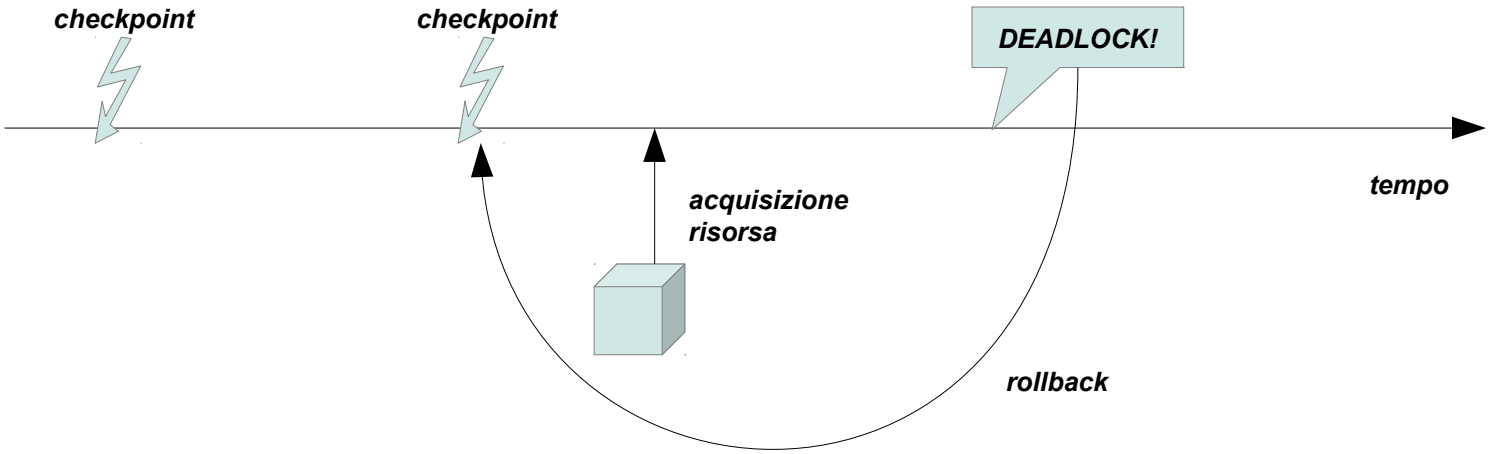
- Se ci si trova in una situazione di deadlock, come ci si deve comportare?

Pre-rilascio (preemption)

- Se è possibile tolgo una risorsa a un processo e la passo ad un altro processo
  - dipende dalla risorsa: per esempio con una stampante si può interrompere la stampa corrente, mettere da parte quanto già stampato, passare il controllo ad un altro processo, e poi riprendere
  - spesso difficile o impossibile

Rollback

- Periodicamento salvo lo stato di esecuzione (memoria, risorse) dei processi (**checkpoint**)
  - se ho un deadlock il processo che ha la risorsa viene reinizializzato (**rollback**) all'ultimo checkpoint prima dell'acquisizione



Uccisione di un processo

- Per risolvere il deadlock uccido uno dei processi in attesa circolare
  - Quale processo? Chi sceglie?

Evitare i deadlock

- Possiamo prevenire i deadlock evitando una delle quattro condizioni necessarie?

Impedire la mutua esclusione

- Senza mutua esclusione non è possibile che un processo blocchi una risorsa
  - ...ma possono nascere dei conflitti
  - ...una possibile soluzione è permettere a un solo processo di utilizzare una risorsa:
    - es. stampante: un processo che vuole stampare salva il documento in una directory di spooling; un demone (server) con accesso esclusivo alla stampante prende un documento alla volta dalla directory e lo stampa
    - ... ma non tutte le risorse possono funzionare così!

Impedire le richieste incrementali

- Se un processo che detiene una risorsa non può richiederne altre possiamo evitare una delle condizioni necessarie per un deadlock
  - non realistico!
- Possiamo richiedere ai processi di definire in anticipo quali risorse richiederanno, così da verificare se il sistema va incontro a uno stato non sicuro
  - non realistico!
- Oppure richiediamo che un processo deve rilasciare tutte le risorse (almeno temporaneamente) prima di richiederne un'altra, e poi provare a ri-acquisirle tutte
  - non efficiente!

Rendere le risorse prerilasciabili

- Rendendo le risorse prerilasciabili perché il sistema non sa in che stato queste verranno tolte ai processi.

Impedire l'attesa circolare

- Per eliminare l'attesa circolare possiamo richiedere che un processo può detenere solo una risorsa alla volta, e se ne vuole un'altra deve prima rilasciare quella corrente
- In alternativa possiamo numerare tutte le risorse e definire la seguente regola di acquisizione:
  - i processi possono richiedere risorse ma solo se in ordine numerico
  - ... funziona, ma è difficile implementare questo meccanismo in modo da soddisfare tutti i processi e gli utenti!

Two-phase locking

- Per evitare i deadlock sono state sviluppate delle tecniche specifiche, come il **two-phase locking**
- L'algoritmo si compone di due fasi:
  - 1. Il processo cerca di acquisire tutte le risorse di cui ha bisogno, una alla volta
    - se ci riesce, passa alla fase 2
    - altrimenti rilascia tutte le risorse finora acquisite
  - 2. Il processo opera sulle risorse e le rilascia

Two-phase locking

- Utilizzato nei sistemi di basi dati
- Non applicabile in ogni situazione
  - non è sempre possibile ricominciare il processo di acquisizione se una delle risorse non è disponibile

Starvation

- In alcuni casi è possibile che un processo rimanga in attesa indefinita anche senza essere in deadlock
- Esempio:
  - consideriamo un servizio tecnico che risponde alle domande degli utenti via email:
    - il sistemista controlla periodicamente la sua casella di posta e prende le domande in entrata, e visto che viene pagato in base al numero di domande che riesce ad evadere, sceglie sempre quelle più corte
    - se c'è un invio continuo di domande corte e semplici, l'utente che invierà una domanda molto lunga potrebbe non ricevere mai una risposta
  - in un sistema di scheduling con politica Shortest-Job-First i lavori lunghi possono dover attendere per sempre
  - Si parla di **starvation** (morte per fame)
    - soluzione: cambiare politica di esecuzione (es. First-come first-served)