Il deadlock

Deadlock (blocco critico)

- Più processi possono entrare in competizione per ottenere l'uso di risorse.
  Se la risorsa richiesta *non è disponibile* il processo viene posto in condizione *di attesa*.
- Se un processo in attesa non cambia più il suo stato, cioè se le risorse richieste sono trattenute da altri processi essi pure in attesa, si ha una *situazione di deadlock* (blocco critico).
- Per blocco critico si intende una situazione nella quale *uno o più processi* rimangono *indefinitamente bloccati* a causa della impossibilità del verificarsi delle condizioni necessarie per il loro proseguimento.
- Nel caso di un solo processo bloccato si parla anche di *blocco individuale*.
- Si possono avere situazioni di deadlock sia nel caso di *interazione indiretta* tra processi (risorse riusabili) che nel caso di *interazione diretta* (risorse consumabili).

Risorse riusabile e risorse consumabili

Risorse riusabili: dopo il loro uso da parte di un processo possono essere usate da altri processi.

Proprietà:

- devono essere usate in modo esclusivo,
- (in genere) non possono essere sottratte al processo durante l'uso.

Esempi:

- risorse fisiche (stampanti, telescriventi, etc.)
- risorse logiche (files, tabelle, etc.).

Sono divise in *tipi*, ciascuno dei quali costituito da una o piu` unita` *equivalenti* (usabili in maniera indifferenziata dai processi).

Risorse consumabili:

- Sono *segnali* o *messaggi* scambiati tra processi (ad esempio i segnali e i messaggi delle pipe in UNIX)
- *Cessano di esistere* non appena acquisite da un processo.
- Sono potenzialmente in *numero infinito*.

Esempio di deadlock

- Due processi A e B necessitano entrambi delle risorse R1 (CDROM) e R2 (stampante)

1. A richiede e ottiene R1
2. B richiede e ottiene R2
3. A richiede R2 e viene bloccato
4. B richiede R1 e viene bloccato

Esempio di deadlock

Deadlock provocato da un uso scorretto delle primitive di sincronizzazione

processo P1	**processo P2**
...	...
wait (s1)	wait (s2)
...	...
wait (s2)	wait (s1)
...	...
signal (s2)	signal (s1)
...	...
signal (s1)	signal (s2)

(valore iniziale: s1=s2=1)

Condizioni per il deadlock

Coffman et al. (1971) hanno dimostrato che è **necessario** che si verifichino quattro condizioni perché vi sia deadlock:

1. **Mutua Esclusione**: ogni risorsa risulta assegnata esattamente ad un processo oppure è disponibile.

2. **Tieni e aspetta (hold and wait)** : i processi che detengono risorse assegnategli in precedenza possono richiederne di nuove.

3. **Assenza di revoca** : le risorse precedentemente assegnate non possono essere revocate forzatamente.

4. **Attesa circolare** : deve essere presente una lista circolare di almeno due processi ognuno dei quali è in attesa di una risorsa posseduta dal processo che segue nella lista.

Strategie per evitare il deadlock

1. **Ignorare il problema**
   - Algoritmo dello struzzo …

Anche UNIX può soffrire di deadlock

- Esempio: si supponga che la tabella dei processi abbia dimensione fissa di 100 elementi (nel kernel Linux 2.6 il limite è la memoria fisica disponibile)
  - Se la tabella è piena, fork() fallisce e il processo può decidere di riprovare dopo un intervallo casuale
  - 10 processi ciascuno dei quali vuole creare 12 processi figli
  - Dopo che ogni processo ha creato 9 figli c'è il deadlock

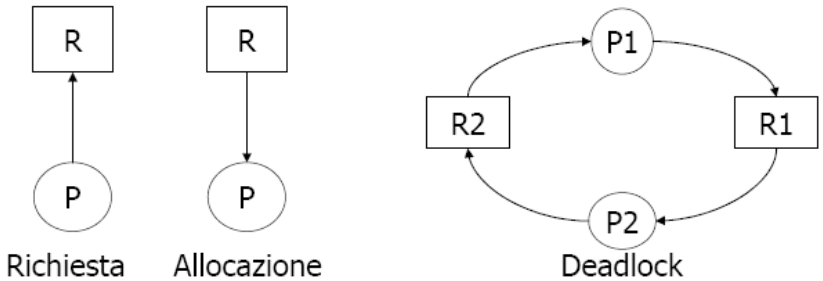
Meglio accettare deadlock occasionali piuttosto che dover utilizzare una sola risorsa alla volta

- compromesso tra utilità correttezza
- eliminare deadlock e' costoso

Strategie per evitare il deadlock

2. Rilevare la presenza di deadlock e risolverlo

- Periodicamente il S.O. controlla il grafo di allocazione delle risorse
- Se c'è un ciclo (deadlock) si terminano processi a caso nel ciclo (recuperando le loro risorse) fino a quando non vi è più deadlock



Richiesta Allocazione Deadlock

Strategie per evitare il deadlock

3. Prevenzione dinamica

- allocazione attenta delle risorse

Vincolare i processi in modo che il deadlock risulti strutturalmente impossibile

– Esempio : algoritmo del banchiere (Dijkstra)

Il banchiere (SO) non soddisfa le richieste di credito (risorse) dei clienti (processi) che possono portare a stati di deadlock

L'algoritmo richiede di conoscere a-priori il massimo numero di risorse richieste da ciascun processo e presuppone che ogni processo chieda sempre il *numero massimo* di risorse e le *mantenga tutte* durante l'esecuzione (caso peggiore).

4. Prevenzione Strutturale (statica)

- Occorre negare una delle 4 condizioni precedenti
- Normalmente si cerca di evitare la condizione di attesa circolare imponendo *un ordine sulla richiesta delle risorse*