CONCORRENZA: Stallo e Starvation

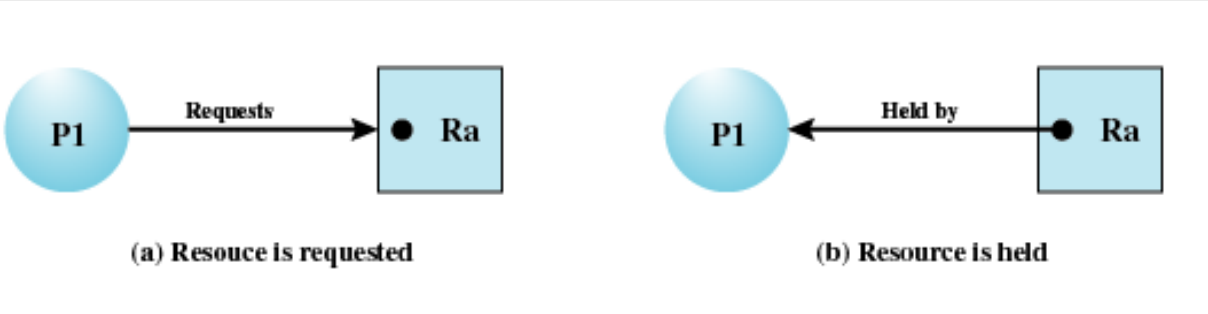
Introduzione allo Stallo

- Definizione :
  Un insieme di processi è in stallo se ogni processo nell'insieme è in attesa di un evento che solo un altro processo nello stesso insieme può generare.
- Generalmente l'evento è il rilascio di una risorsa
- Nessuno dei processi in stallo può …
  - passare in esecuzione
  - rilasciare risorse
  - essere riattivato

Grafici di Allocazione delle risorse:

– Il processo P1 richiede la risorsa Ra (fig. a)

– La risorsa Ra è posseduta dal processo P1 (fig. b)



Condizioni per lo Stallo

Mutua esclusione: un solo processo alla volta può usare una risorsa

"hold & wait" – possesso e attesa: un processo può mantenere il possesso delle risorse allocate mentre attende di averne altre

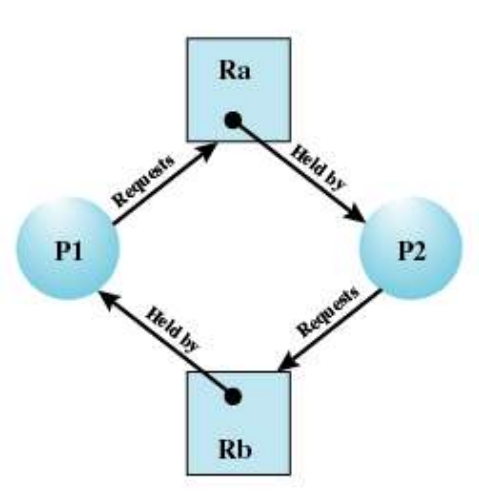
Assenza di prerilascio: i processi non possono essere forzati a rilasciare in anticipo le risorse acquisite

Attesa circolare

- almeno due processi
- ogni processo aspetta una risorsa occupata da un altro processo in attesa circolare

Condizioni necessarie ma non sufficienti

Condizioni necessarie e sufficienti



NB:

- *le prime tre condizioni derivano direttamente dalla progettazione (in molti casi sono auspicabili),*
- *la quarta è un evento che si può verificare e dipende dalla particolare sequenza di richieste e rilasci.*

Stallo

Strategie per affrontare il problema dello stallo

1. **Ignorare il problema (*algoritmo dello struzzo*)**
   - Soluzione adottata dalla maggior parte dei SO inclusi Unix e Windows
   - Soluzione ragionevole se lo stallo capita raramente o evitare lo stallo risulta molto costoso
   - Degrado delle prestazioni: blocco totale del sistema e riavvio manuale
   - Compromesso tra convenienza e correttezza

2. **Prevenirlo**: progettare un SO in modo che la possibilità di avere uno stallo sia esclusa a priori tramite la negazione di una delle 4 condizioni

3. **Esclusione**: le tre condizioni necessarie sono permesse, un algoritmo verifica dinamicamente che una richiesta non produca una situazione di stallo

4. **Consentire il verificarsi dello stallo, rilevarlo e risolverlo**.

Prevenzione dello Stallo

Classi di prevenzione: si impongono vincoli sulla richiesta delle risorse

Metodi indiretti: prevengono il verificarsi di una delle tre condizioni necessarie

Metodi diretti: prevengono il verificarsi dell'attesa circolare

- Mutua Esclusione
  - Un processo non deve mai attendere una risorsa condivisibile
  - Devono essere possibili accessi multipli contemporanei alle risorse condivise
  - Non applicabile….

- Hold and Wait
  - Ogni processo deve richiedere all'inizio della sua esecuzione tutte le risorse
  - Il processo entra in blocked fino a quando tutte le richieste non vengono soddisfatte contemporameamente
  - Approccio inefficiente:
    - Prima di andare in run tutte le risorse devono essere disponibili, in realtà potrebbe procedere utilizzandone solo una parte
    - Le risorse assegnate al processo potrebbero rimanere inutilizzate per molto tempo, gli altri processi sono in attesa indefinita

Prevenzione dello Stallo

- Assenza di prerilascio

  Due possibilità:

  1. Un processo che non riesce ad ottenere le risorse di cui necessita, rilascia quelle che detiene per richiederle nuovamente in un istante successivo
  2. OS può richiedere il pre-rilascio delle risorse al processo che le detiene per assegnarle al richiedente

  – Approccio applicabile solo se lo stato della risorsa è facilmente salvabile (CPU)
  – Non applicabile nel caso di stampanti….

- Attesa Circolare (metodi diretti)
  – Si definisce un ordine lineare (di numerazione) per tutte le risorse
  – Se un processo richiede una risorsa R, successivamente potrà richiedere solo una risorsa che nell'ordinamento segue R
  – Es.: R_i precede R_j se i<j, un processo potrà chiedere le risorse solo nell'ordine R_i, R_j
  – Questo metodo può essere inefficiente

Esclusione dello Stallo
DeadLock Avoidance

Esclusione:

•le tre condizioni necessarie
  – Mutua esclusione
  – Possesso e attesa
  – Assenza di pre-rilascio

sono permesse,

•un algoritmo verifica dinamicamente che una richiesta non produca una situazione attesa circolare

L'esclusione permette più concorrenza rispetto alla prevenzione

Approcci per escludere (evitare) lo stallo:

1. **Non avviare un processo** le cui richieste possono provocare lo stallo
2. **Non concedere ulteriori risorse** ad un processo se la loro allocazione può portare ad uno stallo

DeadLock Avoidance: Rifiuto del permesso di esecuzione

n – processi, *m* – tipi di risorse

$Risorse = (R_1, R_2, ..., R_m)$ *Numero di istanze per ogni risorsa*

$Disponibili = (V_1, V_2, ..., V_m)$ *Numero di istanze disponibili per ogni risorsa*

$$Richieste = \begin{pmatrix} C_{11} & C_{12} & \dots & C_{1m} \\ C_{21} & C_{22} & \dots & C_{2m} \\ \dots & \dots & \dots & \dots \\ C_{n1} & C_{n2} & & C_{nm} \end{pmatrix}_{nxm}$$

C_{ij} – richieste complessive da parte del processo i sulla risorsa j

$$Assegnazioni = \begin{pmatrix} A_{11} & A_{12} & \dots & A_{1m} \\ A_{21} & A_{22} & \dots & A_{2m} \\ \dots & \dots & \dots & \dots \\ A_{n1} & A_{n2} & & A_{nm} \end{pmatrix}_{nxm}$$

A_{ij} – assegnazioni correnti

Info dichiarate all'inizio

Risulta:

$$R_i = V_i + \sum_{k=1}^{n} A_{ki}, \quad \forall i$$
$$C_{ki} \le R_i, \quad \forall k,i$$
$$A_{ki} \le C_{ki}, \quad \forall k,i$$

Un processo P_{n+1} può essere avviato <u>se e solo se</u>: $R_i \ge C_{(n+1)i} + \sum_{k=1}^{n} C_{ki}, \quad \forall i$

Il processo può essere avviato se e solo se è possibile (per ogni risorsa) soddisfare la sua richiesta e quelle dei processi correnti.

Strategia non ottimale: si suppone che i processi necessitino delle risorse nello stesso istante.

Difficoltà: le richieste devono essere note all'avvio dei processi…

DeadLock Avoidance: Rifiuto di allocazione delle risorse

- Stato del sistema: assegnazione corrente di risorse ai processi, si costituisce delle matrici:
  - *Risorse*
  - *Disponibili*
  - *Richieste*
  - *Assegnazioni*

Lo stato in cui si trova il sistema può essere:

- SICURO: esiste almeno una sequenza di esecuzione dei processi che ne consente la terminazione senza incorrere in una situazione di stallo
- NON SICURO

Quando un processo richiede una risorsa disponibile, il sistema :
-Simula l'assegnazione delle risorse e l'aggiornamento dello stato
-Verifica che il nuovo stato sia uno stato sicuro
-In caso positivo assegna le risorse,
-In caso negativo rifiuta la richiesta e il processo viene bloccato fino a che la risorsa non si rende disponibile

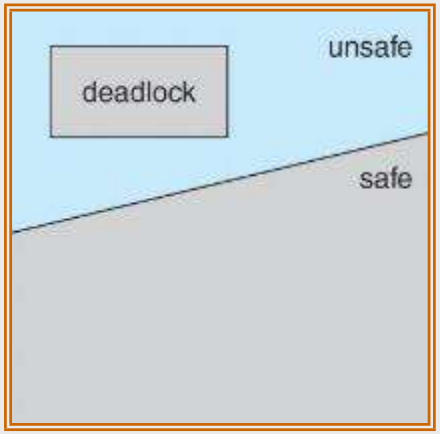
Stato Sicuro

Osservazioni:

- Se le risorse richieste da P_i non sono immediatamente disponibili, allora P_i deve attendere la terminazione di un processo che rilascia un numero sufficiente di risorse.
- Quando un processo P_j termina, P_i può ottenere le risorse necessarie all'esecuzione e terminare a sua volta rilasciando le risorse detenute.

- Stato sicuro $\Rightarrow$ no deadlocks.
- Stato non sicuro $\Rightarrow$ possibilità di deadlock.
- Avoidance $\Rightarrow$ assicurarsi che il sistema non entri in uno stato non sicuro.

Uno stato non sicuro non rappresenta uno stato di stallo, ma uno stato in cui lo stallo è possibile. La strategia di esclusione dello stallo non si basa sulla certezza che uno stato sia di stallo, ma sulla possibilità che uno stato non sicuro determini uno stallo.



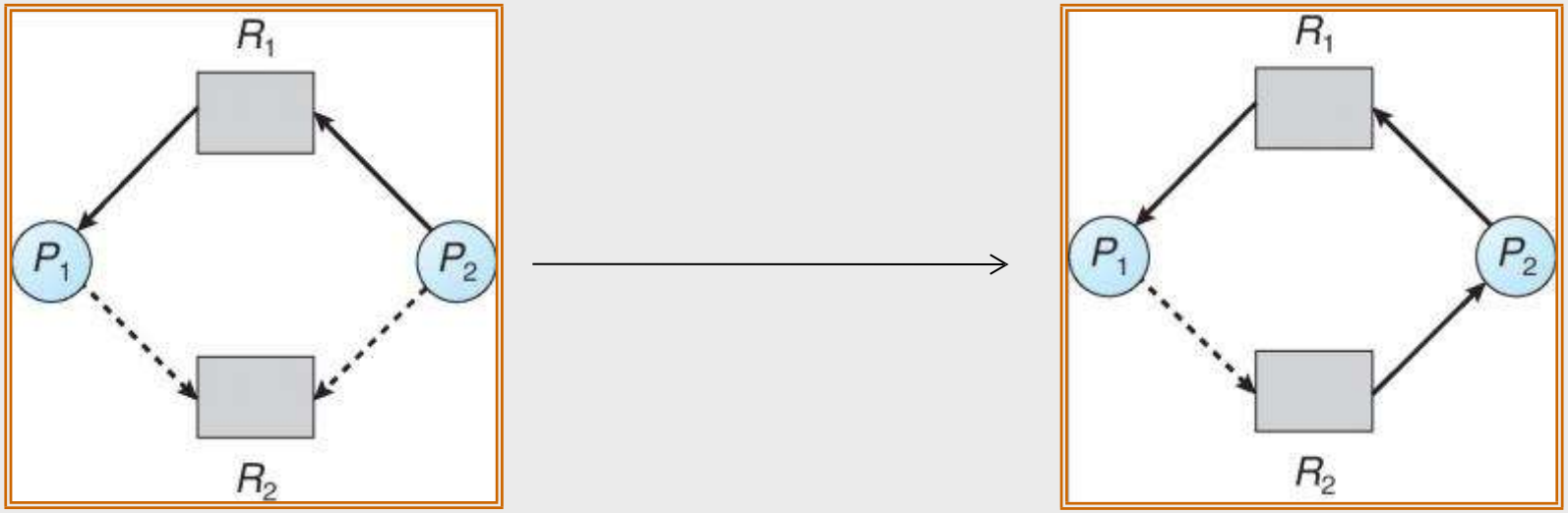
Esclusione dello stallo
Grafo di assegnazione delle risorse

Singola istanza per ciascuna risorsa.

- Arco di reclamo: $P_i \rightarrow R_j$ indica che il processo P_j può richiedere la risorsa R_j; linea tratteggiata.
- L'arco di reclamo diventa un arco di richiesta quando la risorsa viene effettivamente richiesta.
- L'arco di richiesta diventa di assegnazione quando la risorsa viene effettivamente assegnata.
- Le risorse devono essere reclamate a priori nel sistema.

- Si supponga che P_i richieda R_j

  La richiesta può essere garantita solo se la conversione di un arco di richiesta in un arco di assegnazione non comporta la presenza di un ciclo nel grafo di assegnazione delle risorse (stato non sicuro).



Logica di esclusione dello stallo Algoritmo del Banchiere

Istanze multiple per le risorse.

```
struct state
{
        int resource[m];
        int available[m];
        int claim[n][m];
        int alloc[n][m];
}
```

stato del sistema

(a) global data structures

```
if (alloc [i,*] + request [*] > claim [i,*])
     < error >;                          /* total request > claim*/
else if (request [*] > available [*])
     < suspend process >;
else                                     /* simulate alloc */
{
     < define newstate by:
     alloc [i,*] = alloc [i,*] + request [*];
     available [*] = available [*] - request [*] >;
}
if (safe (newstate))
     < carry out allocation >;
else
{
     < restore original state >;
     < suspend process >;
}
```

(b) resource alloc algorithm

Si verifica che la somma tra le risorse attualmente assegnate e richieste non superi la dichiarazione iniziale

Se non ci sono risorse sufficienti, il proceso viene sospeso

Si definisce il nuovo stato ipotizzando di assegnare le risorse al processo richiedente

Si verifica se il nuovo stato è sicuro

Logica di esclusione dello stallo
Algoritmo del Banchiere

Algoritmo per la determinazione di stato sicuro/nonsicuro

```
boolean safe (state S)
{
    int currentavail[m];
    process rest[<number of processes>];
    currentavail = available;
    rest = {all processes};
    possible = true;
    while (possible)
    {
        <find a process Pk in rest such that
            claim [k,*] - alloc [k,*] <= currentavail;>
        if (found)                          /* simulate execution of Pk */
        {
            currentavail = currentavail + alloc [k,*];
            rest = rest - {Pk};
        }
        else
            possible = false;
    }
    return (rest == null);
}
```

(c) test for safety algorithm (banker's algorithm)

Algoritmo del Banchiere: esempio

- 5 processi: $P_0,\ldots,P_4$; 3 tipi di risorse: *A* (10 istanze), *B* (5 istanze), *C* (7 istanze).
- Snapshot a T_0:

	alloc[5,3]	*claim[5,3]*	*available[1,3]*	*request=claim-alloc*
	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

- If request>available suspend process:
  - P0, P2, P4 vengono sospesi
- Per P1 risulta Request $\leq$ Available (that is, (1,2,2) $\leq$ (3,3,2) $\Rightarrow$ true).

	alloc	*request*	*available*
	A B C	*A B C*	*A B C*
P_0	0 1 0	7 4 3	2 1 0
P_1	3 2 2	0 0 0	
P_2	3 0 1	6 0 0	
P_3	2 1 1	0 1 1	
P_4	0 0 2	4 3 1	

- Per determinare se questo è uno stato sicuro si verifica se altri processi, dopo la terminazione di P1 e il rilascio delle risorse potranno essere completati.

Considerazioni sulla logica di esclusione dello stallo

Vantaggi:

- Non’è necessario interrompere processi e riportarli in uno stato precedente (problema presente nelle strategie di rilevamento dello stallo)

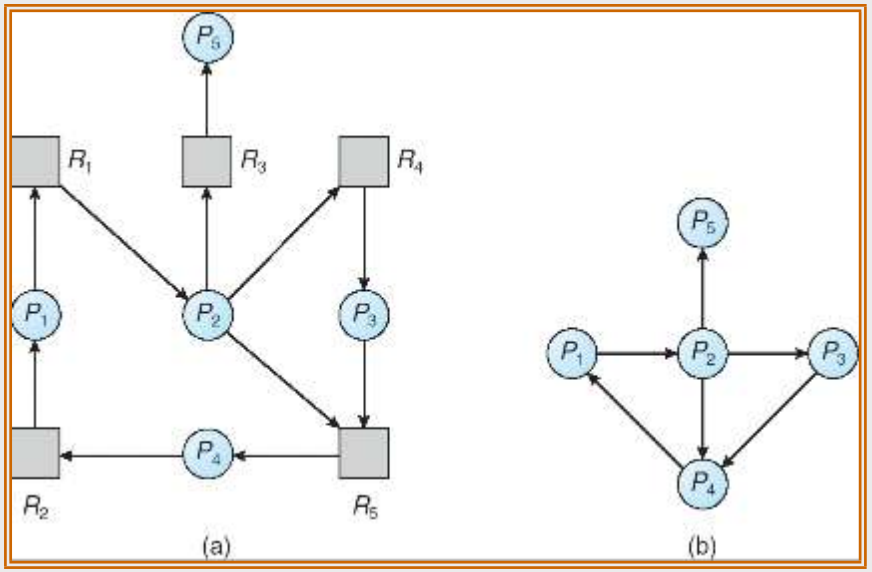
Restrizioni:

- Il numero massimo di risorse necessitate da ogni processo deve essere noto a priori (prima dell’esecuzione)
- I processi devono essere indipendenti: l’ordine di esecuzione non deve essere vincolato a esigenze di sincronizzazione
- Deve esserci un numero fissato di risorse da allocare
- Quando un processo richiede risorse potrebbe essere posto in attesa
- Quando un processo ottiene tutte le risorse di cui necessita, deve rilasciarle in un tempo finito.

Deadlock Detection: Rilevamento dello stallo

1. Quando è possibile le richieste vengono sempre soddisfatte
2. Il SO verifica periodicamente che non si sia verificato uno stallo
3. Il SO applica uno schema di recovery per risolvere lo stallo

Individuare lo stallo con una sola risorsa di ogni tipo



Resource-Allocation Graph *Corresponding wait-for graph*

- Un ciclo nel grafo di attesa denota uno stallo
- L'algoritmo può essere invocato periodicamente
- Numero di operazioni pari a n^2 dove n=numero di processi

Deadlock Detection: Rilevamento dello stallo

Individuare lo stallo con più istanze per ogni risorsa

Consideriamo le matrici:

- *Available:* vettore di lunghezza *m*: ogni elemento indica il numero di istanze disponibili per ciascuna risorsa in un dato istante
- *Allocation:* matrice *nxm*: ogni elemento indica il numero di istanze di ciascuna risorsa assegnate a ciascun processo
- *Request:* matrice *nxm*: ogni elemento indica la richiesta di risorse da parte dei processi.

L'algoritmo di rilevamento indaga su ogni possibile sequenza di assegnazione per i processi che devono ancora essere completati. Vengono marcati i processi che non sono in stallo. Inizialmente nessun processo è marcato.

1. Si marca ogni processo che ha una riga di 0 nella matrice *Allocation* (il processo non ha risorse assegnate);
2. Si inizializza *Work=Available;*
3. Si cerca un indice *i* (processo) tale che:
   1. *i* non'è marcato (può essere in stallo);
   2. *Request[i,:]≤Work;*

   Se tale processo non esiste l'algoritmo termina
4. Se si trova il processo al punto 3. esso viene marcato e *Work=Work+Allocation[i,:]*

Un processo è in stallo solo se alla fine dell'algoritmo esso non risulta marcato.

L'algoritmo richiede un numero di operazioni dell'ordine di $O(m \times n^2)$

Deadlock Detection: Rilevamento dello stallo

Individuare lo stallo con più istanze per ogni risorsa

Filosofia dell'algoritmo:

- Trovare un processo le cui richieste possono essere soddisfatte con le disponibilità attuali
- Supporre che il processo vada a conclusione e rilasci le risorse
- Cercare un altro processo da soddisfare

	R1	R2	R3	R4	R5
P1	0	1	0	0	1
P2	0	0	1	0	1
P3	0	0	0	0	1
P4	1	0	1	0	1

Request matrix **Q**

	R1	R2	R3	R4	R5
P1	1	0	1	1	0
P2	1	1	0	0	0
P3	0	0	0	1	0
P4	0	0	0	0	0

Allocation matrix **A**

R1	R2	R3	R4	R5
2	1	1	2	1

Resource vector

R1	R2	R3	R4	R5
0	0	0	0	1

Available vector

Figure 6.10 Example for Deadlock Detection

P4 viene marcato (non detiene alcuna risorsa)
Work=Available= [0, 0, 0, 0, 1]
Request[P3,:] ≤Work, P3 viene marcato e Work=Work+Allocation[P3,:]=[0, 0, 0, 1, 1]
Non'è possibile determinare alcun altro indice (processo) per il quale soddisfare le richieste
L'algoritmo termina.
P1, P2 non sono marcati: P1, P2 sono in stallo

Deadlock Detection: Rilevamento dello stallo

Utilizzo dell'algoritmo di rilevamento dello stallo

- Quando e quanto spesso invocare l'algoritmo dipende da:
  - Frequenza presunta con la quale si verifica lo stallo
  - Numero di processi coinvolti nello stallo

- In generale uno stallo si verifica quando un processo avanza una richiesta che non può essere soddisfatta immediatamente.

  UTILIZZO DELL'ALGORITMO AD OGNI RICHIESTA. Consente la determinazione dello stallo e del processo la cui richiesta ha cagionato lo stallo.

  Aumento notevole del carico… (overhead)

- <u>INVOCARE L'ALGORITMO QUANDO LA PERCENTUALE DI UTILIZZO DELLE RISORSE SCENDE AL DISOTTO DI UNA SOGLIA</u>

- INVOCARE L'ALGORITMO AD ISTANTI ARBITRARI: nel grafo di assegnazione delle risorse potrebbero esistere molti cicli e diventa difficile determinare quale processo ha "causato" lo stallo…

Deadlock Detection: Ripristino

Terminazione dei processi:

- Abort di tutti i processi coinvolti nello stallo. Soluzione più comunemente adottata.
- Abort di un processo alla volta fino a quando il ciclo non'è eliminato. Dopo ogni abort si deve rieseguire l'algoritmo di determinazione dello stallo. Ordine con cui abortire i processi dato da funzione di minimo costo basata su
  - Priorità dei processi
  - Tempo trascorso in esecuzione e necessario al completamento
  - Risorse già utilizzate e risorse richieste (processo in fase di stampa…)
  - Tipo di processo (interattivo, ecc.)

Prelazione di risorse:

- Selezionare un processo vittima – minimize cost.
- Rollback del processo a cui è stata sottratta la risorsa – return ad uno stato sicuro per poterlo riavviare in seguito.

  Salvataggio periodico dello stato dei processi. In caso di stallo:

  Si ripristina il processo all'ultimo stato salvato

  Si fa ripartire il processo…il non-determinismo dei processi concorrenti dovrebbe garantire il non ri-verificarsi dello stallo

  In generale conviene uccidere il processo.
- Starvation – alcuni processi potrebbero essere selezionati costantemente come vittime, include number of rollback in cost factor.

Strategie a confronto

Table 6.1 Summary of Deadlock Detection, Prevention, and Avoidance Approaches for Operating Systems [ISLO80]

Approach	Resource Allocation Policy	Different Schemes	Major Advantages	Major Disadvantages
Prevention	Conservative; undercommits resources	Requesting all resources at once	•Works well for processes that perform a single burst of activity •No preemption necessary	•Inefficient •Delays process initiation •Future resource requirements must be known by processes
		Preemption	•Convenient when applied to resources whose state can be saved and restored easily	•Preempts more often than necessary
		Resource ordering	•Feasible to enforce via compile-time checks •Needs no run-time computation since problem is solved in system design	•Disallows incremental resource requests
Avoidance	Midway between that of detection and prevention	Manipulate to find at least one safe path	•No preemption necessary	•Future resource requirements must be known by OS •Processes can be blocked for long periods
Detection	Very liberal; requested resources are granted where possible	Invoke periodically to test for deadlock	•Never delays process initiation •Facilitates on-line handling	•Inherent preemption losses