Il deadlock (o stallo) è una condizione in cui ogni processo in un gruppo è bloccato in attesa di un evento che può essere generato solo da un altro processo del gruppo, tipicamente il rilascio di risorse fisiche o logiche.
Affinchè il deadlock si possa verificare devono essere soddifatte contemporaneamente 4 condizioni:

- Mutua esclusione: almeno una delle risorse coinvolte è utilizzata in modo mutualmente esclusivo dai processi
- Possesso e attesa: un processo che possiede delle risorse si mette in attesa per averne altre
- Nessun rilascio anticipato: il sistema operativo non può forzare il rilascio di risorse assegnate ai processi
- Attesa circolare: n processi attendono in modo che P1 attende P2, P2 attende P3, ..., Pn attende P1

Vediamo ora come si può affrontare il problema.
Ignorare il deadlock:
Nella maggior parte dei sistemi, il deadlock è una condizione rara e quindi si può risparmiare tempo e aumentare l'uso della risorse ignorandolo e resettando il sistema se si verifica.
Prevenire il deadlock:
Si agisce invalidando una delle 4 condizioni che causano il deadlock imponendo dei vincoli sulle richieste, che però comportano un potenziale sottoutilizzo delle risorse e sono scomodi per il programmatore.
Mutua esclusione:
Alcune risorse sono intrinsecamente condivisibili (ad esempio file in lettura), e per queste si può invalidare; tuttavia, in generale non è possibile farlo poiché non tutte lo sono.
Possesso e attesa:
Si può invalidare imponendo che i processi richiedano tutte le risorse all'avvio, oppure imponendo che un processo, prima di fare una richiesta, rilasci tutte le risorse che possiede e le richieda tutte in blocco. In entrambi i casi si ha un sottoutilizzo delle risorse e possibile starvation; inoltre, nel secondo caso, si richiede ai programmatori di rilasciare le risorse in stato consistente poiché potrebbero essere assegnate ad altri processi.
Nessun rilascio anticipato:
Si può invalidarla in 2 modi: quando un processo viene messo in attesa per una richiesta, tutte le sue risorse sono rilasciate anticipatamente, e quindi attende per la richiesta più le risorse che già possedeva. In alternativa, si possono rilasciare anticipatamente solo le risorse di processi in attesa che sono richieste da un processo in esecuzione.
Il problema della consistenza descritto prima rimane, tuttavia è il sistema operativo che rilascia le risorse, e non il programmatore.
Questo metodo funziona bene con risorse il cui stato può essere salvato e ripristinato.
Attesa circolare:
Si può invalidare imponendo un ordinamento sulle risorse, ossia una definendo una funzione che assegni ad ogni risorsa un numero intero, e imponendo questo protocollo: quando un processo richiede risorse di ordine i, questi può ottenerle solo se sono disponibili e non possiede già risorse di ordine >=i. Se ne possiede, le deve rilasciare e richiedere in ordine corretto (con tutti i problemi relativi). In questo modo, l'attesa circolare non può verificarsi, ma la funzione va definita in base ai consueti ordini di utilizzo delle risorse, e può non essere facile.
Evitare il deadlock:
I metodi per evitare il deadlock non pongono vincoli su come fare le richieste, ma chiedono al processo informazioni supplementari sull'utilizzo delle risorse (quali, quante, in che ordine, ...), e le utilizzano per verificare se lo stato risultante da una richiesta è sicuro. Il problema è che queste informazioni potrebbero non essere note a priori.
Uno stato si dice sicuro per n processi se esiste almeno una sequenza sicura, ossia una sequenza di n processi per cui le richieste di ogni processo Pi della sequenza possono essere soddisfatte con le risorse attualmente disponibili più quelle detenute dai processi Pj, con j<i (poiché le rilasceranno). Se non esiste una tale sequenza, lo stato si dice non sicuro. Uno stato sicuro garantisce l'assenza di deadlock,

ma uno stato non sicuro non significa necessariamente deadlock.
Verranno ora presentati 2 metodi che richiedono di sapere quali e quante risorse saranno usate da un processo.
In caso di istanze singole delle risorse, si può utilizzare una variante del grafo di allocazione delle risorse. Quest'ultimo è un grafo orientato che ha come vertici i processi e le risorse, e come archi ha archi di assegnazione (da una risorsa a un processo) e archi di richiesta (da un processo a una risorsa). Questa variante introduce gli archi di prenotazione (da un processo a una risorsa). All'avvio, un processo deve specificare quali risorse utilizzerà, così da generare i suoi archi di prenotazione. Quando richiederà una risorsa (disponibile), si controlla se la conversione del relativo arco di prenotazione ad arco di assegnazione causa cicli nel grafo. Se ne causa, lo stato non è sicuro e il processo deve attendere.
In caso di istanze multiple si può utilizzare il più complesso algoritmo del banchiere.
Fa uso di alcune strutture dati per mantenere le informazioni sui processi(n= numero processi, m=numero risorse):

- Available: array di m elementi contenente il numero di istanze disponibili per ogni risorsa
- Allocation: matrice n*m contenente le allocazioni per ogni processo
- Max: matrice n*m contenente per ogni processo le risorse massime che può utilizzare
- Need: matrice n*m contenente per ogni processo le risorse massime che può ancora richiedere

Quando un processo fa una richiesta, si controlla se è valida (<= need di quel processo) e se è soddisfacibile (<= available), quindi modifica le strutture dati per simulare l'assegnazione della risorsa, ed esegue l'algoritmo di verifica dello stato sicuro: si cercano tutti i processi il cui fabbisogno (need) può essere soddisfatto con le attuali disponibilità più le risorse assegnate ai processi analizzati precedentemente (che le rilasceranno al termine). Se alcuni processi rimangono scoperti, lo stato non è sicuro, si ripristinano i vecchi valori delle strutture dati e il processo deve attendere. In caso contrario, il processo ottiene le risorse richieste.
Rilevamento e risoluzione del deadlock:
Un ultimo approccio che consente di aumentare l'utilizzo delle risorse è lasciare che il deadlock si verifichi, e usare degli algoritmi per rilevarlo e risolverlo.
In caso di istanze singole delle risorse, si può usare il grafo di attesa: una variante del grafo di allocazione con i nodi risorsa collassati per mostrare le attese tra i processi. Se nel grafo ci sono cicli, quei processi sono coinvolti in un deadlock.
In caso di istanze multiple si può usare un algoritmo simile all'algoritmo di verifica dell'algoritmo del banchiere. Utilizza le stesse strutture dati (tranne need e max, poiché non sono note), più la matrice request n*m che contiene le attese di ogni processo. Si cercano tutti i processi le cui richieste possono essere soddisfatte con le attuali disponibilità più le risorse detenute dai processi analizzati precedentemente (si suppone che terminino e le rilascino, se non lo fanno, il deadlock verrà rilevato successivamente). Se alcuni processi rimangono scoperti, questi sono coinvolti in un deadlock.
Per risolverlo si possono terminare tutti i processi coinvolti (viene risolto, ma si possono perdere i risultati di lunghe computazioni), oppure terminarli uno per volta in base a qualche criterio finché non si risolve il deadlock (evitando di terminare sempre gli stessi causandone la starvation).
L'algoritmo di rilevamento può essere eseguito ogni n secondi, oppure quando l'utilizzo della cpu scende sotto una certa soglia (se c'è un deadlock, sempre più processi saranno coinvolti quindi continuerà a scendere).