Transazioni

L'esecuzione concorrente è essenziale per garantire buone prestazioni nei database, poichè i dischi sono lenti ed è bene tenere la CPU impegnata.

Una transazione è una astrazione per un DBMS che indica una sequenza di letture e scritture fatte da un'applicazione sul DBMS.

Una transazione possiede le proprietà ACID:

Atomicità: proprietà di tutto o niente, assicura che in presenza di fallimenti lo stato è riportato al precedente stato consistente, prima dell'inizio della transazione.

Consistenza: assicura che le transazioni siano schedulate per eseguire senza interferenze, in modo equivalente ad una esecuzione sequenziale (serializzata).

Isolamento: proprietà di contenimento, assicura che gli aggiornamenti fatti da una transazione diventano visibili solo dopo che la transazione termina.

Durabilità: proprietà di persistenza, assicura che i risultati prodotti da una transazione terminata con successo non siano distrutti da successivi fallimenti.

Serializzabilita'

Date due transazioni P1 e P2, e sia inizialmente x = 5 e y = 2

P1 { 1: x = x + y; 2: z = x; }

P2 { 1: y = x – y; 2: q = y; }

Se P1 e P2 sono eseguite sequenzialmente una dopo l'altra, possiamo ottenere le sequenze:

S1: P1 seguita da P2 produce: x = z = 7, y = q = 5

S2: P2 seguita da P1 produce: x = z = 8, y = q = 3

Se P1 e P2 sono eseguite nello stesso momento si possono avere le tre esecuzioni concorrenti di P1 e P2 (indicate con P1 ll P2)

A: {P1(1) ll P2(1)}; {P1(2) ll P2(2)} produce x = z = 7, y = q = 3

B: {P1(1); P2(1)}; {P1(2) ll P2(2)} produce i risultati di S1

C: {P2(1); P1(1)}; {P1(2) ll P2(2)} produce i risultati di S2

B e C sono esecuzioni serializzabili poichè producono gli stessi risultati di esecuzioni seriali di programmi. Mentre A non è serializzabile.

Per garantire che le transazioni non interferiscano esse devono essere serializzabili. Occorre per esse una tecnica di controllo che permette solo esecuzioni serializzabili per transazioni che eseguono in parallelo.

Questa tecnica di controllo è la tecnica di two phase locking.

Two Phase Locking (2PL) - 1

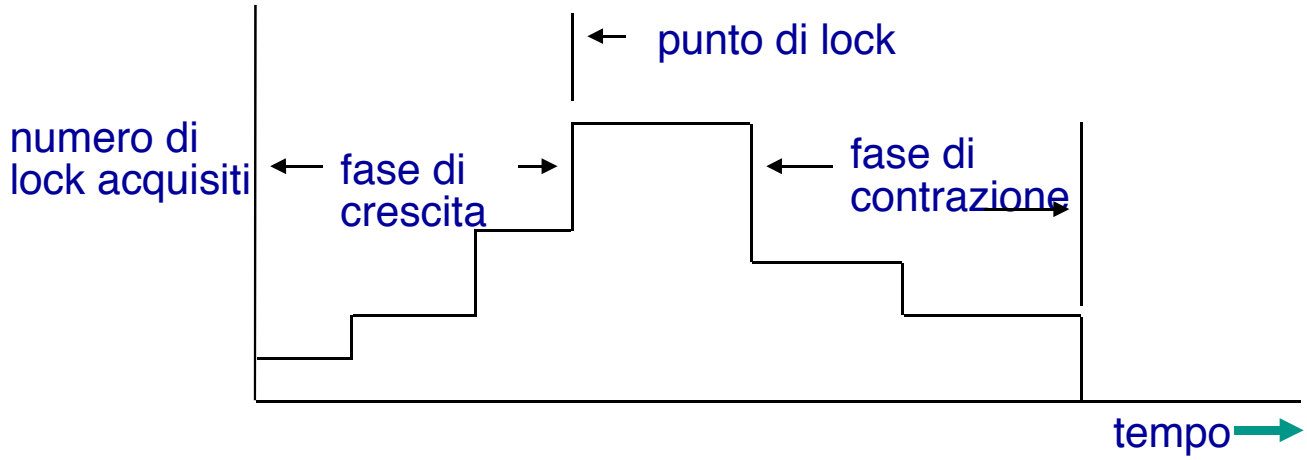
E' una tecnica di controllo della concorrenza che consente solo esecuzioni serializzabili. Consiste di due fasi.

La prima fase è detta fase di crescita:

- un processo cerca di fare il lock (in lettura o scrittura) su tutti i record di cui ha bisogno, uno alla volta
- se un record necessario è trovato inaccessibile, esiste già un lock su esso, si riparte da capo (rilasciando prima i lock acquisiti)
- nessun lavoro è svolto in questa fase. Se l'acquisizione di un lock fallisce, la transazione può essere annullata semplicemente rilasciando i lock già ottenuti (senza effetti collaterali).

Se la prima fase ha successo, viene cominciata la seconda fase, detta fase di contrazione:

- vengono fatti gli aggiornamenti dei record
- vengono rilasciati i lock
- dopo il rilascio di un lock non possono essere acquisiti nuovi lock



Teorema: Se ogni transazione di un insieme S soddisfa la tecnica 2PL, allora S è serializzabile.

Two Phase Locking (2PL) - 2

Per le transazioni P1 ed P2 viste prima, nella fase di crescita, solo una tra P1 e P2 acquisisce i lock per x, y, quindi le transazioni sono eseguite necessariamente in modo sequenziale.

Con la tecnica di 2PL, il deadlock è evitato, poichè se una transazione non può acquisire un lock, essa rilascia tutti i lock già acquisiti, consentendo ad altre transazioni in attesa su uno dei suoi lock di continuare.

- Poichè nessun aggiornamento di record viene fatto durante la fase di crescita, il rilascio dei lock lascia lo stato consistente.

Varianti di 2PL:

- Rilasciando i lock tutti insieme si ha la strict-two-phase-locking (S2PL). Questo assicura la proprietà di Isolamento e previene l'aborto in cascata delle transazioni.
- La conservative-two-phase-locking è la tecnica che richiede ad una transazione di dichiarare il set di risorse che necessita e tenta di acquisirle tutte insieme prima dell'inizio dell'esecuzione.
  - Se il lock su una risorsa non può essere acquisito, nessun lock sarà acquisito e si aspetterà che tutte le risorse siano disponibili.

Proprieta' dei programmi concorrenti

Safety

I processi non devono interferire tra loro nell'accesso alle risorse condivise.

I meccanismi di sincronizzazione servono a garantire la proprietà di safety, se usati in modo appropriato.

Liveness

I meccanismi di sincronizzazione usati non devono ostacolare l'avanzamento del programma. La mutua esclusione risolve i problemi legati all'interferenza nell'uso di una risorsa condivisa, ma bisogna evitare che:

- tutti i processi si blocchino in attesa di eventi generabili da altri processi bloccati;
- un processo attenda indefinitamente per l'accesso ad una risorsa condivisa.

Deadlock (stallo) - 1

Esistono potenziali conflitti tra i processi che usano risorse condivise.

Le risorse possono essere di diverso tipo: dispositivi di I/O, file, memoria, una tupla di un database, etc. Vengono comunemente classificate in due tipi:

- risorse con prerilascio: possono essere tolte ai processi senza problemi (memoria, CPU)
- risorse senza prerilascio: se vengono tolte ai processi si ha il fallimento dell'elaborazione (stampanti, lettori di nastro).

La sequenza di passi necessari per l'uso di una risorsa è:

1. Richiesta della risorsa
2. Utilizzo della risorsa
3. Rilascio della risorsa

Se la risorsa non è disponibile, il processo viene fatto attendere. In alcuni sistemi il processo viene automaticamente bloccato e poi risvegliato quando la risorsa torna disponibile; in altri il processo deve esplicitamente gestire la situazione.

Definizione di deadlock: un insieme di processi è in deadlock se ogni processo dell'insieme è in attesa di un evento che solo un altro processo appartenente allo stesso insieme può causare.

Il deadlock è una condizione da evitare.

Deadlock (stallo) - 2

Un processo in deadlock non procede mai e non finisce mai la sua esecuzione. Processi in deadlock possono bloccare il sistema.

I processi possono bloccarsi nel tentativo di accedere ad una risorsa in modo esclusivo.

Esempio:

> Date due risorse (A e B) e due processi (P1 e P2) che hanno bisogno di accedere alle due risorse contemporaneamente.
>
> Supponiamo che P1 richiede ed ottiene la risorsa A, e P2 richiede ed ottiene la risorsa B.
>
> Successivamente P1 richiede B e viene messo in attesa (poichè B è bloccata da P2). Analogamente P2 viene messo in attesa quando richiede A. I due processi sono bloccati indefinitamente (in attesa circolare): sono in deadlock.

Coffman e altri (1971) hanno dimostrato che le seguenti sono condizioni necessarie affinché esista un deadlock:

1. Mutua esclusione: un solo processo alla volta può utilizzare la risorsa
2. Prendi e aspetta (Hold and Wait): i processi che detengono risorse possono chiederne altre
3. Assenza di prerilascio (No Preemption): le risorse possono essere rilasciate solo dal processo che le detiene, il rilascio non può essere forzato
4. Attesa circolare (Circular Wait): deve esistere una lista circolare di processi e risorse

Rappresentazione del Deadlock

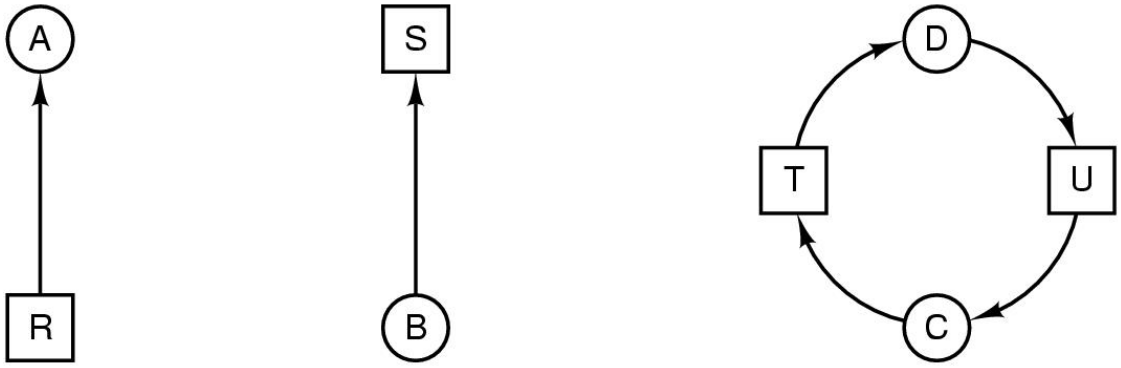
Holt (1972) ha mostrato che le quattro condizioni possono essere rappresentate da grafi orientati

I nodi possono essere di due tipi:

- risorse (quadrati)
- processi (cerchi)

Gli archi possono solo connettere nodi di tipo diverso.
Necessariamente anche gli archi assumono significato diverso:

- archi uscenti da processi: il processo ha richiesto la risorsa
- archi uscenti da risorse: la risorsa è allocata al processo



(a) (b) (c)

- la risorsa R è assegnata al processo A
- il processo B sta richiedendo ed aspetta la risorsa S
- i processi C e D sono in deadlock sulle risorse T e U

Se esiste un ciclo vi è un deadlock. Un deadlock può verificarsi solo se esiste un ciclo.

Trattamento del deadlock - 1

In generale sono utilizzate quattro strategie per trattare i deadlock:

1. Non porsi il problema (algoritmo dello struzzo)

   E' valido se il deadlock avviene molto raramente.

2. Individuare il deadlock e risolverlo

   Ogni volta che una risorsa viene richiesta o rilasciata si controlla il grafo delle risorse per verificare l'esistenza di un ciclo, se esiste si elimina uno dei processi coinvolti. Tuttavia, non sempre è possibile eliminare un processo senza effetti dannosi.

3. Prevenire il deadlock in modo dinamico

   Bisogna assicurarsi a run-time che una delle quattro condizioni non si verifichi. In tal caso il deadlock è impossibile.

1. Prevenire il deadlock impedendo una delle quattro condizioni necessarie

Trattamento del deadlock - 2

Impedire la mutua esclusione: se non vi sono risorse assegnate in modo esclusivo ad un singolo processo non vi saranno deadlock.

Un possibile approccio è lo Spooling

- Occorre un processo deamon (demone) e una directory di spooling
- Per la stampa di un file, il processo genera l'intero file e lo memorizza nella directory di spooling
- Il deamon è l'unico processo ad avere accesso alla directory di spooling e ha il compito di avviare la stampa
- Lo spooling viene usato anche nei trasferimenti via rete (posta elettronica, ftp, ...)

Esempio stampante:

Effettuando lo spooling sulle uscite per la stampante, più processi possono produrre le loro uscite contemporaneamente

- Dato che esiste un solo processo che utilizza la stampante e non ha bisogno di altre risorse il deadlock è impossibile
- Non tutti i processi possono essere gestiti attraverso lo spooling (es. la tabella dei processi)

Rimane comunque la possibilità di deadlock sulla risorsa disco

Trattamento del deadlock - 3

Impedire prendi ed aspetta: occorre evitare che processi che detengono risorse rimangano in attesa di ulteriori risorse. Ci possono essere varie soluzioni:

- Un processo richiede immediatamente tutte le risorse di cui ha bisogno e se non sono disponibili attende.

  Nascono nuovi problemi:

  - Un processo non sempre sa quello di cui ha bisogno nel corso dell'esecuzione
  - L'uso delle risorse non è ottimizzato (rimangono risorse spesso inutilizzate)
  - Vi è il rischio che un processo che ha bisogno di molte risorse rimanga in attesa per tempi molto lunghi

- Un modo leggermente diverso è quello di imporre che un processo rilasci temporaneamente le risorse detenute prima di una nuova richiesta

Impedire l'assenza di prerilascio è di fatto inutilizzabile: non si può togliere di forza una risorsa (es. stampante) mentre un processo la sta usando.

Impedire l'attesa circolare: l'attesa circolare può essere eliminata in vari modi:

- Un processo può richiedere una sola risorsa per volta
- Le risorse sono numerate e possono essere richieste solo secondo l'ordine numerico
- Una variante richiede che la nuova risorsa debba avere una etichetta maggiore di tutte quelle detenute

Trattamento del deadlock - 4

Riepilogo

Situazione	Approccio
mutua esclusione	tecniche di spooling
prendi e aspetta	richiedere tutte le risorse all'inizio
assenza di prerilascio	farsi restituire le risorse
attesa circolare	ordinare le risorse per indice

Starvation (morte per fame – inedia)

Si verifica quando un processo attende indefinitamente l'accesso ad una risorsa. Il sistema complessivo non è bloccato, nel senso che ci sono processi che procedono nella loro esecuzione, ma uno o più processi attendono perennemente una risorsa.

Si può evitare adottando una opportuna politica di assegnamento delle risorse. Es. nel caso in cui la risorsa è la CPU, la priorità del processo è aumentata se il processo aspetta per un tempo lungo