TANKERINO.com

Lezione 196

https://www.tankerino.com/it/corsi/10/tepi-4/lezioni/196/istruzioni-assembly

Istruzioni Assembly nella mutua esclusione

Quando più processi devono accedere a una risorsa condivisa, come un file o una memoria, è fondamentale garantire che solo uno alla volta possa utilizzarla. Questo concetto è noto come mutua esclusione. Se non lo facciamo, i dati possono corrompersi o i processi possono bloccarsi.

Per implementare la mutua esclusione, possiamo utilizzare tecniche hardware e software. Le istruzioni assembly giocano un ruolo fondamentale in questa operazione, poiché lavorano direttamente con l'hardware per garantire l'esecuzione atomica delle operazioni critiche.

La mutua esclusione serve a garantire che una risorsa condivisa sia utilizzata da un solo processo alla volta.

Un esempio classico è quello di un sistema operativo che deve gestire la memoria condivisa tra processi. Se due processi tentano di scrivere nello stesso momento, il risultato sarà imprevedibile.

Le istruzioni assembly forniscono meccanismi come Test-and-Set o Compare-and-Swap, che assicurano che le operazioni su risorse condivise siano eseguite in modo sicuro.

In questa lezione, esamineremo alcune istruzioni assembly utilizzate per la mutua esclusione, analizzando come funzionano e vedendo esempi pratici di applicazione. Non preoccuparti se sembra complesso: faremo tutto un passo alla volta!

Vediamo ora come le istruzioni assembly implementano la mutua esclusione in dettaglio.

L'istruzione Test-and-Set

L'istruzione Test-and-Set è una delle operazioni più comuni per implementare la mutua esclusione. Questa istruzione è atomica, il che significa che viene eseguita completamente senza interruzioni. Serve per garantire che solo un processo alla volta possa accedere a una risorsa condivisa.

Immagina di voler accedere a una stampante condivisa. Prima di poterla usare, devi verificare se è libera. Se lo è, la prenoti per te. Questo è esattamente ciò che fa l'istruzione Test-and-Set.

L'istruzione Test-and-Set verifica il valore di una variabile e lo modifica contemporaneamente, in modo che nessun altro processo possa interferire.

In questa lezione vedremo come funziona, un esempio pratico e perché è importante nei sistemi multitasking. Test-and-Set è comunemente usata per implementare un meccanismo chiamato lock, che garantisce l'accesso esclusivo a una risorsa.

Il funzionamento di Test-and-Set è molto semplice e si basa su tre passaggi principali:

1. Verifica il valore attuale di una variabile, chiamata solitamente lock.
2. Imposta il valore della variabile a 1 (bloccando la risorsa).
3. Restituisce il valore originale della variabile.

Se la variabile era 0, significa che la risorsa era libera. Se era 1, significa che la risorsa era già occupata.

Pseudo-codice di Test-and-Set:

```
function TestAndSet(lock):
  temp = lock # Memorizza il valore attuale

  lock = 1 # Imposta lock a 1 (risorsa bloccata)

  return temp # Restituisce il valore originale
```

Vediamo un caso concreto: se un processo esegue `TestAndSet` su una variabile `lock` inizialmente impostata a `0`, la funzione restituisce `0` e imposta `lock` a `1`.

Un altro processo che esegue `TestAndSet` subito dopo riceverà `1` e non potrà accedere alla risorsa.

Esempio 1 pratico di Test-and-Set

Supponiamo di avere due processi, A e B, che vogliono accedere alla stessa risorsa (ad esempio, una stampante). La variabile `lock` viene usata per indicare se la risorsa è libera o occupata:

- `lock = 0`: La risorsa è libera.
- `lock = 1`: La risorsa è occupata.

Ecco cosa succede:

1. Il processo A esegue `TestAndSet(lock)`:

   - Il valore di `lock` è `0`, quindi TestAndSet restituisce `0`.
   - Il processo A imposta `lock = 1` e entra nella sezione critica.

2. Durante l'esecuzione di A, il processo B tenta di accedere alla risorsa:

   - `TestAndSet(lock)` restituisce `1` (risorsa occupata).
   - Il processo B non può entrare nella sezione critica e deve aspettare.

3. Quando il processo A termina, imposta `lock = 0` (risorsa libera).

4. Ora il processo B può accedere alla risorsa.

Questo garantisce che solo un processo alla volta possa usare la risorsa, evitando conflitti.

Passiamo ora ai vantaggi e ai limiti di questa istruzione.

Vantaggi di Test-and-Set

Test-and-Set è un'istruzione molto utile perché è semplice ed efficace. Ecco alcuni vantaggi principali:

1. Atomicità: L'operazione è garantita come indivisibile. Nessun altro processo può interferire durante l'esecuzione.

2. Semplicità: È facile da implementare sia a livello hardware che software.

Test-and-Set è ideale per garantire la mutua esclusione in sistemi a singola CPU.

3. Portabilità: Molti processori supportano istruzioni simili, rendendo Test-and-Set una scelta universale per molte architetture.

Nei sistemi operativi embedded, Test-and-Set viene usato per gestire risorse come la memoria condivisa o i dispositivi hardware.

Compare-and-Swap

Un'altra istruzione assembly usata per la mutua esclusione è Compare-and-Swap. Questa istruzione è leggermente più avanzata di Test-and-Set, perché controlla se il valore di una variabile è uguale a un valore atteso e, in caso affermativo, lo modifica.

Tuttavia, richiede un po' più di overhead rispetto a Test-and-Set e deve essere implementata con attenzione per evitare problemi di performance.

Limitazioni delle istruzioni assembly

Nonostante siano strumenti potenti, le istruzioni assembly per la mutua esclusione hanno alcune limitazioni. La prima è che sono strettamente legate all'hardware, il che le rende non portabili tra diverse architetture di processore.

Inoltre, l'uso esclusivo di istruzioni come Test-and-Set o Compare-and-Swap può portare al problema del busy waiting, dove un processo rimane in un ciclo infinito, consumando risorse, mentre aspetta che la risorsa diventi disponibile.

Il busy waiting si verifica quando un processo consuma inutilmente risorse di CPU mentre aspetta l'accesso a una risorsa condivisa.

Un altro problema è la complessità nei sistemi multiprocessore, dove la sincronizzazione tra CPU diventa più difficile. In questi casi, si utilizzano meccanismi più avanzati come semafori o mutex.

In un sistema multiprocessore, un'istruzione assembly potrebbe non essere sufficiente per garantire che tutte le CPU rispettino lo stato della variabile condivisa. In questi casi, si utilizza una combinazione di hardware e software per sincronizzare i processi.

Vedremo ora come le istruzioni assembly si integrano con soluzioni software per superare queste limitazioni.

Integrazione con soluzioni software

Le istruzioni assembly, come Test-and-Set e Compare-and-Swap, sono spesso combinate con soluzioni software per risolvere problemi di sincronizzazione più complessi. Ad esempio, i semafori utilizzano queste istruzioni per gestire l'accesso alle risorse condivise.

Un semaforo è una struttura dati che utilizza istruzioni atomiche per controllare l'accesso a una risorsa condivisa.

I semafori evitano il busy waiting, permettendo ai processi di "dormire" mentre aspettano l'accesso a una risorsa. Questo migliora l'efficienza complessiva del sistema.

Un semaforo può utilizzare l'istruzione Test-and-Set per verificare se una risorsa è libera. Se non lo è, il processo viene messo in coda e attivato solo quando la risorsa diventa disponibile.

Un'altra soluzione software è l'uso dei mutex (mutual exclusion locks), che garantiscono che solo un processo alla volta possa accedere a una sezione critica. I mutex sono implementati utilizzando Compare-and-Swap o istruzioni simili.

Queste integrazioni rendono le istruzioni assembly più potenti e applicabili anche nei sistemi complessi.

Questa lezione e' stata realizzata grazie al contributo di:

Baobab.School
Risorse per la scuola
https://www.baobab.school

FrancescoBelloni.it software development
Siti web a Varese
https://www.francescobelloni.it