Variabili atomiche

Thursday, 16 March 2023 08:51

Supportano operazioni atomiche = No operazioni che si possono dividere
Esistono 2 tipologie:

- Una sola istruzione di CPU
- Virtuale, il thread crede di avere accesso an una variabile atomica

Utilizzare:

- Oggetti sono semplici
  Se abbiamo oggetti complicati, o in un cluster di oggetti semplici, NAH
  Ameno che non creiamo una classe per gli oggetti semplici -> AtomicReference

Problematiche:

- Race condiction, cioè più thread cercano di modificare o leggere una variabile
- Visibilità, noi qui dobbiamo far si che le modifiche apportate sono visibili a tutti i thread

Soluzione:

- Usando syncronized si risolvono tutti due
  Però paghiamo in velocità
- Volatile, con questo diciamo che una variabile non deve essere un cache però la modifica deve essere visibile a tutti, però con questo non è sincronizzato

E questo volatile è importante siccome, è qui che vengono create le variabile atomiche:

- Sono variabili volatile migliori
- Non richiedono il lock, se ci sono conflitti

E da questo viene creato l'algoritmo alternativo del locking: optimistic retrying

- No sincronizzazione lettura
- Se scrittura:
  - Copio variabile in locale
  - Aggiorno copia
  - Scrivo nella variabile originale, se c'è una collisione ritorno al copiare

Ed una operazione atomica che viene usata è il Compare and Swap (CAS):

- Input:
  - Posizione memoria V
  - Valore atteso E
  - Nuovo valore N
- Noi aggiorniamo se e soltanto se il valore in memoria è lo stesso di E
- Restituisce un valore se c'è stato un cambiamento

Abbiamo la sorella compare and set

- Restituisce true se l'operazione si è conclusa con successo

```
public class IntSimulatedCAS {

   private int value;

   public int compareAndSwap (int exp, int newValue) {
        int oldValue = value;
        synchronized(this) { if (oldValue == exp) value= new; }
        return oldValue ;
   }

   public boolean compareAndSet(int exp,int newValue) {
        return (exp == compareAndSwap(exp, newValue));
   }
    ...
}
```

Problema:

- Abbiamo 3 thread
- Il primo thread entra ed incrementa
- Il secondo entra, fallisce, e dopo incrementa
- Il terzo entra, fallisce 2 volte, e dopo incrementa

Metodi AtomicInteger:

- Get
- Set
- GetAndSet
- compareAndSet
- getAndIncrement

```
public class AtomicPseudoRandom implements IPseudoRandom{
    private AtomicInteger seed;

    AtomicPseudoRandom(int seed) {
        this.seed = new AtomicInteger(seed);
    }
    @Override
    public int nextInt(int n) {
        while (true) {
            int s = seed.get();
            int nextSeed = calculateNext(s);
            if (seed.compareAndSet(s, nextSeed)) {
               int resto = s % n;
               return resto> 0 ? resto: resto+ n;
            }
        }
    }
    private int calculateNext(int s) {return s + 1;}
}
```

Utilizza CAS (lock-free)