

Algorithmen und Programmierung IV

SS 2006

26. Juni 2006

Musterlösung zum Aufgabenblatt 5

Aufgabe 1

Eine einfache Sperre benötigt als Zustand lediglich einen Wahrheitswert `locked`, der angibt, ob die Sperre offen oder geschlossen ist. Bei der Anforderung der Sperre über `tryLock` wird überprüft, ob sie noch offen ist, und gegebenenfalls geschlossen. Diese Methode muss über einem kritischen Abschnitt (`synchronized`) gesichert sein, um zu verhindern, dass mehrere Fäden die Überprüfung passieren können, bevor die Sperre geschlossen wird. Die Freigabe der Sperre über `unlock` muss nicht in einem kritischen Abschnitt stehen, da sie unteilbar ist und die Variable `unlocked` als `volatile` gekennzeichnet ist, so dass Änderungen an ihr sofort sichtbar werden.

```
public class PoorLock implements (some methods) Lock {  
  
    private volatile boolean locked;  
  
    public synchronized boolean tryLock() {  
        if (locked) return false;  
        locked = true;  
        return true;  
    }  
  
    public unlock() {  
        locked = false;  
    }  
}
```

Falls man sicherstellen möchte, dass nur derjenige Faden eine Sperre wieder öffnen kann, der sie geschlossen hat, muss die Sperre statt des Wahrheitswertes den entsprechenden Faden speichern. Auch hier muss die Methode `unlock` nicht in einen kritischen Abschnitt eingeschlossen werden, da nur der Faden, der die Sperre geschlossen hat, Schreiboperationen in der Methode durchführen kann.

```
public class PoorLock implements (some methods) Lock {  
  
    private volatile Thread owner;  
  
    public synchronized boolean tryLock() {  
        if (owner != null) return false;  
        owner = Thread.currentThread();  
        return true;  
    }  
  
    public unlock() {  
        if (owner != Thread.currentThread())  
            throw new IllegalMonitorStateException();  
        owner = null;  
    }  
  
}
```

Man kann mit beiden Implementierungen den wechselseitigen Ausschluss kritischer Abschnitte realisieren. Sofern jeder Faden nur dann in einen kritischen Abschnitt eintritt, wenn `tryLock` den Wert `true` geliefert hat, ist ein wechselseitiger Ausschluss gewährleistet. Da `tryLock` jedoch eine nicht blockierende Operation ist, bietet die Sperre (außer durch aktives Warten) keine Möglichkeit einen Faden solange anzuhalten, bis der kritische Abschnitt betreten werden kann.

Aufgabe 2

Man spricht von konservativem Zwei-Phasen-Sperren, wenn alle benötigten Sperren am Transaktionsbeginn unteilbar angefordert werden. Konservatives Sperren kann nicht zu Verklemmungen führen, jedoch benachteiligt es Fäden, die Sperren zu mehr als einem Objekt anfordern.

```
// Faden 1           // Faden 2           // Faden 3
// Operationen auf Konto a   // Operationen auf Konto b   // Überweisung von a nach b
-----
WLOCK(a);
a.bal += amount;
-----
                                           WLOCK(a,b); // blockiert
-----
                                           WLOCK(b);
-----
UNLOCK(a);
-----
                                           b.bal -= amount;
-----
WLOCK(a);
-----
                                           UNLOCK(b);
-----
...                               ...
-----
UNLOCK(a);
-----
                                           // kann erst jetzt fortfahren
                                           a.bal -= amount;
                                           b.bal += amount;
                                           UNLOCK(a,b);
```

Wenn zu jedem Zeitpunkt mindestens eine Sperre geschlossen ist, die Faden 3 angefordert hat, hat dieser keine Chance fortzufahren. Das Verfahren ist deswegen unfair, weil Fäden, die einzelne Sperren zu einem späteren Zeitpunkt anfordern, sich „vordrängeln“ und Faden 3 an der weiteren Ausführung behindern.

Aufgabe 3

(keine Musterlösung)