Praktikum 6: Low Level Synchronisation

M.Thaler, 2/2016, ZHAW

1 Einführung

In diesem Praktikum werden wir zwei wichtige Synchronisationsmechanismen kennen lernen, die sowohl von modernen Prozessoren, aber auch von Hochsprachen wie z.B. Java in der einen oder anderen Form unterstützt werden:

TAS Test and Set
CAS Compare and Swap

TAS und CAS sind atomare Instruktionen, die als Grundlage für die Implementierung von Synchronisationsmechanismen benötigt werden. Mit CAS können zudem *Lock-* und *Wait-Free* Zugriffe auf gemeinsame Daten realisiert werden, dabei entspricht das Verhalten von CAS einer Transaktion.

In einem ersten Teil werden wir das Zeitverhalten von TAS- und CAS- basierten Mechanismen anhand des Even/Odd Zählers untersuchen. In einem zweiten Teil, werden wir eine Problemstellung mit mutex-basierter Synchronisationen analysieren.

2 TAS und CAS

Das *Funktionalität* von TAS und CAS lässt sich wie folgt beschrieben (die Ausführung dieser Funktionen wird als **atomar** (nicht unterbrechbar) angenommen):

```
int testAndSet(*var) {
    if (*var == 0) {
        *var = 1;
        return 1;
   } else
        return 0;
}
```

```
int compAndSwap(*var, old, new) {
    if (*var == oldval) {
        *var = newval;
        return 1;
    } else
        return 0;
}
```

Im Fall von TAS muss `var` mit 0 initialisiert werden und natürlich auch wieder auf 0 zurückgesetzt werden (`var=0`), das Schreiben von `var` ist atomar und muss nicht speziell behandelt werden.

Mit TAS kann z.B. ein Spinlock implementiert werden, mit CAS ein atomarer Zähler (siehe unten).

2.1 TAS, CAS und Intel (x86)

Bei Intel-Prozessoren stehen für die Implementierung von TAS und CAS die Instruktionen `xchg` bzw. `cmpxchg` zur Verfügung. Vereinfachter Assemblercode für TAS und CAS:

```
tas:    MOV     AX, 1
        XCHG    AX, var
        XOR     AX, 1
        RET
```

```
cas:    MOV           AX, old
        MOV           DI, new
        LOCK CMPXCHG  var, DI
        SETZ          AL
        RET
```

- Die Instruktion `XCHG` benötigt kein Lock-Präfix, der Bus wird automatisch gesperrt.
- Die Instruktion `CMPXCHG` benötigt jedoch ein Lock-Präfix auf Multicores.
- Die Instruktion `SETZ` kopiert das Zero Flag ins Register `AL`.

2.2 Beispiel zu TAS und CAS

Unten stehende Beispiele realisieren einen synchronisierten gemeinsamen Zähler (`count`) mit TAS und CAS. Der aktuelle Zählerstand wird für die weitere Verwendung in die lokale Variable `lc` kopiert. Die Variablen `lock` und `count` sind global bzw. gemeinsam und beide Verfahren verwenden Spinning, d.h. die Locks werden solange abgefragt, bis sie offen sind (Spin Locks).

```
int lock = 0, count = 0;
...
int lc;
while(!testAndSet(&lock) {}
    lc = count++;
lock = 0;
```

```
int count = 0;
...
int old, new, lc;
do {
    oldval = count;
    newval = oldval+1;
until (compAndSwap(&count, old, new);
lc = newval;
```

3 Zeitverhalten mit Mutex-, TAS- und CAS-Synchronisation

Wir haben für sie drei verschiedene, thread-basierte Implementationen eines Even/Odd Zählers vorbereitet (Verzeichnisse in `lowLevelSync/a1/*`):

- Synchronisation mit einem Pthread-Mutex
- Synchronisation mit einem Spinlock (TAS, File `spinlockIntel.h`)
- Synchronisation mit einem atomaren Zähler (CAS-Counter, File `atomicIntel.h`)

Die Programme stehen zudem in zwei Varianten zur Verfügung:

1. mit einem *Work Counter* pro Thread zur Simulation von Thread-Workload
2. mit Worker-Threads in einem eigenen Prozess zur Simulation eines belasteten Rechners

Im folgenden sollen die Auswirkungen verschiedener Lastfälle auf die Laufzeiten der Programme für unterschiedliche Synchronisationsmechanismen untersucht und diskutiert werden.

Wichtiger Hinweis: die folgenden Programme werden automatisch aufgrund der Anzahl CPUs konfiguriert und gestartet, die gesamte Workload der Threads wird konstant gehalten.

3.1 Aufgaben

Arbeiten sie in diesem Praktikum auf dem Compute Server der Vorlesung um vernünftige Rechenzeiten und einigermassen stabile Resultate zu erhalten.

a) Im Verzeichnis `lowLevelSync/a1/workCounter` finden sie die Even/Odd Zähler mit einem variablen Work Counter. Übersetzen sie das Programm mit `make` und starten sie es wie folgt:

```
main.e | tee filename.txt
```

Die Ausgabe des Programms wird sowohl auf dem Bildschirm angezeigt als auch im File `filname.txt` gespeichert.

Analysieren und diskutieren sie, wie sich die Ausführungszeit mit zunehmendem `work count` verändert.

b) Im Verzeichnis `lowLevelSync/a1/workerThreads` finden sie die Even/Odd Zähler mit einer variablen Anzahl zusätzlicher Worker Threads. Starten sie das Programm wieder wie oben mit `tee`.

   Analysieren und diskutieren sie, wie sich die Ausführungszeiten mit zunehmender Anzahl Worker Threads verändern.

c) Vergleichen und diskutieren sie nun die Auswirkungen des Work Counters und der zusätzlichen Worker Threads auf die Ausführungszeiten. Was sind die wesentlichen Unterschiede zwischen Mutex, Atomic Counter und Spinlock? Was sind Vorteile, was sind Nachteile? Unter welchen Bedingungen soll bzw. kann welcher Synchronisationsmechnismus verwendet werden.

4 Scheduling und Locking

In `lowLevelSync/a2` finden sie ein Programm mit dem untersucht wird, was geschieht, wenn das Betriebssystem einen Thread auslagert, der ein Lock hält. Dazu wird während der Lockphase die System-Funktion `nanosleep()` zufällig aufgerufen, wobei die Wahrscheinlichkeit des Aufrufs wählbar ist. Auch hier wird die gesamte Workload konstant gehalten. Der Overhead durch den Aufruf der Funktion `nanosleep()` wird zu Beginn mit Hilfe der Funktion `test()` geschätzt und die Simulationsresultate entsprechend korrigiert

4.1 Aufgabe

Starten sie das Programm wiederum mit **`main.e | tee filename.txt`** Das Programm gibt für verschiedene Wahrscheinlichkeiten p und eine unterschiedliche Anzahl von Threads die mittlere Rechenzeit pro Thread aus. Die Anzahl Threads wird aufgrund der Anzahl verfügbaren CPUs n bestimmt: gewählt werden n, $n+1$ und $n+2$ Threads.

n CPUs	p in %	n Threads	$n+1$ Threads	$n+2$ Threads
	0.0			
	0.001			
	0.01			
	0.1			
	1.0			

Analysieren und diskutieren sie das Resultat. Welche Rückschlüsse können gezogen werden, was ist dabei zu aber zu beachten bzw. zu relativieren.

5 Take Home

Was ist bei der Verwendung von Synchronisationsmechnismen zu beachten?

Notieren sie sich die wichtigsten Punkte, die sie im Zusammenhang mit diesem Praktikum kennengelernt haben.