

Betriebssysteme Übungsblatt 1

Von den vier zur Auswahl stehenden Systemoperationen habe ich aktives Warten („Spinlocks“) gewählt. Grund dafür ist, dass es unter MacOS zwei Implementierungen gibt, C11-Atomics und os_unfair_lock, deren Vergleich von Interesse ist.

Aktives Warten oder „busy waiting“ ist eine Methode, einen Thread warten zu lassen, bis eine erwartete Kondition eintritt. Hierbei wird der Thread nicht blockiert sondern durchläuft aktiv eine Warteschleife, bis die erwartete Bedingung erfüllt wird.

C11-Atomics nutzen dabei atomare CPU-Instruktionen um das eintreten der Release-Kondition innerhalb des User-Space in jedem Schritt zu überprüfen. Dazu bot Apple mit OSSpinLock eine spezifische Implementierung für Mac- und iOS-System, die allerdings anfällig für Priority Inversion Probleme war.

Beim Auftreten eines solchen Problems wartet ein hoch priorisierte Thread auf eine Sperre die von einem niedrig priorisierten Prozess freigegeben werden muss. Wird dieser vom Scheduler aufgrund der niedrigen Priorität niemals aufgerufen, wartet der hochpriorisierte Thread unendlich. Aus diesem Grund ist OSSpinLock seit MacOS Sierra (veröffentlicht in 2016) deprecated und wurde durch die neue Implementation os_unfair_lock ersetzt, die diese Anfälligkeit nicht aufweist.

Nach einer kurzen Phase des aktiven Wartens setzt der Scheduler hier den wartenden Thread auf eine Warteliste im Kernel und kann so die CPU-Ressourcen frei geben. Tritt die Releasekonditon ein, erhält der Scheduler diese Info vom Kernel und gibt den Thread wieder frei.

Um im Folgenden die Performance von Spinlocks zu messen und zwischen den Methoden vergleichbar zu machen, soll die Latenz beider Ansätze gemessen werden. Sowohl C11-Atomics als auch Apples Unfair Lock sollen im Nanosekunden-Bereich quantifiziert werden. Dazu wurden auf jeder Variante mehrere Durchläufe mit unterschiedlicher Menge an Iterationen ausgeführt und die durchschnittliche Lock-Unlock-Dauer ermittelt.

Implementierung

Die Implementierung greift auf die Funktionen mach_absolute_time() und mach_timebase_info() des Mach-Kernels zurück um die Zeit in Nanosekunden zu messen.

Der Aufruf des Spinlocks selbst erfolgt über die Spinlock API mit den Befehlen my_lock() / my_unlock() mit atomic_flag für den Atomics-basierten Ansatz und os_unfair_lock für eben diesen.

Zur Berechnung der Statistiken wird der Welford-Algorithmus verwendet um Mittelwert und Varianz auf den Onlinedaten zu berechnen.

Das Programm führt eine Messung mit n Samples durch und gibt die gemessenen Werte für Minimum, Maximum, Median, Standardabweichung und 95% Konfidenzintervall zurück. Eine Kommandozeilen Option erlaubt es neben der Festlegung von n zudem die erhobenen Daten in eine CSV-Datei zu schreiben. Ein Beispielhafter Aufruf sieht wie folgt aus:

```
./aw_unfair -csv result.csv 1000000
```

Es findet hier eine Messung mit Unfair Lock über eine Millionen Samples statt. Das Ergebnis wird der Datei result.csv angehangen.

Versuchsaufbau

Die Datenerhebung wurde auf einem MacBook Air mit M1-Prozessor durchgeführt, Betriebssystem MacOS Tahoe Version 26.1. Gemessen wurde mit 1.000, 10.000, 100.000, 1.000.000, 10.000.000 und 100.000.000 Samples, jeweils 100 mal für sowohl C11-Atomic als auch Unfair Lock.

Bei Betrachten der Daten fällt auf, dass das Minimum durch alle Messreihen durchweg 0,0ns beträgt. Die Maximalwerte sind zwischen den Messreihen ebenfalls vielmals ähnlich, es existieren jedoch Ausreißer, die um ein Tausendfaches höher sind als der Durchschnitt. Diese Ausreißer beeinflussen in ihren Testreihen Standardabweichung 95%-Ci erheblich. Abbildung 2 verdeutlicht die Seltenheit solcher Ausreißer. Die Meisten Messenreihen gruppieren sich um den Median.

Auffällig ist zudem, das der durchschnittliche Maximalwert mit der Menge an Samples pro Durchlauf steigt. Die Mediane der Messreihen bleiben allerdings unabhängig von der Menge der Samples stabil.

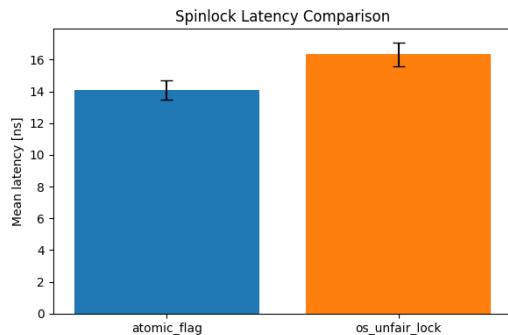


Abb.1 Latenz beider Ansätze im Vergleich

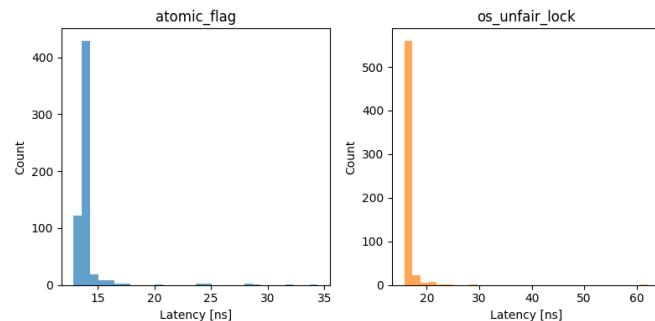


Abb.2 Per-Lauf Mittelwerte

Wie aus der graphischen Darstellung der Messwerte leicht ersichtlich wird, ist der Atomic-Ansatz im Mittel 2-3 Nanosekunden schneller als der MacOS spezifische Unfair Lock-Ansatz, dessen stärken außerhalb der reinen Latenzperformance liegen.

Fazit

Für reinen Low-Latency-Einsatz reicht der Atomic-Ansatz aus und liefert sogar bessere Werte. Allgemein empfiehlt sich aber trotzdem die Unfair-Lock-Implementation zu verwenden, da sie Schutzes gegen Priority-Inversion bietet und durch Einbeziehung des Kernels eine effizientere Verwaltung der CPU-Zeit erlaubt. In beiden Fällen liegen die durchschnittlichen Lock-Unlock-Zyklen im niedrigen zweistelligen Nanosekunden-Bereich.