Fakultät für Informatik und Mathematik
HOCHSCHULE FÜR ANGEWANDTE WISSENSCHAFTEN MÜNCHEN

Betriebssysteme, SoSe 2019

Praktische Übung 7: Compare-and-Swap

Ziele des Labortermins

- Nicht-blockierende Datenstrukturen

Aufgabe 7.1: Verständnisfragen

a) Was ist der wichtigste Vorteil von Nicht-Blockierenden Datenstrukturen gegenüber Locking?

b) Beschreiben Sie die Funktionsweise von Compare-and-Swap?

c) Lässt sich Compare-and-Swap alleine mit C-Code implementieren?

d) Was ist das ABA-Problem?

e) Wie lässt sich das ABA-Problem lösen?

f) Was ist der Vorteil von Load-Linked/Store-Conditional gegenüber Compare-and-Swap?

g) Gibt es Load-Linked/Store-Conditional in x86?

Aufgabe 7.2: Counter-Implementierung Compare-and-Swap (praktisch)

Sie können in GCC (dem Standard-Compiler unter Linux) das folgende Intrinsic verwenden um Compare-and-Swap zu benutzen:

```
type __sync_val_compare_and_swap (type *ptr, type oldval type newval)
```

Implementieren Sie einen Counter, der von 16 Threads geändert wird mithilfe von Compare-and-Swap. Verwenden Sie dabei kein Locking.

Aufgabe 7.3: Lock-Implementierung mit Compare-and-Swap (praktisch)

Implementieren Sie die Funktionen `lock(lock_t *lock)` und `unlock(lock_t *lock)` für eine von Ihnen definierte Datenstruktur und verwenden Sie in der Implementierung ein Compare-and-Swap.