Universität Augsburg
University

Vertiefte Multicore-Programmierung

Universität Augsburg
Institut für Informatik
Prof. Dr. Theo Ungerer
Dr. Sebastian Weis

04. Mai 2016

Aufgabenblatt 3

1. Aufgabe (4 + 4 + 2 Punkte)

a) Implementieren Sie eine lock-freie und thread-sichere Datenstruktur `queue`, die folgende Methoden unterstützt:

- `void push_back(T i);` Fügt ein neues Element an das Ende der Schlange an.
- `void push_front(T i);` Fügt ein neues Element an den Anfang der Schlange an.
- `bool pop_back(T &i);` `true`, falls die Schlange nicht leer ist. `i` enthält danach das letzte Element der Schlange.
- `bool pop_front(T &i);` `true`, falls die Schlange nicht leer ist. `i` enthält danach das erste Element der Schlange.
- `auto at(int i);` Liefert das `i`-te Element in der Schlange zurück. Existiert kein Element an der Stelle `i` soll ein `nullptr` zurückgegeben werden.

Die Schlange soll als einfach verkettete Liste implementiert werden. Verzichten sie vollständig auf Locks und verwenden Sie stattdessen `std::atomic` um die Korrektheit der Schlange zu gewährleisten. Zunächst soll nur ein einzelner Thread Daten in der Queue ablegen (`push`) und ein anderer Thread entfernen (`pop`) dürfen.

b) Erweitern Sie die Schlange aus Teilaufgabe a), um den parallelen Zugriff mehrerer Threads mit den o. g. Methoden zu ermöglichen. Achten Sie auf das ABA-Problem und verwenden Sie *reference counting* mit der `shared_ptr`-Klasse.

c) Optimieren Sie die atomaren Zugriffe in der Schlange aus Teilaufgaben a) und b) durch Verwendung *schwächerer Speicherordnung* (*Acquire-Release* oder *Relaxed Ordering*).

Vergleichen Sie die Ausführungszeit aller Teilaufgaben mit unterschiedlicher Anzahl Threads und variierender Eingabegrößen (Parameter `-t` und `-n`).

Hinweis

Die Abnahme der Aufgaben soll bis 10. Mai erfolgen.