SS 2023 20.04.2023

Übungen zur Vorlesung
Modern Concurrency Theory
Blatt 1

Prof. Dr. Roland Meyer
Anton Opaterny Abgabe bis 26.04.2023 um 13:15 Uhr

Aufgabe 1.1 (Gruppenbildung)

Bilden Sie Gruppen von zwei bis drei Personen zur Bearbeitung der Übungsaufgaben. Zum Abgeben schicken Sie ihre Source-files an anton.opaterny@tu-braunschweig.de. Andere Lösungen können Sie entweder als PDF-Datei an die selbe E-Mail-Adresse schicken, oder in Papierform zu Beginn der Übung abgeben. Stellen Sie sicher, dass in jeder Datei, bzw auf jedem Zettel, Ihr *Name*, *MatrNr* und *Studiengang* angegeben sind.

Aufgabe 1.2 (Michael&Scott's Queue)

Implementieren Sie eine nebenläufige Queue, also eine einfach verkettete Liste, welche Elemente nach dem first-in-first-out Prinzip verwaltet. Die zu implementierende Queue soll das in der Vorlesung vorgestellt Konzept der fine-grained Concurrency umsetzten. Gehen Sie wie folgt vor:

a) Verwenden Sie die Implementierung aus der Vorlesung als Vorlage (zu finden in den Vorlesungsnotizen). Benutzen Sie insbesondere `struct Node`, `atomic` Variablen und `atomic_compare_exchange_weak`.

b) Definieren Sie zwei globale Pointer `Head` und `Tail`.

c) Um unnötige Fallunterscheidungen zu vermeiden, soll Ihre Queue immer einen `Node` enthalten. Schreiben Sie eine Funktion `init`, die einen Dummy-`Node` erzeugt und `Head` und `Tail` damit initialisiert. Sie können davon ausgehen, dass `init` zu Beginn der Ausführung ein einziges Mal ohne Nebenläufigkeit ausgeführt wird.

d) Implementieren Sie eine Methode `enqueue`, die einen neuen `Node` ans Ender der Queue anhängt. Der Datenwert des neuen `Node` ist Parameter von `enqueue` (vgl. `finePush` bei Treiber's Stack). Das Ende der Queue identifizieren Sie mittels `Tail`.

Hinweis: Ihr Implementierung wird nicht garantieren können, dass `Tail` immer auf den letzten `Node` der Queue zeigt. Sie müssen diesen Umstand also erst prüfen und ggf. `Tail` an das Ende der Queue schieben, bevor Sie mit dem eigentlichen Hinzufügen eines neuen `Node` beginnen können.

e) Implementieren Sie eine Methode `dequeue`, die den ersten Datenwert der Queue entfernt. Schieben Sie dazu den `Head` "um eins nach hinten" und löschen Sie den nun unbenutzten Knoten, der vormals von `Head` referenziert wurde.

Hinweis: Der von `Head` referenzierte `Node` ist ein Dummy, sein Datenwert ist nicht Teil der Queue. Fangen Sie den Sonderfall einer leeren Queue ab. Zur Vereinfachung können Sie ihre Queue so implementieren, dass `Tail` nie von `Head` "überholt" wird.

f) Testen Sie ihre Implementierung

Aufgabe 1.3 (Optional: Performance Vergleich)

Implementieren Sie naive Varianten von `enqueue` und `dequeue` und evaluieren Sie den Performance-Unterschied.

Aufgabe 1.4 (Optional: das ABA-Problem)

Ist ihre Implementierung von `enqueue` und `dequeue` anfällig für das ABA-Problem? Falls dem so ist: welche Pointer müssen Sie mit Version-Countern ausstatten, um das Problem zu beseitigen? Genügt es Version-Counter für `Head` und `Tail` einzuführen?

Abgabe bis 26.04.2023 um 13:15 Uhr in der Übung oder per Mail an anton.opaterny@tu-braunschweig.de.