

Übungen zur Vorlesung
Modern Concurrency Theory
Blatt 1

Prof. Dr. Roland Meyer
Sören van der Wall

Abgabe bis 09.11.2021 um 23:59 Uhr

Aufgabe 1.1 (Gruppenbildung)

Bilden Sie Gruppen von zwei Personen zur Bearbeitung der Übungsaufgaben. Interessierte erreichen Sie über folgenden, für die Vorlesung erstellten Chat: <https://go.rocket.chat/invite?host=messenger.tu-braunschweig.de&path=invite%2Fq52aR4>. Zum Abgeben laden Sie ihre Source-files und PDF-Dateien unter <https://cloudstorage.tu-braunschweig.de/preparefilelink?folderID=2JMw94NhSVEhHTnRQ6Ydb> hoch. Stellen Sie sicher, dass in jeder Datei Ihr *Name*, *MatNr* 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 vorgestellte Konzept der fine-grained Concurrency umsetzen. Gehen Sie wie folgt vor:

- Verwenden Sie die Implementierung aus der Vorlesung als Vorlage. Benutzen Sie insbesondere `struct Node`, `atomic` Variablen und `atomic_compare_exchange_weak`.
- Definieren Sie zwei globale Pointer `Head` und `Tail`.
- 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.
- Implementieren Sie eine Methode `enqueue`, die einen neuen `Node` ans Ende 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`.

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

- 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.

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 09.11.2021 um 23:59 Uhr unter <https://cloudstorage.tu-braunschweig.de/preparefilelink?folderID=2JMw94NhSVEhHTnRQ6Ydb>.