Ludwig-Maximilians-Universität München
Institut für Informatik
Lehrstuhl für Mobile und Verteilte Systeme
Prof. Dr. Claudia Linnhoff-Popien

mobile and distributed systems group

Übungsblatt 3
Betriebssysteme im WiSe 2020/2021

Zum Modul D

Abgabetermin: 22.11.2020, 18:59 Uhr

Besprechung: Besprechung der Übungsaufgaben in den Übungsgruppen vom 30. November – 04. Dezember 2020

Aufgabe Ü7: Threads in Java

(4 Pkt.)

Laden Sie das Java-Programm `SimpleThread.java` von der Betriebssysteme-Homepage herunter.

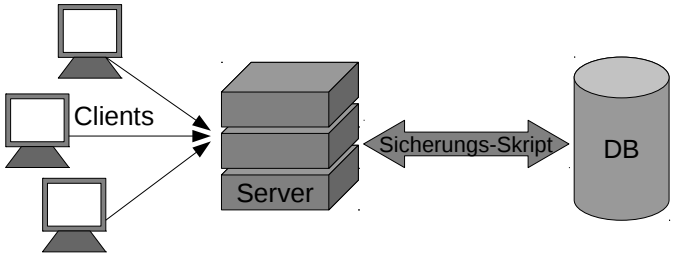
a. Betrachten Sie das Java-Programm! Als Eingabeparameter verlangt es eine Integer-Zahl. Wie hängt diese Zahl mit der Ausgabe zusammen? Welche Art der Ausgabe erwarten Sie für verschiedene Integer-Eingaben (zum Beispiel 1, 2, 100 oder 10000)?

b. Basierend auf dem Java-Programm aus Teilaufgabe a:
Geben Sie in Abhängigkeit von `c` eine allgemeine Formel für die Anzahl der möglichen Konsolenausgaben an.

Aufgabe Ü8: Threads in Java

(11 Pkt.)

In dieser Aufgabe soll eine einfache Client/Server Kommunikation in Java simuliert werden, bei welcher mehrere Clients Anfragen an eine Server-Schnittstelle stellen können, um dort Daten abzulegen. Die Daten werden in regelmäßigen Abständen von einem Sicherungs-Skript auf eine Datenbank geschrieben. Das beschriebene Szenario ist in folgender Abbildung schematisch dargestellt:



Aus Performanzgründen erlaubt der Server nur, dass eine maximale Anzahl von `maxClients` gleichzeitig Daten auf dem Server ablegen darf. Eine Client-Anfrage muss also ggf. mit dem Beginn der Datenablegung warten, damit diese Bedingung nicht verletzt wird.
Das Sicherungs-Skript wird regelmäßig aktiviert und speichert die abgelegten Daten zu einem dedizierten Zeitpunkt auf die Datenbank. Dazu meldet das Skript einen Sicherungswunsch am Server an, so dass kein weiterer Client mehr Daten ablegen darf. Anfragen können jedoch weiterhin gestellt werden und Clients, die zum Zeitpunkt des Sicherungswunsches bereits Daten ablegen, können natürlich ihre Aufgabe noch erledigen.
Das aktive Sicherungs-Skript muss solange warten, bis kein Client mehr Daten ablegt. Nach Beenden der Sicherung wird das Skript wieder deaktiviert und damit der Sicherwunsch aufgehoben, so dass die anfragenden Clients wieder Daten ablegen können.
Im Folgenden soll eine Klasse `Server` implementiert werden. Laden Sie sich bitte dazu zunächst von der Betriebssysteme-Homepage die Dateien `Simulation.java, Client.java, Sicherung.java` sowie den Code-Rahmen der Server-Klasse `ServerAbstract.java` herunter. Die Beispielimplementierungen der Klassen `Client`, `Sicherung` und `Simulation` sollen Ihnen verdeutlichen, wie die Klasse `Server` verwendet werden kann.
Bearbeiten Sie nun die folgenden Aufgaben:

a. Implementieren Sie den Konstruktor der Klasse `Server`. Verwenden Sie dabei den gegebenen Code-Rahmen! Die Klassenattribute sind dort bereits deklariert und müssen durch den Konstruktur initialisiert werden.

b. Implementieren Sie die Server-Methode `daten_ablegen(Client c)`, welche die Client Anfrage zum Daten ablegen modelliert, sowie die die Methode `daten_ablegen_beenden()`, welche vom Client aufgerufen wird, nachdem die Daten abgelegt wurden. Beachten Sie dazu die folgenden Randbedingungen:

   - Alle oben genannten Anforderungen müssen beachtet werden.
   - Maximal `maxClients` dürfen gleichzeitig Daten ablegen. Die Variable wird in der Klasse `Simulation` festgelegt.
   - Es darf kein neuer Client mehr Daten ablegen, sobald das Sicherungs-Skript den Sicherungswunsch geäußert hat. Neue Anfragen können aber weiterhin gestellt werden und noch laufende Daten-Ablegungen werden noch vollständig beendet.

   Ergänzen Sie dazu den Code-Rahmen am Ende der Aufgabe.
   Hinweis: Sie können davon ausgehen, dass die Methoden `daten_ablegen(Client c)` bzw. `daten_ablegen_beenden()` immer in einer sinnvollen Reihenfolge aufgerufen werden (siehe Beispielimplementierung der Klasse `Client`).

c. Implementieren Sie nun die Methoden für das Sicherungs-Skript. Vervollständigen Sie dazu den Code-Rahmen für die Methoden `sicherungAktivieren()` und `sicherungDeaktivieren()` in dem Code-Rahmen am Ende der Aufgabe.
   Hinweis: Sie können davon ausgehen, dass die Methoden `sicherungAktivieren()` und `sicherungDeaktivieren()` immer in einer sinnvollen Reihenfolge aufgerufen werden (siehe Beispielimplementierung der Klasse `Sicherung`).

d. Zeigen Sie zwei kritische Bereiche in ihrem Programm auf. Wie wird hier sichergestellt, dass die Bedingung der wechselseitige Ausschluss erfüllt ist?

Aufgabe Ü9: Einfachauswahlaufgabe: Threads in Java

(5 Pkt.)

Für jede der folgenden Fragen ist eine korrekte Antwort auszuwählen („1 aus n“). Nennen Sie dazu in Ihrer Abgabe explizit die jeweils ausgewählte Antwortnummer ((i), (ii), (iii) oder (iv)). Eine korrekte Antwort ergibt jeweils einen Punkt. Mehrfache Antworten oder eine falsche Antwort werden mit 0 Punkten bewertet.

a) Was ist kein Bestandteil eines Monitors (Softwaremodul)?			
(i) eine oder mehrere Prozeduren	(ii) lokale Daten	(iii) eine Warteschlange für ankommende Prozesse	(iv) eine Liste der Prozesse, die den Monitor verlassen haben
b) Wie viele Prozesse dürfen sich zu jeder Zeit maximal in einem Monitor befinden?			
(i) 0	(ii) 1	(iii) 2	(iv) 3
c) Wie muss die Methode einer von `java.lang.Thread` abgeleiteten Klasse heißen, die den Code beinhaltet, der parallel ausgeführt werden soll?			
(i) *start()*	(ii) *notify()*	(iii) *run()*	(iv) *wait()*
d) Mit welchem Schlüsselwort müssen Eintrittspunkte von Monitoren in Java gekennzeichnet werden?			
(i) `pooled`	(ii) `synchronized`	(iii) `harmonize`	(iv) `locked`
e) Was ist keine Java Direktive, um das Betreten bzw. Verlassen von in Java synchronisierten “Monitor”-Methoden/Blöcken zu organisieren?			
(i) *wait()*	(ii) *notify()*	(iii) *notifyAll()*	(iv) *run()*