SoSe 2017

Nichtsequentielle und verteilte Programmierung

6. Übungsblatt

Prof. Dr. Margarita Esponda

Ziel: Auseinandersetzung mit Deadlocks.

1. Aufgabe (2 P.)

Begründen Sie, dass die Guards der **await**-Anweisungen in Folie 8 der 13. Vorlesung (Erzeuger-Verbraucher-Problem) korrekt sind, indem Sie diese mit Hilfe der *wp*-Funktion selber berechnen.

2. Aufgabe (8 P.)

Die Lösung für das Leser/Schreiber-Problem, das in der Vorlesung besprochen worden ist, hat das Problem, dass die Schreiber-Threads verhungern können. Das kann passieren, wenn pausenlos Leser-Threads ankommen und die Schreiber keine Möglichkeit mehr bekommen, die Daten zu aktualisieren.

Verbessern Sie die zwei Java-Lösungen (mit Semaphoren und Monitoren) aus der Vorlesung, sodass, wenn ein Schreiber kommt, alle Leser-Threads, die bereits da waren, noch zu Ende lesen können aber neue Leser, die nach dem Schreiber gekommen sind, warten, bis der Schreiber dran gewesen ist.

3. Aufgabe (4 P.)

Betrachten Sie folgendes System mit 5 Threads, vier verschieden Ressourcen und folgendem Ressourcen-Vektor, Belegungsmatrix und Anforderungsmatrix (was noch maximal angefordert werden kann):

$$E = [3, 15, 12, 11]$$

Belegungsmatrix

$$C = \begin{array}{c} \\ T_1 \\ T_2 \\ T_3 \\ T_4 \\ T_5 \end{array} \begin{bmatrix} 0 & 0 & 1 & 1 \\ 1 & 1 & 0 & 0 \\ 1 & 3 & 5 & 4 \\ 0 & 6 & 3 & 2 \\ 0 & 0 & 1 & 4 \end{bmatrix}$$

Max. Anforderungsmatrix

$$A = \begin{bmatrix} 0 & 0 & 0 & 1 \\ 0 & 6 & 5 & 0 \\ 1 & 0 & 0 & 1 \\ 0 & 0 & 2 & 0 \\ 0 & 6 & 4 & 2 \end{bmatrix}$$

a) Berechnen Sie zuerst die noch vorhandenen Ressourcen.

b) Befindet sich das System in einem sicheren Zustand?

c) Soll eine Teilanforderung von Thread T_2 von (0, 3, 2, 0) in dem aktuellen Zustand bedient werden? Begründen Sie Ihre Antwort.

4. Aufgabe (16 P.)

a) (0 P.) Importieren Sie das Framework „Resource Allocation“ in eine IDE Ihrer Wahl und lesen Sie die dazugehörige Dokumentation. Die Frameworks (**ResourceAllocation_v1.zip**) sowie dazugehörige Dokumentation (**FW2_ResourceAllocation_v1.pdf**) finden Sie auf der Veranstaltungsseite unter “*Ressourcen —> Material”*.

b) (2 P.) Betrachten Sie den in der Klasse „ExampleAlgo“ implementierten Algorithmus.
- Beschreiben Sie das Vorgehen des Algorithmus in 1-2 Sätzen/Stichpunkten.
- Unter welcher Voraussetzung gewährleistet der Algorithmus ein erfolgreiches Abarbeiten aller Prozesse?

c) (6 P.) Betrachten Sie den in der Klasse „ExampleAlgo2“ implementierten Algorithmus.
- Beschreiben Sie das Vorgehen des Algorithmus in 3-4 Sätzen/Stichpunkten.
- Unter welcher Voraussetzung gewährleistet der Algorithmus ein erfolgreiches Abarbeiten aller Prozesse?
- Wie könnte man den Algorithmus verbessern, so dass weiterhin zwei Prozesse abgearbeitet werden können, es zwischen diesen Prozessen aber nicht zu einem Deadlock kommt? Nennen Sie einen naiven Ansatz.
- Implementieren Sie eine Methode *checkIfSafeState*, die nach jeder Iteration berechnet, ob sich das System gegenwärtig in einem sicheren Zustand befindet. Geben Sie das Ergebnis nach jeder Iteration mit Hilfe der Rückgabe von *nextStep* aus.

d) (8 P.) Implementieren Sie den in der Vorlesung behandelten Bankieralgorithmus. Vervollständigen Sie dazu die Klasse BankersAlgo. Unter welcher Voraussetzung gewährleistet der Algorithmus ein erfolgreiches Abarbeiten aller Prozesse?

Hinweise zu dieser Aufgabe:

1. Lesen Sie sich die zum Framework gehörige Dokumentation sorgfältig durch. Vielen Fragen und Fehlern kann so zuvorgekommen werden.
2. Für die digitale Abgabe des Codes müssen lediglich die von Ihnen bearbeitete Algorithmen - Implementierung als .java-Datei eingereicht werden.
3. Achten Sie auf angemessene Kommentierung Ihres Quelltexts. Sollte es mangels Erklärungen / Kommentaren zu Verständnisproblemen kommen, können Punkte abgezogen werden.

5. Aufgabe (7 Bonuspunkte)

a) (3 P.) Schreiben Sie ein Java-Programm, das bei Eingabe eines „Resource-Allocation“-Graph Zyklen erkennt.

b) (4 P.) Ergänzen Sie Ihre Lösung indem nach der Erkennung von Zyklen entschieden wird, ob das System ein Deadlock hat oder nur Deadlock gefährdet ist.

Allgemeine Wichtige Hinweise zu den Übungsabgaben:

1. Es muss der Quellcode hochgeladen werden (.java-Dateien), nicht der kompilierte Bytecode (.class-Dateien).

2. Zusätzlich zur Online-Abgabe muss der selbstprogrammierten Quellcode ausgedruckt werden. Der ausgedruckte Quellcode muss lesbar sein, insbesondere ist auf automatische Zeilenumbrüche zu achten.

3. Für die gute Verständlichkeit des Quellcodes sind sinnvolle Bezeichnernamen zu wählen und der Code ausreichend zu kommentieren.

4. Das Programm muss Testläufe enthalten, die die Aufgabe vollständig abdecken. Das heißt, zum Testen sollen keine Änderungen am Code durch die Tutorin/den Tutor mehr notwendig sein. Die Testläufe müssen in einer eigenen Klasse Main enthalten sein.

5. Das Programm muss mit Java 8 kompatibel sein.

6. Alle Übungsaufgaben, die nicht Programmieraufgaben sind, müssen ebenfalls digital und nicht handschriftlich sowohl online als auch ausgedruckt zu dem im KVV angegebenen Datum (in der Regel Dienstag um 11:55) abgegeben werden. Zu spät abgegebene Lösungen werden mit 0 Punkten bewertet.