PFP-Team
pfp@i2.cs.fau.de

Übungsblatt 5

Abgabe bis 05.06.2019, 18:00 Uhr

Aufgabe 5.1: Verklemmungen und Verklemmungsbedingungen

a) Erklären Sie informell den Begriff der Verklemmung.

b) Überlegen Sie sich (neben den Vorlesungsbeispielen) noch weitere Verklemmungssituationen im Alltag oder in Code.

c) Welche drei Bedingungen müssen gelten, damit es zu einer Verklemmung in einem parallelen System kommen kann?

d) Nach welcher zusätzlichen Bedingung tritt eine Verklemmung auf?

e) Welche Möglichkeiten gibt es, diese Situationen zu vermeiden?

f) Welche Möglichkeiten sind davon in Java umsetzbar? Wie?

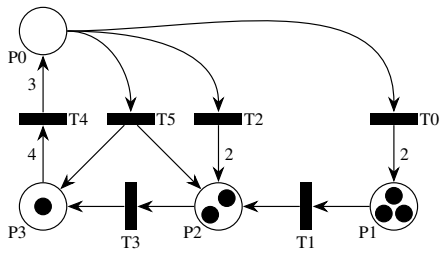
g) Was unterscheidet einen Deadlock von einem Lifelock?

Aufgabe 5.2: Verklemmungen in Petri-Netzen

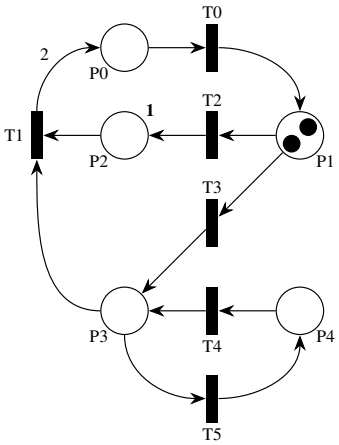
a) Entwerfen Sie ein Petri-Netz mit einem Deadlock.

b) Bei welchen der folgenden Petri-Netze kann es zu einem Deadlock kommen? Bei welchen zu einem Lifelock?

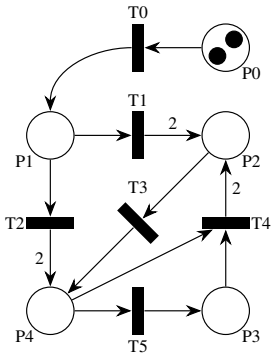
(I)



(II)

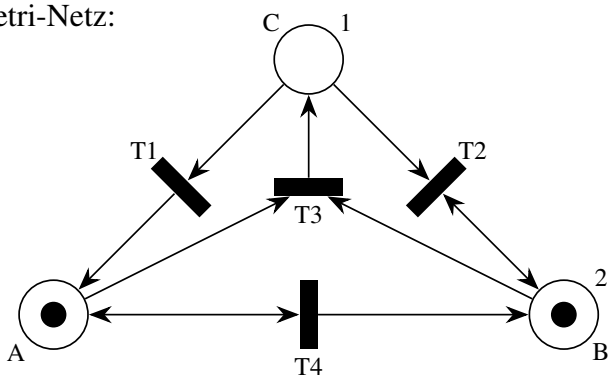


(III)



Aufgabe 5.3: Rechnen mit Petri-Netzen

Gegeben sei folgendes Petri-Netz:



Erstellen Sie zu dem Petri-Netz die zugehörige Matrix. Unter Berücksichtigung der dort eingetragenen Startbelegung I, wenden Sie den Schaltvektor $T_1 = (1, 1, 2, 2)^{\mathrm{T}}$ auf das Petri-Netz an. Geben Sie den berechneten Ergebnisvektor E an.

Bonusaufgabe 5.4: Philosophen **6 Punkte**

In dieser Aufgabe sollen Sie vier verschiedene Lösungen des Philosophen-Problems implementieren. Vervollständigen Sie dafür die vier statischen Methoden in der Klasse `Dinner`. Alle Philosophen-Klassen sollen eine Unterklasse der vorgegebenen abstrakten `Philosopher`-Klasse sein. Zum Verdeutlichen der einzelnen Philosophen-Aktionen soll bei der `eat`-Methode wie bei der vorgegebenen `think`-Methode der Schritt auf `System.out` ausgeben werden. Verwenden Sie als Gabeln die vorgegebene `Fork`-Klasse, die von `ReentrantLock` abgeleitet ist. Die Gabel soll durch einen Aufruf der `lock`-Methode aufgenommen werden.

a) In der Methode `startDinnerDeadlock` soll die naive Version des Philosophen-Problems implementiert werden. Diese Version soll früher oder später zum Deadlock führen. Die Philosophen sollen in der Klasse `PhilosopherDeadlock` realisiert werden.

b) In der Methode `startDinnerQuickPhilosophers` soll eine weitere Version des Philosophen-Problems implementiert werden. Die Philosophen sollen in der Klasse `PhilosopherQuick` realisiert werden. Wenn ein Philosoph essen will, so soll er beide Gabeln aufnehmen können, ohne dass ein weiterer Philosoph eine Gabel aufnehmen bzw. zu essen beginnen kann. Philosophen sollen dennoch gleichzeitig essen können.

c) In der Methode `startDinnerOrderedForks` soll eine weitere Version des Philosophen-Problems implementiert werden. Die Philosophen sollen in der Klasse `PhilosopherOrdered` realisiert werden. Alle Gabeln sind durchnummeriert und ein Philosoph nimmt immer zuerst die Gabel mit der höheren Nummerierung auf.

d) In der Methode `startDinnerWaiter` soll die vierte Version des Philosophen-Problems implementiert werden. Die Philosophen sollen in der Klasse `PhilosopherWaiter` realisiert werden. Wenn ein Philosoph essen will, so muss er zuvor eine Bedienung (Klasse `Waiter`) um Erlaubnis fragen. Die Bedienung kennt den Zustand des ganzen Tisches und kann deshalb entscheiden, ob es zu einem Deadlock kommt, wenn der Philosoph die Gabel nimmt, oder nicht. Um Deadlocks zu vermeiden, kann dem Philosophen die Gabel verweigert werden. Die genaue Implementierung ist freigestellt, jedoch soll die Bedienung nicht die Gabeln anstelle der Philosophen nehmen. Jeder Philosoph soll außerdem nach einer endlichen Anzahl von Anfragen essen können. Zudem sollen möglichst viele Philosophen gleichzeitig essen dürfen.

In einer einfachen, möglichen Lösung gibt die Bedienung jeweils nur einem Philosophen die Erlaubnis eine Gabel zu nehmen. Das geschieht so lange, bis dieser Philosoph 2 Gabeln hat. Erst danach bekommt der nächste Philosoph die Erlaubnis eine Gabel zu nehmen. Eine Gabel abzulegen ist immer erlaubt.

Hinweis: Achten Sie darauf, dass jeder Philosoph von einem eigenen Thread realisiert wird!

6 Punkte