FAU, Informatik 2, Marc Wörlein
woerlein@informatik.uni-erlangen.de

5. Übung

Abgabe bis 26.05.2009, 12:00 Uhr

Aufgabe 5.1: Verklemmungsbedingungen

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

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

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

Aufgabe 5.2: Verklemmungen

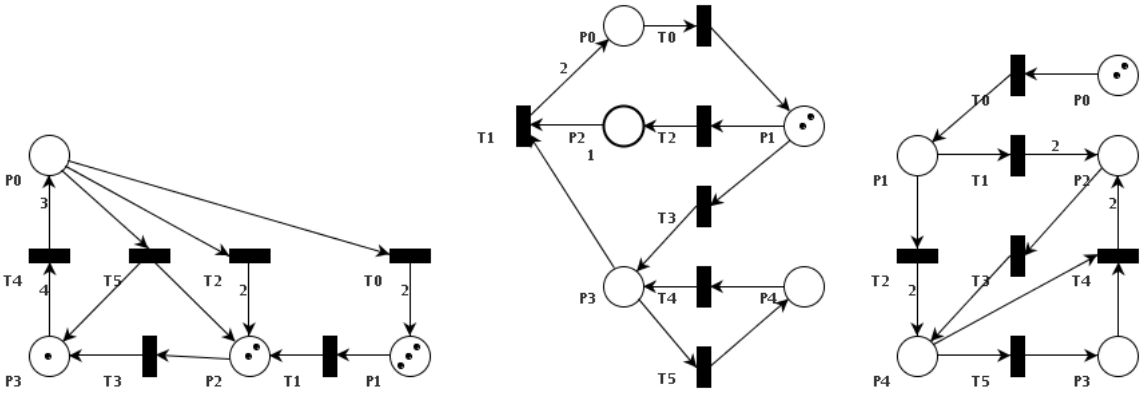
a) Überlegen Sie sich (neben den Vorlesungsbeispielen) noch weitere Verklemmungssituationen im Alltag oder in Code.

b) Was unterschieded einen dead-lock von einem life-lock.

Aufgabe 5.3: Verklemmungen in Petri-Netzen

a) Entwerfen Sie ein Petri-Netz mit einem dead-lock.

b) Bei welchen der folgenden Petri-Netze kann es zu einem dead-lock kommen? Bei welchen zu einem life-lock?



Aufgabe 5.4: Philosophen

Implementieren Sie in der Datei `PhilosophImpl.java` mit Hilfe des zur Verfügung gestellten Interfaces `Philosoph` eine Version des in der Vorlesung vorgestellten Philosophenproblems. Hierfür müssen jedem Philosophen genau zwei `Gabeln` (im Konstruktor) zugeordnet werden, die er sich dann in der `eat`-Methode greifen (`lock()`) und wieder weglegen (`unlock()`) kann. Die `eat`- und `think`-Methoden sollen zum Verdeutlichen ihre einzelnen Schritte sprechend auf `System.out` ausgeben. Die `run`-Methode eines Philosophen stellt seinen Tagesablauf dar: ein ständiges Abwechseln zwischen essen und denken.

Implementieren Sie eine `main`-Methode, die über die Kommandozeile die Anzahl der zu simulierenden Philosophen (und damit auch die Anzahl der `Gabeln` und der zu verwendenden `Threads`) erhält, die entsprechenden Objekte erzeugt, verknüpft und startet.
Tritt in Ihrer Implementierung eine Verklemmung ein? Wenn ja, wann? Wenn nein, wieso nicht?
Hinweis: Erzeugen Sie gerne, sowohl fehlerhafte als auch richtige Implementierungen, z.B. als `main`, `main2`, ...

Aufgabe 5.5: Handy

Implementieren Sie in der Datei `HandyImpl.java` das auf der Übungsseite zur Verfügung gestellte Interface `Handy.java`. Die beiden Methoden des Interface stellen zwei Funktionalitäten eines Handys dar, die nicht gleichzeitig ablaufen dürfen. Mit der Methode `sendeSMS` kann man ein Handy auffordern, einen Text an ein angegebenes Handy zu versenden. Anschließend wartet es auf die Rückkehr der `sendeEmpfangsbestaetigung`-Methode des anderen Handys, um dessen Rückgabewert als Status des Versendens zurückzuliefern.
Implementieren Sie eine `main`-Methode, die für jeden in der Komandozeile übergebenen Namen ein `Handy` erzeugt. Jedes `Handy` sendet in der `run`-Methode immer wieder zufällig an ein anderes Handy eine SMS mit dem Text „SMS von <NAME>“.
Können bei dieser Implementierung Verklemmungen auftreten?

Aufgabe 5.6: Spedition

In dieser Aufgabe soll das Anliefern und Abholen von verschiedenen Produkten in einem `Lager` simuliert werden. Dies geschieht, indem `LKW`s bestimmte Produkte (`Lager.Produkt`) in bestimmter Anzahl transportieren. Implementieren Sie hierfür in der Klasse `LKWImpl` das Interface `LKW` und in der Klasse `LagerImpl` das Interface `Lager`.
Die `LKWImpl`-Klasse erhält über den Konstruktor ein `Lager` (s.u.), ein Produkt das abgeholt/geliefert werden soll und die entpsrechende Anzahl des Produkts. Bei einer positiven Anzahl liefert dieser `LKW` etwas zum `Lager`, bei negativer holt er etwas ab. Jeder `LKW` ist eine eigenständige Aktivität, die regelmäßig (jede Sekunde) versucht, zu seinem `Lager` zu fahren (`fahreHin`). Wenn dies erfolgreich war, wird alle 5 Sekunden überprüft, ob die Lagerarbeiter den Ladevorgang schon abgeschlossen haben, wonach der `LKW` vom `Lager` wieder wegfahren kann (`fahreWeg`). Dies soll nach einer Pause von 10s wiederholt werden, bis das gesamte Programm beendet wird.
Ein `Lager` hat nur eine bestimmte Anzahl an Laderampen, an denen ein `LKW` halten kann, um be- bzw. entladen zu werden. Die Klasse `LagerImpl` soll einen Konstruktor besitzen, dem die Anzahl der Laderampen und die Anfangsmengen der vorrätigen Produkte (z.B. als `java.util.Map` mitgegeben werden. In der Methode `fahreHin` soll ein freier Platz für einen `LKW` gesucht werden. Wird keiner gefunden liefert die Methode `false` zurück. Ansonsten wird die Laderampe vom dem `LKW` besetzt und mittels einer eigenen Aktivitität (einem „Lagerarbeiter“) mit dem Be- bzw. Entladen begonnen. Diese überprüft regelmäßig (jede Sekunde), ob ausreichend Waren im `Lager` vorhanden sind um den LKW zu belanden. Die Kapazität eines Lagers ist quasi unbegrenzt, so dass ein Ausladen immer möglich ist. Das be-/entladen dauert, sobald es möglich ist) weitere 5 Sekunden (Wartezeit), um den Ladevorgang abzuschließen, die Waren in das `Lager` zu bringen oder dem `Lager` zu entnehmen und abschließend den `LKW` mittels dessen Methode `setzeFertig` zu benachrichtigen. Durch die Methode `fahreWeg` wird dem `Lager` mitgeteilt, dass die Rampe des entsprechenden `LKW`s wieder frei ist und für andere `LKW`s zur Verfügung steht.
Kann bei dieser Simulation eine Verklemmung auftreten? Wenn ja, warum? Wie kann dieses Problem umgangen werden?