Grundlagen der Programmierung 2 SS 2005 - Aufgabenblatt 7

Ausgabe: 27.05.05

Aufgabe 24 (Fragen zu Monitoren)

1. Was bedeutet "gegenseitiger Ausschluss"?
2. Welche Einschränkung gilt für die Aufrufe von Methoden eines Monitors?
3. Was bedeutet "Bedingungssynchronisation"?
4. Was bewirken die Aufrufe `wait()` und `notifyAll()`?
5. Was ist eine Verklemmung?
6. Warum sollte man bei dem in der Vorlesung vorgestellten Monitorschema immer `notifyAll()` und nicht `notify()` benutzen?

Aufgabe 25 (Verklemmungen)

In dieser Aufgabe beschäftigen wir uns mit dem Alltag einer Familie mit drei Kindern, die im Winter draußen spielen wollen. Damit sie sich nicht erkälten, braucht jedes Kind eine Mütze, ein Paar Handschuhe und einen Schal, um draußen spielen zu können. Leider ist ihre Oma mit dem Stricken noch nicht soweit, so dass jedes Kind im Moment erst eines dieser drei Kleidungsstücke besitzt und zwar jedes Kind ein anderes. Außerdem haben die Kinder im Kleiderschrank noch eine alte Mütze, ein altes Paar Handschuhe und einen großen Schal gefunden. Wenn nun ein Kind nach draußen will, braucht es noch zwei von den Sachen aus dem Schrank:

- entweder Schal und Mütze,
- oder Mütze und Handschuhe,
- oder Handschuhe und Schal.

Geben Sie eine deadlock-freie Lösung dieses Problems in einer Pseudo-Programmiersprache an (ähnlich wie auf Folie 152). Verwenden Sie dazu für gemeinsame Ressourcen zwei Operationen `reserve()` und `release()`: `reserve()` belegt die Ressource und `release()` gibt die Ressource wieder frei (Semaphorenprinzip). Begründen Sie, warum Ihr System deadlock-frei ist.

Aufgabe 26 (Monitor, Bedingungssynchronisation: Entwurf)

Ziel: Die Sitzplatzbelegung in einem Restaurant soll simuliert werden. Besucher betreten in Gruppen das Restaurant, suchen sich einen Tisch, essen und verlassen das Restaurant wieder.

Modellierung:

- **Tische:** An jedem Tisch steht eine bestimmte Anzahl von Stühlen, einige davon sind belegt, die restlichen Stühlen sind frei.
- **Restaurant:** Ein Restaurant besteht aus mehreren Tischen. Alles andere ist für diese Simulation nicht relevant.
- **Besuchergruppen:** Die Gruppen bestehen aus einer bestimmten Anzahl Personen und wissen im voraus, wie lange Sie im Restaurant bleiben werden (nachdem sie einen Platz gefunden haben). Besuchergruppen werden als Prozesse (*Threads*) modelliert. (Auch in der Realität handeln die Leute gleichzeit und unabhängig voneinander).

Verhalten der Besuchergruppen

1. Restaurant betreten
2. Platzsuche gemäß "westfälischer" Strategie (siehe unten)
3. Aufenthalt im Restaurant
4. Restaurant verlassen

Strategie bei der Platzsuche einer Gruppe aus *n* Personen

Bei der westfälischen Sitzplatzsuche trennen sich Gruppen niemals auf. Außerdem will eine Gruppe lieber allein am Tisch sitzen als sich mit einer fremden Gruppe einen Tisch zu teilen. Später soll sich möglichst kein Fremder zu der Gruppe dazusetzen können. Formal läuft die Platzsuche also so ab:

1. Versuche, einen noch völlig leeren Tisch mit *n* Stühlen zu finden.
2. Versuche, einen völlig leeren Tisch mit mindestens *n* Stühlen zu finden.
3. Versuche, einen Tisch mit mindestens *n* freien Stühlen zu finden.
4. Beginne wieder ab 1. bis Sitzplätze gefunden wurden

Aufgaben: (Entwurf)

a) Welche Objekte der Simulation sollen als Monitor wirken (siehe Folie 143)?
b) Welche Monitor-Operationen werden benötigt (siehe Folie 143)? An welchen Stellen ist Bedingungssynchronisation erforderlich (siehe Folie 145)? Wie lauten diese Bedingungen?

Hinweis: Legen Sie für die zusätzlichen Methoden zunächst nur die Signaturen und die zu erledigenden Aufgaben fest. Überlegen Sie für jede dieser Methoden, ob sie `synchronized` sein muss (Begründung!). Denken Sie beim Entwurf an die umzusetzende Strategie für die Platzsuche.

Aufgabe 27 (Monitor, Bedingungssynchronisation: Implementierung)

a) Implementieren Sie Ihren Entwurf aus Aufgabe 26 in Java. Ergänzen Sie dazu die Klassen `Tisch`, `Restaurant` und `BesucherGruppe` um die noch fehlenden Methoden. Vervollständigen Sie die `run`-Methode, so daß sich jede Gruppe wie oben beschrieben verhält.

   Die Datei `Tisch.java` enthält den Rahmen für eine entsprechende Klasse `Tisch`. Einen geeigneten Rahmen für die Klasse `BesucherGruppe` finden Sie in der Datei `BesucherGruppe.java`, einen Rahmen für die Klasse `Restaurant` in der Datei `Restaurant.java`.

b) Testen Sie Ihr Programm mit der `main`-Methode aus der Klasse `Restaurant`.