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

Übungsblatt 2

Abgabe bis 02.05.2018, 18:00 Uhr

Aufgabe 2.1: Flynns Taxonomie

- Was bedeuten SISD und MIMD? Was unterscheidet diese beiden Ansätze grundsätzlich von einander?
- Nach Flynn gibt es auch noch SIMD und MISD. Wie können diese beiden Programmieransätze aussehen?
- Was unterscheidet SIMD von SPMD?

Aufgabe 2.2: Petri-Fabrik

In dieser Aufgabe geht es um die Modellierung einer einfachen Industrieanlage. Rohstoffe werden in ein Materiallager angeliefert, das beliebig viel aufnehmen kann. Aus diesem Lager nehmen zwei identische Maschinen jeweils genau ein Teil und verarbeiten es zu einem Produkt weiter. Ist die Verarbeitung abgeschlossen, wird das Produkt aus der Maschine ausgeworfen und die Verarbeitung des nächsten Teils beginnt. Die ausgeworfenen Teile werden von einem Greifarm auf ein Laufband transportiert. Maximal ein Produkt kann sich gleichzeitig auf dem Laufband befinden. Von diesem Band aus kommen die Teile in einen anderen Teil der Fabrik, der nicht mehr zu modellieren ist.

a) Entwerfen Sie ein Petri-Netz, das das Verhalten der Anlage nachbildet. Erstellen Sie hierzu Stellen für das Materiallager , für die Verarbeitung durch die beiden Maschinen, für die Ausgabe jeder Maschine und für das Laufband . Fügen Sie dann die benötigten Transitionen und möglicherweise Stellenbeschränkungen hinzu, um den richtigen Ablauf zu gewährleisten.

b) An welchen Stellen ist **Nebenläufigkeit** der verschiedenen Vorgänge möglich? Wie lassen sich solche Stellen erkennen?

c) Es soll nur dann ein neues Teil in die Maschinen eingezogen werden, wenn die Ausgabe wieder leer ist. Wie lässt sich dies modellieren?

Bonusaufgabe 2.3: 99 Bottles of Beer — 4 Punkte

a) Implementieren Sie eine Klasse `SingerThread`, deren `main`-Methode mit Hilfe von verschiedenen `Threads` den Text des Liedes "99 bottles of beer" (http://99-bottles-of-beer.net/lyrics.html) ausgibt. Hierbei soll die Anzahl N der zu startenden `Threads` als Parameter auf der Konsole übergeben werden. „`THREADNUMMER`" soll für jeden erzeugten `Thread` eine eindeutige Nummer n mit $0 \leq n < N$ sein.

Ausgabe des Threads mit `THREADNUMMER==0`:

```
No more bottles of beer on the wall, no more bottles of beer. Go to the
store and buy some more, 99 bottles of beer on the wall.
```

Ausgabe des Threads mit `THREADNUMMER==1`:

```
1 bottle of beer on the wall, 1 bottle of beer. Take one down and pass it
around, no more bottles of beer on the wall.
```

Ausgabe des Threads mit `THREADNUMMER==2`:

```
2 bottles of beer on the wall, 2 bottles of beer. Take one down and pass
it around, 1 bottle of beer on the wall.
```

Alle anderen `Threads` erzeugen ihre Ausgabe nach folgendem Muster:

```
THREADNUMMER bottles of beer on the wall, THREADNUMMER bottles of beer.
Take one down and pass it around, THREADNUMMER-1 bottles of beer on the
wall.
```

Hinweis: Eine korrekte Abfolge der Strophen soll nicht erzwungen werden.

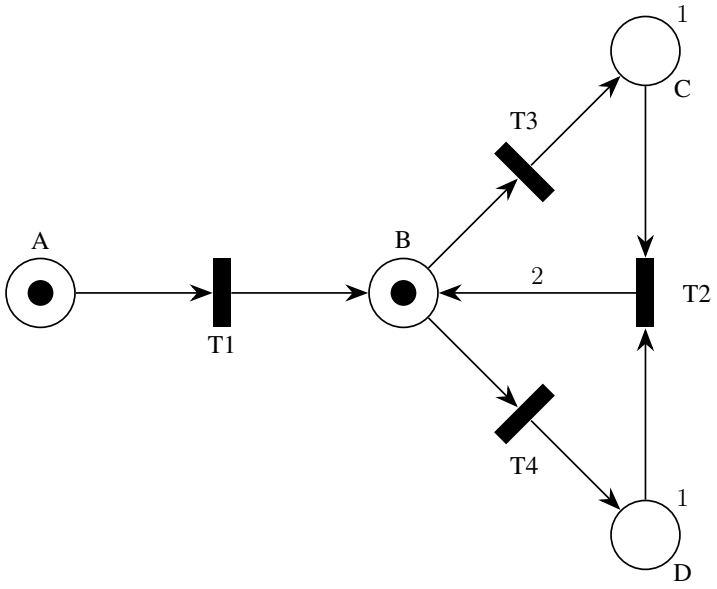
b) Implementieren Sie eine Klasse `SingerRunner`, deren `main`-Methode sich genauso verhält wie `SingerThread`, nur dass die Ausgabe nicht mit `Thread`-Objekten, sondern über das `Runnable`-Interface realisiert werden soll. `SingerRunner` darf keine abgeleitete Klasse von `SingerThread` oder `Thread` sein.

c) Bauen Sie `SingerRunner` zu `SingerExecutor` um, so dass die Arbeitspakete nicht mehr durch explizite `Thread`-Erzeugung, sondern durch einen `newFixedThreadPool` der Größe N ausgeführt wird.

Hinweis: Der Haupt-`Thread` soll bei allen drei Lösungen nichts tun, außer die entsprechenden Arbeitspakete zu starten, und danach auf deren Beendigung zu warten.

Bonusaufgabe 2.4: Beschränktheit **2 Punkte**

Gegeben sei folgendes Petri-Netz:



Ist das gegebene Petri-Netz beschränkt? Begründen Sie ihre Entscheidung. Geben Sie Ihre Lösung in der Datei **pfp_abgabe_2.pdf** ab.

Unbepunktete Programmieraufgaben:

Die folgende Aufgabe gibt keine Bonuspunkte. Ihre Lösungen, die Sie über das EST abgeben können, werden aber dennoch korrigiert. Auf diese Weise können Sie weitere Konzepte üben, die durch die Bonusaufgaben nicht abgedeckt werden.

Aufgabe 2.5: Array-Summe

Implementieren Sie eine Klasse `ArraySumImpl`, die das auf der Übungsseite zur Verfügung gestellte Interface `ArraySum` implementiert. Die Methode `long sum(long[] array, int threads)` soll mit Hilfe eines `newFixedThreadPool` der Größe `threads` und `threads` `Future`-Objekten die Summe der Werte in einem Array berechnen. Jedes Arbeitspaket soll einen Teil der Werte summieren und das Teilergebnis zurückliefern. Die `sum`-Methode führt diese Teilergebnisse dann zum Gesamtergebnis zusammen. Überlegen Sie, wie die Werte auf die einzelnen `Future`-Objekte aufgeteilt werden können und testen Sie Ihre Implementierung mit dem JUnit-Testcase `ArraySumTest`.

6 Punkte