Schule Informationstechnik

der Bundeswehr

Sprachausbildung Java

Übungen

"Multithreading"

Stand: April 2018

1 Primzahltest

Diese Aufgabe ist die Vorbereitung eines Primzahltests mit Hilfe von Threads.

1. Schreiben Sie eine Klasse Primzahltest. Ein Objekt dieser Klasse bekommt bei der Erzeugung eine Ganzzahl zugewiesen.
2. Schreiben sie in dieser Klasse eine Methode isPrim(), die die enthaltene Zahl überprüft, ob es sich dabei um eine Primzahl handelt. Das Ergebnis wird in einem Objektattribut gespeichert, sowie direkt von der Funktion zurückgeliefert.
3. Testen Sie die Klasse. Schreiben Sie eine Testfunktion, welche eine Zahl einliest, und diese mit Hilfe der Primzahlklasse auf Prim testet.

2 Paralleler Primzahltest

Nun kommen die Threads ins Spiel.

1. Machen Sie aus der Primzahltest-Klasse eine Thread-Klasse. Sobald der Thread gestartet wird, soll die enthaltene Zahl auf Prim getestet werden. Bauen Sie vor der Prüfung eine zufällige Verzögerung <=50ms ein. (Der Primzahltest für sehr große Zahlen ist extrem rechenintensiv. Deshalb würde man für den Test von vielen Zahlen mehrere Rechner verwenden (also ein Cluster). Die Verzögerung soll dies simulieren (Zeit für Übertragung und Berechnung)).
2. Schreiben Sie einen Test, der mit Hilfe von 10 Threads, 10 Zahlen ab einem eingegebenen Startwert testet und Primzahlen ausgibt.
3. Schreiben Sie einen Test, der mit maximal 10 Threads gleichzeitig, beliebig viele Zahlen (vom User eingegeben) ab einem eingegebenen Startwert testet und Primzahlen ausgibt.
4. Verändern Sie das Programm derart, dass alle Primzahlen zwischen einem eingegebenen Startwert und Endwert (mit maximal 10 Threads gleichzeitig) ausgegeben werden.

3 Soldatenleben I

1. Schreiben Sie eine Klasse `Soldat`:
   - Ein `Soldat` hat eine eindeutige Nummer (Personalnummer) zur Identifikation.
   - Unser Soldat ist ziemlich einfach strukturiert und kann lediglich seine Ausrüstung packen. Dieses Packen dauert eine zufällige Zeitspanne. Vor Beginn des Packens meldet er sich ab und bei Rückkehr zurück (Ausgaben auf Konsole).
2. Schreiben Sie eine Testklasse `GrpFhr`.
   - Hier sollen 8 Soldaten erstellt und zum Packen geschickt werden.
   - Der Gruppenführer wartet bis alle Soldaten fertig gepackt haben und meldet sobald alle fertig sind.

4 Soldatenleben II

Erweitern wir nun Aufgabe 3.

1. Implementieren Sie eine Klasse `Gruppe`. Eine solche `Gruppe` besteht aus:
   a. Einer eindeutigen ID
   b. einem Soldaten, dem Gruppenführer
   c. bis zu 8 Soldaten.
   - Beginnt der Gruppenführer mit seiner Arbeit, schickt er alle Soldaten zum Packen. Anschließend wartet er bis alle fertig sind.
   - Dann geht er mit seiner Gruppe essen, wobei die Gruppe eine zufällige Zeitspanne zur Kantine marschiert.
   - Ist die Gruppe dann angekommen, meldet der Gruppenführer, dass seine Gruppe beim Essen ist.
   - Nach einer zufälligen Zeitspanne fürs Essen und den Rückmarsch meldet sich der Gruppenführer zurück.
2. Schreiben Sie eine Testklasse `Zugfuehrer`.
   - Hier sollen 2 Gruppen à 8 Soldaten erstellt und zum Essen geschickt werden.
   - Der `Zugfuehrer` wartet, bis alle Gruppen vom Essen zurück sind. Wenn alle fertig sind, meldet der `Zugfuehrer`, dass alle zurück sind.

5 Soldatenleben III

Erweitern wir nun Aufgabe 4. Die Küche in unserer Kaserne ist sehr klein. D.h. es darf immer nur eine Gruppe in der Küche essen, der Rest muss warten.

1. Erzeugen Sie eine Klasse `Küche`. Diese Klasse hat lediglich eine Methode, nämlich `essen(int grpID)`. Diese Methode gibt nur aus, dass eine bestimmte Gruppe beim Essen ist.
2. Synchronisieren Sie den Bereich Essen aus Aufgabe 4 mit Hilfe eines Küchenobjektes.

6 Soldatenleben IV

Die letzte Erweiterung.

Erweitern Sie folgende Aspekte:

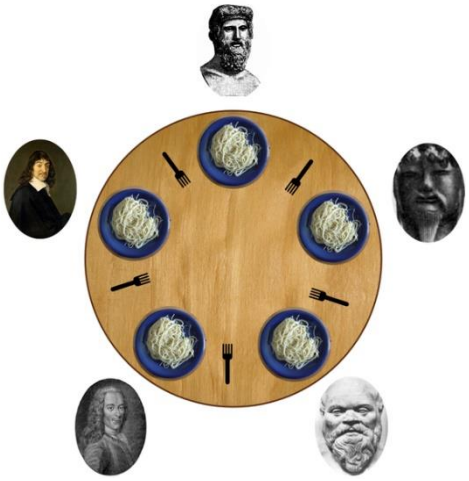
1. Soldaten benötigen beim Essen unterschiedlich Zeit.
2. Erweitern Sie das Packet um die Klasse Zug. Ein Zug besteht aus <= 6 Gruppen.
3. Wird ein Zugobjekt gestartet, werden alle Gruppen zum Essen geschickt. Sind die Gruppen zurück, meldet dies der Zugführer.
4. Testen Sie ihr Programm mit einem Zug.

7 Cloning and Killing

Implementieren Sie eine Cloning Machine, die Scratchys produziert. Jeder Scratchy hat eine eindeutige Nummer. Die Cloning Machine produziert je einen Scratchy in 200 ms. Dieser wird dann auf ein Fließband gesetzt und zur Killing Machine transportiert. Dort wird dann in 100 ms ein Scratchy „verarbeitet“. Das geht natürlich nur, wenn ein Scratchy auf dem Band sitzt. Realisieren Sie das Fließband als Queue (LinkedList). Die Killing Machine wartet stets darauf, dass in der Queue mindestens ein Scratchy ist.

8 Das Philosophenproblem

Fünf Philosophen sitzen an einem runden Tisch. Vor jedem Philosophen befindet sich ein Teller mit Spaghetti. Zwischen zwei Tellern liegt jeweils eine Gabel.



Jeder Philosoph erlebt der Reihe nach folgende drei Zustände: "Denken", "Hungrig" und "Essen". Zum Denken braucht er keine Gabel. Wenn er hungrig wird, greift er zu den Gabeln an seiner rechten und linken Seite. Wenn sie nicht frei sind, bleibt er hungrig und wartet, bis beide seine Nachbarn nicht mehr essen. Wenn die Gabeln frei sind, fängt er an zu essen. Dann müssen seine beiden Nachbarn gegebenenfalls hungrig warten, bis er fertig ist. Anschließend versinkt er wieder in den Denkzustand, bis er hungrig wird. Die Frage ist, ob es möglich ist, die Aktivitäten der Philosophen untereinander so zu koordinieren, dass niemand verhungert.

Das Problem ist, wenn alle Philosophen jeweils zur gleichen Zeit hungrig werden und sich gleichzeitig eine Gabel nehmen, in der Hoffnung, anschließend die zweite Gabel zu bekommen, kommt es zu einer Verklemmung (Deadlock).

Versuchen Sie dieses Problem zu implementieren. Dazu brauchen Sie fünf Objekte der Klasse Philosoph. Ein Philosoph hat einen Namen, zwei Objekte der Klasse Gabeln (Klasse bzw. Objekt hat nur eine ID) und einen Zustand. Bedenken Sie, dass sich je zwei Philosophen eine Gabel teilen, es existieren also genau fünf Gabel-Objekte. Das Denken und das Essen dauern eine zufällige Zeit. Die run()- Methode der Philosophen ist hier eine fast endlose Schleife (die man nur mit einem interrupt beenden kann), die das simple Leben der Philosophen darstellt (denken, hungrig, essen).

Versuchen Sie nun durch Manipulation der Zeiten für Essen und denken einen Deadlock zu produzieren. Überlegen Sie sich, wie man diesen verhindern kann.