Grundlagen der Systemsoftware

SoSe 2010 · Übungsblatt 3 · Abgabe: Mi. 12.05.2010 (bis 12:00 Uhr)

Aufgabe 1: Speicherverwaltung (20 Punkte)

Früher war ein Computer darauf ausgelegt lediglich ein Programm zur Zeit auszuführen. Wenn ein Programm ausgeführt wurde, konnte dieses den gesamten Arbeitsspeicher nutzen, da alle Ressourcen exklusiv zur Verfügung standen. Im Laufe der Entwicklung wuchs der Bedarf auch mehrere Programme gleichzeitig laufen zu lassen, mit der Konsequenz, dass der Arbeitsspeicher für gewöhnlich zu klein ist, um alle Prozesse samt ihrer Daten zu beherbergen. Um diesem Problem beizukommen wurde eine Technik namens *Paging* entwickelt.

a) Beschreiben Sie diese Technik (mit max. 150 Wörtern). Gehen Sie dabei auch auf die Begriffe *Seite*, *Seitenrahmen* und logischer (virtueller) Speicher ein. (8 Punkte)

b) Kann durch Paging das Problem der *externen Fragmentierung* des Speichers gelöst werden (falls ja, inwiefern, falls nein, warum nicht)? (4 Punkte)

c) Woraus resultieren Seitenfehler (engl. *Page Fault*)? Nennen Sie mindestens drei Faktoren, die die Wahrscheinlichkeit für das Auftreten von Seitenfehlern signifikant beeinflussen können. (8 Punkte)

Aufgabe 2: Seitenersetzungsalgorithmen (25 Punkte)

a) Gegeben sei ein Seitenspeicher der Größe 3. Zum Zeitpunkt 0 sei der Speicher leer. Illustrieren Sie (z. B. anhand von Tabelle 1) den Zustand des Seitenspeichers bei Verwendung von (a) des *optimalen Seitenersetzungsalgorithmus* (10 Punkte) und (b) des LRU-Algorithmus (*Least Recently Used*) (10 Punkte) zu den Zeitpunkten 1 bis 15 für die Referenzkette 1, 2, 3, 4, 5, 6, 1, 3, 1, 6, 3, 5, 4, 2, 1. Geben Sie jeweils die Zahl der Seitenalarme (engl. „page faults“) an.

t	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
Page	1	2	3	4	5	6	1	3	1	6	3	5	4	2	1
* = Fault															
Seiten	?	?	?	?	?	?	?	?	?	?	?	?	?	?	?
im		?	?	?	?	?	?	?	?	?	?	?	?	?	?
Speicher			?	?	?	?	?	?	?	?	?	?	?	?	?

Tabelle 1: Zustand des Seitenspeichers für t = 1 bis 15.

b) Woraus kann sich aus Ihrer Sicht die unterschiedliche Zahl der Seitenalarme bei der Verwendung des optimalen und des LRU-Algorithmus ergeben? (max. 3 Sätze) (3 Punkte)

c) Kann der optimale Seitenersetzungsalgorithmus für reale Betriebssysteme effektiv eingesetzt werden? Begründen Sie kurz (max. 3 Sätze) ihre Antwort. (2 Punkte)

Aufgabe 3: Synchronisation (45 Punkte)

Ein Druckdienst soll eingehende Druckaufträge auf eine feste Anzahl von Druckern verteilen. Dabei muss der Druckdienst dafür Sorge tragen, dass jeder Drucker stets nur maximal einen Auftrag zur Zeit verarbeitet. Der Druckdienst kann von mehreren Prozessen bzw. Threads gleichzeitig aufgerufen werden, wobei der Aufruf solange blockiert, bis das Dokument gedruckt wurde.

Ein solcher Druckdienst soll in Java implementiert werden. Dazu finden Sie auf der Website ein Paket mit einigen Hilfklassen, die für die Implementierung verwendet werden sollen:

- **Paket** `de.uhh.informatik.vsis.gss.tools`
  - **Klasse** `Drucker`: Druckt etwas virtuell (Druckdauer ist zufällig) und vermerkt Beginn und Ende in einem Log.
  - **Klasse** `Tools`: Stellt eine Methode für aktives Warten (*busy waiting*) bereit.
- **Paket** `de.uhh.informatik.vsis.gss.sync`
  - **Klasse** `TSL`: Simuliert die Assembler-Operation *Test and Set Lock*
  - **Klasse** `Semaphore`: Stellt die Methoden `P()` und `V()` der klassischen Semaphore bereit[1].
  - **Klasse** `Mutex`: Stellt einen Mutex bereit, der auf obiger Semaphore-Klasse basiert.
- **Paket** `de.uhh.informatik.vsis.gss.aufgaben`
  - **Interface** `Druckdienst`: Diese Schnittstelle soll implementiert werden
  - **Klasse** `DruckdienstDummy`: Vorlage für die Implementierung
  - **Klasse** `DruckdienstNaiv`: Bereits implementierter Druckdienst, der jedoch *Race Conditions* besitzt.
  - **Klasse** `DruckTest`: Test-Klasse mit `main`-Methode, die initialisierte Drucker an einen konkreten Druckdienst übergibt und mehrere Threads zum Drucken startet. Anschließend werden die Logs der Drucker ausgegeben.

Hinweis: Der Befehl `Tools.busyWait(250);` kann zwischen dem Vergleichen und Ändern von Variablen eingefügt werden, um die Wahrscheinlichkeit zu erhöhen, dass *Race Conditions*[2] zu einem sichtbaren Problem werden. Sie können dann an dem Drucker-Log erkennen, ob Druckaufträge fälschlicherweise parallel verarbeitet wurden (bei jedem einzelnen Drucker darf zwischen Druckbeginn und Druckende eines Dokumentes kein weiteres Dokument gestartet oder beendet werden). Bedenken Sie jedoch, dass das Ausbleiben von sichtbaren Problemen nicht bedeutet, dass keine Race Conditions existieren!

[1] Dabei wird eine Klasse `Semaphore` verwendet, die seit Java 2 SE 5.0 Bestandteil der Java-Plattform ist. Sie benötigen daher ein entsprechendes JDK!

[2] Race Conditions sind Wettlaufsituationen, bei denen das Ergebnis einer Operation vom zeitlichen Verhalten während der Ausführung abhängt. Sie führen unter anderem dann zu Problemen, wenn der Scheduler den laufenden Prozess bzw. Thread zu einem für diesen ungünstigen Zeitpunkt verdrängt.

a) **Test and Set Lock** (15 Punkte)
Einige Prozessoren unterstützen einen Befehl zum gleichzeitigen Lesen und Verändern eines Speicherwortes. Da Java jedoch auf einer virtuellen Maschine ausgeführt wird, wird dieser Befehl durch die Klasse `TSL` simuliert. Verwenden Sie diese Klasse um den Druckdienst zu implementieren (`DruckdienstTSL`). Verwenden Sie dabei *keine* anderen Konstrukte zur Synchronisation, wie beispielsweise *synchronized*, *Semaphore*, *Mutex*, *wait* oder *notify*. Das Verwenden von aktivem Warten ist erlaubt.

b) **Semaphore** (15 Punkte)
Semaphore können einen Prozess blockieren um aktives Warten zu vermeiden. Darüber hinaus kapseln sie einen Zähler, der zur Verfügung stehende Ressourcen repräsentieren kann. Ein *Mutex* ist ein Semaphore, dessen Zähler mit 1 initialisiert ist. Verwenden Sie die Klassen *Semaphore* und/oder *Mutex* um den Druckdienst zu implementieren (`Druckdienst-Semaphore`). Verwenden Sie dabei *keine* anderen Konstrukte zur Synchronisation, wie beispielsweise *TSL*, *synchronized*, *wait* oder *notify*. Das Verwenden von aktivem Warten ist *nicht* erlaubt.

c) **Monitor** (15 Punkte)
Da der Umgang mit Semaphoren genaue Aufmerksamkeit erfordert, besitzen moderne Programmiersprachen das sprachliche Konstrukt *Monitor*. Auch Java verfügt über entsprechende Konstrukte und Methoden. Verwenden Sie diese um den Druckdienst zu implementieren (`DruckdienstMonitor`). Verwenden Sie dabei *keine* anderen Konstrukte zur Synchronisation, wie beispielsweise *TSL*, *Semaphore* oder *Mutex*. Das Verwenden von aktivem Warten ist *nicht* erlaubt.

Die zu erstellenden Klassen sollen sich in einem Package befinden, dessen Namen sich von `de.uhh.informatik.vsis.gss.aufgaben.g`*XY* ableitet, wobei *X* die Nummer Ihrer Übungsgruppe und *Y* der Buchstabe Ihrer Kleingruppe ist. Geben Sie als Lösung dieser Aufgabe den Quellcode für die drei von Ihnen implementierten Java-Klassen `DruckdienstTSL`, `Druckdienst-Semaphore` und `DruckdienstMonitor` in einer ZIP-komprimierten Archiv-Datei per E-Mail an den jeweiligen Übungsgruppenleiter ab.

Aufgabe 4: Deadlocks (10 Punkte)

Ein *Deadlock* bezeichnet die Situation, in der eine Menge von Prozessen blockiert sind, weil jeder Prozess auf ein Ereignis wartet, das nur einer der anderen Prozesse dieser Menge auslösen kann.

a) Es gelten vier Rahmenbedingungen, die erfüllt sein müssen, damit ein Deadlock überhaupt entstehen kann. Welche Bedingungen sind dies? Erläutern Sie diese vier Bedingungen in jeweils einem Satz. (4 Punkte)

b) Beschreiben Sie eine Deadlock-Situation anhand eines realen Beispiels mit drei Prozessen und drei Betriebsmitteln (max. 5 Sätze). (3 Punkte)

c) Modellieren Sie Ihr obiges Beispiel anhand eines *Betriebsmittel-Allokationsgraphen* (siehe Abbildung 1). Kann man anhand eines solchen Graphen die mögliche Gefahr eines Deadlocks erkennen (falls ja, wie, falls nein, warum nicht)? (max. 3 Sätze) (3 Punkte)

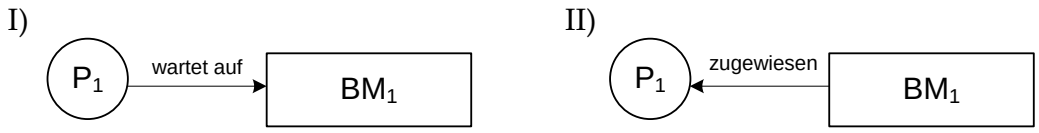

Abbildung 1: BetriebsmittelAllokationsgraph: Abb. I) Prozess wartet auf ein Betriebsmittel. Abb. II) Prozess hält ein Betriebsmittel