Ludwig-Maximilians-Universität München
Institut für Informatik
Lehrstuhl für Mobile und Verteilte Systeme
Prof. Dr. Claudia Linnhoff-Popien

mobile and distributed systems group

Betriebssysteme im Wintersemester 2019/2020
Übungsblatt 10

Abgabetermin: 13.01.2020, 18:00 Uhr

Besprechung: Besprechung der T-Aufgaben in den Tutorien am 23. Dezember 2019 und vom 07. – 10. Januar 2020
Besprechung der H-Aufgaben in den Tutorien vom 13. – 17. Januar 2020

Aufgabe 47: (T) Wechselseitiger Ausschluss

(– Pkt.)

In dieser Aufgabe soll das Szenario **zweier** Prozesse P_1 und P_2, die auf einen kritischen Abschnitt zugreifen, zu einem Erzeuger/Verbraucher-Szenario erweitert werden.
Der Prozess P_1 erzeugt Daten und schreibt diese in einen gemeinsamen Speicher mit 5 Plätzen. Der Prozess P_2 liest diese Daten. Dazu werden die beiden Zählsemaphoren `platz` und `bestand` eingeführt und wie folgt initialisiert:

```
Semaphore platz, bestand;
init(platz, 5);
init(bestand, 0);
```

Ergänzen Sie die folgenden Prozessdefinitionen unter Verwendung dieser Semaphoren so, dass P_1 nicht in den vollen Speicher schreiben und P_2 nicht aus einem leeren Speicher lesen kann.

```
P1:                                  P2:
<rechne im unkritischen Bereich>     <rechne im unkritischen Bereich>


wait(mutex);                         wait(mutex);


<erzeuge Element>                    <lies Element>


signal(mutex);                       signal(mutex);


<rechne im unkritischen Bereich>     <rechne im unkritischen Bereich>
```

Aufgabe 48: (T) Nachbildung von Zählsemaphoren mittels Monitoren in Java

(– Pkt.)

Schreiben Sie eine Java Klasse `SimpleCountingSemaphore`, die es ermöglicht, Instanzen zu erzeugen, welche in Analogie zu der von Dijkstra vorgeschlagenen Datenstruktur eines Zählsemaphor verwendet werden können. Verwenden Sie dazu den Java `synchronized`-Mechanismus. Verwenden Sie außerdem eine *minimale* Anzahl an `wait()`- und `notify()`-Aufrufen!

Hinweis: Für die Lösung dieser Aufgabe reicht es aus, wenn Sie das auftreten einer etwaigen `InterruptedException` ohne weitere Fehlerbehandlung abfangen.

Des Weiteren sollen Sie sich in dieser Aufgabe die Unterschiede und Gemeinsamkeiten zwischen Java-Synchronisation und Monitoren klar machen.

a. Beschreiben Sie das grundlegende Konzept der Synchronisation in Java. Gehen Sie dabei auf die Verwendung von Objekt-Locks, Threads und Warteschlangen ein.

b. Wie werden `wait()` und `signal()` in Java umgesetzt?

c. Erläutern Sie den grundlegenden Unterschied des Java Synchronisationsmechnismus im Gegensatz zu „echten“ Monitoren (ohne Beachtung der Signalisierungsmechanismen).

d. Beschreiben Sie die Einschränkungen bei der Verwendung von `wait()` und `notify()` gegenüber „echten“ Monitoren.

   Hinweis: Überlegen Sie sich dazu, wie im Falle echter Monitore bzw. im Falle des Java Synchronisierungsmechanismus der nächste aktive Thread selektiert wird.

e. Überlegen Sie sich, wie diese Einschränkungen umgangen werden können.

f. Es gibt zwei Modelle, wie die Ausführung in einem Monitor nach dem Aufruf von `signal()` fortfährt (A: signalisierender Prozess, B: aufgeweckter Prozeß):

   - Signal-and-wait: A muß nach seiner Signalisierung den Monitor sofort freigeben und warten, bis B den Monitor verlassen hat oder auf eine andere Bedingung wartet.
   - Signal-and-continue: B muß warten, bis A den Monitor verlassen hat oder auf eine andere Bedingung wartet.

   Welchem Modell folgt Java?

Aufgabe 49: (T) User-Level-, Kernel-Level-Threads und Prozesse

(– Pkt.)

Betrachten Sie nun folgendes Fallbeispiel: Gegeben ist eine Anwendung, die bisher single-threaded abläuft, also nur aus einem einzigen Thread besteht. Diese Anwendung bietet eine Eingabe-Schnittstelle, über die der Benutzer mathematische Ausdrücke auswerten lassen kann. Die Anwendung soll nun so modifiziert werden, dass der Benutzer während einer laufenden Auswertung eines Ausdrucks weitere Berechnungen starten oder auch Zwischenergebnisse einer der aktiven Auswertung abfragen kann.

Im Folgenden wird angenommen, dass ein Benutzer die Berechnung der ersten hunderttausend Primzahlen anfordert.

a. Warum muss man bei diesem Szenario unbedingt einen neuen Thread oder Prozess für die Berechnung starten?

b. Es werden folgende Alternativen erwogen:

- Berechnung der Primzahlen in einem neuen User-Level-Thread starten.
- Berechnung der Primzahlen in einem neuen Kernel-Level-Thread starten.
- Berechnung der Primzahlen in einem neuen Prozess starten.

Bewerten Sie jede der drei Möglichkeiten hinsichtlich der folgenden Gesichtspunkte. Verwenden Sie dazu eine geeignete tabellarische Darstellung.

(i) Aufwand für die Generierung des neuen Threads/Prozesses

(ii) Kommunikation/Datenaustausch zwischen der bisherigen Anwendung und dem neuen Thread/Prozess

(iii) Abwicklung des Schedulings (Wer ist für das Scheduling des neuen Threads/Prozesses verantwortlich?)

(iv) Ausführung auf einer Multiprozessorumgebung

c. Basierend auf Ihrer Bewertung in der Teilaufgabe b), welche Möglichkeit würden Sie hier wählen? Begründen Sie Ihre Entscheidung.

Aufgabe 50: (H) Threads/Monitore in Java

(19 Pkt.)

In dieser Aufgabe soll der Arbeitsablauf auf einer stilisierten Apfelplantage in Java mit dessen Implementierung des Monitor-Konzepts simuliert werden. Auf der Apfelplantage arbeiten 2 Feldarbeiter und ein Koch. Feldarbeiter müssen essen, um Äpfel pflücken zu können. Eine Portion Apfelmus reicht aus, damit sich ein Feldarbeiter genug gestärkt fühlt, um 2 Äpfel pflücken zu können (außer Apfelmus gibt es nichts zu essen). Vor dem Pflücken muss ein Arbeiter satt sein. Nach dem Pflücken ist der Feldarbeiter wieder hungrig. Der Koch kann aus 12 Äpfeln 8 Portionen Apfelmus kochen. Er beginnt immer erst dann mit dem Kochen, wenn er mindestens 12 Äpfel hat. Um die schwere Kocharbeit verrichten zu können, muss sich der Koch selbst zuvor auch mit einer Portion Apfelmus stärken. Danach ist er wieder hungrig. Zu Beginn sind sowohl der Koch als auch die Feldarbeiter hungrig. Die Anzahl der Äpfel bzw. die Anzahl der Portionen Apfelmus im Lager darf in Ihrer Implementierung nicht unter 0 fallen.
Im Folgenden soll die Klasse `Lager` implementiert werden. Die Beispielimplementierungen der Klassen `Apfelplantage`, `Koch` und `Feldarbeiter` sollen Ihnen verdeutlichen, wie die Klasse `Lager` verwendet werden kann. Dieses können auf der Vorlesungswebsite heruntergeladen werden.
Bearbeiten Sie die folgenden Aufgaben. Ein Rahmen zur Bearbeitung der Aufgaben a), b), c und d) steht auf der Vorlesungswebsite zur Verfügung.

a. Implementieren Sie den Konstruktor der Klasse `Lager`. Die Klassenattribute sind dort bereits deklariert und sollen durch den Konstruktor explizit initialisiert werden.

b. Implementieren Sie die Methode `apfelmusEntnehmen(int id, int anzahl)`, welche das Entnehmen von Apfelmus aus dem Lager zum Verzehr modelliert.

c. Implementieren Sie die Methode `aepfelEntnehmen(int anzahl)`, welche das Entnehmen von Äpfeln aus dem Lager durch den Koch modelliert.

d. Vervollständigen Sie nun die Methode `apfelmusEinlagern(int anzahl)`. Auch hier sollten die oben genannten Anforderungen Beachtung finden.

e. In der Methode `aepfelEinlagern(int anzahl)`, die vom Feldarbeiter aufgerufen wird, ist bereits der Aufruf des Befehls *notifyAll()* vorgegeben. Welche andere Semantik hat der *notify()* Befehl im Vergleich dazu und welche Änderungen im Ablauf würden sich potenziell ergeben, wenn *notify()* anstelle von *notifyAll()* aufgerufen werden würde.

Aufgabe 51: (H) Einfachauswahlaufgabe: Prozesskoordination

(5 Pkt.)

Für jede der folgenden Fragen ist eine korrekte Antwort auszuwählen („1 aus n“). Nennen Sie dazu in Ihrer Abgabe explizit die jeweils ausgewählte Antwortnummer ((i), (ii), (iii) oder (iv)). Eine korrekte Antwort ergibt jeweils einen Punkt. Mehrfache Antworten oder eine falsche Antwort werden mit 0 Punkten bewertet.

a) Wie betritt ein Prozess einen Monitor?			
(i) Durch den Zugriff auf **public** Variablen.	(ii) Durch den Zugriff auf bestimmte Monitorprozeduren.	(iii) Durch den Aufruf von *notify()*.	(iv) Durch den Aufruf von *notifyAll()*.
b) Welches Kommunikationsschema liegt vor, wenn der Sender nach dem Absenden mit seiner Ausführung fortfahren kann aber der Empfänger bis zum Erhalt einer Nachricht wartet.			
(i) Blocking Send, Blocking Receive.	(ii) Blocking Send, Nonblocking Receive.	(iii) Nonblocking Send, Blocking Receive.	(iv) Nonblocking Send, Nonblocking Receive.
c) Welcher der folgenden Betriebssystemmechanismen dient vorrangig der Prozesssynchronisation und weniger der Prozesskommunikation?			
(i) Message Queues	(ii) Shared Memory	(iii) Semaphore	(iv) Pipes
d) Wie bezeichnet man die Thread-Implementierung, bei der das Threadmanagement Aufgabe der jeweiligen Anwendung ist und der Betriebssystemkern keine Informationen über die Existenz solcher Threads hat bzw. haben muss?			
(i) User-Level-Threads	(ii) Kernel-Level-Threads	(iii) Hardware-Level-Threads	(iv) Low-Level-Threads
e) Wie bezeichnet man die Phase, in der sich zu einem Zeitpunkt nur ein Prozess befinden darf, da sich sonst z.B. inkonsistente Zustände bei gemeinsam genutzten Datenstrukturen ergeben?			
(i) einfacher Bereich	(ii) schwieriger Bereich	(iii) unkritischer Bereich	(iv) kritischer Bereich