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 2017/2018
Übungsblatt 11

Abgabetermin: 22.01.2018, 18:00 Uhr

Besprechung: Besprechung der T-Aufgaben in den Tutorien vom 15. – 19. Januar 2018
Besprechung der H-Aufgaben in den Tutorien vom 22. – 26. Januar 2018

Aufgabe 47: (T) Leser-/Schreiberproblem mit Petrinetz

(– Pkt.)

Beim Leser-/Schreiberproblem operieren **n** Leserprozesse und **m** Schreiberprozesse auf ein und derselben Datei. Um Inkonsistenzen der Dateiinhalte zu vermeiden, müssen die folgenden beiden Bedingungen erfüllt sein:

- Mehrere Leserprozesse dürfen zur gleichen Zeit auf die Datei zugreifen.
- Ein Schreiberprozess darf nur dann auf die Datei zugreifen, wenn gerade kein anderer Leser- oder Schreiberprozess auf die Datei zugreift.

Bearbeiten Sie unter Berücksichtigung dieser beiden Bedingungen die folgenden Aufgaben

a. Modellieren Sie das Leser-/Schreiberproblem als ein Petrinetz. Gehen Sie dabei von der folgenden Situation aus:

   - Es gibt drei Leserprozesse und drei Schreiberprozesse, die auf ein und dieselbe Datei lesend bzw. schreibend zugreifen wollen. Es handelt sich dabei um disjunkte Prozesse.
   - Es können maximal zwei Leserprozesse gleichzeitig auf die Datei zugreifen.

   Hinweise:

   - Leserprozesse (Schreiberprozesse) können entweder auf die Datei lesend (schreibend) zugreifen, oder warten auf ihren Lesezugriff (Schreibzugriff). Überlegen Sie sich basierend darauf zunächst, welche Zustände Ihr Petrinetz modellieren muss.
   - Modellieren Sie die Leser- und Schreiberprozesse als Marken!
   - Zählen Sie unter Verwendung einer separaten Stelle in Ihrem Petrinetz mit, wieviele Leserprozesse bzw. Schreiberprozesse noch Zugriff auf die Datei erhalten dürfen.
   - Verwenden Sie an geeigneter Stelle gewichtete Transitionen.

b. Skizzieren Sie den Erreichbarkeitsgraphen für das in Aufgabe a) modellierte Petrinetz

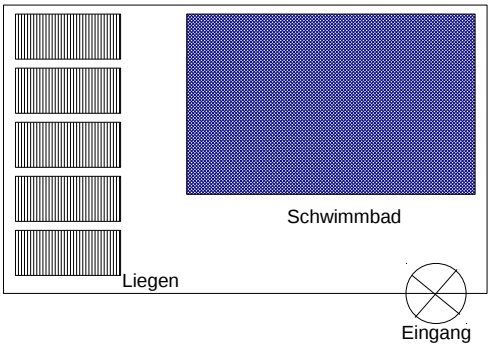
c. Handelt es sich bei Ihrer Modellierung aus Aufgabe a) um ein *faires* Petrinetz, d.h. können hier Prozesse verhungern? Begründen Sie Ihre Antwort! Für den Fall, dass Ihre Lösung kein faires Petrinetz darstellt: Wie könnte man ein Verhungern unterbinden?

Aufgabe 48: (T) Schwimmbad

(– Pkt.)

In dieser Aufgabe sollen Sie das Konzept der Semaphore am Beispiel eines Schwimmbads umsetzen. Dazu soll das Betreten und Verlassen eines Schwimmbads simuliert werden, welches `5` Liegen bereitstellt, die von den Badegästen genutzt werden können. Die Badegäste (welche hier als Prozesse angesehen werden können) können das Schwimmbad über ein Drehkreuz betreten bzw. verlassen, das zu jedem Zeitpunkt nur Platz für eine Person bietet. Daher kann zu einem bestimmten Zeitpunkt immer nur eine Person durch das Drehkreuz gehen. Es dürfen sich stets auch nur maximal soviele Personen im Schwimmbad befinden (inklusive einer Person im Drehkreuz), wie Liegen vorhanden sind.

Die nachfolgende Abbildung zeigt den schematischen Aufbau.



Beantworten Sie nun auf Grundlage dieses Szenarios folgende Aufgaben:

a. Welches klassische Problem aus der Informatik wird hier beschrieben?

b. Was sind die kritischen Bereiche bei diesem Problem?

c. Wieviele Semaphoren benötigt man um die Badegäste, die durch das Drehkreuz gehen wollen zu synchronisieren? Um welche Art von Semaphor handelt es sich jeweils?

d. Geben Sie in Pseudocode an, wie die benötigten Semaphore initialisiert werden müssen. Wählen Sie sinnvolle Bezeichner für Ihre Semaphore. Verwenden Sie folgende Notation für die Initialisierung:

Pseudocode	Beispiel	Bedeutung
`init(<semaphor>,<value>);`	`init(mutex,1);`	Initialisiert den Semaphor `mutex` mit dem Wert 1.

e. Vervollständigen Sie nun den folgenden Pseudocode, so dass dieser das Betreten bzw. Verlassen eines Badegastes simuliert und mehrere Gäste stets synchronisiert werden.

```
1  badegast() {
2      while(true) {
3          ...
4          <das Schwimmbad betreten>;
5          ...
6          <Liege belegen>;
7          ...
8          <das Schwimmbad verlassen>;
9          ...
10     }
11 }
```

Verwenden Sie für den Zugriff auf Ihre Semaphore folgende Notation:

Pseudocode	Beispiel	Bedeutung
`wait(<semaphor>);`	`wait(mutex);`	Erniedrigt den Wert des Semaphor `mutex` um eins
`signal(<semaphor>);`	`signal(mutex);`	Erhöhe den Wert des Semaphor `mutex` um eins

f. Das Schwimmbad will sein Image verbessern und familienfreundlicher werden. Deshalb bietet es nun auch spezielle Kinderliegen an, die auch nur von Kindern benutzt werden dürfen. Dazu werden zwei der fünf vorhandenen Liegen zu Kinderliegen verkleinert. Es gibt fortan also fünf Liegen wovon zwei nur von Kindern genutzt werden können. Kinder können natürlich auf allen Liegen PLatz nehmen. Erwachsene Badegäste hingegen sind für die Kinderliegen zu groß und dürfen nur auf großen Liegen Platz nehmen. Unabhängig davon kann aber immer nur eine Person eine Liege besetzen. Gehen Sie davon aus, dass eine Person stets über die boolesche Variable `is_Kinderliege` testen kann, ob es sich bei einer freien Liege um eine Kinderliege oder um eine große Liege handelt.

(i) Damit das Belegen von Liegen im Schwimmbad weiterhin synchronisiert erfolgen kann, benötigt man weitere Semaphore. Wählen Sie geeignete Bezeichner für die neuen Semaphore und initialisieren Sie diese in Analogie zu Aufgabe d).

(ii) Geben Sie in Analogie zum Pseudocode für Badegäste aus Aufgabe e) den Pseudocode für Kinder an, die ins Schwimmbad wollen. Nennen Sie die entsprechende Funktion `kind()`.

(iii) Da erwachsene Badegäste nun auf die Kinder Rücksicht nehmen müssen, ist es erforderlich, dass Sie ihren Pseudocode `badegast()` aus Aufgabe e) so anpassen, dass erwachsene Badegäste bzw. Kinder (d.h. die ausführenden Prozesse der Funktionen `badegast()` und `kind()`) synchronisiert werden.

Aufgabe 49: (H) Algorithmus von Peterson

(15 Pkt.)

Druckaufträge werden vom Betriebssystem in einer (FIFO-)Warteschlange `W` verwaltet. Die Warteschlange verwaltet selbst lediglich eine Liste von Zeigern, die auf den Speicherbereich verweisen, an dem die zu druckenden Daten liegen. Die Variable `next` enthält den Index der nächsten freien Position in der Warteschlange.
Gegeben seien zwei Prozesse P_1 und P_2, die jeweils eine Datei drucken möchten. Die folgende Tabelle illustriert die Programmausschnitte, die die Prozesse P_1 bzw. P_2 dazu jeweils ausführen.

Prozess $\mathbf{P_1}$

```
1 ...
2 W[next] = pointer_file1;
3 next = next + 1;
4 ...
```

Prozess $\mathbf{P_2}$

```
1 ...
2 W[next] = pointer_file2;
3 next = next + 1;
4 ...
```

a. Welches Problem kann auftreten, wenn P_1 und P_2 im Mehrprogrammbetrieb parallel ausgeführt werden? Modellieren Sie einen Ablauf, der dieses Problem illustriert. Verwenden Sie dazu folgende Darstellung:

aktiver Prozess	ausgeführte Codezeile	Inhalt von `W`	Wert von `next`	Kommentar
...	...	...	...	...

Stellen Sie den Inhalt von `W` als Liste der Form `[ptr_1, ptr_2, ...]` dar.

b. Synchronisieren Sie die Prozesse mit dem Algorithmus von Peterson. Geben Sie dazu (in Analogie zu den Code-Ausschnitten der Angabe) die Codeausschnitte der Prozesse P_1 und P_2 an.

c. Welchen erheblichen Nachteil hat der Peterson-Ansatz?

Aufgabe 50: (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.

<table>
<tr><td colspan="4">a) Wie bezeichnet man die Eigenschaft einer korrekten Lösung des wechselseitigen Ausschlusses, die besagt, dass sich zu jedem Zeitpunkt nur ein Prozess im kritischen Bereich befinden darf?</td></tr>
<tr><td>(i) Bounded Waiting</td><td>(ii) Progress</td><td>(iii) Mutual Exclusion</td><td>(iv) Circular Wait</td></tr>
<tr><td colspan="4">b) Ein Computer habe vier Bandlaufwerke und n Prozesse, von denen jeder zwei Bandlaufwerke gleichzeitig für seine Ausführung benötigt. Bei einer Anfrage bekommt ein Prozess ein beliebiges freies Bandlaufwerk zugewiesen. Für einen Prozess ist es irrelevant welche Bandlaufwerke er verwendet, solange es zwei sind. Nachdem er die zwei Bandlaufwerke erhalten hat, terminiert er nach endlicher Zeit. Für welchen Wert von n besteht die Möglichkeit eines Deadlocks?</td></tr>
<tr><td>(i) 1</td><td>(ii) 2</td><td>(iii) 3</td><td>(iv) 4</td></tr>
<tr><td colspan="4">c) Für eine korrekte Lösung des wechselseitigen Ausschlusses müssen die drei Bedingungen Mutual Exclusion, Progress, und Bounded Waiting erfüllt sein. Welche Bedingung(en) erfüllt der Algorithmus von Decker (erster Ansatz) nicht?</td></tr>
<tr><td>(i) Progress</td><td>(ii) Mutual Exclusion</td><td>(iii) Bounded Waiting</td><td>(iv) alle drei</td></tr>
<tr><td colspan="4">d) Für eine korrekte Lösung des wechselseitigen Ausschlusses müssen die drei Bedingungen Mutual Exclusion, Progress, und Bounded Waiting erfüllt sein. Was trifft auf den Algorithmus von Peterson zu?</td></tr>
<tr><td>(i) Er erfüllt keine der Bedingungen.</td><td>(ii) Er erfüllt alle Bedingungen.</td><td>(iii) Er erfüllt nur die Mutual Exclusion Bedingung.</td><td>(iv) Er erfüllt nur die Progress Bedingung.</td></tr>
<tr><td colspan="4">e) Was ist keine der 3 atomaren Operationen, mit denen ein Semaphor <code>S</code> verändert werden kann?</td></tr>
<tr><td>(i) <code>block(S)</code> oder auch <code>B(S)</code></td><td>(ii) <code>wait(S)</code> oder auch <code>P(S)</code></td><td>(iii) <code>init(S, Anfangswert)</code></td><td>(iv) <code>signal(S)</code> oder auch <code>V(S)</code></td></tr>
</table>