

Einführung in die Informatik: Systeme und Anwendungen
SS 2017

Übungsblatt 3: Semaphore

Abgabe bis **21.05.2017, 23:59 Uhr (VERLÄNGERT)**

Besprechung am 22./23.05.2017

Aufgabe 3-1 *Semaphore* (8 Punkte)
Hausaufgabe

Das Problem des *schlafenden Barbiers* ist ein bekanntes Beispiel für die Anwendung von Semaphoren zur Synchronisation von Prozessen. Der Barbier arbeitet in einem Laden mit einem Barbierstuhl und einem Warteraum, in dem Platz für vier Kunden vorhanden ist. Es gelten folgende Verhaltensregeln:

- Der Barbier verbringt seine Zeit mit Haare schneiden, Kassieren und Schlafen (=warten bis ein Kunde ihn durch das Setzen auf den Barbierstuhl weckt).
- Ein Kunde darf den Laden nicht betreten, wenn der Warteraum voll ist.
- Hat ein Kunde den Laden betreten, nimmt er im Warteraum Platz.
- Wenn der Barbier frei (bereit) ist, verlässt der Kunde den Warteraum und nimmt auf dem Barbierstuhl Platz (macht sich für den Barbier bereit).
- Der Barbier rasiert den auf dem Stuhl hockenden Kunden.
- Wenn ein Kunde bedient wurde, muss dieser den Barbier bezahlen.
- Wenn der Barbier bezahlt ist, kann er den nächsten Kunden bedienen.

Gehen Sie davon aus, dass sich zu Beginn kein Kunde im Laden befindet und der Barbier bereit ist.

Geben Sie Prozessbeschreibungen für den Barbier und die Kunden (jeder Kunde entspricht einer Instanz der gleichen Prozessbeschreibung) an. Die Synchronisation soll dabei über Binär- sowie Zählsemaphoren erfolgen. Folgende (globalen) Semaphorvariablen stehen zur Verfügung:

GLOBAL VARIABLES

```
warteraumFrei : Semaphore  
barbierBereit, kundeBereit, rasurFertig, zahlung: BinarySemaphore
```

Aufgabe 3-2 *Semaphore*
Hausaufgabe

(2+6 Punkte)

Betrachten Sie die folgende Variante des Erzeuger-Verbraucher-Problems. Gegeben seien ein Erzeuger E , der Produkte erzeugt und in eine Zwischenablage legt, sowie ein Verbraucher V , der Produkte aus der Zwischenablage nimmt und dann verbraucht. Die Kapazität der Zwischenablage ist auf ein Produkt beschränkt.

Der Erzeuger E führt über die Anzahl der erzeugten Produkte und die Anzahl der in der Zwischenablage abgelegten Produkte Buch. Beim Erzeugen wird die Integer-Variable *erzeugt* um 1 erhöht. Beim Ablegen wird die Integer-Variable *abgelegt* um 1 erhöht.

Der Verbraucher V führt über die Anzahl der aus der Zwischenablage genommenen Produkte und die Anzahl der verbrauchten Produkte Buch. Beim Nehmen wird die Integer-Variable *genommen* um 1 erhöht. Beim Verbrauchen wird die Integer-Variable *verbraucht* um 1 erhöht.

Erzeuger und Verbraucher seien durch folgende Prozessbeschreibungen gegeben. Die einzelnen Aktionen sind dabei zusätzlich mit einer Nummer versehen.

```
GLOBAL VARIABLES nichtVoll, nichtLeer: BinarySemaphore
GLOBAL OPERATION
  init(nichtLeer, 0);
  init(nichtVoll, 1);
```

| | |
|--|---|
| <pre>PROZESS E VARIABLES erzeugt, abgelegt: Int BEGIN erzeugt = 0; abgelegt = 0; WHILE true DO { (1) erzeugt = erzeugt + 1; (2) wait(nichtVoll); (3) abgelegt = abgelegt + 1; (4) signal(nichtLeer); } END</pre> | <pre>PROZESS V VARIABLES genommen, verbraucht : Int BEGIN genommen = 0; verbraucht = 0; WHILE true DO { (1) wait(nichtLeer); (2) genommen = genommen + 1; (3) signal(nichtVoll); (4) verbraucht = verbraucht + 1; } END</pre> |
|--|---|

- (a) In dieser Variante des Erzeuger-Verbraucher-Problems ist der Platz in der Zwischenablage auf ein Produkt beschränkt. Begründen Sie kurz, warum in diesem Fall allein durch die Semaphoren *nichtVoll* und *nichtLeer* der wechselseitige Ausschluss der Zugriffe auf die Zwischenablage gewährleistet ist.
- (b) Vervollständigen Sie die die Prozessablauftabelle auf dem Arbeitsblatt, so dass genau der angegebene Endzustand erreicht wird. Insbesondere sollen am Ende des Ablaufs 4 Produkte erzeugt, 3 Produkte abgelegt, 2 Produkte genommen und 1 Produkt verbraucht worden sein.

Hinweise: Sie benötigen nicht alle zur Verfügung stehenden Zeilen für eine richtige Lösung. In die Spalten *nichtVoll* und *nichtLeer* sind sowohl die Werte als auch die Warteschlangen der Semaphoren einzutragen (z.B. leere Warteschlange : [], Warteschlange in der die Prozesse A und B warten: $[A, B]$) Benutzen Sie eine neue Zeile für jede einzelne Aktion der Prozesse. Die Zustandsänderungen der Variablen sollen eindeutig nachvollziehbar sein. Die Menge *ready* enthält stets alle laufenden Prozesse, die nicht blockiert sind (vgl. Prozesszustände). Es gibt verschiedene alternative Lösungen.