Aufgabe K15: Semaphor mit Argumenten

Ein erweitertes Semaphor ist als Kernobjekt implementiert. Die Operationen zum Hoch- und Runterzählen erhalten jeweils ein Argument, um wieviel der Zählerstand verändert werden soll. Das Kernobjekt bietet folgende Operationen an:

`decrementSyn`: Runterzählen synchron. Argument ist eine positive Zahl R.

`decrementTry`: Runterzählen versuchend. Argument ist eine positive Zahl R. Rückgabewert ist entweder Erfolg oder Misserfolg.

`incrementAsyn`: Hochzählen asynchron. Argument ist eine positive Zahl H.

Die Implementierung des Kernobjekts verwendet folgende Attribute:

Z: Zählerstand, nicht negativ. Initial wählbar.

WR: Wartemenge für Runterzähler. Initial leer.

Aufrufe der `decrement`–Operationen werden reihenfolgetreu durchgeführt. Ein Aufrufer darf also keinen anderen, noch blockierten Aufrufer überholen.

a) Welche Strategie verwendet die Wartemenge WR? Welche zusätzlichen Funktionen muss WR unterstützen, gegenüber einem Semaphor ohne Argumente R und H?

WR verwendet eine Reihenfolgestrategie (FIFO, engl.: *first in — first out*), sonst könnten sich Aufrufer überholen. Für jeden wartenden Prozess muss auch die Zahl R gespeichert sein, um die er runterzählt. Die Elemente der Menge sind also nicht Prozesse P, sondern Tupel (P,R). Die Zahl R für den ersten Prozess muss abfragbar sein, ohne das Tupel aus der Menge zu nehmen.

b) Welches sind die ungültigen Fälle des Kernobjekts?

Es gibt nur einen ungültiger Fall:

- WR nicht leer und das erste R darin ist kleiner oder gleich Z.

Das heißt, der erste wartende Prozess könnte Z weit genug runterzählen.

c) Beschreiben Sie den Ablauf von `decrementSyn` mit Argument R.

Den Zählerstand zu prüfen reicht nicht. Der Aufrufer darf auch nicht überholen. Es ist nicht verlangt, den Wertebereich von R zu kontrollieren, aber guter Stil.

- falls R ≤ 0 : Abbruch mit Fehler
- WR leer und Z $\geq$ R ? +2

ja: – Z um R runterzählen +1

nein: – Aufrufer blockieren +0.3

– (Aufrufer,R) in WR legen +0.7

d) Beschreiben Sie den Ablauf von `incrementAsyn` mit Argument H.

H muss zum aktuellen Zählerstand addiert werden, um zu entscheiden, ob einer oder mehrere der wartenden Prozesse deblockieren.

- falls H ≤ 0 : Abbruch mit Fehler *guter Stil, siehe oben*
- Z um H hochzählen +1
- solange WR nicht leer und Z $\geq$ R aus WR: +2
  - (Prozess,R) aus WR nehmen +0.7
  - Z um R runterzählen +1
  - Prozess deblockieren +0.3

e) Beschreiben Sie den Ablauf von `decrementTry` mit Argument R.

Die Fallunterscheidung ist die gleiche wie bei `decrementSyn`. Hier wird aber in beiden Fällen ein Ergebnis zurückgegeben, entweder Erfolg oder Misserfolg.

- falls R ≤ 0 : Abbruch mit Fehler *guter Stil, siehe oben*
- WR leer und Z $\geq$ R ? +2

ja:
- Z um R runterzählen +1
- Erfolg zurückgeben +0.5

nein:
- Misserfolg zurückgeben +0.5

Achtung: Misserfolg ist *kein* Abbruch oder Fehler, sondern einer von zwei möglichen, korrekten Abläufen. Implementierungen verwenden vielleicht den gleichen Mechanismus, um Fehler und Misserfolg an den Aufrufer zurückzugeben. Zum Beispiel erlaubt C nur einen einzigen direkten Rückgabewert. Meist bedeutet dann 0 den fehlerfreien Erfolg. Bei anderen Werten muss aber der Aufrufer unterscheiden, ob es sich um den eingeplanten Misserfolg oder um einen tragischen Fehler handelt.

f) Statt der Reihenfolge soll eine andere Strategie zum Einsatz kommen, die Aufrufer mit niedrigem R bevorzugt. Wo und wie ändern sich dabei die Abläufe?

Bei `incrementAsyn` ändert sich nichts. Die `decrement`–Operationen ändert sich für den Fall, dass die Wartemenge nicht leer ist. Falls der Aufrufer beim Einfügen in die Wartemenge an deren Spitze gelangen würde, darf er trotz der Wartenden runterzählen, sofern der Zählerstand hoch genug ist. Diese Möglichkeit ist auch bei `decrementTry` zu prüfen, obwohl der Aufrufer dort niemals blockiert.

g) Statt der bisherigen soll eine Strategie zum Einsatz kommen, welche die Prioritäten der aufrufenden Prozesse berücksichtigt. Was ist zu bedenken?

Prioritäten können sich ändern, während Prozesse in der Wartemenge stecken. Wenn sich die Reihenfolge in der Wartemenge ändert, muss neu geprüft werden, ob nun Runterzähler deblockiert werden dürfen.