Institut für Informatik

Arbeitsgruppe Autonome Intelligente Systeme

Freiburg, 5. Dezember 2017

Systeme I: Betriebssysteme
Übungsblatt 6

Aufgabe 1 (1,5 Punkte)

Erweitern Sie die in der Vorlesung vorgestellte Softwarelösung zum wechselseitigen Ausschluss aus *Versuch 1b*, so dass der wechselseitige Ausschluss für n Prozesse garantiert ist. Geben Sie hierzu an, wie der i-te Prozess aussieht ($0 \leq i < n$).

Aufgabe 2 (2+1 Punkte)

In der Vorlesung wurden mehrere Lösungsversuche vorgestellt, mit denen eine Softwarelösung für den wechselseitigen Ausschluss gefunden werden sollte. In dieser Aufgabe geht es um den *Versuch 3*:

Initialisierung

```
1  flag[0] := false;
2  flag[1] := false;
```

Prozess 0

```
3  wiederhole
4  {
5    flag[0] := true;
6    solange (flag[1] = true)
7      tue nichts;
8
9    Anweisung 1
10   Anweisung 2      } kritische Region
11   ...
12
13   flag[0] := false;
14
15   Anweisung 3
16   Anweisung 4      } nichtkritische Region
17   ...
18 }
```

Prozess 1

```
wiederhole
{
  flag[1] := true;
  solange (flag[0] = true)
    tue nichts;

  Anweisung 5
  Anweisung 6      } kritische Region
  ...

  flag[1] := false;

  Anweisung 7
  Anweisung 8      } nichtkritische Region
  ...
}
```

Es ist sichergestellt, dass kein Prozess unendlich lange in seiner kritischen oder nichtkritischen Region bleibt.

a) Beweisen Sie, dass der wechselseitige Ausschluss auf die kritische Region garantiert ist, das heißt, dass zu keinem Zeitpunkt beide Prozesse gleichzeitig in der kritischen Region sein können. Achten Sie darauf, den Beweis vollständig aufzuschreiben.

Hinweis: Eine Möglichkeit zur Beweisführung besteht darin, den Wert der beiden Flags zu betrachten zu den Zeitpunkten, an denen die Prozesse zum letzten Mal die `solange`-Schleife verlassen haben, um zu zeigen, dass dies zu einem Widerspruch führt.

b) Erläutern Sie in eigenen Worten, was der grundlegende Nachteil ist, den alle reinen Softwarelösungen zum wechselseitigen Ausschluss aufweisen.

Aufgabe 3 (1+3+1 Punkte)

Sie möchten von zwei Prozessen aus auf eine kritische Region zugreifen und überlegen sich folgenden Code:

Initialisierung

```
1  variable := 1;
```

Prozess 0

```
2  wiederhole
3  {
4      wiederhole
5      {
6          variable--;
7          wenn (variable = 0) {
8              verlasse innere Schleife
9              (d.h. weiter mit Zeile 14)
10         }
11         variable++;
12     }
13
14     Anweisung 1   } kritische Region
15     Anweisung 2   }
16     ...           }
17
18     variable++;
19
20     Anweisung 3   } nichtkritische Region
21     Anweisung 4   }
22     ...           }
23 }
```

Prozess 1

```
wiederhole
{
    wiederhole
    {
        variable--;
        wenn (variable = 0) {
            verlasse innere Schleife
            (d.h. weiter mit Zeile 14)
        }
        variable++;
    }

    Anweisung 5   } kritische Region
    Anweisung 6   }
    ...           }

    variable++;

    Anweisung 7   } nichtkritische Region
    Anweisung 8   }
    ...           }
}
```

Wir nehmen *für diese Aufgabe* an, dass die Operationen „`++`" und „`--`" atomar sind und es sichergestellt ist, dass kein Prozess unendlich lange in seiner kritischen Region bleibt.

a) Begründen Sie, warum eine Iteration der äußeren Schleife eines Prozesses immer auf den

gleichen Wert von `variable` führt, sofern der andere Prozess `variable` währenddessen gleich oft inkrementiert und dekrementiert.

b) Beweisen Sie entweder formell, dass niemals beide Prozesse gleichzeitig in der kritischen Region sein können oder zeigen Sie anhand eines Gegenbeispiels, mit welcher Ablauffolge beide Prozesse gleichzeitig in die kritische Region gelangen können.

c) Sie möchten nun mehr als zwei Prozesse laufen lassen, welche alle den obigen Code verwenden, um auf die kritische Region zuzugreifen. Ist das im Hinblick auf den wechselseitigen Ausschluss grundsätzlich möglich? Welche Probleme könnten dabei auftreten? Begründen Sie Ihre Antwort. Ein formeller Beweis ist nicht notwendig.

Aufgabe 4 (1+1+1 Punkte)

Betrachten Sie folgenden Versuch für den wechselseitigen Ausschluss:

Initialisierung

```
turn    := 0;
flag[0] := false;
flag[1] := false;
```

Prozess 0

```
wiederhole
{
    flag[0] := true;
    solange (flag[1] = true) {
        wenn (turn = 1) {
            flag[0] := false;
            solange (turn = 1)
                tue nichts;
            flag[0] := true;
        }
    }

    Anweisung 1
    Anweisung 2      } kritische Region
    ...

    turn := 1;
    flag[0] := false;

    Anweisung 3
    Anweisung 4      } nichtkritische Region
    ...
}
```

Prozess 1

```
wiederhole
{
    flag[1] := true;
    solange (flag[0] = true) {
        wenn (turn = 0) {
            flag[1] := false;
            solange (turn = 0)
                tue nichts;
            flag[1] := true;
        }
    }

    Anweisung 5
    Anweisung 6      } kritische Region
    ...

    turn := 0;
    flag[1] := false;

    Anweisung 7
    Anweisung 8      } nichtkritische Region
    ...
}
```

Es ist sichergestellt, dass `turn`, `flag[0]` und `flag[1]` in den kritischen und nichtkritischen Regionen nicht geändert werden und dass jeder Prozess nur endlich lange in der kritischen Region bleibt.

Dieser Algorithmus garantiert, dass die beiden Prozesse niemals gleichzeitig in ihren kritischen Regionen sind. Das brauchen Sie an dieser Stelle nicht zu zeigen.

Prüfen Sie, ob die übrigen drei Anforderungen für das Problem der kritischen Region erfüllt sind und begründen Sie jeweils Ihre Antwort:

a) Es dürfen keine Annahmen über Geschwindigkeit und Anzahl der Rechenkerne gemacht werden.

b) Kein Prozess, der außerhalb seiner kritischen Region läuft, darf andere Prozesse blockieren.

c) Kein Prozess sollte unendlich lange darauf warten müssen, in seine kritische Region einzutreten.

Abgabe: Als PDF-Datei über Ilias bis 11. Dezember 2017, 23:59:00 Uhr.