Institut für Informatik

Prof. Dr. Christoph Scholl
Dr. Tim Welschehold
Alexander Konrad
Niklas Wetzel

Freiburg, 23.12.2022

Betriebssysteme
Musterlösung zu Übungsblatt 10

Wir wünschen Ihnen Frohe Weihnachten und einen erfolgreichen Start ins Neue Jahr!

Aufgabe 1 (3 Punkte)

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

Lösung:

Prozess i

```
1  wiederhole
2  {
3    solange (turn ≠ i)
4        tue nichts;
5    /* kritische Region */
6    turn := (i + 1)  mod  n;
7    /* nichtkritische Region */
8  }
```

Jeweils Punkte für:

- Zeile 3 korrekt. [1]
- Inkrementierung von `i` in Zeile 6. [1]
- Modulo in Zeile 6. [1]

Bei anderen Fehlern entsprechend Abzug.

Aufgabe 2 (4+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  while(1)
4  {
5    flag[0] = true;
6    while (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

```
while(1)
{
  flag[1] = true;
  while (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 seinem kritischen oder nichtkritischen Abschnitt bleibt.

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

Hinweis: Hinweis: Führen Sie ähnlich wie in der Vorlesung bei den Versuchen 1 und 5 einen Widerspruchsbeweis.

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

Lösung:

a) **Behauptung:** Der wechselseitige Ausschluss ist garantiert.

Beweis: Durch Widerspruch.

Annahme: Es gebe einen Zeitpunkt t, zu dem beide Prozesse im kritischen Abschnitt sind.

Sei t_i der Zeitpunkt, zu dem Prozess i zum letzten Mal die `solange`-Schleife verlassen hat. O.B.d.A. sei $t_0 < t_1$.

Zum Zeitpunkt t_0 muss `flag[0] = true` gewesen sein, da Prozess 0 das Flag vor der `solange`-Schleife auf `true` gesetzt hat und erst nach dem kritischen Abschnitt (also nach t) wieder auf `false` setzen wird. `flag[0]` wird sonst an keiner Stelle verändert. Da $t_0 < t_1 < t$ gilt, muss auch zum Zeitpunkt t_1 das Flag `flag[0] = true` sein. Dann kann Prozess 1 jedoch die `solange`-Schleife zum Zeitpunkt t_1 nicht verlassen haben.

Widerspruch: Dass Prozess 1 die `solange`-Schleife zum Zeitpunkt t_1 nicht verlassen hat, ist ein Widerspruch zur Definition von t_1.

Schlussfolgerung: Daraus folgt, dass die Annahme falsch sein muss, also ist der wechselseitige Ausschluss garantiert.

[4 Punkte]

b) Aktives Warten (busy waiting): Die Prozesse prüfen ständig den Wert bestimmter Flags, was unnötig Rechenzeit verbraucht [1 Punkt]. Besser wäre es, die Prozesse durch das Betriebssystem schlafen zu legen und erst wieder auszuführen, wenn sie in die kritische Region eintreten dürfen.

Aufgabe 3 (2+2+2 Punkte)

Betrachten Sie folgenden Versuch für den wechselseitigen Ausschluss:

Initialisierung

```
1  turn    = 0;
2  flag[0] = false;
3  flag[1] = false;
```

Prozess 0

```
4  while(1)
5  {
6      flag[0] = true;
7      while (flag[1] == true)
8      {
9          if (turn == 1)
10         {
11             flag[0] = false;
12             while (turn == 1)
13                 ; /*tue nichts*/
14             flag[0] = true;
15         }
16     }
17
18     Anweisung 1
19     Anweisung 2      } kritische Region
20     ...
21
22     turn = 1;
23     flag[0] = false;
24
25     Anweisung 3
26     Anweisung 4      } nichtkritische Region
27     ...
28 }
```

Prozess 1

```
while(1)
{
    flag[1] = true;
    while (flag[0] == true)
    {
        if (turn == 0)
        {
            flag[1] = false;
            while (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 Abschnitten nicht geändert werden und dass jeder Prozess nur endlich lange im kritischen Abschnitt bleibt. Dieser Algorithmus garantiert, dass die beiden Prozesse niemals gleichzeitig in ihren kritischen Abschnitten 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 (unter der Voraussetzung eines fairen Schedulers, der es vermeidet, einem Prozess unendlich lange keine Rechenzeit zuzuteilen):

2. Wenn ein Prozess in den kritischen Abschnitt will, so muss er nur endliche Zeit darauf warten.

4. Wenn kein Prozess im kritischen Abschnitt ist, so wird ein interessierter Prozess ohne Verzögerung akzeptiert.

5. Alles funktioniert unabhängig von der relativen Ausführungsgeschwindigkeit der Prozesse.

Lösung:

Keine Annahmen über Geschwindigkeit und Anzahl CPUs: Erfüllt. Der Algorithmus funktioniert unabhängig von der Ausführungsgeschwindigkeit der beiden Prozesse und der wechselseitige Ausschluss ist auch dann garantiert, wenn Befehle parallel auf mehreren CPUs ausgeführt werden. Insbesondere gibt es keine Konflikte beim Schreiben der Variablen, da jeder Prozess nur sein eigenes Flag schreibt und das Setzen von `turn` direkt nach der kritischen Region an einer Stelle stattfindet, an der sich nur ein Prozess gleichzeitig befinden kann. [2]

Nicht blockieren: Erfüllt. Will ein Prozess nicht in die kritische Region, so ist sein Flag `false` und der andere Prozess kann ungehindert in die kritische Region. Wollen beide Prozesse gleichzeitig in die kritische Region, so legt die Variable `turn` den Vorrang fest, sodass einer der Prozesse sein Flag zurückziehen muss und der andere Prozess ohne Verzögerung in die kritische Region eintreten kann. Nach der kritischen Region wird der Vorrang weitergegeben und das Flag gelöscht, sodass der andere Prozess ohne Verzögerung in den kritischen Abschnitt eintreten kann. [2] *Bemerkung:* Streng genommen können sich beide Prozesse in den Ein- und Austrittsbereichen zur kritischen Region (Zeile 6–14, 20–21) blockieren, da diese nicht zur kritischen Region gehören. Dies wird jedoch nicht berücksichtigt.

Nicht ewig warten: Erfüllt. Kein Prozess darf sein Flag unendlich lange auf `true` lassen: Prozesse, die auf die kritische Region warten und nicht an der Reihe sind, müssen das Flag in Zeile 6 auf `false` setzen und warten, bis sie an der Reihe sind. Prozesse, die nicht auf die kritische Region warten, müssen ihr Flag spätestens beim Verlassen der Schleife auf `false` setzen. Ist das Flag des anderen Prozesses auf `false`, so hat der Prozess freien ungehinderten Zugang zur kritischen Region. Nach dem Durchlaufen der kritischen Region muss der Prozess seinen Vorrang abgeben und kann sich den Vorrang niemals selbst geben, sodass der andere Prozess auf jeden Fall zum Zug kommt. [2]

Hier genügen schlüssige Begründungen, ein formeller Beweis war nicht verlangt.

Nice to know (aber nicht für die Vorlesung relevant): Der Algorithmus ist als „Dekker-Algorithmus“ bekannt und war der erste veröffentlichte Algorithmus, der das Problem des Wechselseitigen Ausschlusses für zwei Prozesse korrekt löste. Er wurde 1965 von Theodorus J. Dekker entdeckt und von Dijkstra veröffentlicht.

Aufgabe 4 (3+3 Punkte)

In der Vorlesung haben Sie den *Peterson-Algorithmus* für den wechselseitigen Ausschluss für zwei Prozesse kennengelernt. Folgend ist eine vermeintliche Erweiterung auf drei Prozesse aufgeführt.

Initialisierung

```
1 f[0] = false;
2 f[1] = false;
3 f[2] = false;
4 turn = 0;
```

Prozess 0

```
while(1)
{
  f[0] = true;
  turn = 0;
  while((f[1]==true || f[2]==true)
          && turn==0)
  {
      tue nichts;
  }

  Anweisung 1
  Anweisung 2      } kritische Region
  ...

  f[0] = false;

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

Prozess 1

```
while(1)
{
  f[1] = true;
  turn = 1;
  while((f[0]==true || f[2]==true)
          && turn==1)
  {
      tue nichts;
  }

  Anweisung 5
  Anweisung 6      } kritische Region
  ...

  f[1] = false;

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

Prozess 2

```
while(1)
{
  f[2] = true;
  turn = 2;
  while((f[0]==true || f[1]==true)
          && turn==2)
  {
      tue nichts;
  }

  Anweisung 9
  Anweisung 10     } kritische Region
  ...

  f[2] = false;

  Anweisung 11
  Anweisung 12     } nichtkritische Region
  ...
}
```

a) Ist bei der gezeigten Variante für drei Prozesse der wechselseitige Ausschluss gewährleistet? Begründen Sie Ihre Antwort.

b) Ist der wechselseitige Ausschluss garantiert, wenn man in der oben gezeigten Variante in den `while`-Schleifen bei den Vergleichen mit `turn` statt Gleichheit Ungleichheit fordert (für Prozess `i` `turn` $\neq$ `i`, also beispielsweise für Prozess 0: `turn` $\neq$ `0`)? Begründen Sie Ihre Antwort.

Lösung:

a) Der wechselseitige Ausschluss ist nicht garantiert. Angenommen, nach der Initialisierung wird Prozess 0 ausgeführt. Da die Flags `f[1]` und `f[2]` jeweils nicht gesetzt sind, kann P0 die kritische Region betreten. Nun muss P0 innerhalb dieser die CPU abgeben und Prozess 1 wird ausgeführt. Dieser Prozess bleibt zunächst in der `while`-Schleife. Wird nun Prozess 2 ausgeführt, so wird dieser ebenfalls zunächst in seiner `while`-Schleife verbleiben, durch `turn = 2` verlässt aber nun Prozess 1 seine `while`-Schleife und ist gemeinsam mit Prozess 0 im kritischen Bereich.

b) Der wechselseitige Ausschluss ist auch hier nicht garantiert. Es lässt sich ein ähnliches Gegenbeispiel konstruieren. Prozess 0 wird ausgeführt und führt nicht die `while`-Schleife aus, weil die Bedingung `turn!=0` nicht erfüllt ist. Prozess 0 geht in den kritischen Bereich. Wenn nun Prozess 1 ausgeführt wird, ist das Verhalten analog. Auch er geht nicht in die `while`-Schleife und kommt gleich in den kritischen Bereich. Somit sind jetzt Prozess 0 und Prozess 1 im kritischen Bereich.

Jeweils 1 Punkt für die richtige Begründungsidee (nicht für eine Ja/Nein-Antwort) und 2 Punkte für eine korrekte Begründung.

Abgabe: als PDF im Übungsportal bis 13.01.2023 um 12:00