Fabian Hupfeld (35523329) und Konstantin Stitz (35532242)

Übungsaufgaben 6

Aufgabe 1.)

Die Anforderungen an eine gute Lösung sind wie folgt definiert:

1. es darf höchstens ein Prozess gleichzeitig im KA sein

2. die Lösung muss für eine beliebige Anzahl von Prozessen funktionieren

3. andere Prozesse außerhalb des KA dürfen den Ablauf nicht blockieren

4. jeder Prozess muss irgendwann drankommen

a.)

Das Prozesssystem I erfüllt diese Anforderungen nicht. Zwar ist höchstens ein Prozess gleichzeitig im kritischen Abschnitt und man könnte das Verfahren für mehrere Prozessoren erweitern, jedoch kann es zu einem Deadlock kommen, wenn zB bei Prozess P1 an folgender markierter Stelle ein Prozesswechsel zu P2 erfolgt:

Prozess P_1

```
flag[2] = true;
while(flag[1]) {
  flag[2] = true;
  flag[2] = false; }
```

Prozesswechsel zu P2

```
printf("Process 1 inside
        criticalsection.");
```

```
flag[2] = false;
```

Wird P2 ausgeführt und der Prozess stürzt an der markierten Stelle ab, so kann das Programm nicht mehr weiterlaufen, da P1 nie in den KA wechseln kann, da es in der while-Schleife hängen bleibt:

Prozess P_2

```
flag[1] = true;
while(flag[2]) {
  flag[1] = true;
  flag[1] = false; }
```

Prozessabsturz

```
printf("Process 2 inside
        critical section.");
```

```
flag[1] = false;
```

Somit sind die Anforderungen an eine gute Lösung nicht erfüllt, auch, da es ohne Prozessabsturz durch ungünstige Prozesswechsel dazu kommen kann (zB wenn immer nach der 3. Zeile ein Prozesswechsel stattfindet), das nie ein Prozess in den KA eintreten kann (Deadlock). Auch die Bedingung der Fairness ist verletzt, weil nicht garantiert ist, dass jeder Abschnitt irgendwann eintreten kann. Es könnte so durch ungünstige Schaltvorgänge hier auch passieren, dass ein Prozess mehrmals ausgeführt wird und der andere gar nicht.

b.)

Das Prozesssystem II erfüllt die Anforderungen ebenfalls nicht, da es auch hier dazu kommen kann, dass Prozesse außerhalb eines KA blockiert werden, zB wenn nach Ablauf des KA von P1 der Prozess abstürzt wird var nicht auf 2 gesetzt und es kommt zu einem Deadlock.--

Aufgabe 2.)

a.) (siehe Quellcode)

Die Tasks laufen immer genau der Reihe nach ab, würde man einen Task stoppen, so würde sich das Programm an dieser Stelle aufhängen und nicht mehr weiterlaufen (Deadlock).

b.)

Nein, der Ansatz wird den gestellten Anforderungen nicht gerecht, da ein untätiger oder ausgefallener Prozess alle anderen Prozesse blockiert und damit gegen Punkt 3 der Anforderungen verstößt. Alle anderen Punkte dagegen sind erfüllt, denn durch die selectedTask wird sichergestellt, dass nur ein Prozess im KA sein kein. Außerdem kann eine beliebige Anzahl an Prozessen gewählt werden und durch die Zählvariable selectedTask wechseln sich die Prozesse fair ab.