Aufgabenblatt 7
Technische Informatik II

Tobias Petsch

Aufgabe 7.1

a)

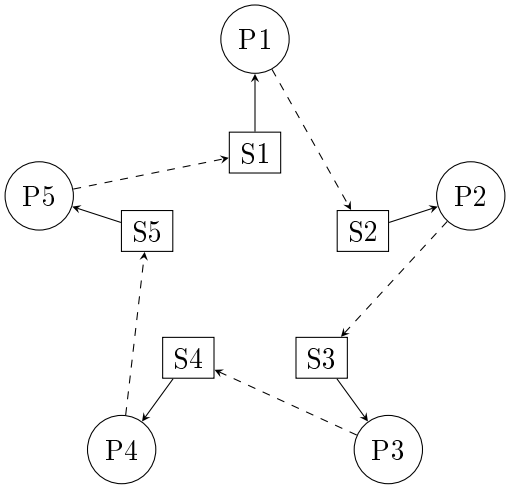
Die Parkplatz Simulation bricht ab, da durch eine Race Condition der sem−>value negativ wird und so ein Deadlock entsteht. Wenn mehrere Threads feststellen, dass sem−>value >= 0 ist dann werden sie den Wert verringern was zu einem negativen Wert führt.

Aufgabe 7.2

a)

Da jeder Philosoph erst sein linkes Stäbchen nimmt, kommt es zu Situationen bei denen das zweite Stäbchen vom Nachbarn genommen wird und das solange bis jeder Philosoph genau ein Stäbchen in der Hand hat, da er es nicht ablegen darf bis er gegessen hat, verhungert er.

b)



c)

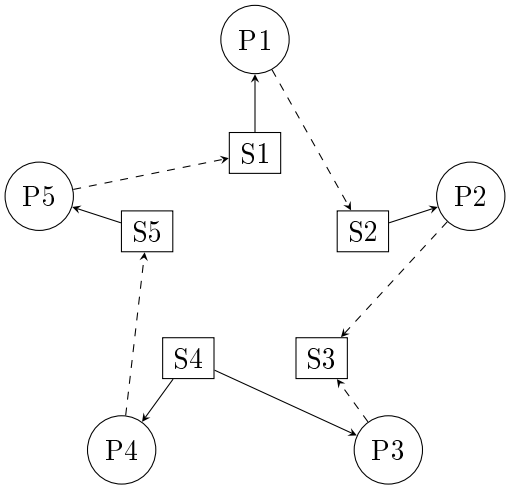
```
while true do
    wait until keiner der Nachbarn einen Hut trägt
```

```
setze lustigen Hut auf

warte, bis beide Stäbchen verfügbar sind (keine Konflikte mit Nachbarn)

nimm das linke Stäbchen
nimm das rechte Stäbchen

esse Reis, bis du satt bist

lege beide Stäbchen zurück auf den Tisch
setze den Hut ab
denke nach, bis du wieder hungrig bist
```

d)

Wenn zwei Philosophen abwechselnd immer essen, ohne dabei den mittleren Philosophen zu berücksichtigen, dann wird dieser verhungern da er nie zum Zug kommt.

e)



Es kann kein Deadlock mehr entstehen, da kein zyklischer Deadlock entstehen kann.

Aufgabe 7.3

Die `<stdatomic>` Bibliothek erlaubt es atomare Operationen auf Variablen auszuführen, die frei von Race Conditions sind. Sie erlaubt das sichere Schreiben und Lesen von Variablen mithilfe von `atomic_int` oder ähnlichen. Sie ist hilfreich für multithreaded Programme oder auch Zähler.

Aufgabe 7.4

a)

- Mutual exclusion: Ressourcen können immer nur von einem Prozess gleichzeitig genutzt werden.
- Hold and Wait: Ein Prozess hält bereits mindestens eine Ressource und wartet zusätzlich auf weitere, die von anderen Prozessen belegt sind.
- No Preemption: Ressourcen können einem Prozess nicht zwangsweise entzogen werden. Sie müssen freiwillig freigegeben werden.
- Circular Wait: Es existiert eine zyklische Wartekette, in der jeder Prozess auf eine Ressource wartet, die von einem anderen Prozess in der Kette gehalten werden.

b)

- **Mutual Exclusion vermeiden:**
  - Betriebsmittel werden, wenn möglich, so gestaltet, dass sie gleichzeitig von mehreren Prozessen genutzt werden können (z. B. durch Read-Only-Zugriffe).
  - Dadurch ist keine exklusive Nutzung erforderlich.
  - *Die Bedingung ist nie erfüllt, weil keine Ressource exklusiv vergeben wird.*
- **Hold and Wait vermeiden:**
  - Prozesse müssen vor Ausführung alle benötigten Ressourcen auf einmal anfordern.
  - Falls nicht alle verfügbar sind, wird der Prozess blockiert und bekommt keine.
  - *Die Bedingung ist nie erfüllt, da Prozesse keine Ressourcen halten, während sie auf andere warten.*
- **No Preemption vermeiden:**
  - Ressourcen können einem Prozess entzogen werden, wenn dieser auf weitere wartet.
  - Das Betriebssystem kann die Ressource zwangsweise freigeben und später neu zuteilen.
  - *Die Bedingung ist nie erfüllt, da Prozesse Ressourcen nicht dauerhaft behalten dürfen.*
- **Circular Wait vermeiden:**
  - Eine feste Reihenfolge (Total Order) für die Anforderung von Ressourcen wird definiert.
  - Prozesse dürfen nur in aufsteigender Reihenfolge nach Ressourcen fragen.
  - *Die Bedingung ist nie erfüllt, weil dadurch keine zyklischen Wartekanten entstehen können.*