Grundlagen der Praktischen Informatik **Sommersemester 2023**
Dr. Henrik Brosenne Georg-August-Universität Göttingen
Institut für Informatik

Übungsblatt 03

E-Learning

Absolvieren Sie die Tests bis Di., 16.05., 8 Uhr

Die Tests sind in der Stud.IP-Veranstaltung *Grundlagen der Praktischen Informatik (Informatik II)* unter *Lernmodule* hinterlegt.
Sie können einen Test **nur einmal durchlaufen**. Sobald Sie einen Test starten steht Ihnen nur eine **begrenzte Zeit** zu Verfügung, um den Test zu bearbeiten.
Alle Punkte, die Sie beim Test erreichen, werden ihnen angerechnet.

ILIAS – 40 Punkte

Betriebssysteme - Prozess-Synchronisation, Speicherverwaltung

Absolvieren Sie die folgenden Tests.

- *GdPI 03 - Betriebssysteme - Prozess-Synchronisation*
- *GdPI 03 - Betriebssysteme - Speicherverwaltung*

(40 Punkte)

Achtung

Zum ordnungsgemäßen Beenden eines Ilias-Test müssen Sie die Schaltfläche **Test beenden** betätigen.

Wenn Sie einen Ilias-Test einmal vollständig durchlaufen haben bekommen Sie auf die Seite *Testergebnisse*. Starten Sie den Test erneut aus Stud.IP, ist jetzt auch eine Schaltfläche *Testergebnisse anzeigen* vorhanden, die auf diese Seite führt.

Auf der Seite *Testergebnisse* können Sie sich unter *Übersicht der Testdurchläufe* zu jedem Testdurchlauf *Details anzeigen* lassen.

Prüfen Sie, insbesondere bei den *Markdown+AsciiMath*-Aufgaben, ob Ihre Lösung korrekt übermittelt wurde. Nur Lösungen, die hier angezeigt werden, können zur Korrektur in das Korrektursystem Grady übertragen werden.

Falls eine **Musterlösung** vorhanden ist, führt der Titel einer Aufgabe in der Auflistung der Aufgaben zur Musterlösung.

Hinweis

- Eine häufige Fehlerquellen ist das Schließen des Browser-Fensters vor **Test beenden**.
- Wenn Sie einen JavaScript Blocker einsetzen, sollten Sie für Ilias eine Ausnahme hinterlegen.

Übung

Abgabe bis Di., 16.05., 8 Uhr

Abgabe in Stud.IP

Aufgabe 1 muss digital in der Stud.IP-Veranstaltung **Ihrer Übungsgruppe** abgegeben werden.

Geben Sie Ihre Lösung (pdf, jpg, etc.) im **Vips-Modul** unter *Übung 03*, *Aufgabe 1* in der Stud.IP-Veranstaltung Ihrer Übungsgruppe ab.

Es gilt auch hier das übliche Verfahren.

- **Nur wer etwas abgibt kann Punkte bekommen.**
- Jedes Gruppenmitglied gibt mindestens die Liste der Gruppenmitglieder ab.
- Mindestens ein Gruppenmitglied gibt die vollständige Lösung der Aufgaben ab. Die Lösung wird **nicht** über mehrere Gruppenmitglieder verteilt abgegeben.
- **Eine** Abgabe wird von Ihrer Tutorin/Ihrem Tutor korrigiert, alle anderen Abgaben der Gruppe erhalten dieselbe Punktzahl.

Aufgabe 1 – 35 Punkte

Synchronisation

Betrachten Sie den Pseudo-Code der Prozesse I (Init), A (Alice), B (Bob).

I

```
1  global int i = 1;
2  global mutex a = false;
3  global mutex b = true;
```

A

```
4  while(i < 4) {
5    down(a)
6    i = i / 2 + 2
7    up(b)       }
```

B

```
8  while(i < 4) {
9    down(b)
10   i = i + 1
11   up(a)       }
```

Hinweis. Die Variablen `i`, `a` und `b` sind global, d.h. in jedem Prozess verfügbar.

Das System verwaltet die Prozesse, die darauf warten Rechnenzeit zu bekommen, nach dem Prinzip *First Come First Served* (FCFS).

Die Prozesse werden in der Reihenfolge I, A, B gestartet.

Vervollständigen Sie nachstehende Grafik.

- *Zeile* ist die Nummer der Codezeile, aus den Prozessen I, A oder B, die gerade durchlaufen wird.

  Hinweis. Die Zeilen der Prozesse sind fortlaufend nummeriert, damit genügt die Nummer der Codezeile zur Unterscheidung der Prozesse.

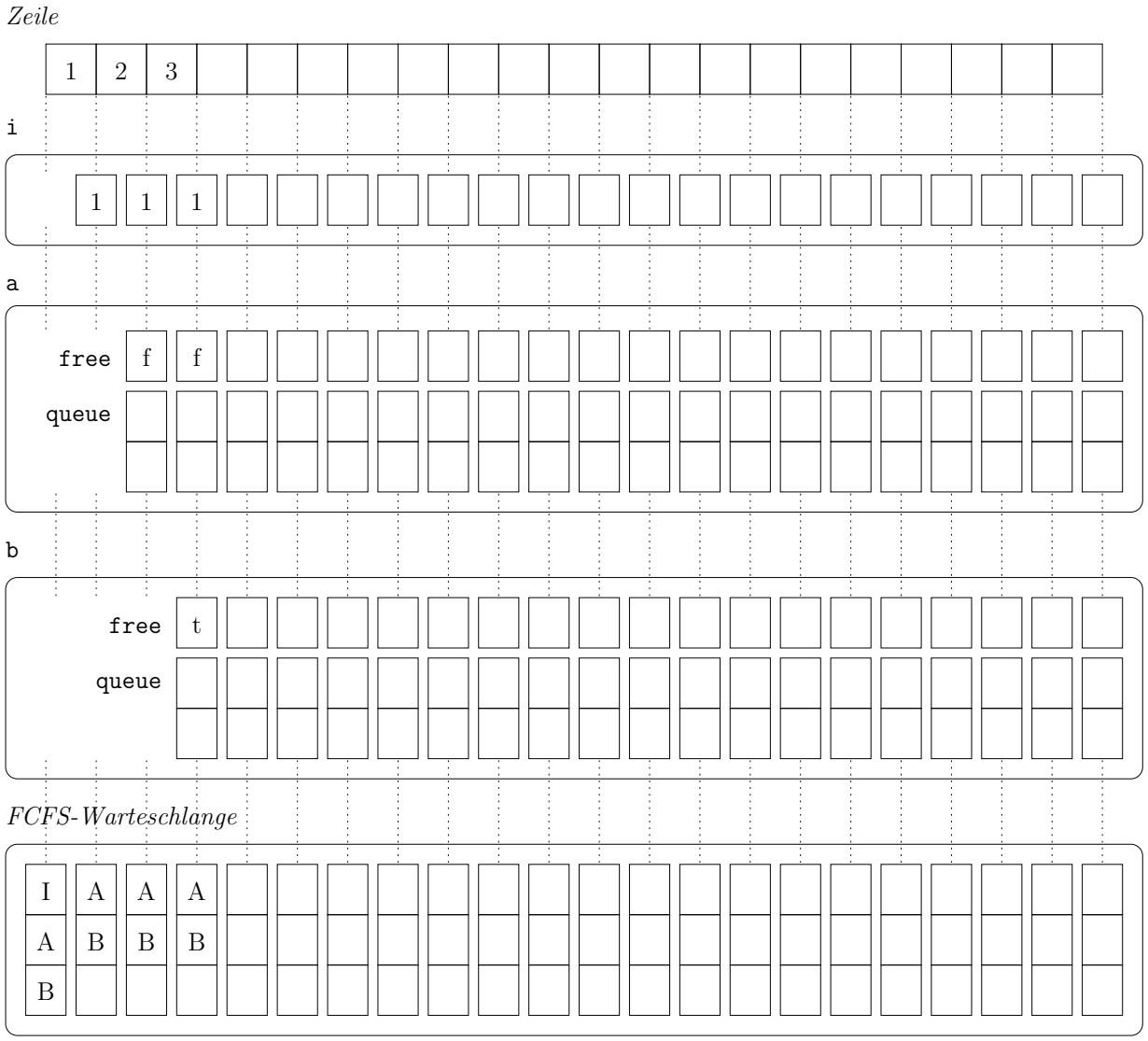
- Tragen Sie unter der Codezeilen-Nummer den Zustand der Variablen `i`, `a` und `b` ein, der sich aus dem Durchlaufen der Codezeile ergibt und aktualisieren Sie die *FCFS-Warteschlange*.

  Hinweise

  Anzahl der dargestellten Plätze in den Datenstrukturen ist so gewählt, dass genug Platz für die zu speichernden Daten vorhanden ist.

  Der Wert `true` kann mit t, der Wert `false` mit f abgekürzt werden.

(35 Punkte)



Hinweise

Die Grafik finden Sie noch einmal als `uebung03-grafik.pdf` in der Stud.IP-Veranstaltung *Grundlagen der Praktischen Informatik (Informatik II)* unter *Übung→uebung03-data*.

Wenn Sie die Grafik nicht ausdrucken und wieder einlesen, sondern am Rechner vervollständigen möchten, eignet sich ein *PDF Annotator*, z.B. Xournal (`http://xournal.sourceforge.net/`, *Ubuntu package* `xournal`). Alternativ kann das pdf auch in ein Bildbearbeitungsprogramm (z.B. Gimp) oder einen Vektorgrafik-Editor (z.B. LibreOffice Draw) importiert werden.

Praktische Übung

Abgabe der Prüfsumme bis Di., 16.05., 8 Uhr
Testat ab Fr., 25.05.

Hilfe zum Bearbeiten der praktischen Übungen können Sie grundsätzlich jeden Tag in den Rechnerübungen bekommen.

Am **Mo., 15.05.**, 10-12 Uhr online (4 Tutor*innen) und 18-20 Uhr in Präsenz (2 Tutor*innen) finden **keine Testate** statt. Diese Rechnerübungen sind ausschließlich **für Fragen** reserviert.

Abgabe der Prüfsumme

- Siehe vorherige Übungen.
- Übermitteln Sie die Prüfsumme mit dem Test *GdPI 03 - Testat*.

Aufgabe 1 – 25 Punkte

Prozess-Scheduling

Bilden Sie das Verhalten verschiedener Scheduling-Verfahren nach.

In der Vorlesung wurde der Typ `Prozess` mit Prozessbezeichner `pid`, Ankunftzeit `arrival` und Rechenzeit `computing` eingeführt. Prozesse sind über Ihre Rechenzeiten vergleichbar und können nach Rechenzeiten geordnet werden.

```
data Prozess = Prozess { pid       :: String
                       ,arrival   :: Int
                       ,computing :: Int } deriving (Show)

instance Eq Prozess where
    Prozess { computing = a } == Prozess { computing = b } = a == b

instance Ord Prozess where
    compare x y
        | computing x < computing y   = LT
        | computing x > computing y   = GT
        | otherwise                   = EQ
```

Verwenden Sie zum Signalsieren, dass der Prozessor mit keinem der Prozesse, denen Rechenzeit zugeteil werden soll, belegt ist, den Idle-Prozess.

```
let idle = Prozess{pid = "IDLE", arrival = -1, computing = -1}
```

Verwenden Sie den Typ `State`, mit den noch nicht angekommenen Prozessen `new`, dem rechnenden Prozess `run`, den wartenden Prozessen `ready`, der aktuellen Zeit `time` und einer Repräsentation des Schedules der Prozesse bis zur aktuellen Zeit `chart`, um den fortlaufenden Status des betrachteten Verfahrens zu beschreiben.

```
data State = State { new       :: [Prozess]
                   ,run       :: Prozess
                   ,ready     :: [Prozess]
                   ,time      :: Int
                   ,chart     :: String }
```

1. Die Standarddarstellung von `State`, die man mit `show` bekommen würde, wenn man `deriving (Show)` der Deklaration von `State` hinzufügt, ist sehr unübersichtlich.

   Sorgen Sie selbst für eine übersichtlichere Darstellung mit `show`.

   ```
   instance Show State where
       show s = ...
   ```

   (3 Punkte)

   Beispiel

   Scheduling-Beispiel aus der Vorlesung.

Prozesse	P_1	P_2	P_3	P_4	P_5	P_6
Ankunftszeit a_i	0	2	4	12	16	19
Rechenzeit t_i	6	6	5	4	3	6

   Umgesetzt in den initialen Status für das Scheduling.

   ```
   let ps = [Prozess{pid="P1", arrival=0, computing=6},...]
   let lectureExample = State { new=ps
                              ,run=idle
                              ,ready=[]
                              ,time=0
                              ,chart="" }
   ```

   Dann könnte eine übersichtlichere Ausgabe wie folgt aussehen.

   ```
   print lectureExample
   ```

   ```
   -- new
   Prozess {pid = "P1", arrival = 0, computing = 6}
   Prozess {pid = "P2", arrival = 2, computing = 6}
   Prozess {pid = "P3", arrival = 4, computing = 5}
   Prozess {pid = "P4", arrival = 12, computing = 4}
   Prozess {pid = "P5", arrival = 16, computing = 3}
   Prozess {pid = "P6", arrival = 19, computing = 6}
   -- run
   Prozess {pid = "IDLE", arrival = -1, computing = -1}
   -- ready
   -- time: 0
   -- chart:
   ```

   Hinweis

   Die Ausgabe ist vor allem für die Fehlersuche hilfreich.

2. Programmieren Sie zum Aktualisieren der wartenden Prozesse die folgende Funktion.

```
update_ready :: State -> State
```

   Alle noch nicht angekommene Prozesse, deren Ankunftszeit mit der aktuellen Zeit übereinstimmt, werden in die Liste der wartenden Prozesse verschoben.
   (4 Punkte)

   Hinweis

   Es ist sinnvoll ein Hilfsfunktion zu verwenden, die abhängig von einer übergebenen Zeit eine übergebene Prozessliste in ein Tupel von Prozesslisten aufteilt.

3. Programmieren Sie eine Funktion

```
update_run :: State -> State
```

   die, falls aktuell der Idle-Prozess gerechnet wird, den vordersten Prozess in der Liste der wartenden Prozesse zum rechnenden Prozess macht.
   (4 Punkte)

4. Programmieren Sie zum Aktualisieren des Status für den nächsten Zeitabschnitt die folgende Funktion.

```
update_time :: State -> State
```

   Die aktuelle Zeit wird um Eins erhöht, die Rechenzeit des rechnenden Prozesses um Eins verringert. In der Repräsentation des Schedules der Prozesse bis zur aktuellen Zeit (`chart`) wird die Prozess-ID des rechnenden Prozesses eingetragen. Ist der rechnende Prozess abgearbeitet, d.h. keine Rechenzeit mehr benötigt, wird der Idle-Prozess zum rechnenden Prozess.
   (4 Punkte)

5. Erzeugen Sie einen Status für das Beispiel aus Aufgabeteil 1. und wenden Sie die Funktionen `update_ready`, `update_run` und `update_time`, in dieser Reihenfolge, solange auf diesen Status an, bis alle Prozesse abgelaufen sind. Geben Sie die Repräsentation des Schedules der Prozesse für den resultierenden Status aus.
   (4 Punkte)

   Hinweis

   Diese Funktion bildet das Scheduling-Verfahren `First Come First Serve` (FCFS) nach.

6. Passen Sie die Funktion `update_ready` an, sodass die Scheduling-Verfahren *Shortest Job First (SJF)* und *Shortest Remaining Time First (SRTF)* simuliert werden (siehe nachstehend beide Verfahren).

   Wenden Sie analog zu Aufgabeteil 5 diese Scheduling-Verfahren auf das Beispiel aus Aufgabeteil 1. an.
   (6 Punkte)

   Hinweis

   Mit der Funktion `sort` können Listen sortiert werden, die Elemente eines Typs enthalten, der eine Instanz von `Ord` ist. Diese Funktion steht nach dem Import des Moduls `Data.List` zu Verfügung (`import Data.List`).

Shortest Job First (SJF)

- Nicht-unterbrechendes Scheduling.
- Es wird jeweils der Prozess mit der kürzesten Rechenzeit als nächstes auf dem Prozessor ausgeführt.

Shortest Remaining Time First (SRTF)

- Unterbrechendes Scheduling.
- Es wird jederzeit der Prozess auf dem Prozessor ausgeführt, der die geringste verbleibende Rechenzeit hat.
- Neu eintreffende Prozesse, die eine kürzere Rechenzeit haben, als der gerade bearbeitete Prozess, lösen den aktuell rechnenden Prozess ab.