Grundlagen der Praktischen Informatik **Sommersemester 2024**
Dr. Stefan Siemer Georg-August-Universität Göttingen
Timo Specht Institut für Informatik

Übungsblatt 02

E-Learning

Absolvieren Sie die Tests bis Di., 29.04., 14 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 – (30 Punkte)

Betriebssysteme - Prozess-Scheduling

Absolvieren Sie den Test *GdPI 02 - Betriebssysteme - Prozess-Scheduling.*
(30 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.

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., 29.04., 14 Uhr

Die Aufgaben müssen in **Dreiergruppen** abgegeben werden. Vierergruppen sind ebenfalls möglich.

Es ist **wichtig**, dass Sie sich an folgendes **Verfahren für die Abgabe** halten.
Die Lösungen werden in geeigneter Form in der Stud.IP-Veranstaltung Ihrer Übungsgruppe über das Vips-Modul hochgeladen. Sie müssen diese Abgaben nicht mit Markdown+AsciiMath erstellen. Sie können Ihre Bearbeitungen auch mit LaTeX formatieren, es ist aber auch die direkte Eingabe von Text oder der Upload von Text- und Bilddateien in gängigen Formaten möglich.

Weitere Hinweise zur Abgabe der Lösungen finden Sie in den Aufgabenstellungen.

Vorbereitung - Rechenintensive Prozesse und Scheduling

Allgemeine Begriffe

- Ein Prozess wird **aktiviert**, wenn er Prozessorzeit zugeteilt bekommt, d.h. der Zustand des Prozesses wechselt von bereit nach rechnend.
- Die **Ankunftszeit** eines Prozesses ist der Zeitpunkt, ab dem der Prozess vom Scheduling berücksichtigt wird. Der Prozess wird erzeugt, ist rechenbereit und wird der Menge der Prozesse mit Zustand bereit hinzugefügt. Ist dieser Prozess der einzige in der Menge der Prozesse mit Zustand bereit, wird der Prozess sofort aktiviert.

  Beispiel. Kommt der Prozess P zum Zeitpunkt t an, ist die Bereitliste leer und kein Prozess belegt den Prozessor, dann wird der Prozess P zum Zeitpunkt t aktiviert.
- Die **Endzeit** ist der Zeitpunkt an dem ein Prozess vollständig abgelaufen ist.
- Die **Wartezeit** eines Prozesses ist die Zeit zwischen Ankunfts- und Endzeitpunkt, während der der Prozess keine Prozessorzeit zugeteilt bekommen hat.

Nicht-unterbrechendes Scheduling

- Ein Prozess läuft nach der Aktivierung vollständig ab ohne zu blockieren.
- Kommt eine neuer Prozess an, wird er in die Menge der bereiten Prozesse eingeordnet.
- Beendet sich der aktuell rechnenden Prozesses, muss ein Prozess, aus der Menge der bereiten Prozesse, ausgesucht werden, der aktiviert wird. Ist die Menge leer, läuft der Prozessor (quasi im Leerlauf) weiter, bis wieder ein Prozess in der Menge der bereiten Prozess verfügbar ist.

Unterbrechendes Scheduling

- Der aktuell rechnende Prozess läuft ohne zu blockieren solange bis er unterbrochen wird oder vollständig abgelaufen ist, d.h. sich beendet.
- Kommt eine neuer Prozess an, wird er in die Menge der bereiten Prozesse eingeordnet. Weiterhin wird der aktuell rechnende Prozess unterbrochen und ebenfalls in die Menge der bereiten Prozesse eingeordnet, dabei wird die noch verbleibende Rechenzeit dem Prozess als Rechenzeit zugeordnet.
- Kommt ein neuer Prozess an oder beendet sich der aktuell rechnenden Prozesses, muss ein Prozess, aus der Menge der bereiten Prozesse, ausgesucht werden, der aktiviert wird. Ist die Menge leer, läuft der Prozessor (quasi im Leerlauf) weiter, bis wieder ein Prozess in der Menge der bereiten Prozess verfügbar ist.

Aufgabe 1 – (24 Punkte)

Auf der Suche nach einem guten Scheduling-Verfahren für Großrechner, auf den hauptsächlich rechenintensive Prozesse ablaufen, wird folgendes Beispiel betrachtet.

Prozesse	P_1	P_2	P_3	P_4	P_5	P_6
Ankunftszeit	0	4000	5000	39000	42000	43000
Rechenzeit	15000	20000	5000	50000	25000	10000

1. Lassen Sie die Prozesse $P_1, \ldots, P_6$ mit nicht-unterbrechenden und unterbrechenden Scheduling ablaufen. Stellen Sie jeweils, wenn ein Prozess aktiviert wird, den rechnenden Prozess und die Menge der bereiten Prozesse dar.

   Wählen Sie jeweils den zu aktivierenden Prozess, sodass die Wartezeit für die Prozesse in der Menge der bereiten Prozesse möglichst klein wird.

   Bestimmen Sie die durchschnittliche Wartezeit pro Prozess.
   (14 Punkte)

2. Leiten Sie aus dem Beispiel für nicht-unterbrechendes und unterbrechendes Scheduling ein allgemeines Kriterium für die Wahl des jeweils zu aktivierende Prozesses ab, sodass die durchschnittliche Wartezeit möglichst klein wird.
   (10 Punkte)

Aufgabe 2 – (21 Punkte)

Round-Robin Scheduler verwalten normalerweise eine Liste der Prozesse, die auf Zuteilung von Prozessorzeit warten, in der jeder Prozess genau einmal aufgeführt wird.

1. Beschreiben Sie ausfühlich was passieren, wenn Prozesse mehrfach in der Liste stehen?
   (10 Punkte)

2. Aus welchen Gründen könnte man das mehrfache Eintragen von Prozessen in die Liste erlauben?
   (6 Punkte)

3. Welche Probleme ergeben sich aus Mehrfacheintragungen desselben Prozesses?
   (5 Punkte)

   Hinweis. Was passiert wenn der Prozess blockiert?

Praktische Übung

Abgabe der Prüfsumme Di., 29.04., 14 Uhr

Testat Di., 29.04., ab 18 Uhr

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

Abgabe der Prüfsumme

- Erstellen Sie ein Archiv, dass **alle Dateien** enthält, die Sie beim Testat vorstellen möchten.
- Beim Testat werden nur Dateien aus einem Archiv testiert, dessen Prüfsumme **exakt** der von Ihnen übermittelten Prüfsumme entspricht.
- Berechnen Sie die Prüfsumme des Archivs mit dem `sha1sum` Befehl.
- Übermitteln Sie die Prüfsumme mit dem Test *GdPI 02 - Testat*, der in Stud.IP unter *Grundlagen der Praktischen Informatik (Informatik II)* → *Lernmodule* hinterlegt ist.

Testat

- Um die praktische Übung testieren zu lassen, **müssen** Sie einen Termin über die Stud.IP-Terminvergabe für ein Testat reservieren. Ein Testat ohne Termin ist nicht möglich.
- Ab Übung 02 finden die Testate ebenfalls in **Dreiergruppen** und Vierergruppen statt. Dabei sind die Gruppen identisch zu denen, die auch die theoretischen Aufgaben zusammen bearbeiten. In diesem Fall reserviert nur ein Gruppenmitglied einen Termin.
- Bringen Sie bzw. jedes Mitglied Ihrer Gruppe zu jedem Testat den unten angehängten Kontrollzettel **Praktische Übung - Testate** mit. Ein Einspruch zu nicht oder falsch eingetragenen Punkten erfordert die Vorlage des Kontrollzettels.

Vorbereitung

Codierung

Sei $x = (x_{n-1} \dots x_0)$ mit $x_i \in \{0, 1\}$ eine Bit-Folge der Länge n.

Als Binärzahl in Dualcodierung stellt x die Dezimalzahl $d(x)$ und in Zweierkomplement-Codierung die Dezimalzahl $z(x)$ dar, wobei folgendes gilt.

$$d(x) = \sum_{i=0}^{n-1} x_i \cdot 2^i$$

$$z(x) = -\Big(x_{n-1} \cdot 2^{n-1}\Big) + \sum_{i=0}^{n-2} x_i \cdot 2^i$$

1-Bit-Addierer

Ein 1-Bit-Addierer fast 3 Bits $(x), (y), (z)$ (Bit-Folgen der Länge 1) zu einer Bit-Folge $(c\,s) = (x) + (y) + (z)$ der Länge 2 zusammen, wobei folgendes gilt.

$$\begin{aligned} s' &= x \text{ XOR } y & \qquad c' &= x \text{ AND } y \\ \mathbf{s} &= s' \text{ XOR } z & \qquad c'' &= s' \text{ AND } z \\ \mathbf{c} &= c' \text{ OR } c'' \end{aligned}$$

Hinweis. c' und c'' können nicht gleichzeitig 1 sein

n-Bit-Addierer

Ein n-Bit-Addierer addiert zwei Bit-Folgen $x = (x_{n-1} \dots x_0)$ und $y = (y_{n-1} \dots y_0)$ der Länge n zu einer Bit-Folge $(c\,s_{n-1} \dots s_0) = x + y$ der Länge $(n+1)$, wobei folgendes gilt.

$$\begin{aligned} (c_1\,\mathbf{s_0}) &= x_0 + y_0 + 0 \\ (c_2\,\mathbf{s_1}) &= x_1 + y_1 + c_1 \\ &\vdots \\ (c_{n-1}\,\mathbf{s_{n-2}}) &= x_{n-2} + y_{n-2} + c_{n-2} \\ (\mathbf{c}\,\mathbf{s_{n-1}}) &= x_{n-1} + y_{n-1} + c_{n-1} \end{aligned}$$

Hinweis

In der Regel wird bei n-Bit-Addieren das Ergebnis nur in n Bit zurückgeliefert, damit sind nicht alle Ergebnisse korrekt. Es kommt dann zu einem sogenannten **Überlauf**, der über das Bit c abgeprüft werden kann.

Aufgabe 2 – (25 Punkte)

Bit-Folgen

In der Vorlesung wurde der Typ `Nibble`

```
type Nibble = (Bool, Bool, Bool, Bool)
```

eingeführt.

1. Programmieren Sie eine Funktion

   ```
   showNibble :: Nibble -> String
   ```

   die den `Nibble` auf einen `String` abbildet, der folgendes enthält.

   - Die durch den `Nibble` repräsentierte Bitfolge (0 = `False` und 1 = `True`).
   - Die Dezimalzahl, die durch interpretieren des `Nibble` als Binärzahl in Dualcodierung entsteht.
   - Die Dezimalzahl, die durch interpretieren des `Nibble` als Binärzahl in Zweierkomplement-Codierung entsteht.

   Die Funktion sollte wie im folgenden Beispiel funktionieren.

   ```
   > showNibble (True, False, False, True)
   "1001  9  -7"
   ```

   (4 Punkte)

   Hinweis. Verwenden Sie passenden Hilfsfunktionen.

2. Programmieren Sie eine Funktion

   ```
   bitAdder :: Bool -> Bool -> Bool -> (Bool, Bool)
   ```

   die einen 1-Bit-Addierer realisiert. Verwenden Sie nur die Funktionen/Operatoren `not`, `&&`, ||, `and` und `or` oder daraus konstruierte Hilfsfunktionen/-operatoren.
   (3 Punkte)

3. Programmieren Sie eine Funktion

   ```
   nibbleAdder :: Nibble -> Nibble -> (Bool, Nibble)
   ```

   die einen 4-Bit-Addierer realsiert. Verwenden Sie nur die Funktionen/Operatoren `bitAdder`, `not`, `&&`, ||, `and` und `or` oder daraus konstruierte Hilfsfunktionen/-operatoren.
   (10 Punkte)

   Hinweis. Das Tupel `(Bool, Nibble)` repäsentiert die 5 Bit ($c\ z_3 \dots\ z_0$), die für das Ergebnis der Addition von zwei Nibble ($x_3 \dots\ x_0$) und ($y_3 \dots\ y_0$) benötigt werden.

4. Programmieren Sie eine Testfunktion

```
tableAdder :: (Nibble -> Nibble -> (Bool, Nibble))
              -> [(Nibble, Nibble)] -> String
```

die die übergebene Funktion auf die in der Liste übergebenen Eingaben anwendet und sowohl die Eingabe, als auch den Funktionwert mit der Funktion `showNibble` aus Aufgabeteil 1 zeilenweise ausgibt.

Ein Test soll wie im folgenden Beispiel funktionieren.

```
> putStrLn ( tableAdder nibbleAdder
             [((True , False, False, True),
               (False, False, False, True))] )
1001  9  -7 + 0001  1  1  = False  1010  10  -6
```

Die Ausgabe wird wie folgt erzeugt.

- Nibble `(True, False, False, True)`
  - als Bit-Folge (`1001`)
  - Dezimalzahl in Dualcodierung (`9`)
  - Dezimalzahl in Zweikomplement-Codierung (`-7`)
- Trennzeichen (`+`)
- Nibble `(False, False, False, True)`
  - als Bit-Folge (`0001`)
  - Dezimalzahl in Dualcodierung (`1`)
  - Dezimalzahl in Zweikomplement-Codierung (`1`)
- Trennzeichen (`=`)
- Tupel `(Bool, Nibble)` als Funktionswert von `nibbleAdder`
  - Überlauf? Bool (`False`)
  - Nibble `(True, False, True, False)`
    - als Bit-Folge (`1010`)
    - Dezimalzahl in Dualcodierung (`10`)
    - Dezimalzahl in Zweikomplement-Codierung (`-6`)

(8 Punkte)

Hinweis.

Falls Sie Funktion `showNibble` nicht implementiert haben, ist die Ausgabe der `Nibble` selbst ausreichend.

Praktische Übung - Testate

Bringen Sie einen Ausdruck dieser Seite zu jedem Testat mit.

Die Punkte, die Sie in den Testaten erreichen, werden elektronisch erfasst. Dieser Kontrollzettel gibt Ihnen einen zusätzlichen Überblick über die von Ihnen in den praktischen Übungen erarbeiteten Punkte.

Studentin/Student

Name, Vorname	
Matrikelnummer	
Stud.IP-Benutzername	

Testate

Übung	Punkte	Tutor	Unterschrift
01			
02			
03			
04			
05			
06			
07			
08			
09			
10			
11			
12			