

Prüfung Verteilte Softwaresysteme

Datum	:	29.01.2019, 12:30 Uhr
Bearbeitungszeit	:	90 Minuten
Prüfer	:	Prof. Dr. Oliver Braun
Hilfsmittel	:	Keine
Erreichbare Punkte	:	80

Name: _____

Vorname: _____

Matrikelnummer: _____ Studiengruppe: _____

Hörsaal: _____ Platz Nr.: _____

Unterschrift: _____

Bitte kontrollieren Sie, ob Sie eine vollständige Angabe mit 4 Aufgaben auf 10 Seiten erhalten haben.

Aufgabe	1	2	3	4	Summe
max. Punkte	20	20	20	20	80

Anmerkungen:

- Schreiben Sie die Lösungen in die dafür vorgesehenen Kästchen sofern welche vorhanden sind. Sollte Ihnen der Platz dabei nicht reichen, benutzen Sie die Rückseite **und vermerken Sie das im dazugehörigen Kästchen!**
- Sollte es bei der entsprechenden Aufgabe explizit vermerkt sein, nutzen Sie das zusätzlich ausgeteilte Klausurpapier für Ihre Antworten. **Schreiben Sie dann unbedingt Ihren Namen auf jeden Klausurbogen.**

Aufgabe 1 (20 Punkte)

- (a) Grenzen Sie die Begriffe *Netzwerkbetriebssystem* und *verteiltes Betriebssystem* voneinander ab. (4)

- (b) Nennen Sie zwei Transparenzeigenschaften, welche die Verteilung verbergen und erklären Sie sie kurz. (4)

- (c) Was ist statische Lastverteilung? Erklären Sie einen Ansatz wie statische Lastverteilung umgesetzt werden kann. (4)

- (d) Grenzen Sie die *parallele Verarbeitung* und *nebenläufige Verarbeitung* voneinander ab. (4)

- (e) Nennen Sie zwei Probleme die durch die Offenheit eines verteilten Systems entstehen können und erläutern Sie diese kurz. (4)

Aufgabe 2 (20 Punkte)

(a) Erläutern Sie kurz die Kernidee eines Actor-Systems.

(4)

(b) Welche Schritte laufen beim Empfangen einer Nachricht in einem Actorsystem ab.

(4)

(c) Gegeben sei folgender Proto.Actor-Code unter Nutzung des Proto.Actor-Frameworks.

(12)

Eine Actor-Struct:

```
type Actor struct {  
    id      int  
    value   int  
    actors []*actor.PID  
}
```

Die Nachrichten:

```
type Actors struct{ Actors []*actor.PID }
type Count struct{}
type Switch struct{}
```

Die Methoden zum Verarbeiten der Nachrichten:

```
func (state *Actor) receiveDefault(context actor.Context) {
    switch msg := context.Message().(type) {
    case Actors:
        state.actors = msg.actors
    }
}
```

```
func (state *Actor) Receive(context actor.Context) {
    switch context.Message().(type) {
    case Count:
        state.inc(context)
    case Switch:
        context.PushBehavior(state.ReceiveDecrement)
    default:
        state.receiveDefault(context)
    }
}
```

```
func (state *Actor) inc(context actor.Context) {
    state.value++
    if state.value >= len(state.actors) {
        context.Self().Tell(Switch{})
        context.Self().Tell(Count{})
    } else {
        state.actors[state.value].Tell(Count{})
    }
}
```

```
func (state *Actor) ReceiveDecrement(context actor.Context) {
    switch context.Message().(type) {
    case Count:
        state.dec(context)
    case Switch:
        context.PopBehavior()
    default:
        state.receiveDefault(context)
    }
}
```

```
func (state *Actor) dec(context actor.Context) {
    state.value--
}
```

```

    if state.value <= 0 {
        context.Self().Tell(Switch{})
    } else {
        state.actors[state.value].Tell(Count{})
    }
}

```

Die main-Funktion sieht wie folgt aus:

```

func main() {
    actors := make([]*actor.PID, 4)
    for i := 0; i < 4; i++ {
        props := actor.FromProducer(func() actor.Actor {
            return &Actor{i, 0, nil}
        })
        actors[i] = actor.Spawn(props)
    }
    for _, actor := range actors {
        actor.Tell(Actors{Actors: actors})
    }
    actors[0].Tell(Count{})
    if _, err := console.ReadLine(); err != nil {
        panic(err)
    }
}

```

Geben Sie für alle Actors alle Nachrichten mit Absendern an, in der Reihenfolge in der sie empfangen werden. Schreiben Sie hinter jede Nachricht den Wert des `value`-Feldes in der Actor-Struct **nach** dem Behandeln der Nachricht. Nutzen Sie für die Bezeichnung der Actors den Wert des `id`-Feldes. Also z.B.

Actor 5:

```

...
Count{} von Actor 12, Value = 17
...

```

Nutzen Sie für Ihre Lösung das zusätzlich ausgeteilte Klausurpapier.

Aufgabe 3 (20 Punkte)

- (a) Bei der Bewertung von parallelen Programmen spielt die Laufzeit $T_p(n)$ eine wesentliche Rolle. Geben Sie die sechs Zeiten an, aus denen sie sich zusammensetzt und erläutern Sie diese kurz. (6)

A large empty rectangular box with a thin black border, intended for the student to write their answer to question (a).

- (b) Was ist die Effizienz eines parallelen Programms, wie wird sie berechnet und welche Werte nimmt sie im Normalfall an? (5)

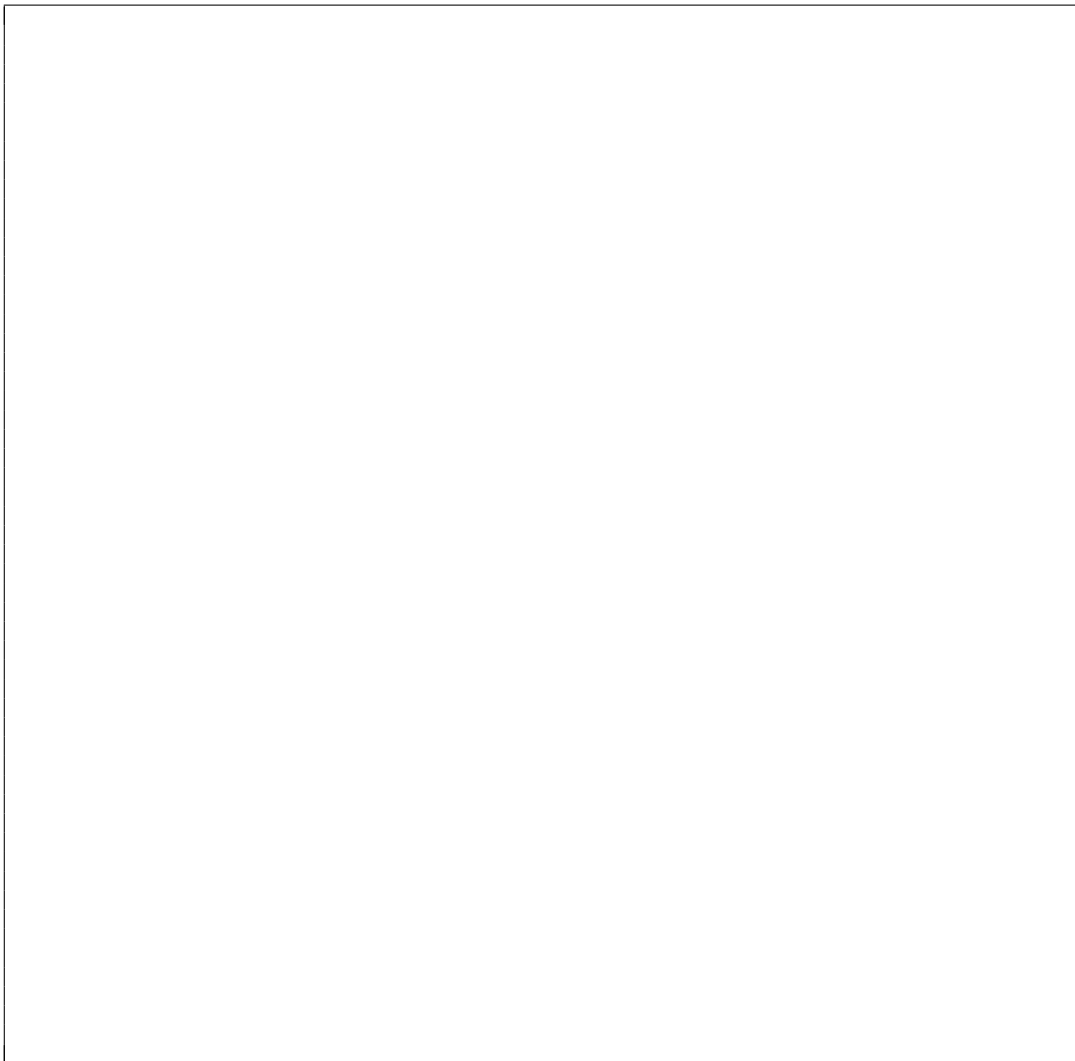
A large empty rectangular box with a thin black border, intended for the student to write their answer to question (b).

- (c) Gegeben sei folgender, allgemein verständlicher, Go-Code zur Multiplikation zweier, quadratischer Matrizen a und b: (9)

```
for i := 0; i < size; i++ {
    for j := 0; j < size; j++ {
        value := 0
        for k := 0; k < size; k++ {
            value += a[i][k] * b[k][j]
        }
        res[i][j] = value
    }
}
```

Wie könnten Sie diesen Code *sinnvoll* parallelisieren?

Beschreiben Sie Ihr Vorgehen, schreiben Sie keinen Code. Wieviele Prozesse würden Sie in Abhängigkeit von der Problemgröße $n = size \cdot size$ verwenden um sie möglichst gleichmäßig auszulasten? Was müssten Sie den einzelnen Prozesse als Parameter geben und was müsste ein Master-Prozess dann noch machen? Wäre ein gemeinsamer Speicher hilfreich?



Aufgabe 4 (20 Punkte)

- (a) Gegeben sind die 4 Prozesse P_1, P_2, P_3 und P_4 die mit Hilfe der Lamportzeit synchronisiert werden sollen. Alle Uhren stehen zu Beginn, also **vor** dem ersten Ereignis auf 0. Die 4 Prozesse verarbeiten die folgenden Ereignisse in der angegebenen Reihenfolge: (11)

Prozess P_1

- Senden der Nachricht N_1
- Bearbeitungsschritt
- Empfangen der Nachricht N_6
- Bearbeitungsschritt
- Empfangen der Nachricht N_8

Prozess P_2

- Empfangen der Nachricht N_3
- Empfangen der Nachricht N_4
- Bearbeitungsschritt
- Empfangen der Nachricht N_7
- Bearbeitungsschritt
- Senden der Nachricht N_2

Prozess P_3

- Empfangen der Nachricht N_{10}
- Senden der Nachricht N_3
- Empfangen der Nachricht N_1
- Senden der Nachricht N_4
- Senden der Nachricht N_5
- Senden der Nachricht N_6
- Empfangen der Nachricht N_{11}
- Bearbeitungsschritt
- Senden der Nachricht N_7
- Bearbeitungsschritt
- Bearbeitungsschritt
- Bearbeitungsschritt
- Senden der Nachricht N_8
- Senden der Nachricht N_9

Prozess P_4

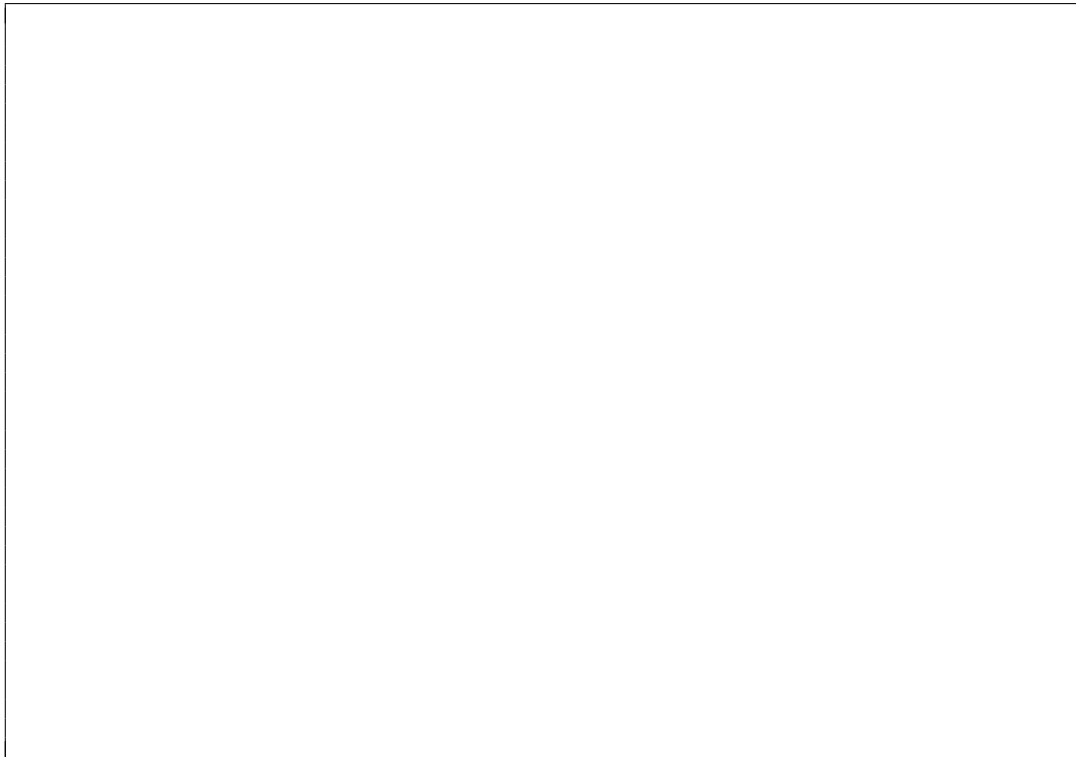
- Senden der Nachricht N_{10}
- Bearbeitungsschritt
- Empfangen der Nachricht N_5
- Senden der Nachricht N_{11}
- Bearbeitungsschritt

- Empfangen der Nachricht N_2
- Empfangen der Nachricht N_9

Nutzen Sie für die Antwort das ausgeteilte Klausurpapier.

Zeichnen Sie die Zeitachsen für die 4 Prozesse, tragen Sie die Ereignisse als Punkte, die Nachrichten als Pfeile ein und beschriften Sie die Nachrichten. Tragen Sie dann die Lamportzeit für jedes Ereignis ein.

- (b) Welchen Nachteil hat die Lamportzeit und wie kann dieser durch Vektoruhren vermieden werden? (4)



- (c) Nehmen wir an in Beispiel von Teilaufgabe (a) werden statt der Lamportzeit Vektoruhren genutzt. Die lokale Uhr soll wieder bei allen Prozessen bei 0 beginnen, d.h. das erste Ereignis erfolgt zur Zeit 1. Geben Sie die 11 Vektoruhren an, wie sie jeweils unmittelbar **nach** dem Empfangen einer Nachricht beim **empfangenden** Prozess vorliegen. (5)

Sie können diese direkt in der Zeichnung zu Aufgabe a) eintragen.