VO 182.022 17. November 2006

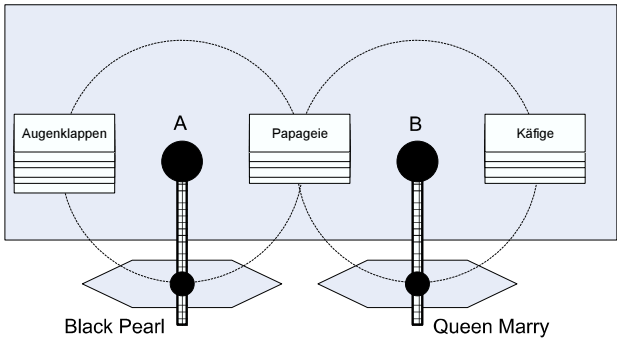
Prüfung Betriebssysteme

KNr. MNr. Zuname, Vorname

Ges.)(100)	1.)(30)	2.)(25)	3.)(23)	4.)(22)	Zusatzblätter:

Bitte verwenden Sie nur dokumentenechtes Schreibmaterial!

1 Synchronisation (30)



Die Reederei *Titanic Transports* besitzt zwei Frachtschiffe namens *Queen Marry* und *Black Pearl*. Beide Frachtschiffe haben einen eigenen Liegeplatz mit dazugehörigem Verladekran. Die Queen Marry liefert regelmäßig *Papageie* und dazugehörige *Käfige* an Tierliebhaber in ein weit entferntes Land. Die Black Pearl liefert an eine von Piraten bewohnte Insel *Papageie* und *Augenklappen* (Piraten tragen ihren Papagei auf der Schulter und brauchen daher keine Käfige).

Die Papageie, Käfige und Augenklappen sind in Containern verpackt, welche auf dem Hafen gestapelt stehen.

Das Beladen der Schiffe läuft nach folgenden Regeln ab:

- Die Queen Marry soll mit einem Container voller Papageie und einem Container voller Käfige beladen werden. Die Black Perl soll mit einem Container voller Augenklappen und mit einem Container voller Papageie beladen werden.
- Zum Beladen werden die Kräne verwendet.
- Der Kran *A*, welcher der Black Pearl zugeordnet ist kann über die Container mit den Papageien, über die Container mit den Augenklappen und über die Black Pearl bewegt werden.
- Der Kran *B*, welcher der Queen Marry zugeordnet ist, kann über die Container mit den Papageien, über die Container mit den Käfigen und über die Queen Marry bewegt werden.

- Um ein Zusammenstoßen der Kräne zu verhindern dürfen niemals beide Kräne gleichzeitig über die Container mit den Papageien bewegt werden.
- Wenn ein Schiff fertig beladen ist, soll es sofort mit seiner Auslieferung beginnen.
- Mit dem Beladen eines Schiffes kann natürlich erst dann wieder begonnen werden wenn das Schiff zurückgekehrt ist. Ein Kran kann allerdings einen Container schon vom Stapel nehmen bevor das Schiff wieder zurückgekehrt ist.

Synchronisieren Sie den Arbeitsablauf der beiden Kräne und der beiden Schiffe mittels **Semaphoren**. Achten Sie auf maximale Parallelität. Verwenden Sie möglichst wenige Synchronisationskonstrukte. Die Verwendung von globalen Variablen ist verboten.

Zu verwendende Funktionen:

`initS(` ***Semaphor,init*** `)` Legt einen Semaphor mit dem angegebenen Namen *Semaphor* an und initialisiert ihn mit der Zahl *init*. Danach können die Funktionen **P(*Semaphor*)** und **V(*Semaphor*)** auf den Semaphor angewendet werden.

`bewege(` ***Richtung*** `)` Bewegt den Kran um 90 Grad in die gewünschte *Richtung*. Als Richtung kann `Uhrzeigersinn (UZS)` oder `Gegen_Uhrzeigersinn (GUZS)` angegeben werden.

`nimm()` Lässt den Kran einen Container von dem Stapel über dem er sich gerade befindet aufnehmen. Funktioniert nur, wenn sich der Kran über einem Containerstapel befindet.

`lege_ab()` Mittels dieser Funktion legt ein Kran den Container, den er gerade hält, an der aktuellen Position ab

`lieferung()` Mit dieser Funktion liefert ein Schiff seine Ware aus, und kehrt anschliessend wieder zurück.

a) Initialisierungen (8)

Initialisieren Sie die notwendigen Semaphore. Der **Anfangszustand** des Systems entspricht dem obigem Bild. Beide Schiffe sind unbeladen und beide Kräne stehen über den jeweiligen Schiffen.

b) (14)

Entwerfen Sie je einen Prozess für *Kran A* und *Kran B*.

Prozess *Kran A*:

```
do forever() {
}
```

Prozess *Kran B*:

```
do forever() {
}
```

c) (8)

Entwerfen Sie die Prozesse für die *Black Pearl* und die *Queen Marry*:

Prozess ***Black Pearl***:

```
do forever() {
                                        }
```

Prozess ***Queen Marry***:

```
do forever() {
                                        }
```

2 Scheduling (25)

2.1 Uniprocessor Scheduling (13)

Gegeben ist nebenstehendes Taskset. Alle Tasks sind periodisch, wobei die Deadlines mit dem Ende der jeweiligen Periode gleichzusetzen sind. Der Overhead für den Taskwechsel ist vernachlässigbar.

Task	Ausführungszeit	Periodendauer
A	1	10
B	1	5
C	1	4
D	2	7

Ermitteln Sie für dieses Taskset die *notwendige* und die *hinreichende* Bedingung für das *Rate Monotonic Scheduling* (RMS) Verfahren. Berechnen Sie die **Zahlenwerte** überschlagsmäßig (ohne Taschenrechner)!

Benutzen Sie dazu folgende Funktionstabelle:

n	1	2	3	4	5	6
$n \cdot (2^{(1/n)} - 1)$	1.00	0.82	0.77	0.75	0.74	0.73

Ist die notwendige Bedingung erfüllt? ◯ Ja ◯ Nein
Ist die hinreichende Bedingung erfüllt? ◯ Ja ◯ Nein

Versuchen Sie das Taskset einmal mit dem RMS und einmal mit dem *Earliest-Deadline-First* (EDF) Verfahren zu schedulen. Verwenden Sie dazu die nachstehenden Vorlagen. Tragen Sie bei jeder Vorlage die aktiven Taskzeiten ein und bezeichnen Sie deutlich eventuelle Deadlineverletzungen. Kreuzen Sie jeweils an ob das Scheduling erfolgreich war. Eine Vorlage dient als Ersatz, streichen Sie gegebenenfalls eine falsch ausgefüllte Vorlage deutlich durch.

Scheduling nach dem **RMS**-Verfahren: Erfolgreich: ◯ Ja ◯ Nein

A																		
B																		
C																		
D																		
	0					5					10					15		

Scheduling nach dem **EDF**-Verfahren: Erfolgreich: ◯ Ja ◯ Nein

A																		
B																		
C																		
D																		
	0					5					10					15		

Ersatzvorlage: Scheduling nach dem [redacted]-Verfahren: Erfolgreich: ◯ Ja ◯ Nein

A																		
B																		
C																		
D																		
	0					5					10					15		

2.2 Verständnisfragen (12)

Beurteilen Sie die folgenden Aussagen für Single-Prozessor Scheduling!
Fehlende Antworten werden negativ, falsche Antworten werden doppelt negativ gewertet!

◯ Ja ◯ Nein Die Prioritäten beim EDF Scheduling ergeben sich durch die Periodendauer der Tasks.

◯ Ja ◯ Nein Die Prioritäten beim RMS Scheduling ergeben sich durch die *Processor Utilization* der Tasks.

◯ Ja ◯ Nein Earliest Deadline First Scheduling findet in Single-Prozessor Systemen immer eine Lösung.

◯ Ja ◯ Nein Earliest Deadline First Scheduling findet in Multi-Prozessor Systemen immer eine Lösung, wenn eine solche existiert.

◯ Ja ◯ Nein Das EDF Scheduling ist non-preemptive.

◯ Ja ◯ Nein Bei Round Robin Scheduling kann das Problem der Starvation auftreten.

◯ Ja ◯ Nein Beim Scheduling nach dem Round-Robin-Verfahren werden I/O-intensive Prozesse benachteiligt.

◯ Ja ◯ Nein Die First-Come-First-Serve-Strategie benachteiligt CPU-intensive Prozesse.

◯ Ja ◯ Nein Das Round-Robin-Verfahren ist preemptive.

◯ Ja ◯ Nein Das Shortest-Process-Next-Verfahren ist für Echtzeitscheduling ungeeignet.

◯ Ja ◯ Nein Das Shortest-Process-Next-Verfahren kommt im Vergleich zu Round Robin mit weniger Interruptaufrufen aus.

◯ Ja ◯ Nein Ein Scheduler nach dem RMS Verfahren benötigt mehr Information pro Task als ein Scheduler nach dem Round-Robin-Verfahren.

3 Deadlock (23)

Das Catering Service *Fein und Gesund* bekommt von verschiedenen Kunden Aufträge um auf diversen Veranstaltungen für das leibliche Wohl zu sorgen.

Um die Aufträge erfolgreich erfüllen zu können müssen bestimmte Ressourcen wie Kellner (K), Teller (T) und Sektgläser (S) koordiniert werden. Dem Unternehmen stehen 15 Kellner, 200 Teller und 500 Sektgläser zur Verfügung.

Im Augenblick sind drei Aufträge in Arbeit:

- Für **Auftrag 1** werden 11 Kellner, 150 Teller und 300 Sektgläser benötigt. Reserviert sind für diesen Auftrag bereits 10 Kellner, 130 Teller und 200 Sektgläser.
- Für **Auftrag 2** werden 5 Kellner, 70 Teller und 300 Sektgläser benötigt. Dieser Auftrag hat noch nicht begonnen.
- Für **Auftrag 3** werden 7 Kellner, 10 Teller und 80 Sektgläser benötigt. Reserviert sind für diesen Auftrag bereits 1 Kellner, 1 Teller und 1 Sektglas.

Process Initiation Denial

Beschreiben Sie *Resource-* und *Available-Vektor* sowie *Claim-* und *Allocation-Matrix*. Die Matrizen sind so auszufüllen, dass die Ressourcen zeilenweise und die Aufträge spaltenweise aufgezählt werden.

$$Resource = \begin{pmatrix} K & \\ T & \\ S & \end{pmatrix} \qquad Available = \begin{pmatrix} & \\ & \\ & \end{pmatrix}$$

$$Claim = \begin{pmatrix} K & & & \\ T & & & \\ S & & & \end{pmatrix} \qquad Allocation = \begin{pmatrix} & & \\ & & \\ & & \end{pmatrix}$$

Darf mit Auftrag 2 gemäß *Process Initiation Denial* begonnen werden?

◯ Ja ◯ Nein

Begründen Sie Ihre Antwort (Antworten ohne Begründung werden nicht gewertet!):

Deadlock Vermeidung

Verwenden Sie nun von obigem Initialzustand ausgehend den *Bankier-Algorithmus* (Banker's Algorithm) zum Schedulen der Aufträge. Geben Sie für *jeden* Schritt die Claim- und Allocation-Matrix, sowie den Available Vector und den als nächstes durchzuführenden Auftrag an. Wenn kein Auftrag mehr auszuführen ist, dann schreiben sie 'fertig' in den nächsten Schritt, falls ein Deadlock auftritt, schreiben Sie 'Deadlock' .

Nächster Auftrag: . Alle für den Auftrag benötigten Ressourcen sind in Verwendung:

$$Claim = \begin{pmatrix} \text{K} & & & \\ \text{T} & & & \\ \text{S} & & & \end{pmatrix} \quad Alloc = \begin{pmatrix} & & \\ & & \\ & & \end{pmatrix} \quad Avail = \begin{pmatrix} \\ \\ \end{pmatrix}$$

Nach der Ausführung des zuletzt markierten Auftrags:

$$Claim = \begin{pmatrix} \texttt{K} & & & \\ \texttt{T} & & & \\ \texttt{S} & & & \end{pmatrix} \quad Alloc = \begin{pmatrix} & & \\ & & \\ & & \end{pmatrix} \quad Avail = \begin{pmatrix} \\ \\ \end{pmatrix}$$

Nächster Auftrag: . Alle für den Auftrag benötigten Ressourcen sind in Verwendung:

$$Claim = \begin{pmatrix} \texttt{K} & & & \\ \texttt{T} & & & \\ \texttt{S} & & & \end{pmatrix} \quad Alloc = \begin{pmatrix} & & \\ & & \\ & & \end{pmatrix} \quad Avail = \begin{pmatrix} \\ \\ \end{pmatrix}$$

Nach der Ausführung des zuletzt markierten Auftrags:

$$Claim = \begin{pmatrix} \texttt{K} & & & \\ \texttt{T} & & & \\ \texttt{S} & & & \end{pmatrix} \quad Alloc = \begin{pmatrix} & & \\ & & \\ & & \end{pmatrix} \quad Avail = \begin{pmatrix} \\ \\ \end{pmatrix}$$

Nächster Auftrag: . Alle für den Auftrag benötigten Ressourcen sind in Verwendung:

$$Claim = \begin{pmatrix} \texttt{K} & & & \\ \texttt{T} & & & \\ \texttt{S} & & & \end{pmatrix} \quad Alloc = \begin{pmatrix} & & \\ & & \\ & & \end{pmatrix} \quad Avail = \begin{pmatrix} \\ \\ \end{pmatrix}$$

Nach der Ausführung des zuletzt markierten Auftrags:

$$Claim = \begin{pmatrix} \texttt{K} & & & \\ \texttt{T} & & & \\ \texttt{S} & & & \end{pmatrix} \quad Alloc = \begin{pmatrix} & & \\ & & \\ & & \end{pmatrix} \quad Avail = \begin{pmatrix} \\ \\ \end{pmatrix}$$

Nächster Auftrag: . Alle für den Auftrag benötigten Ressourcen sind in Verwendung:

$$Claim = \begin{pmatrix} \texttt{K} & & & \\ \texttt{T} & & & \\ \texttt{S} & & & \end{pmatrix} \quad Alloc = \begin{pmatrix} & & \\ & & \\ & & \end{pmatrix} \quad Avail = \begin{pmatrix} \\ \\ \end{pmatrix}$$

Nach der Ausführung des zuletzt markierten Auftrags:

$$Claim = \begin{pmatrix} \texttt{K} & & & \\ \texttt{T} & & & \\ \texttt{S} & & & \end{pmatrix} \quad Alloc = \begin{pmatrix} & & \\ & & \\ & & \end{pmatrix} \quad Avail = \begin{pmatrix} \\ \\ \end{pmatrix}$$

4 Speicherverwaltung (22)

Das betrachtete Speicherverwaltungssystem verwendet zur Adressierung 16-Bit Adressen. Für die angegebenen virtuellen Speicheradressen sind, in Abhängigkeit von der Adressierungsart, die entsprechenden physikalischen Adressen zu ermitteln. Von den angebenen Adressen sind die niederwertigen 8 Bit der Offset der Adresse und die höherwertigen 8 Bit die Segmentnummer bzw. die Seitennummer. Bei Paging sind alle Seiten 256 Bytes (Hexadezimal 0x100) lang. Alle Werte mit führendem **0x** sind als Hexadezimalzahl angegeben. Ergibt sich bei der Umwandlung eine ungültige Adresse, so schreiben Sie bitte **ungültig**

Page# 0x08 --> ist nicht vorhanden in der Adressübersetzungstabelle
UNGÜLTIG

Offset 0x3A

Offset 0xFF

Page# 0x01 --> ist in der Adressübersetzungstabelle
=> Frame# 0x17 append Offset 0xFF
==> 0x17FF **LÖSUNG** Physikalische Adresse

5

Es werden folgende Begriffe (englische Notation) verwendet:

Base Basisadresse des Segmentes

schen Speicher)

Page# Seitennummer (im virtuellen Speicher)

1

2

3

4

a) Paging — Assoziativer Zugriff (associative mapping)

Adressübersetzungstabelle:

Page#	Frame#
0x11	0x33
0x01	0x17
0x17	0x20
0x42	0x08
0x05	0x05

Zu berechnende Adressen:

Virtuelle Adresse	Physikalische Adresse
0x083A	ungültig
0x01FF	0x17FF
0x0505	0x0505
0x063A	ungültig

b) Paging — Direkter Zugriff (direct mapping)

Adressübersetzungstabelle:

Entry	Frame#
0	0x42
1	0x06
2	0x04
3	0x11
4	0x01
5	0xFA
6	0x02

Zu berechnende Adressen:

Virtuelle Adresse	Physikalische Adresse
0x0406	0x0106
0x0611	0x0211
0x11FA	ungültig
0xFA11	ungültig

SegNr -> Entry 0x18 -> dezimal 24 -> es gibt kein Entry 24 in der Adressübersetzungstabelle UNGÜLTIG

Offset 0x02 -> wir schauen in der Adressübersetzungstabelle unter der Entry 4 die Length 0x0F0 nach und
==> wenn 0x02 >= 0x0F0 gilt ==> ungültig
sonst OK

4

c) Segmentierung — Direkter Zugriff (direct mapping)

0x04 ist in der Adressübersetzungstab. vorhanden 4 | 0x01FE | 0x0F0
0x01FE
__0x02 (Offset) + 5
0x0200 LÖSUNG Physikalische Adresse

2 Offset 0x32

1 Zu berechnende Adressen:

Virtuelle Adresse	Physikalische Adresse
0x1832	ungültig
0x0402	0x200
0x4222	ungültig
0x0010	0x1840
0x0305	ungültig
0x01F2	0x0902
0x0650	ungültig

3

Adressübersetzungstabelle:

Entry	Base	Length
0	0x1830	0x020
1	0x0810	0x0FF
2	0x42AC	0x100
3	0x0400	0x004
4	0x01FE	0x0F0
5	0x0010	0x030
6	0x3302	0x050
7	0x4220	0x010

d) Verständnisfragen

Kreuzen Sie bitte die richtigen Antworten an! Achtung! Falsche Antworten werden negativ gewertet!

- Bei Paging kann das Problem der externen Fragmentierung auftreten.
  - ◯ richtig
  - ◯ falsch
- Beim *Fixed Partitioning* sind die Partionen *immer* von gleicher Größe.
  - ⊗ richtig
  - ◯ falsch
- In einem fehlerfreien System können zwei oder mehr virtuelle Adressen auf eine physikalische Adresse abgebildet sein.
  - ◯ richtig
  - ◯ falsch
- Zu welchen Effekten kann es bei Segmentierung kommen?
  - ◯ Unterschiedliche Segmentlängen
  - ◯ Internal Fragmentation
  - ◯ External Fragmentation

  Je größer die "page size" (Seitengröße), desto mehr "pages" (Seiten) und "page frames" (Seitenrahmen) gibt es und desto größer müssen die "page tables" (Seitentabellen) sein.
  - ◯ richtig
  - ⊗ falsch