VO 182.022 25. Jänner 2010

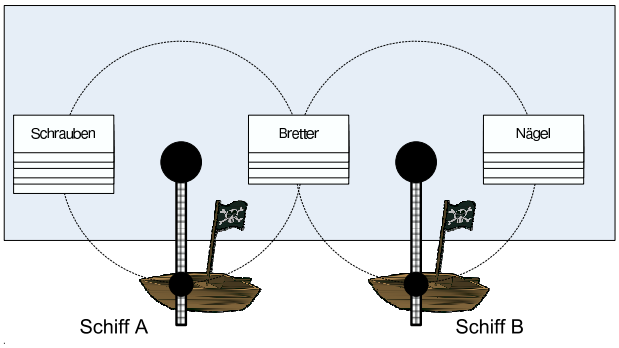
Prüfung Betriebssysteme

KNr. MNr. Zuname, Vorname

Ges.)(100)	1.)(30)	2.)(20)	3.)(25)	4.)(25)	Zusatzblätter:

Bitte verwenden Sie nur dokumentenechtes Schreibmaterial!

1 Synchronisation (30)



Eine Reederei besitzt zwei Frachtschiffe, *Schiff A* und *Schiff B*. Beide Frachtschiffe haben einen eigenen Liegeplatz mit dazugehörigem Verladekran. Das *Schiff A* transportiert *Schrauben* und *Bretter*, das *Schiff B* transportiert *Bretter* und *Nägel*.

Die Schrauben, Bretter und Nägel sind in Containern verpackt, welche auf dem Hafen gestapelt stehen.

Das Beladen der Schiffe läuft nach folgenden Regeln ab:

- Schiff A soll mit einem Container voller Schrauben und einem Container voller Bretter beladen werden. Schiff B soll mit einem Container voller Bretter und mit einem Container voller Nägel beladen werden. Die Schrauben bzw. die Nägel müssen vor den Brettern geladen werden.
- Zum Beladen werden die Kräne verwendet.
- Der Kran *A*, welcher Schiff A zugeordnet ist kann über die Container mit den Schrauben, über die Container mit den Brettern und über das Schiff A bewegt werden.
- Der Kran *B*, welcher Schiff B zugeordnet ist, kann über die Container mit den Brettern, über die Container mit den Nägeln und über das Schiff B bewegt werden.
- Um ein Zusammenstoßen der Kräne zu verhindern dürfen niemals beide Kräne gleichzeitig über die Container mit den Brettern 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 *Schiff A* und *Schiff B*:

Prozess *Schiff_A*:

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

Prozess *Schiff B*:

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

2 Memory Management (20)

Was ist der Unterschied zwischen lokalem und globalem Page Replacement?

Die Page Table eines kleinen Computersystems ist als Inverted Page Table (IPT) mit acht Einträgen realisiert. Beim Auftreten von Page Faults wird in diesem Computersystem der Clock Algorithmus als globale Page Replacement Strategie eingesetzt.

Im zu lösenden Beispiel ist eine Ausgangsbelegung der Page Table (links), sowie eine Folge von Zugriffen auf den virtuellen Speicher, charakterisiert durch Prozessnummer (PID), adressierte Seite (page) und Offset, gegeben. Simulieren Sie die Ausführung der Zugriffsfolge von links nach rechts und tragen Sie für jeden Zugriff auf den virtuellen Speicher (a) die entsprechende physikalische Adresse, (b) den Zustand der IPT und (c) den Wert des Replacement Pointers (rep. pointer) für den Clock Algorithmus nach dem Speicherzugriff an (Sie brauchen bei jedem Schritt nur die Werte in der IPT anzugeben, die sich beim aktuellen Zugriff geändert haben).

#vielleicht beim lesen wird 0 gesetzt look book

PID	page offset
1	0 00a0
phys. address	page fault

	PID	page	use
0	1	5	1
1	3	1	0
2	2	3	0
3	2	1	0
4	1	2	1
5	1	3	1
6	1	4	1
7	3	3	1

rep. pointer 5

PID	page offset
1	0 00a2
phys. address	0x100a2

	PID	page	use
0			0
1	1	0	1
2			0
3			0
4			1
5			0
6			0
7			0

rep. pointer 2

PID	page offset
1	4 0002
phys. address	0x60002

	PID	page	use
0	1	0	1
1			0
2			0
3			0
4			1
5			0
6			0
7			0

rep. pointer 2

PID	page offset
2	3 fffe
phys. address	0x2fffe

	PID	page	use
0	1	0	1
1			0
2			0
3			0
4			1
5			0
6			0
7			0

rep. pointer 2

PID	page offset
2	4 0000
phys. address	page fault

	PID	page	use
0	1	0	1
1			0
2	2	3	1
3			0
4			1
5			0
6			1
7			0

rep. pointer 2

PID	page offset
2	4 0000
phys. address	page fault

	PID	page	use
0	1	0	1
1			0
2			0
3	2	4	1
4			1
5			0
6			1
7			0

rep. pointer 4

3 Deadlock (25)

Im folgenden Beispiel konkurrieren 5 Prozesse um 4 verschiedene Ressourcen. Zum betrachteten Zeitpunkt sind 2 Ressourcen vom Typ 1, und keine Ressource des Typs 2,3 oder 4 verfügbar.

Die *allocation Matrix* A gibt die gegenwärtig von jedem der 5 Prozesse allokierten Ressourcen an, die Matrix Q beschreibt die momentan von den Prozessen angeforderten und noch nicht gewährten Ressourcen (in den Matrizen repräsentiert jede Spalte eine Ressource und jede Zeile einen Prozess).

Kann in dieser Situation ein Deadlock vorliegen ? Führen Sie den *deadlock detection* Algorithmus durch, um diese Frage zu beantworten.

$$A = \begin{pmatrix} 2 & 0 & 1 & 1 \\ 1 & 1 & 0 & 3 \\ 0 & 0 & 0 & 0 \\ 0 & 1 & 2 & 0 \\ 1 & 2 & 0 & 1 \end{pmatrix} \quad Q = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 1 & 0 & 0 & 1 \\ 3 & 3 & 0 & 4 \\ 1 & 2 & 1 & 1 \\ 1 & 0 & 2 & 0 \end{pmatrix}$$

Ein Deadlock kann aufgehoben werden, in dem den Prozessen einige Ressourcen entzogen werden. Wie kann im vorangegangen Beispiel durch das Entziehen genau einer allokierten Ressource sichergestellt werden, dass kein Deadlock mehr vorliegt ?

Welche anderen Strategien zur Aufhebung eines Deadlocks kennen Sie ?

Beschreiben Sie eine Möglichkeit, um das Auftreten der notwendigen Deadlock Bedingung "*circular wait*" auszuschliessen (*direct deadlock prevention*). Die gleichzeitige Nutzung unterschiedlicher Ressourcen soll weiterhin möglich sein.

Welche weiteren Bedingungen sind für einen Deadlock notwendig ? Erklären Sie deren Bedeutung.

4 Input–Output (25)

Nennen Sie die beiden widersprechenden Hauptziele, die bei der Realisierung eines I/O-Systems für ein General Purpose Betriebssystem verfolgt werden. Geben Sie für jedes der beiden Ziele an, welche Auswirkungen es auf das Design des Betriebssystems hat.

Beschreiben Sie die hierarchische Ebenestruktur, die bei der Realisierung von I/O-Funktionen Anwendung findet. Geben Sie Name und Funktion für jede Ebene an.

Was sind die Vor- und Nachteile des Pufferns von I/O-Anfragen?

Nennen Sie zwei Verfahren, die beim Disk-Scheduling zur "Minimierung der Seek Time" verwendet werden (nicht FIFO, LIFO, und Prioritätsverfahren). Beschreiben Sie für jedes der von Ihnen genannten Verfahren kurz Funktionsweise, Vor- und Nachteile.