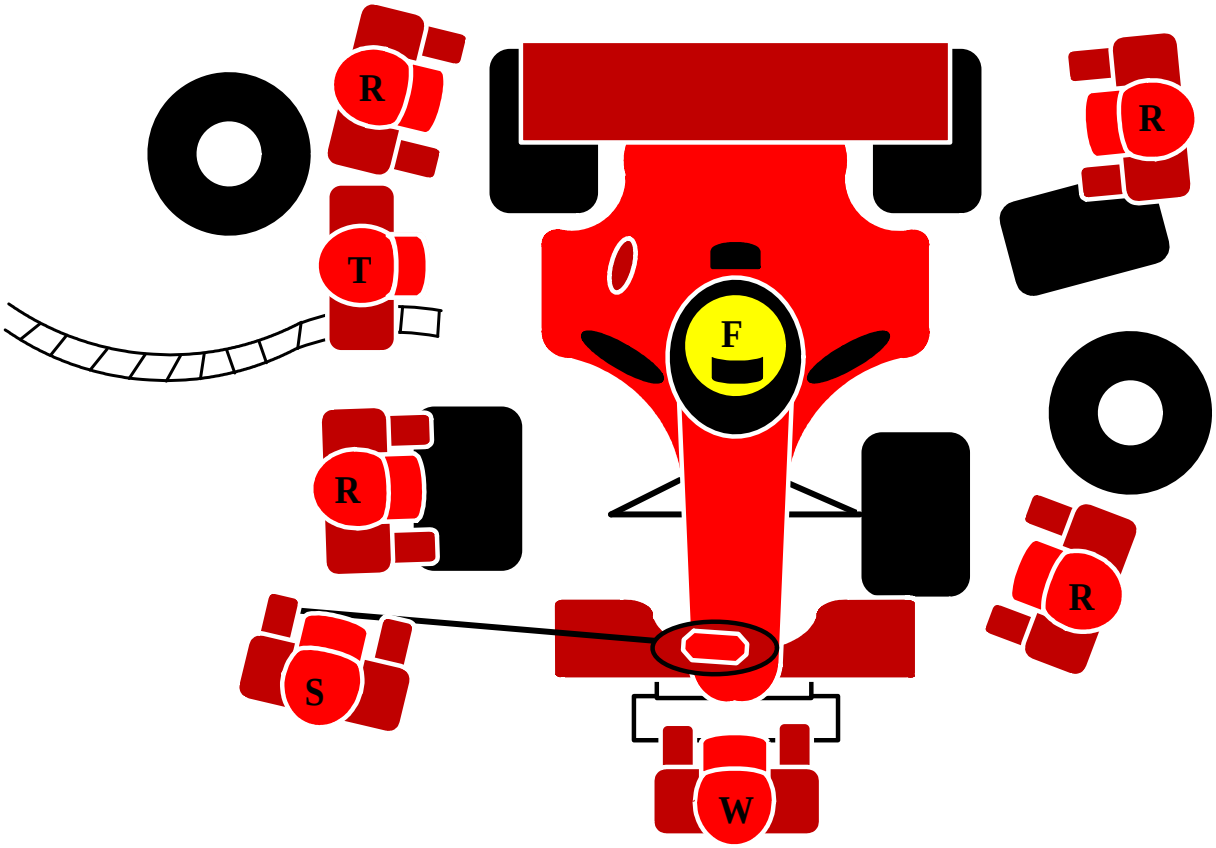
Vorlesungsprüfung aus Systemprogrammierung 2001-10-24

KNr. MNr. Zuname, Vorname

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

Bitte verwenden Sie nur dokumentenechtes Schreibmaterial!

1 Synchronisation (30)



Ein **Boxenstopp** in der Formel 1 läuft folgendermaßen ab: Sobald der Wagen eingefahren ist, beginnt der Tankwart (T) mit dem auffüllen des Tanks, während der Mann mit dem Wagenheber (W) den Wagen anhebt. Ist der Wagen in der Luft, beginnen die vier Reifenwechsler (R) mit dem Austausch der Reifen. Sind alle vier Reifen gewechselt, lässt der Mann mit dem Wagenheber das Fahrzeug wieder zu Boden und geht mit dem Wagenheber

zur Seite. Wenn der Wagenheber beiseite geräumt *und* der Tankvorgang beendet ist, gibt der Signalmann dem Fahrer das Signal zum Wegstarten.

Bedenken Sie bitte, dass es beim **Boxenstopp** auf optimale Synchronisation zwischen den Boxenmitarbeitern untereinander und zwischen Box und Fahrer ankommt, um keine wertvolle Zeit zu verlieren. Verwenden Sie außerdem nur unbedingt notwendige Synchronisationskonstrukte um dieses Ziel zu erreichen.

Zu verwendende Funktionen:

`initE(` ***Name*** `)` Legt einen Eventcounter mit dem angegebenen Namen *Name* an und initialisiert ihn mit der Zahl 0. Danach können die Funktionen **Advance(*Name*)** und **Wait(*Name*, *Wert*)** auf den Eventcounter angewendet werden.

`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.

`reFuel()` Initiert einen Tankvorgang. Diese Funktion wird vom Tankwart ausgeführt und blockiert, bis der Tankvorgang beendet ist und der Tankwart aus dem Gefahrenbereich verschwunden ist.

`changeTire()` Wechselt den Reifen. Diese Funktion wird von den Reifenwechslern ausgeführt und blockiert, bis der Vorgang beendet ist und der Reifenwechsler aus dem Gefahrenbereich verschwunden ist.

`liftCar()` Hebt den Wagen an. Diese Funktion wird vom Mann mit dem Wagenheber ausgeführt und blockiert, bis der Vorgang beendet ist.

`lowerCar()` Lässt den Wagen wieder sinken. Diese Funktion wird vom Mann mit dem Wagenheber ausgeführt und blockiert, bis der Vorgang beendet ist.

`removeJack()` Entfernt den Wagenheber aus dem Gefahrenbereich. Diese Funktion wird vom Mann mit dem Wagenheber ausgeführt und blockiert, bis der Vorgang beendet ist und der bedienende Mann aus dem Gefahrenbereich verschwunden ist.

Synchronisieren Sie den Arbeitsablauf der sieben Mitarbeiter des Boxenteams für *einen einzigen* Boxenstopp mittels **Eventcounter**. Sie können davon ausgehen, dass am Anfang der Wagen zum Stillstand gekommen ist, der Wagenheber bereits unter dem Wagen positioniert ist, der Wagen aber noch nicht angehoben ist und weder Reifenwechsler noch Tankwart mit ihrer Arbeit begonnen haben.

Synchronisieren Sie weiters die Kommunikation zwischen Fahrer und Signalgeber mittels **Semphor(e)**.

a) Initialisierungen (5)

Nehmen Sie hier die notwendigen Initialisierungen, welche beim Einfahren in die Box ausgeführt werden, vor:

b) (5)

Entwerfen Sie hier das Programm für einen beliebigen Reifenwechsler (R):

c) (5)

Entwerfen Sie hier das Programm für den Mann der den Wagenheber bedient (W):

d) (5)

Entwerfen Sie hier das Programm für den Signalgeber (S):

e) (5)

Entwerfen Sie hier das Programm für den Tankwart (T):

f) (5)

Entwerfen Sie hier das Programm für den Fahrer (F):

```
gib_gas();
```

2 Speicherverwaltung (25)

a) Segmentierung (13)

In einem Computersystem mit virtuellem Speicher gibt es drei Tasks, `A`, `B`, und `C`, welche die folgenden Ressourcen benötigen:

Task	**Code** (bytes)	**Data** (bytes)
`A`	1250	798
`B`	3423	3745
`C`	1024	0

Geben Sie den Verschnitt an Speicher (= Speicher, der reserviert, aber nicht tatsächlich benötigt wird) in Bezug zum reservierten Speicher für jedes der Programme für die zwei Fälle in Punkt 1) und 2) in Prozent an.

Nehmen Sie an, dass Code und Daten getrennt gespeichert werden. Geben Sie unbedingt das Ergebnis in Form einer *Dezimalzahl* mit einer Stelle hinter dem Komma (z.B. 13.1 %) an.

1) Verwendung von Paging, Seitengröße 2 *KB* (=2048 *Bytes*) (6)

Task `A`: Anzahl der Seiten = ⇒ Verschnitt = . %

Task `B`: Anzahl der Seiten = ⇒ Verschnitt = . %

Task `C`: Anzahl der Seiten = ⇒ Verschnitt = . %

2) Segmentierung des Speichers (3)

Verschnitt von Task `A` = . %

Verschnitt von Task `B` = . %

Verschnitt von Task `C` = . %

3) Erklärung (4)

Erklären Sie einerseits für Paging, andererseits für Speichersegmentierung, durch welchen Effekt es dazu kommen kann, dass der vorhandene physikalische Speicher nicht optimal genützt wird. Bitte Namen der Effekte und Erklärung angeben.

Paging:

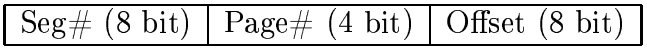
Segmentierung:

b) Kombination aus Segmentierung und Paging (12)

Es werden folgende Begriffe (englische Notation) aus dem Buch zur Vorlesung verwendet:

Base	Basisadresse Seitentabelle des Segmentes
Length	Länge des Segmentes (Anzahl der Seiten des Segmentes)
Virt.Addr.	Virtuelle Adresse
Frame#	Seitenrahmennummer (im physischen Speicher)
Page#	Seitennummer (im virtuellen Speicher)
Seg#	Segmentnummer

Das in der Folge betrachtete Speicherverwaltungssystem verwendet zur (physikalischen und virtuellen) Adressierung 20-bit Adressen. Für das Paging sind alle Seitenrahmen 256 Bytes (hexadezimal `0x00 100`) groß. Das verwendete Adressformat ist folgendes:



Hierbei wird assoziativer Zugriff (associative mapping) auf die Segmenttabelle und direkter Zugriff (direct mapping) auf die Seitentabelle verwendet.

Verwenden Sie für die Adressumsetzung folgende Segmenttabelle und Seitentabelle (alle Werte sind als Hexadezimalzahlen angegeben):

Segmenttabelle		
Seg#	Base	Length
0x02	0x04 21F	0x4
0xFC	0x7D 000	0x2
0x3F	0xC1 0E0	0x3
0x09	0x20 221	0xF

Seitentabelle	
Address	Frame#
...	...
0x04 21F	0x451
0x04 220	0x023
0x04 221	0x845
0x04 222	0x734
0x04 223	0x475
...	...
0x20 221	0x311
0x20 222	0xF00
...	...
0x20 22E	0x000
0x20 22F	0xDDD

Fortsetzung Seitentabelle	
Address	Frame#
0x20 230	0x666
0x20 231	0x765
...	...
0x7D 000	0x471
0x7D 001	0x871
0x7D 002	0x111
...	...
0xC1 0E0	0x394
0xC1 0E1	0x588
0xC1 0E2	0x127
0xC1 0E3	0x600
...	...

Ermitteln Sie unter Benützung obiger Tabellen die physikalischen Adressen zu folgenden virtuellen Adressen (ergibt sich bei der Umwandlung eine ungültige Adresse, so schreiben Sie bitte **ungültig** in das entsprechende Feld):

Virtuelle Adresse	Physikalische Adresse (zu ermitteln)
0x0123F	
0x3F311	
0x09EFF	
0x090D7	
0xFC000	
0x0238A	

3 Deadlock (25)

Gegeben sind zwei Prozesse, P_1 und P_2, die jeweils die Ressourcen R_1 und R_2 benötigen. Jede der zwei Ressourcen ist zweimal vorhanden. Benötigt ein Prozess eine vom anderen Prozess belegte Ressource, so wird er auf jeden Fall bis zum Freiwerden der Ressource verzögert. Die Abbildung unten zeigt für jeden der beiden Prozesse, zu welchem Zeitpunkt seiner Abarbeitung er jeweils die einzelnen Ressourcen benötigt. Die Anforderungen von Prozess P_1 sind entlang der x-Achse, die Anforderungen von Prozess P_2 entlang der y-Achse aufgetragen.

Der Fortschritt von P_1 und P_2 bei der (quasi)parallelen Abarbeitung kann als Kantenzug zwischen den Punkten *0* und *end* in der Grafik eingetragen werden (siehe Buch zur Vorlesung: W. Stallings, Operating Systems).

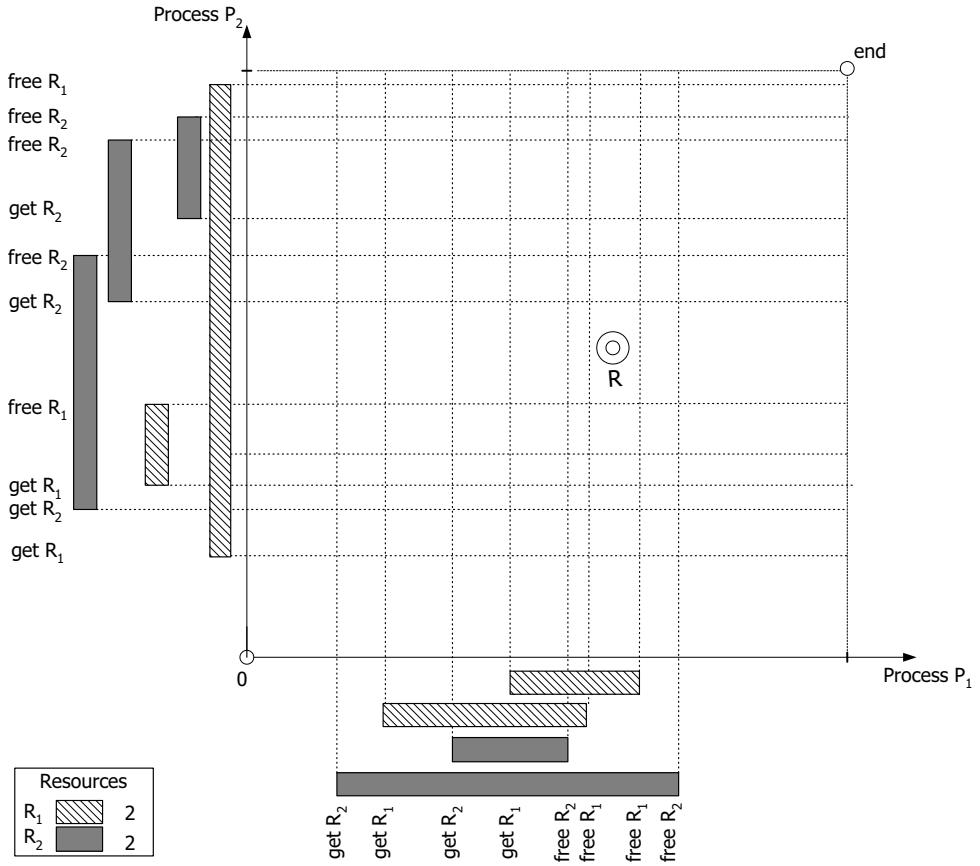
1. Umranden und schraffieren Sie in der Grafik jene Bereiche, durch die ein solcher Kantenzug aufgrund von Ressourcenkonflikten nicht gehen kann.

2. Kennzeichnen Sie auf unterschiedliche Weise die Bereiche, die von einem Kantenzug nicht passiert werden dürfen, wenn eine Abarbeitung von P_1 und P_2 deadlockfrei erfolgen soll. Beschriften Sie diese Bereiche deutlich mit einem "D".

3. Zeichnen Sie einen Kantenzug für eine gültige, deadlockfreie Abarbeitung von P_1 und P_2 in der Grafik ein.

4. In der Grafik ist ein Punkt R mit einem Kreis markiert. Kann dieser Punkt bei der Abarbeitung der Prozesse erreicht werden? Begründen Sie Ihre Antwort!

Achten Sie bitte darauf, dass alle Lösungen gut erkennbar und die Lösungen zu den Teilaufgaben 1 und 2 *deutlich unterscheidbar* sind.


Process P2
end
free R1
free R2
free R2
get R2
free R2
get R2
free R1
get R1
get R2
get R1
R
0
Process P1
Resources
R1 2
R2 2
get R2
get R1
get R2
get R1
free R2
free R1
free R1
free R2

4 Konzepte von Betriebssystemen (20)

a) (4)

Geben Sie zwei Abstraktionen an, die ein Betriebssystem zur Verfügung stellt. Beschreiben Sie für jede dieser Abstraktionen, welche Aufgaben sie dem Benutzer abnimmt.

b) (3)

Wodurch gewährleistet ein Betriebssystem die Portabilität von Programmen?

c) (5)

Geben Sie die drei Fälle an, in denen die Kontrolle über einen Rechner von einem Benutzerprogramm an das Betriebssystem wechselt.

d) (8)

Was ist ein Mode Switch? Was ist ein Process Switch? Was passiert bei jedem dieser Switches? Wie stehen Mode Switch und Process Switch in Beziehung?