1)
a

Auf 1 (Folge April 2002)

c) Prozesse werden durch Synchronisationsanforderungen blockiert
~~Durch~~ Durch diese Blockierungen können die Synchronisationsanforderungen niemals erfüllt werden.

d) Eine Funktion / Prozedur wird auf einen anderen Rechner gestartet und ausgeführt. Der aktuelle Prozess wird blockiert bis die aufrufende Prozedur ein Ergebnis zurückliefert

f) Damit die Betriebsmittelvergabe statisch geplant werden kann müssen alle Prozessparameter vorher bekannt sein. Sonst kann nur dynamisch geplant werden.

(a) **(2 Punkte)** Welche Eigenschaft weist das vorliegende Programm nicht auf, um nebenläufig mehrfach ausgeführt werden zu können? Begründen Sie Ihre Antwort!

(b) **(2 Punkte)** Welche Programmiertechniken müssen eingesetzt werden, um die Probleme zu beheben? Begründen Sie Ihre Antwort!

(c) **(6 Punkte)** Weitere Untersuchungen haben ergeben, daß das Unterprogramm auf dem neuen System wie folgt aufgerufen werden muß:

```
void _tue_gutes (int *D1wert, int *D2wert)
```

Die Parameterübergabe soll nicht mehr über die festen Adressen 0x2700 und 0x2710 sondern mittels der Variablen `D1wert` und `D2wert` erfolgen.

Ändern Sie das Programm so, daß es auf dem neuen System auch bei nebenläufiger Ausführung zu anderen Programmen und mehrfacher nebenläufiger Ausführung fehlerfrei läuft. Es wird die Programmierumgebung wie aus dem Übungsbetrieb angenommen (MC68000-Rechner mit C-Programmierumgebung). Sie müssen bei Ihrer Lösung nur die Register berücksichtigen, die im sichtbaren Programm verwendet werden; in den ausgeblendeten Zeilen mit ... | `tue gutes` werden keine Register verwendet.

Tragen Sie die notwendigen Änderungen in die Zeilen unter dem bisherigen Programmtext ein und streichen Sie bitte nicht benötigten Programmtext durch. Wenn Änderungen nicht nötig sind, dann lassen Sie die betreffenden Zeilen stehen. Nicht alle Zeilen müssen notwendigerweise geändert werden. Kommentieren Sie Ihre Programmzeilen.

a) keine Start und End Symbole
keine Reentranz (wieder Unterbrechbarkeit)
Bei nebenläufiger Programmierung sind mehrere Aufruf von dem Program abhängig sind - - - S57 Skript

b) benutzung des Kellers damit jeder Aufruf einen eigenen Speicherbereich besitzt, Parameter über den Stack, Zwischenwerte im Stack speichern, Register Retten.

c)

a) Reentranz. mit festen Adressen gearbeitet wird die bei nebenläufiger Ausführung von mehreren Prozessen benutzt werde
- die Parameterübergabe erfolgt über feste Adressen
- die Register werden nicht gerettet.

Aufgabe 1: Allgemeine Fragen (9 Punkte)

(a) **(1 Punkt)** Welche Funktion hat der Programmzähler (PC)?

(b) **(3 Punkte)** Nennen Sie 3 Konzepte zum Realisieren eines gegenseitigen Ausschlusses!

(c) **(1 Punkt)** Wann spricht man von einer Verklemmung (dead lock)?

(d) **(2 Punkte)** Was ist ein Prozedurfernaufruf (remote procedure call, remote method invocation)? Wie verhält sich die aufrufende Prozedur?

(e) **(1 Punkt) Welche grundlegende Bedingung muß ein Betriebssystem erfüllen, um „echtzeitfähig" zu sein?**

(f) **(1 Punkt)** Statische Planung hat eine hohe Effizienz und benötigt während der Laufzeit keine Rechenzeit. Warum kann statische Planung nicht für alle Probleme der Betriebsmittelvergabe angewendet werden?

Aufgabe 2: Assembler (10 Punkte)

Auf einer alten Rechenanlage mit einem Prozessor und einem ganz einfachen Betriebssystem, das nicht mehrprozeßfähig war, lief ein Unterprogramm, von dem hier die interessanten Stellen wiedergegeben werden. Die Zeilen mit ... | tue gutes stehen für ausgeblendeten Programmtext, der für die Bearbeitung der Aufgabe nicht interessiert.

```
_tue_gutes:  MOVEA.L #0x2700, A1     |   Speicheradressen in Register
             MOVEA.L #0x2710, A2     | ablegen
             MOVEA.L #0x2720, A3     |
             MOVE.L (A1), D1         | Werte aus Speicherzellen in
             MOVE.L (A2), D2         | Datenregister laden
                      ...            | tue gutes
             MOVE.L D1, D3           | kopiere D1 nach D3
             ADD.L D2, D1            | D1 := D1 + D2
             MOVE.L D3, D2           | Kopiere D3 nach D2
                      ...            | tue gutes
             MOVE.L D3, (A3)         | Zwischenwert aus D3
                                     | sichern
                      ...            | tue gutes
             ADD.L D1, D3            | D3 := D1 + D3
                      ...            | tue gutes
             MOVE.L (A3), D3         | Zwischenwert von D3 zurückholen
             MOVE.L D3, (A2)         | D3 als Ergebnis zurücklegen
             MOVE.L D1, (A1)         | D1 als Ergebnis zurücklegen
             RTS                     | das war es
```

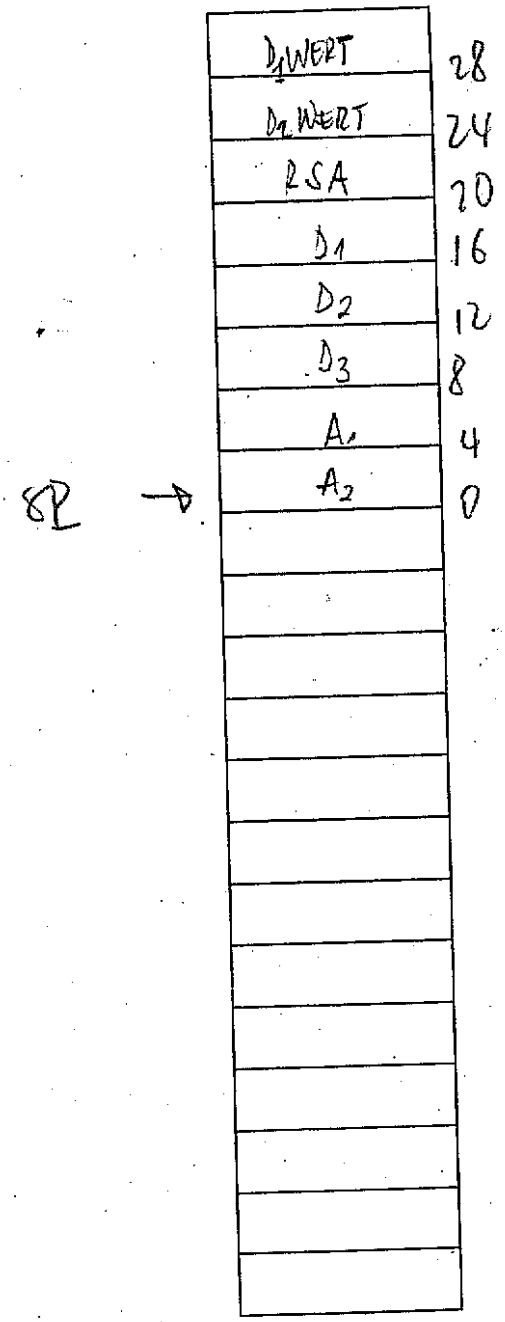
Dieses Programm wird auf einen modernen Einprozessorrechner mit einem modernen Betriebssystem portiert, das die nebenläufige Programmausführung erlaubt. Das Programm arbeitet korrekt, sofern es nur einmal und als einziges Programm zur Ausführung kommt. Wird das Programm jedoch mehrfach gestartet oder wenn es nebenläufig mit anderen Programmen läuft, kommt es zu Rechenfehlern.

```
_tue_gutes:  MOVEM.L D1-D3/A1-A2,(-SP) Register retten
MOVEA.L #0x2700, A1   | Speicheradressen in Register
MOVEAL 24(SP), A1     | D1wert ab. in A1 ablege
MOVEA.L #0x2710, A2   | ablegen
MOVEAL 28(SP), A2     | D2wert in A2 ablege
MOVEA.L #0x2720, A3
                      |
MOVE.L (A1), D1       | Werte aus Speicherzellen in
                      |
MOVE.L (A2), D2       | Datenregister laden
                      |
...                   | tue gutes
MOVE.L D1, D3         | kopiere D1 nach D3
                      |
ADD.L D2, D1          | D1 := D1 + D2
                      |
MOVE.L D3, D2         | Kopiere D3 nach D2
                      |
...                   | tue gutes
MOVE.L D3, (A3)       | Zwischenwert D3
                      | in (A3) speichern
MOVE.L D3, -(SP)      | Zwischenwert D3 auf Stack
                      |
...                   | tue gutes
ADD.L D1, D3          | D3 := D1 + D3
                      |
...                   | tue gutes
MOVE.L (A3), D3       | Zwischenwert von D3 zurückholen
MOVE.L (SP)+, D3      | Wert vom Stack zurückholen
MOVE.L D3, (A2)       | D3 als Ergebnis zurücklegen
                      |
MOVE.L D1, (A1)       | D1 als Ergebnis zurücklegen
MOVEM.L (SP)+, D1-D3/A1/A2 Register zurücksetzen.
RTS                   | das war es
```

nicht rechtzeitig keine Ahnung mehr

(d) **(2 Punkte)** Zeichnen Sie den Kellerspeicher auf, wie er unmittelbar nach Aufruf des Unterprogramms aussieht.

Ein Kästchen entspricht 4 Byte!



(12 Punkte)

Aufgabe 3: Semaphore

Im Raum EN268 steht ein Rechner für Info-4-Übungsaufgaben. An diesem Rechner versammeln sich nacheinander maximal 4 Studis, die spontan eine Arbeitsgruppe bilden. Jede(r) Studi kann den Raum jederzeit wieder verlassen. Wenn eines der am Rechner versammelten Arbeitsgruppenmitglieder den Rechner verlässt, darf sich kein fremdes Arbeitsgruppenmitglied an diesen Rechner setzen. Erst wenn der Rechner wieder völlig frei ist, darf sich dort die nächste Arbeitsgruppe bilden.

Im folgenden ist ein Arbeitgruppenmitglied als Prozesstyp **Studi** modelliert und mit Hilfe von Semaphoren synchronisiert. Eine neue Instanz dieses Prozesses wird für jede(n) den Raum betretende(n) Studi gestartet. Anhand des gegebenen Programms lassen sich die oben geforderten Synchronisationsanforderungen formal so charakterisieren:

1. Es dürfen nie mehr als 4 Prozesse gleichzeitig in Zeile 270 stehen (Arbeit am Rechner). K A
2. Nachdem ein Prozess Zeile 440 passiert hat (Verlassen des Rechners), darf so lange kein später gestarteter Prozess die Zeile 270 (Arbeit am Rechner) erreichen, bis alle dort aktiven Prozesse die Zeile 270 verlassen haben.

Die angegebene Implementierung hat genau **drei Fehler**. Jeder dieser Fehler lässt sich durch die **Änderung eines einzelnen Zeichens** (Buchstabe, Ziffer oder Symbol) im Quelltext beheben. **Zeilennummern, Kommentare** und **Leerzeichen** sind nicht betroffen!

Empfehlung:

Am leichtesten und am schnellsten geht es, wenn Sie alle Aufgabenteile in der vorliegenden Reihenfolge bearbeiten! Bitte suchen oder korrigieren Sie **nicht** zuerst die Fehler, sondern lösen Sie **zuerst Aufgabenteil (a)**!

Sie können die Seite mit dem Programm ruhig **herausreißen**, um das Blättern zu sparen!

Globale Deklarationen:

```
100 ArbeitZähler  : INTEGER = 0   // Anzahl Studis in der Gruppe
110 FertigZähler  : INTEGER = 0   // Anzahl schon weggegangener Studis
120 ZählerSchutz  : Semaphor(1)
130 Studis        : Semaphor(4)
140 Halt          : Semaphor(1)
```

Prozess Studi:

```
200 -- Studi betritt den Raum
210 Studis.P        -- warten bei mehr als 4 Studis
220 Halt.P          -- warten nach erstem Weggang
230 Halt.V
240 ZählerSchutz.P
250   ArbeitZähler = ArbeitZähler + 1
260 ZählerSchutz.V
270 -- Gruppenarbeit am Rechner
280 ZählerSchutz.P
290   FertigZähler = FertigZähler + 1
300   IF FertigZähler = 1 THEN
310     -- erster fertiger Studi sperrt weiteren Zugang
320     Halt.P
330   END IF
340   IF ArbeitZähler = FertigZähler THEN
350     -- letzter weggehender Studi gibt alles wieder frei
360     Halt.V
370     WHILE ArbeitZähler > 0 DO
380       Studis.V
390       ArbeitZähler = ArbeitZähler - 1
400       FertigZähler = FertigZähler - 1
410     END WHILE
420   END IF
430 ZählerSchutz.V
440 -- Studi verlässt den Rechner
```

d.h. alle arbeitenden Mitglieder sind fertig.

Alles freigeben.

(b) **(6 Punkte)** Die gegebene Implementierung hat genau 3 Fehler, die durch eine Änderung jeweils eines einzelnen Zeichens (Buchstabe, Ziffer oder Symbol) behebbar sind. Auswirkungen aller Fehler haben sich in Aufgabenteil (a) gezeigt, wenigstens im Endzustand der Variablen. Geben Sie in der folgenden Tabelle **Zeilennummer** und nötige **Zeichenänderung** an und beschreiben Sie die **praktische Auswirkung** jedes Fehlers sehr kurz in Stichpunkten, ohne Begründung.

Programmzeile und nötige Korrektur	**praktische Auswirkung der fehlerhaften Zeile**
Z 300: if F Z = 1	F Z = 0 bed. keiner ist fertig, obwohl einer dies schon ist => keine Sperre.
Z 360: Hilf. V.	keine rechtzeitige Hilfe.
Z 400: F Z = F Z - 1	Nachdem der letzte Mitglied der nächsten Gruppe fertig ist wird der Zugang trotzdem nicht blockiert.

(4 Punkte)

Aufgabe 4: Monitore

(a) **(2 Punkte)** Es existiert ein Objekt der unten dargestellten Klasse *Counter*, die zwei Methoden mit dem Schlüsselwort *synchronized* enthält. Welche Auswirkungen ergeben sich daraus für die Ausführung der beiden Methoden *up()* und *down()*, wenn diese in beliebiger Reihenfolge von zwei Prozessen ausgeführt werden? Begründen Sie Ihre Antwort!

die Methode werden unabhängig voneinander ausgeführt

```
class Counter {
   private long count = 0;

   public synchronized void up() {
      count = count + 1;
   }

   public synchronized void down() {
      count = count - 1;
   }
}
```

weil synchronized

(b) **(2 Punkte)** Angenommen es gibt zwei Objekte der Klasse *Counter* aus Aufgabenteil (a). Wie viele anonyme Bedingungsvariablen (Sperren) gibt es? Begründen Sie Ihre Antwort!

(5 Punkte)

Aufgabe 5: Monitore

Modellierung eines Tischtennisspiels

Bei einem Tischtennisspiel mit zwei Mitspielern wird ein Ball abwechselnd hin- und hergespielt. Dieser Ablauf wird in diesem Beispiel mit Hilfe von Prozessen (Java-Threads) und den zur Verfügung stehenden Synchronisationsmethoden nachgebildet. Zwei Prozesse (mit eindeutigen Namen) existieren für die beiden Mitspieler, die jeweils in einer Endlosschleife die Methode *hit(Gegner)* aufrufen, wobei *Gegner* der Name des anderen Mitspieler-Prozesses ist. Außerdem gibt es die unten dargestellte Klasse *PingPong*, die diese Methode *hit(Gegner)* beinhaltet und damit die Synchronisation zwischen den beiden Spieler-Prozessen übernimmt.

Hinweise:

- Mit der Methode *Thread.currentThread().getName()* kann der Name des aktuellen Prozesses abgefragt werden.
- Die Methode *String.compareTo(String)* liefert 0, wenn beide Strings identisch sind.

```
public class PingPong {
   private String WerIstDran = null;

   public synchronized boolean hit(String Gegner) {
      String Name = Thread.currentThread().getName();

      // Erster Prozess wird erster Spieler
      if (WerIstDran == null) {
         WerIstDran = Name;
         return true;                    // weiterspielen
      }

      if (Name.compareTo(WerIstDran) == 0) {
         System.out.println("PING! "+Name);
         WerIstDran = Gegner;
         notifyAll();
      }
      else {
         try {
            long t1 = System.currentTimeMillis();
            wait(2500);
            if ((System.currentTimeMillis() - t1) > 2500)
               System.out.println("****** TIMEOUT!!");
         } catch (InterruptedException e) { }
      }
      return true;                       // weiterspielen
   }
}
```

(a) **(3 Punkte)** Die beiden existierenden Spieler spielen noch. Angenommen es gibt einen dritten Spieler-Prozess mit einem neuen eindeutigen Namen. Was passiert mit dem Spielablauf, wenn dieser Prozess keinen Gegner (*Gegner=null*) hat? Und was passiert, wenn dieser Prozess als Gegner einen der beiden ersten Mitspieler hat? Begründen Sie Ihre Antworten!

(b) **(3 Punkte)** Wenn in der Implementierung der PingPong-Klasse nicht *notifyAll()* sondern *notify()* verwendet werden würde, welche Auswirkungen hätte dies. wenn nur die ersten beiden Mitspieler existieren? Und was wäre, wenn der dritte Spieler ohne Gegner hinzukommen würde? Begründen Sie Ihre Antworten!

Aufgabe 6 : Speicherverwaltung

(7 Punkte)

(a) **(1 Punkte)** Gilt bei seitenverwalteten Systemen folgende Aussage?
„Je größer die Anzahl von Kacheln, desto kleiner wird die Anzahl der Seitenwechsel."

es gibt hier keine Auswahlprozesse, da nur Prozesse von einem einzigen Typ vorhanden sind.

(a) **(6 Punkte)** Simulieren Sie von Hand die in der Tabelle angegebene Abfolge von Prozessaktivitäten für die gegebene Implementierung. Pro Tabellenzeile läuft **nur ein einziger Prozess**. Er läuft genau bis zu dem Zeitpunkt, zu dem entweder **Zeile 270** (Arbeit am Rechner) oder **Zeile 440** (Prozessende) erreicht ist oder aber eine **Blockierung** eintritt. Geben Sie für diesen Zeitpunkt jeweils an: den Wert der beiden INTEGER-Variablen, den der drei Semaphorzähler (`<Semaphorname>.n`), die in dieser Implementierung erreichte Programmzeile und die von einer **korrekten Implementierung gemäß der Aufgabe** zu erreichende Zeile (falls abweichend).

Unveränderte Werte brauchen nicht aufgeschrieben zu werden. Als Hilfestellung ist Platz für jeweils einen Zwischenwert (**nicht immer für alle Zwischenwerte!**) vorgesehen, aber nur der **Endzustand** pro Tabellenzeile wird bewertet.

	Ereignis	`ArbeitZähler`	`FertigZähler`	`ZählerSchutz.n`	`Studis.n` Anzahl Studis	`Halt.n` Gruppe n - Verwaltung	Prozess erreicht Zeile	richtig wäre Zeile
	Anfangszustand	0	0	1	4	1		
G1.	Studi 1 kommt an	1		0	3	0	270	
				1		1		
G1.	Studi 2 kommt an	2	0	0	2	0	270	
				1		1		
Verwandt!	Studi 2 verlässt den Rechner		1	0			440	
				1				
G2	Studi 3 kommt an	3		0	1	0	270	220
				1		1		
	Studi 1 verlässt den Rechner		2	0			440	
				1				
G2	Studi 3 läuft weiter		3	0		0	440	270
		0	6	1	4			
G2.	Studi 4 kommt an	0			3	-1	270 blockiert	270

300: Fertigzähler = 1

360 ; Halt. V

400 - Fertigzähler - 1

keine Sperre nach erstem Weggang

Deadlock bei letzten Studis der ersten oder ersten der zweite Gruppe

keine Sperre nach zweiten Weggang ab 2. Gruppe (oder Fertigzähler wird nicht auf 0 zurückgesetzt)

	AZ	FZ	Schub	Studis Max: 4	Halt Vermeidung	erreicht Ziel	richtig Ziel erreicht
Init.	0	0	1	4	1	/	/
S1 an	1		0 1	3 3	0 1	270	
S2 an	2		0 1	2	0 1	270	
S2 weg		1 abtitiert	0 1		0	440	
S3 an	3 – ? sollte 1 werden		0 1	1	0 1	270	220 = S3 sollte bl. werden
S1 weg	– sollte 0 sein	2	0 1			440	
S3 weiter	0	3 6	0 1	4	0	440	270
S4 an				3	1 – S4 block	220	270