Die folgenden Aufgaben entstammen aus alten Klausuren und sind nach Themen geordnet. Der Informatik 4-Teil zur Informatik B Prüfung bestand aus jeweils 5 bis 6 Fragen verschiedener Themen, die zusammen 50 Punkte ergaben.

Zu einigen Fragen sind Lösungsvorschläge gegeben, die dann grau hinterlegt sind. Diese Lösungsvorschläge erheben nicht den Anspruch, mustergültig zu sein, helfen jedoch beim Üben des Stoffs für diesen Prüfungsteil. Die Fragen ohne Lösungsvorschlag sind zum Vertiefen gedacht.

Da es sich um alte Aufgaben handelt, wird ausdrücklich darauf hingewiesen, daß sich der Prüfungsstoff über alle Gebiete der Vorlesung und des Übungsbetriebs des jeweiligen Veranstaltungsjahrs erstreckt. Es ist möglich, daß Fragen zu Themen vorkommen, die hier nicht aufgeführt sind.

Viel Erfolg!

Grundlagen von Betriebssystemen **(4 Punkte)**

Definieren oder beschreiben Sie die folgenden Begriffe **kurz** in ein oder zwei Sätzen. Erläutern Sie dabei, was der Begriff darstellt und wozu so etwas von Betriebssystemen verwendet wird.

(a) Maskierung von Unterbrechungen beim MC680x0

unterstützt 7 verschiedene Unterbrechungsebenen.

Ebene 1 bis 6 können jeweils von einer Unterbrechung höherer Priorität selbst wieder unterbrochen werden.

maskierbare Unterbrechungen: Die Bearbeitung der aktuellen Unterbrechung wird erst nach Beendigung der dringlicheren Unterbrechung fortgesetzt

nicht maskierbare Unterbrechung: Unterbrechungen der Ebene 7 haben höchste Priorität und können sich selbst unterbrechen

(b) Prozeßbeschreibungsblock (Erläutern Sie den Begriff und nennen Sie 4 Komponenten)

Betriebssystem legt Verwaltungsdaten über Prozesse an. Für jeden Prozeß wird ein Prozeßbeschreibungsblock angelegt.

Komponenten: Eindeutige Identifikation – Benutzer, der Prozeß gestartet hat – Startparameter – Identifikation des Elternprozesses – bisherige Laufzeit – Priorität – Zustand – Prozessorstatus – benötigte Betriebsmittel

(c) Verklemmung

Durch Synchronisationsmaßnahmen können Prozesse blockiert werden. In ungünstigen Fällen können Prozesse nicht fortgesetzt werden. Dies ist der Fall, wenn Prozesse auf Ereignisse warten, die nicht mehr eintreten können. Diese Situation wird Verklemmung genannt.

(d) Belegungsstrategie bei der Speicherzuordnung

Belegungsstrategie legt fest, wo Information im Hauptspeicher abgelegt wird. Wählt einen Teil eines freien Speicherbereichs aus.

Grundlagen von Betriebssystemen **(4 Punkte)**

Definieren oder beschreiben Sie die folgenden Begriffe **kurz** in ein oder zwei Sätzen. Erläutern Sie dabei, was der Begriff darstellt und wozu so etwas von Betriebssystemen verwendet wird.

(a) Prozeßabzweigung

In einem Prozeß (Elternprozeß) wird ein weiterer Prozeß (Kindprozeß) erzeugt, der nebenläufig zum Elternprozeß ausgeführt wird.

(b) Prozeßzustand (Erläutern Sie den Begriff und nennen Sie 4 Zustände)

Zustand charakterisiert Ablauffähigkeit eines Prozesses (unbekannt, existent, bereit, blockiert, laufend, beendet).

(c) Geräte-Treiber

Programm bzw. Prozeß, der die Ein-/Ausgabe von/auf einem Peripheriegerät bearbeitet. E/A-Aufträge werden in Warteschlange eingereiht, Treiber arbeitet Warteschalnge ab.

(d) Freispeicherliste

Liste mit den freien Speicherbereichen in einem segmentverwalteten System.

Grundlagen von Betriebssystemen **(4 Punkte)**

Beantworten Sie folgende Fragen **kurz** in ein oder zwei Sätzen.

(a) **(1 Punkt)** Wann ist ein Programm **deterministisch**?
(b) **(1 Punkt)** Wann ist ein Programm **determiniert**?
(c) **(1 Punkt)** Was versteht man unter dem **Lokalitätsprinzip**?
(d) **(1 Punkt)** Wodurch unterscheiden sich **asynchrone** und **synchrone** Unterbrechungen?

Grundlagen von Betriebssystemen **(5 Punkte)**

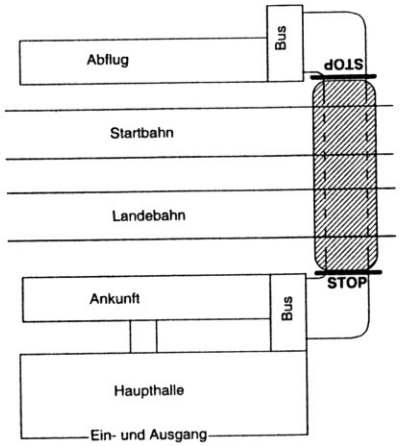
Beantworten Sie folgende Fragen **kurz** in ein oder zwei Sätzen.

(a) **(1 Punkt)** Was versteht man unter **Nebenläufigkeit**?
(b) **(1 Punkt)** Wann ist ein Unterprogramm **wiedereintrittsfest**?
(c) **(1 Punkt)** Was versteht man unter **Koroutinen**?
(d) **(1 Punkt)** Wann spricht man von **synchronem** Nachrichtenaustausch, wann von **asynchronem**?
(e) **(1 Punkte)** Skizzieren Sie den **Prozeßfolgegraph** für folgendes Prozeßsystem:

```
P1;
Fork P3;
Fork P5;
P2;
Join P3;
P4;
Join P5;
P6;
```

Semaphore **(12 Punkte)**

Der unten aufgezeichnete Flughafen hat eine Haupthalle, über die alle Fluggäste zum Abflug- bzw. Ankunftbereich gelangen. Aus der Haupthalle ist der Ankunftbereich zu Fuß zu erreichen. Zwischen der Haupthalle und dem Abflugbereich verkehren Busse, die aber die Start- bzw. Landebahn kreuzen müssen.



Es gelten folgende Regeln für den Flug- und Busbetrieb:

1. Es darf höchstens ein Flugzeug auf der Landebahn landen.
2. Es darf höchstens ein Flugzeug auf der Startbahn starten.
3. Busse dürfen die Start- und Landebahn nur überqueren, wenn kein Flugzeuge startet oder landet.
4. Wenn sich mindestens ein Bus auf dem Rollfeld im gestrichelten Bereich befindet, darf nicht gestartet oder gelandet werden.
5. Um eine möglichst kurze Zeit zur Räumung der Start- und Landebahnen zu gewährleisten, dürfen maximal 4 Busse in den schraffierten Bereich zwischen den STOP-Linien einfahren.

(a) Schreiben Sie ein nebenläufiges Programm, das einen sicheren Betrieb auf dem Flughafen nach den obigen Regeln sicherstellt. Die startenden und landenden Flugzeuge sowie die Busse sind als nebenläufige Prozesse darzustellen.


```
Startbahn, Landebahn, BZ_Schutz : sema (1);
Busse sema (4);
BZ : INTEGER := 0;
```

```
LandendesFlugzeug: PROCESS
BEGIN
  Landebahn•P
  -- Flieger landet
  Landebahn•V
END PROCESS
```

```
StartendesFlugzeug: PROCESS
BEGIN
  Startbahn•P
  -- Flieger startet
  Startbahn•V
END PROCESS
```

```
Bus: PROCESS
BEGIN
  Busse•P
  BZ_Schutz•P
    INC (BZ)
    IF BZ = 1 THEN
      Startbahn•P
      Landebahn•P
    ENDIF
  BZ_Schutz•V

  -- tucker tucker

  BZ_Schutz•P
    DEC (BZ)
    IF BZ = 0 THEN
      Startbahn•V
      Landebahn•V
    ENDIF
  BZ_Schutz•V
  Busse•V
END PROCESS
```

(b) Regel 5 wird wie folgt geändert:

5. Um eine möglichst kurze Zeit zur Räumung der Start- und Landebahnen zu gewährleisten, dürfen maximal 100 Busse in den schraffierten Bereich zwischen den STOP-Linien einfahren.

Danach kommt es zu enormen Verspätungen im Flugbetrieb. Was ist passiert? Beschreiben Sie kurz verbal ein Konzept (kein Programm!), wie mittels Semaphoren Abhilfe geschaffen werden kann!

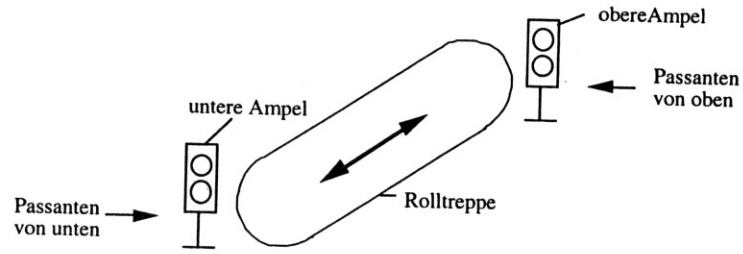
Sobald ein Bus in den Sperrbereich einfährt, werden die Landebahnen gesperrt und Flugzeuge gestoppt. Es können bis zu 100 Busse in den Bereich einfahren, und solange auch nur ein Bus im Sperrbereich ist, können immer wieder neue Busse einfahren und so werden die Flugzeuge ausgehungert.

Abhilfe: Mittels Semaphoren sind die Flieger so zu priorisieren, daß der Start- bzw. Landewunsch eines Flieger dazu führt, daß keine weiteren Busse in den Sperrbereich einfahren dürfen. Dann können die Busse noch den Sperrbereich verlassen, danach starten und landen Flieger (und können die Busse aushungern!).

Priorisierung kann durch Einführen von „Vorhalt" bei den Bussen realisiert werden.

Semaphore **(8 Punkte)**

Das Bild zeigt eine gegenläufige Rolltreppe, die über zwei Ampeln gesteuert wird:



Es gilt:.

- Passanten kommen von unten und von oben.
- Die Rolltreppe läuft entweder auf- oder abwärts oder steht still.
- Die beiden Ampeln steuern den Zugang zur Rolltreppe: bei grün darf die Rolltreppe betreten werden, bei rot muß gewartet werden.
- Ein Passant darf die Rolltreppe betreten, wenn sie bereits in die richtige Richtung läuft oder wenn sie stillsteht (wird dann eingeschaltet).
- Es dürfen maximal 20 Passanten die Rolltreppe betreten.
- Wenn der letzte Passant die Rolltreppe verläßt, wird die Rolltreppe angehalten.

Der Ablauf soll durch ein nebenläufiges Programm in der aus der Vorlesung bekannten Notation beschrieben werden. Die Synchronisation soll durch Semaphore erfolgen. Die Passanten werden durch zwei Prozeßtypen dargestellt:

- **type** passant_unten_typ = **process**
- **type** passant_oben_typ = **process**

Hinweis: Das Ein- und Umschalten der Rolltreppe und das Schalten der Ampel kann in Kommentaren beschrieben werden.

Schreiben Sie ein Programm unter Verwendung von Semaphoren, so daß Passanten von unten bevorzugt werden:

- Wenn die Rolltreppe abwärts läuft, und unten ein Passant ankommt, so schaltet die obere Ampel auf rot.
- Nachdem der letzte abwärts fahrende Passant die Rolltreppe verlassen hat, wird die Laufrichtung umgeschaltet und die unten wartenden Passanten dürfen die Rolltreppe betreten.

```
-- Variablen und Semaphore:

pu_z, po_z : integer := 0;

pu, po : semaphore(20);

pu_z_schutz, po_z_schutz,
rolltreppe, halt, vorhalt :
semaphore(1);
```

```
type passant_unten_typ = process

pu•P;

  pu_z_schutz•P;
    pu_z++;
    if pu_z=1 then
      halt•P;
      rolltreppe•P;
      -- obere Ampel auf rot
      -- Rolltreppe aufwärts
    end if;
  pu_z_schutz•V

  -- hochfahren

  pu_z_schutz•P;
    pu_z--;
    if pu_z=0 then
      rolltreppe•V;
      Halt•V;
      -- Rolltreppe aus
      -- obere Ampel auf grün
    end if;
  pu_z_schutz•V

pu•V

end process;
```

```
type passant_oben_typ = process

po•P;

 vorhalt•P;
  halt•P;
   po_z_schutz•P;
    po_z++;
    if po_z=1 then
     rolltreppe•P;
     -- untere Ampel auf rot
     -- Rolltreppe abwärts
    end if;
   po_z_schutz•V
  halt•V;
 vorhalt•V;

 -- runterfahren

 po_z_schutz•P;
  po_z--;
  if po_z=0 then
   rolltreppe•V;
   -- Rolltreppe aus
   -- untere Ampel auf grün
  end if;
 po_z_schutz•V

po•V

end process;
```


Semaphore **(12 Punkte)**

Eine Brücke kann von Autos und Fahrrädern befahren werden. Auf der Brücke dürfen sich nie mehr als 4 Autos und nie mehr als 10 Fahrräder befinden. Wenn mehr als zwei Autos auf der Brücke sind, dürfen dort gleichzeitig keine Fahräder sein. Weil Radfahrer vorsichtig sein müssen, befahren neu eintreffende Fahrräder **nicht** die Brücke, wenn schon ein oder mehr Autos darauf sind. Autofahrer neigen dazu, sich vorzudrängeln, Autos haben also Priorität vor Fahrrädern: Neu eintreffende Autos können wartende Fahrräder überholen.

Im folgenden ist eine **fehlerhafte** Implementierung dieses Synchronisationsproblems angegeben:

Globale Deklarationen:

```
100 Fahrrad_Schutz    :     Semaphor(1)
110 Auto_Schutz       :     Semaphor(1)
120 Fahrrad           :     Semaphor(10)
130 Auto              :     Semaphor(4)
140 Ausschluß         :     Semaphor(1)
150 Halt              :     Semaphor(1)
160 Vorhalt           :     Semaphor(1)
170 Fahrrad_Zähler    :     INTEGER = 0
180 Auto_Zähler       :     INTEGER = 0
```

Prozeß Fahrrad:

```
200 Vorhalt.P
210   Halt.P
220     Fahrrad_Schutz.P
230       INC(Fahrrad_Zähler)
240       IF Fahrrad_Zähler = 1 THEN
250         Ausschluß.P
260       END IF
270     Fahrrad_Schutz.V
280   Halt.V
290 Vorhalt.V
300 Fahrrad.P
310   -- Brücke benutzen
320 Fahrrad.V
330 Fahrrad_Schutz.P
340   DEC(Fahrrad_Zähler)
350   IF Fahrrad_Zähler = 0 THEN
360     Ausschluß.V
370   END IF
380 Fahrrad_Schutz.V
```


Prozeß Auto:

```
400 Auto_Schutz.P
410   INC(Auto_Zähler)
420   IF Auto_Zähler = 1 THEN
430     Ausschluß.P
440   ELSE IF Auto_Zähler = 3 THEN
450     Halt.P
460   END IF
470 Auto_Schutz.V
480 Auto.P
490   -- Brücke benutzen
500 Auto.V
510 Auto_Schutz.P
520   DEC(Auto_Zähler)
530   IF Auto_Zähler = 0 THEN
540     Ausschluß.V
550   ELSE IF Auto_Zähler = 2 THEN
560     Halt.V
570   END IF
580 Auto_Schutz.V
```

(a) **(3 Punkte)** Simulieren Sie von Hand die in der Tabelle angegebene Abfolge von nacheinander eintretenden Ereignissen **für die gegebene Implementierung**. Prozesse verlassen die Brücke nur dann, wenn das links angegeben ist. Geben Sie pro Zeile jeweils den Endzustand an, bei dem jeder Prozeß entweder blockiert oder im kritischen Abschnitt angekommen oder komplett abgearbeitet ist. Werte, die mit der Vorgängerzeile überstimmen, brauchen nicht aufgeschrieben zu werden. Die Notation „**n**" steht für die Semaphorzähler.

Ereignis	**Fahrrad_Zähler**	**Auto_Zähler**	**Fahrrad_Schutz.n**	**Auto_Schutz.n**	**Fahhrad.n**	**Auto.n**	**Ausschluß.n**	**Halt.n**	**Vorhalt.n**	**Fahrräder auf Brücke**	**Autos auf Brücke**
Anfangszustand (Programmstart)											
2 Autos kommen an											
1 Fahrrad kommt an											
1 Auto kommt an											
2 Autos verlassen die Brücke											

(b) **(9 Punkte)** Die gegebene Implementierung hat 3 logische Fehler (die jedoch mehr als 3 Stellen im Programmtext betreffen und nicht alle in Teil a aufgetreten sein müssen). Geben Sie in der folgenden Tabelle zu jedem dieser Fehler zunächst seine Auswirkung an, indem Sie die das unerwünschte Verhalten beschreiben, das der Fehler bedingt. Beschreiben Sie dann die nötige Korrektur **in Stichpunktform,** Programmtext ist also **nicht** erforderlich. (Es gibt auch einen Teil der Punkte, wenn nur das Fehlverhalten oder nur die Korrektur beschrieben sind.)

Fehlverhalten	**Korrekturmaßnahme**

Monitore **(10 Punkte)**

Im Folgenden ist die *main*-Methode einer Klasse implementiert, die einen neuen Thread mit Hilfe der Klasse *MyThread* erzeugt. Anschließend werden einige Berechnungen durchgeführt, die hier nur durch den Kommentar „arbeiten…" dargestellt sind.
Die *MyThread*-Klasse ist weiter unten dargestellt und läuft ebenfalls in einer Endlosschleife. Dort wird zwischen den Berechnungen (die wieder nur durch einen Kommentar angedeutet sind) durch aktives Warten in einer WHILE-Schleife gewartet, solange die Variable *warten* auf *true* gesetzt ist.

```
public void main()
{
   // Thread erzeugen und starten
   MyThread mythread = new MyThread();
   mythread.start();

   while (true)
   {
      // arbeiten…

      // Thread anhalten
      mythread.warten = true;

      // arbeiten…

      // Thread weiterlaufen lassen
      mythread.warten = false;

      // arbeiten…
   }
}


public class MyThread extends Thread
{
   public boolean warten = false;

   // wird ausgeführt, wenn der Thread gestartet
wurde
   public void run()
   {
      while (true)
      {
         // arbeiten…

         // ich warte wenn ich soll
         while (warten)
         {
         }

         // arbeiten…
      }
   }
}
```


(a) **(2 Punkte)** Bei dem gegebenen Beispielprogramm auf der vorherigen Seite werden nach dem Start der *main*-Methode zwei Prozesse nebenläufig ausgeführt. Ist in diesem Fall eine Synchronisation zwischen beiden Prozessen notwendig? Begründen Sie Ihre Antwort!

Eine Synchronisation ist notwendig, wenn man davon ausgeht, dass Zuweisungs- und Abfragefunktionen nicht atomar sind, denn beide Prozesse greifen auf die Variable *warten* zu, wobei der eine Prozess die Variable ändert und der andere diese abfragt.

Wenn man davon ausgeht, dass die Zuweisungs- und Abfragefunktionen atomar sind, dann ist keine Synchronisation notwendig, weil es in diesem Fall zu keinen Inkonsistenzen zwischen den Änderungen der Variable warten und den Abfragen des anderen Prozesses kommen kann.

(b) **(5 Punkte)** In dem gegebenen Beispiel wartet der Thread *MyThread* durch aktives Warten. Ergänzen bzw. ändern Sie das Beispielprogramm so, dass das Warten über die *wait*-Methode erfolgt! Kommentieren Sie Ihre eigenen Änderungen, indem sie deutlich den Sinn und Zweck einer Änderung und falls immer möglich auch das Synchronisationsobjekt nennen!
Beachten Sie, in welchem Kontext die *wait*-Methode ausgeführt werden darf und an welchen Programmstellen Ausnahmen (*Exceptions*) abgefangen werden müssen!
Die Ergänzungen bzw. Änderungen können im folgenden Quelltextfragment (welches beliebig verändert werden kann!) oder auf einer Extraseite gemacht werden.

```
public void main()
{
   MyThread mythread = new MyThread();
   mythread.start();
   while (true)
   {
      // arbeiten…

      // Thread anhalten
      synchronized (mythread) {
         mythread.warten = true;
      }

      // arbeiten…

      // Thread weiterlaufen lassen
      synchronized (mythread) {
         mythread.warten = false;
         mythread.notify();
      }

      // arbeiten…
   }
}

public class MyThread extends Thread
{

   public boolean warten = false;

   public void run()
   {

      while (true)
      {
         // arbeiten…

         // ich warte wenn ich soll
         synchronized (this) {
            while (warten)
            {
               try {
                  wait();
               } catch (Exception e) {}
            }
         }

         // arbeiten…
      }
   }
}
```


(c) **(2 Punkte)** Angenommen es würde mehrere Abhängigkeiten zwischen beiden Prozessen in den nicht näher spezifizierten Zeilen mit dem Kommentar „arbeiten…" geben und man würde die beiden Methoden *main* und *run* durch Zuhilfenahme der *synchronized*-Anweisung wie folgt spezifizieren:

```
public synchronized void main() {…}
public synchronized void run() {…}
```

Wären dann alle Synchronisationsprobleme beider Prozesse ohne Rücksicht auf die Effizienz gelöst? Begründen Sie Ihre Antwort!

Nein, die Synchronisationsprobleme wären in diesem Fall nicht gelöst, weil beide Prozesse auf ein anderes Synchronisationsobjekt synchronisieren würden und es deshalb zu keiner Synchronisation zwischen den Prozessen kommt.

(d) **(1 Punkt)** Welche der folgenden Variablen können als Synchronisationsobjekte benutzt werden?

a) int syncint;
b) Object syncobject;
c) String syncstring;
d) Integer syncinteger;

b, c und d können als Synchronisationsobjekte benutzt werden, weil es Klassen sind, die automatisch von der Klasse Object erben.
Eine Variable vom Typ int kann nicht als Synchronisationsobjekt benutzt werden, weil dies ein eingebauter Datentyp von Java ist.

Nachrichten

(10 Punkte)

Drei Prozesse *Produzent1*, *Produzent2* und *Konsument* tauschen Daten als Nachrichten aus. Dabei erzeugen die Prozesse *Produzent1* und *Produzent2* Daten, die sie an den Prozeß *Konsument* senden. Jedes Datum wird sofort nach dem Erzeugen versendet. Das Erzeugen geschieht zyklisch, das heißt unendlich häufig. Der Prozeß *Konsument* empfängt die Daten in der Reihenfolge des Versendens, verarbeitet sie, und ist dann wieder empfangsbereit.

Zur Nachrichtenübertragung stehen zwei Puffer der Länge N zur Verfügung. Das Senden in einen Puffer geschieht mit der Operation

```
PROCEDURE send(k: TKanal, Datum: TNachricht): INTEGER.
```

Analog kann eine Nachricht empfangen werden durch Aufruf der Prozedur

```
PROCEDURE receive(k: TKanal, VAR Datum: TNachricht): INTEGER.
```

Beim Empfangen werden die Nachrichten automatisch aus dem Puffer entfernt.
Beide Prozeduren geben einen Fehlerwert (-1) zurück, wenn versucht wurde, in einen vollen Puffer zu schreiben bzw. aus einem leeren Puffer zu lesen. Sonst ist der Rückgabewert 0.

Das Senden und Empfangen soll folgende Bedingungen sicherstellen:

- Es dürfen keine Daten verloren gehen.
- Der *Konsument* soll nicht blockiert werden, wenn der Puffer von einem der beiden sendenden Prozesse leer ist. (D.h. wenn von *Produzent1* keine aktuellen Daten vorliegen, müssen noch Daten von *Produzent2* empfangen werden können und umgekehrt.)
- Ein Produzent soll blockiert werden, solange er ein Datum wegen eines vollen Puffers nicht versenden kann.
- Keiner der Prozesse *Produzent1* oder *Produzent2* soll vom Prozeß *Konsument* bevorzugt werden.

(a) (1 Punkt)

Geben Sie an, ob die Sende- und Empfangsoperationen blockierend, pufferblockierend oder nicht blockierend sein sollten.

send:

Entweder **pufferblockierend** oder **nicht blockierend** mit busy waiting.

receive:

Nicht blockierend, damit Konsument nicht bei einem leeren Puffer blockiert wird.

(b) (9 Punkte)

Vervollständigen Sie die Vorgaben auf den folgenden Seiten für die drei Prozesse. Verwenden Sie die Sende - und Empfangsoperationen wie Sie sie in Aufgabenteil (a) festgelegt haben. Sie dürfen die Anweisung zur nichtdeterministischen Auswahl verwenden, falls es sinnvoll ist. Auch die Rückgabewerte der Sende- und Empfangsprozeduren dürfen abgefragt und zur Steuerung des Programmablaufs verwendet werden.


```
-- Hier ist Platz für alle globalen Variablen!!
kanal1, kanal2: TKanal;

Produzent1: process
VAR Datum: TNachricht;
  dummy: INTEGER;
BEGIN
REPEAT
  -- Datum produzieren
  -- Hier: pufferblockierendes Senden
  dummy := send (kanal1, datum);
  --alternativ:
  --nicht blockierendes Senden
  --repeat dummy :=
  --send (kanal1, datum); until dummy
  <> -1)
UNTIL FALSE;
end process;

Produzent2: process
VAR Datum: TNachricht;
  dummy: INTEGER;
BEGIN
REPEAT
  -- Datum produzieren
  -- Hier: pufferblockierendes Senden
  dummy := send (kanal2, datum);
  --alternativ:
  --nicht blockierendes Senden
  --repeat dummy :=
  --send (kanal2, datum); until dummy
  <> -1
UNTIL FALSE;
end process;

Konsument: process
VAR Datum: TNachricht;
BEGIN
REPEAT
  select
    receive(kanal1, Datum) <> -1 →
    -- Datum von Prozeß Produzent1 verarbeiten
  [] receive(kanal2, Datum) <> -1 →
    -- Datum von Prozeß Produzent2 verarbeiten
  else NOP;
  end select;
end process;
```

Nachrichten **(9 Punkte)**

In einem System aus Sende- und Empfangsprozeß soll ein Vermittlerprozeß eingeschaltet werden, um die Sendeblockierung des Senders zu vermeiden. Die Wartezeit soll im Sendeprozeß dazu benutzt werden, zyklisch die Prozedur `TueEtwasSinnvolles` aufzurufen. Der Vermittler soll den Sender hierzu über zwei zusätzliche Kanäle über die Sendeblockierung zum Empfänger bzw. über die erfolgte Sendung informieren.

Der Empfangsprozeß ist vorgegeben. Entwickeln Sie den Sende- und Vermittlerprozeß anhand der vorbereiteten Schablonen. Beide Prozesse arbeiten in einer Endlosschleife.

Zum Nachrichtenaustausch stehen die Prozeduren

```
b_send ( nachricht: T_Nachricht; kanal: T_Kanal )
```

für das blockierende Senden und

```
b_receive ( nachricht: T_Nachricht; kanal: T_Kanal )
```

für das blockierende Empfangen zur Verfügung.
Zusätzlich können Sie die nicht-deterministische Auswahl `select` verwenden.

Globale Vereinbarungen:

```
type T_Kanal;
type T_Nachricht;
```

```
Sender : prozess




  repeat
    ErstelleAuftrag;









  end repeat;

end prozess;
```

```
Vermittler : prozess




  repeat

  end repeat;

end prozess;
```

```
Empfaenger: process

  daten : T_Nachricht;

  repeat
    b_receive(daten,DatenKanal2);
    BearbeiteAufrag;
  end repeat;

end prozess;
```


Prozesserzeugung und Nachrichten **(12 Punkte)**

(a) (2 Punkte) Nennen Sie kurz den wesentlichen funktionalen Unterschied zwischen einem Socket und einem ServerSocket in Java.

Ein ServerSocket wird benötigt, um in Java eine Kommunikation über Sockets aufzubauen. Dabei „lauscht" ein ServerSocket auf einem bestimmten Port. Wird auf diesem Port ein Verbindungsaufbau unternommen, so wird sowohl dem Client als auch dem Server ein gemeinsamer, neuer Socket für die Kommunikation miteinander zugewiesen.

(b) (8 Punkte) Schreiben Sie ein Programm, welches einen eigenständigen Server- und einen eigenständigen Client-Thread erzeugt.

Der Client-Thread soll einen String von der Tastatur einlesen und diesen an den Server-Thread schicken. Anschließend soll er den vom Server-Thread zurückgegebenen String ausgeben. Danach soll sich der Thread beenden.

Der Server-Thread soll einen String von einem Client-Thread empfangen und diesen String wieder an den Server-Thread zurücksenden (Echo-Server). Danach soll sich der Thread beenden.
Bemerkung: Es genügt, wenn in jeder Methode die IO-Exceptions nur einmal global abgefangen werden. Benutzen Sie bitte die folgenden Vorlagen.

```
//Hauptprogramm
    public class Hauptprogramm
{
   public static void main (String[] args) throws
java.io.IOException
  {
    try
    {
         // Threads erzeugen
         ServerThread prozess1 = new ServerThread();
         ClientThread prozess2 = new ClientThread();

         // P1 starten und auf das Ende warten
         prozess1.start();
         prozess2.start();
         prozess1.join();
         prozess2.join();
    } catch (Exception e) {
      System.out.println("Exception: " + e.getMessage());
         e.printStackTrace();
} }
}
```

```
// Server
import java.io.*;
import java.net.*;

public class ServerThread extends Thread {

  public void run() {

    Socket clientSocket = null;
    ServerSocket myServer = null;
    PrintWriter out = null;
    BufferedReader in = null;

    try {
      // Server und Client Sockets erzeugen
      myServer = new ServerSocket(5555);
      clientSocket = myServer.accept();

      // IO-Streams anbinden
      out=new PrintWriter(clientSocket.getOutputStream(), true);
      in=new BufferedReader(new InputStreamReader(
          clientSocket.getInputStream()));

      // Eingabe ausgeben
      out.println(in.readLine());

      // Alle Streams und Sockets schließen
      out.close();
      in.close();
      clientSocket.close();
      myServer.close();

    } catch (IOException e) {
        System.out.println("Exception: "+e.getMessage());
        e.printStackTrace();
}
```



```
// Client
import java.io.*;
import java.net.*;

public class ClientThread extends Thread {

public void run() {

    Socket mySocket = null;
    PrintWriter out = null;
    BufferedReader in = null;

    try {
      // Client Socket erzeugen und IO-Streams anbinden
      mySocket = new Socket("localhost", 5555);
      out=new PrintWriter(mySocket.getOutputStream(), true);
      in = new BufferedReader(new InputStreamReader(
          mySocket.getInputStream()));

      // stdin für Tastatureingabe erzeugen
      BufferedReader stdIn = new BufferedReader(
         new InputStreamReader(System.in));
      String userInput;

      // Eingabe einlesen und senden,
      // Ergebnis empfangen und ausgeben
      System.out.print("Client sendet: ");
      userInput = stdIn.readLine();
      out.println(userInput);
      System.out.println("empfangen: "+in.readLine());

      // Alle Streams und Sockets schließen
      out.close();
      in.close();
      stdIn.close();
      mySocket.close();

    } catch (IOException e) {
      System.out.println("Exception: " + e.getMessage());
      e.printStackTrace();
} }
}
```

(c) **(2 Punkte)** Beschreiben Sie in wenigen Stichpunkten, wie der Server-Thread geändert werden muss, damit aus dem Server ein Multiserver wird, d.h. damit der Server mehrere Clients bedienen kann?

Da die accept()-Methode des ServerSockets den aktuellen Prozess blockiert, muss in diesem Fall für jeden neuen Client vom Server ein neuer Thread erzeugt werden, der diesen Client bedient.
Diese erzeugten Threads kann man als „Inner Class" oder als eigene „normale" Klasse implementieren. Der Multiserver macht dann nix anderes, als in einer Schleife die ServerSocket.accept()-Methode auszuführen und den dabei generierten Socket an einen neu-generierten Server-Thread zu übergeben, der dann mit dem Client kommuniziert.

Speicherverwaltung **(8 Punkte)**

(a) **(1 Punkt)** Der Hauptspeicher eines Rechners verfügt über 3 Kacheln, die im Anfangszustand alle leer sind. Die Seitenanforderungen kommen in dieser Folge:

0 1 2 3 2 4 1 3 0 2 3 4

Welche Belegung der Kacheln ergibt sich, wenn zur Seitenersetzung die FIFO-Strategie angewendet wird?

K_1											
K_2											
K_3											

Markieren Sie durch Unterstreichen die Seitenfehler. (Die erste Belegung der freien Kacheln sind auch Seitenfehler.) Wieviel Seitenfehler ergeben sich?

(b) **(1 Punkt)** Der Hauptspeicher eines Rechners verfügt nun über 4 Kacheln, die im Anfangszustand alle leer sind. Die Seitenanforderungen kommen wieder in dieser Folge:

0 1 2 3 2 4 1 3 0 2 3 4

Welche Belegung der Kacheln ergibt sich, wenn zur Seitenersetzung wieder die FIFO-Strategie angewendet wird?

K_1											
K_2											
K_3											
K_4											

Markieren Sie durch Unterstreichen die Seitenfehler. (Die erste Belegung der freien Kacheln sind auch Seitenfehler.) Wieviel Seitenfehler ergeben sich?

(c) **(1 Punkt)** Der Hauptspeicher eines Rechners verfügt über 4 Kacheln, die im Anfangszustand alle leer sind. Die Seitenanforderungen kommen in dieser Folge:

0 1 2 3 2 4 1 3 0 2 3 4

Welche Belegung der Kacheln ergibt sich, wenn zur Seitenersetzung die Strategie Least Recently Used (LRU) angewendet wird?

K_1											
K_2											
K_3											
K_4											

Markieren Sie durch Unterstreichen die Seitenfehler. (Die erste Belegung der freien Kacheln sind auch Seitenfehler.) Wieviel Seitenfehler ergeben sich?

(d) **(1 Punkt)** Der Hauptspeicher eines Rechners verfügt über 4 Kacheln, die im Anfangszustand alle leer sind. Die Seitenanforderungen kommen in dieser Folge:

0 1 2 3 2 4 1 3 0 2 3 4

Welche Belegung der Kacheln ergibt sich, wenn zur Seitenersetzung die Strategie Second Chance (SC) angewendet wird?

K_1											
K_2											
K_3											
K_4											

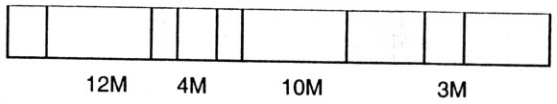
Markieren Sie durch Unterstreichen die Seitenfehler. (Die erste Belegung der freien Kacheln sind auch Seitenfehler.) Wieviel Seitenfehler ergeben sich?

(e) **(1 Punkt)** In realen Systemen findet die Seitenersetzungsstrategie LRU nur selten Verwendung, während von Second Chance abgeleitete Strategien häufig implementiert werden. Nennen sie mögliche Gründe dafür!

(f) **(3 Punkte)** Belegungsstrategien für segmentverwaltete Systeme
Im Speicher eines segmentverwalteten Systems existieren (geordnet nach aufsteigenden Adressen) freie Speicherbereiche der Größen:

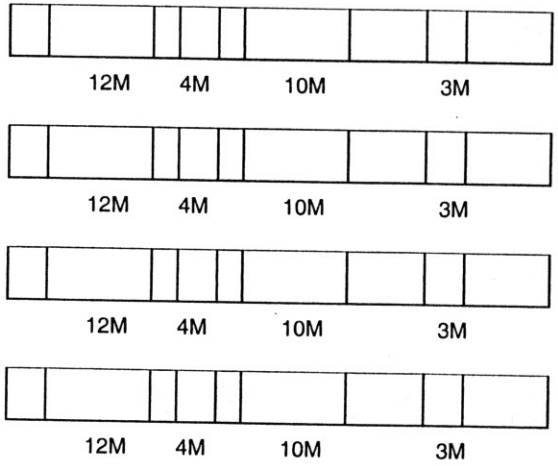
12M, 4M, 10M, 3M.

In einer groben Skizze sieht das folgendermaßen aus. (Wobei die initial belegten Speicherbereiche grau unterlegt sind.)

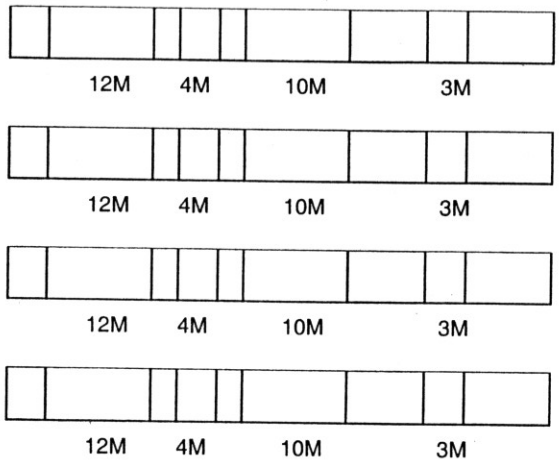


Nacheinander werden nun Segmente der Größen **7M, 4M, 3M, 9M** angefordert.

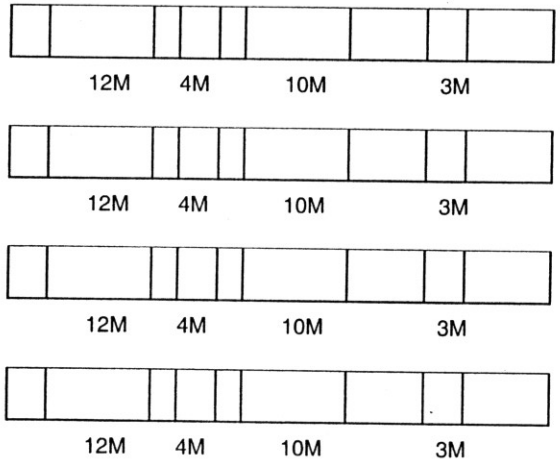
(i) Skizzieren Sie, wie die Speicherbelegung erfolgt, wenn die **First Fit** Strategie angewendet wird.



(ii) Skizzieren Sie, wie die Speicherbelegung erfolgt, wenn die **Worst Fit** Strategie angewendet wird und Segmente der Größen **7M, 4M, 3M, 9M** einzulagern sind.



(iii) Skizzieren Sie, wie die Speicherbelegung erfolgt, wenn die **Best Fit** Strategie angewendet wird und Segmente der Größen **7M, 4M, 3M, 9M** einzulagern sind.



Assembler **(12 Punkte)**

Benutzernamen – Schreiben Sie folgende Prozedur `filter` in MC68000-Assembler zur Einbindung in C-Programme (Es soll der in der Übung benutzte C-Compiler verwendet werden.):

```
void filter (int *Anzahl, int *Vergessliche);
```

Für die Lehrveranstaltung Informatik 4 haben sich die Studenten am Computer angemeldet. Folgende Angaben wurden erfragt: Matrikelnummer, Name, Vorname und Benutzername. Einige Studierenden haben leider vergessen, ihre Matrikelnummer anzugeben. Da die Veranstalter die Matrikelnummer zur Benotung der Klausur benötigen, sollen die Benutzernamen der vergesslichen Studenten aus der Liste aller Benutzernamen herausgesucht werden, damit die Matrikelnummer erfragt werden kann.

Beispiel für die Datensätze:

Matrikelnummer	Name	Vorname	Benutzername
134545	Meier	Sarah	samie
0	Vergesslich	Peter	schlunzi
100678	Stutz	Hans	hansi
.			
.			
999999	Ende		

Es soll ein Assemblerprogramm entwickelt werden, das die Benutzernamen aller Studierenden, die vergessen haben ihre Matrikelnummer anzugeben, hintereinander an eine vorgegebene, feste Adresse `cBenutzernamen` im Speicher schreibt. Es sollen immer alle 8 Zeichen des Benutzernamens kopiert werden. Ausserdem soll als Rückgabewert die Anzahl aller Info-4-Teilnehmer und die Anzahl der Studierenden, die ihre Matrikelnummer nicht angegeben haben, berechnet werden. Die Liste der Daten der Info-4-Teilnehmer befindet sich im Speicher an der vorgegebenen, festen Adresse `cDaten`. Die Matrikelnummer 999999 markiert das Ende der Daten und 0 bezeichnet eine nicht vorhandene Matrikelnummer.

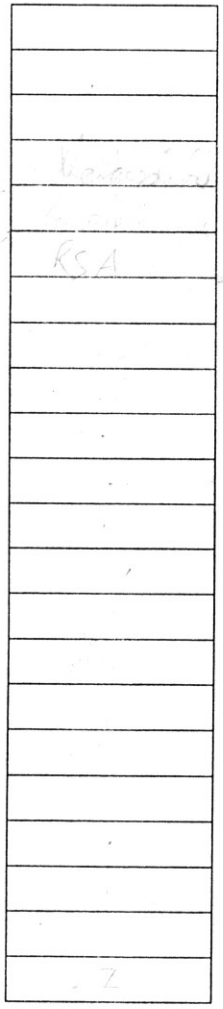
```
#define cBenutzernamen      0x30000
#define cDaten              0x60000
```

Ein Datensatzeintrag sieht wie folgt aus:

```
struct {
       int      matrikelnummer;
       char     name[10];
       char     vorname[20];
       char     benutzername[8];
       } eintrag;
```

[**Hinweis:** Variablen vom Typ `char` sind bei dem in den Übungen verwendeten Compiler 1 Byte groß, Variablen vom Typ `int` sind 4 Bytes groß, Adressen sind ebenfalls 4 Bytes groß.]

(a) **(2 Punkte)** Geben Sie den Kellerinhalt und den Kellerzeiger nach dem Aufruf der Prozedur und unmittelbar nach Sichern der Register an. Sie können die vorgegebenen Kästchen als Zeichenhilfe verwenden. Ein Kästchen sei dabei **4** Byte groß.



(b) **(10 Punkte)** Schreiben Sie die Prozedur in Motorola 68000-Assembler. Berücksichtigen Sie die oben angegebenen Anforderungen. Sie sind für die Verwaltung des Kellers verantwortlich!
Prozeduren ohne Dokumentation werden mit Null Punkten bewertet!
Hinweis nur für Wiederholer: Die Prozedur kann sinngemäß auch in Modula-2-Inline-Assembler implementiert werden. Geben Sie dazu den Prozedurkopf an!

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.

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

Das Programm ist nicht wiedereintrittsfest, da mit festen Adressen gearbeitet wird, die bei nebenläufiger Ausführung von mehreren Prozessen benutzt werden.

Die Parameterübergabe erfolgt über feste Adressen.

Die Register werden nicht gerettet.

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

Parameterübergabe über den Stack, Zwischenwerte im Stack speichern, Register retten.

(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.

```
_tue_gutes:
        MOVEM.L D1-D3/A1-A2, -(SP)   | Register retten
        MOVEA.L #0x2700, A1          | Speicheradressen in Register
        MOVE.L 24(SP), A1            | D1wertadr in A1 ablegen
        MOVEA.L #0x2710, A2          | ablegen
        MOVE.L 28(SP), A2            | D2wertadr in 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 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
```

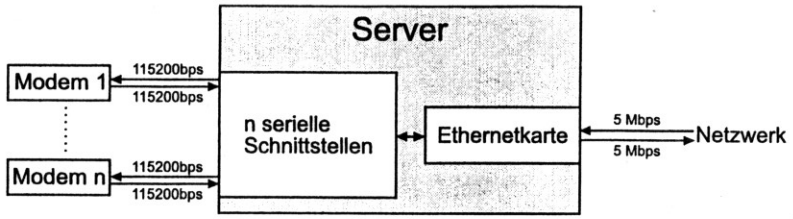
(Unterstrichen sind die zur Lösung hinzugefügten Zeilen)

(d) **(2 Punkte)** Zeichnen Sie den Kellerspeicher auf, wie er **unmittelbar nach Aufruf** des Unterprogramms aussieht.
Ein Kästchen entspricht 4 Byte!

28	D2wert	
24	D1wert	
20	RTS	
16	D1	
12	D2	
8	D3	
4	A1	
0	A2	← SP

Hardware **(5 Punkte)**

Die Fakultät IV möchte einen neuen Server anschaffen, der den Studierenden weitere Einwahlmöglichkeiten sprich modembestückte Telefonleitungen zur Verfügung stellt.



Die Modems werden über serielle Schnittstellen am Server angeschlossen, die mit einer Datenrate von jeweils 115.200 bps (Bit pro Sekunde) bidirektional betrieben werden. Somit kann jedes Modem 11.520 cps (Zeichen pro Sekunde) senden und 11.520 cps empfangen.

An das Fachbereichsnetz ist der Server mit einer eigenen 10 Mbps (Millionen Bits pro Sekunde) (Summe der Sende- und Empfangsdatenrate) schnellen Ethernetverbindung angeschlossen, die mit eine maximalen Gesamtdatenrate von 1.000.000 cps sicher betrieben werden kann. Der Server reicht die von den Modems empfangenen Daten an das Netzwerk weiter und umgekehrt.

Der lesende oder schreibende Zugriff auf die serielle Schnittstelle benötigt 50 Taktzyklen je Zeichen. Der Zugriff erfolgt im Interrupt. Pro Interrupt wird 1 Zeichen gelesen und geschrieben. Die Interruptbehandlungsroutine benötigt weitere 150 Taktzyklen.

Die Ethernetkarte verbraucht unabhängig vom verwendeten Prozessor 25 % der vorhandenen Rechenleistung.

Der Rechenweg muß klar erkennbar sein!

(a) **(2 Punkte)** Welche Anzahl an Modems kann ein Server betreiben, dessen Prozessor mit 200 MHz (200 Millionen Taktzyklen pro Sekunde) getaktet ist?

(b) **(2 Punkte)** Welche Anzahl an Modems kann ein Server betreiben, dessen Prozessor mit 400 MHz getaktet ist?

(c) **(1 Punkt)** Welcher der beiden Server bietet das bessere Preis-Leistungs-Verhältnis, wenn der schnellere 1,5 mal soviel wie der langsamere kostet?

Hardwarenahe Programmierung **(5 Punkte)**

Auf einem System mit Seitenverwaltung arbeiten Programme, die zu ihrem Ablauf mehr Speicher benötigen, als in dem vorliegenden System physikalisch (RAM) vorhanden ist.

Das System arbeitet mit einer Seitengröße von 4 KB, für den Datenbereich der Programme stehen 128 MB Speicher zur Verfügung.

Die Seitenverwaltung lagert nur den Datenbereich der Programme aus. Je nach Art des Programms wird der dem Programm zugehörige Datenbereich verschieden häufig und mit verschiedenen Zugriffsmustern benutzt.

Die Seitenverwaltung hat ihren Auslagerungsbereich auf einer dem System zugehörigen Festplattenlaufwerk und arbeitet nach der LRU-Strategie.

Das Lesen einer ganzen Seite aus dem Speicher dauert 10 ns.

Das Einlagern und Lesen einer ganzen Seite dauert 20 ms.

($1ns = 10^{-9}s$, $1ms = 10^{-3}s$)

Bei Programmstart sind die 128 MB Speicher des System schon mit dem (ggf. ersten) Teil der Daten belegt. Die Rechenzeiten des Programmes können gegenüber den Speicherzugriffszeiten vernachlässigt werden.

Rechenwege nicht vergessen!

(a) **(1 Punkt)** Wie lange läuft ein Programm mit einem Datenbereich von 128 MB, das diesen Bereich nur einmal abarbeitet?

(b) **(2 Punkte)** Wie lange läuft ein Programm mit einem Datenbereich von 256 MB, das diesen Bereich dreimal linear durchliest?

(c) **(2 Punkte)** Wie lange läuft ein Programm mit einem Datenbereich von 256 MB maximal, das auf diesen Bereich zufällig in 4 KB Häppchen zugreift und jedes Datum genau einmal anfaßt?

Wie lange minimal?