Name: | Vorname: | Matr.–Nr.:

Technische Universität München
Fakultät für Informatik
Prof. Dr. A. Knoll

WS 2010/2011
08. Februar 2011

Klausur zu Echtzeitsysteme

Aufgabe 1 Wissensfragen (20 Punkte)

a) Gegeben sind folgende Aussagen über Echtzeitsysteme:

(i) Harte Echtzeitsysteme sind Systeme, die besonders schnell sind, also besonders kurze Berechnungszeiten haben.

(ii) Harte Echtzeitsysteme sind Systeme, bei denen ein bestimmtes Zeitverhalten garantiert werden kann, wobei das Zeitverhalten aber langsam sein kann.

Welche dieser Aussagen ist richtig? Geben Sie für die richtige Aussage ein Beispiel an, das ein Echtzeitsystem ist. Geben Sie für die falsche Aussage ein Beispiel an, das zwar die Aussage erfüllt, aber kein Echtzeitsystem ist.

b) Reaktive, synchrone Sprachen wie Esterel basieren auf der Synchronitätshypothese. Erläutern Sie diese Hypothese, erklären Sie den wesentlichen Vorteil dieser Hypothese und beschreiben Sie, wie eine Implementierung aussieht.

c) Gegeben seien folgende drei Ausführungsmodelle

(i) Synchronous Dataflow (Synchroner Datenfluss)

(ii) Synchronous Reactive

(iii) Time-Triggered Execution (zeitgesteuerte Ausführung)

und folgende drei Anwendungen:

1. Fahrzeugsteuerung: Mehrere periodische Prozesse werden im verteilten System ausgeführt.
2. Transaktionssystem: Eine Ampelsteuerung mit einem zentralen Steuerrechner soll entwickelt werden. Die Steuerung soll auf verschiedenste Ereignisse reagieren können und abhängig von diesen Ereignissen verschiedene Aktionen ausführen.
3. Regelungsanwendung: Eine Spannungsquelle soll periodisch geregelt werden.

Finden Sie eine passende Zuordnung von den Ausführungsmodellen (jeweils nur einmal verwenden) zu den Anwendungen und begründen Sie kurz Ihre Antwort.

Aufgabe 2 Scheduling **(20 Punkte)**

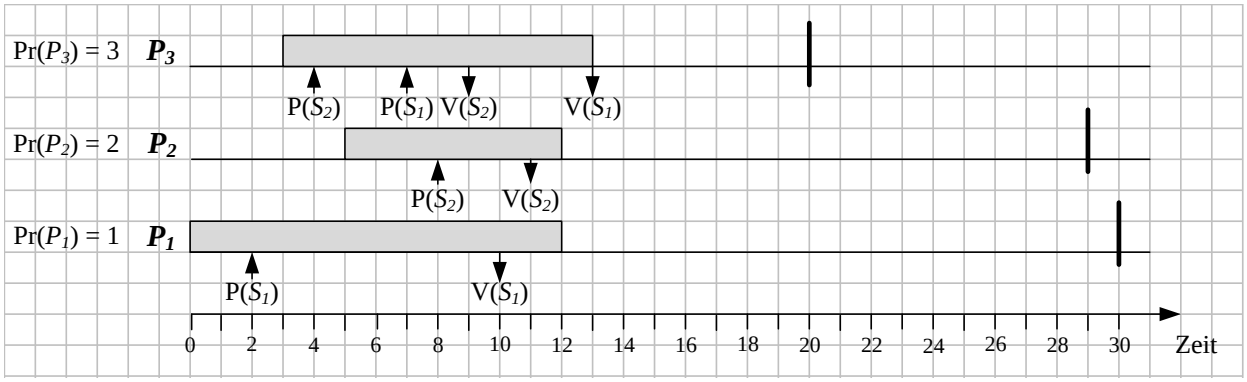

Abbildung 1: Szenario: Die senkrechten Balken kennzeichnen jeweils die Fristen der Prozesse, die Pfeile kennenzeichnen das Anfordern bzw. die Freigabe von Semaphoren.

Gegeben sei das im Folgenden beschriebene Szenario aus Abbildung 1, bei dem die Prozesse P_1, P_2 und P_3 nebenläufig auf einem Prozessor ausgeführt werden. **Die Abbildung zeigt die Ausführung jedes Prozesses so, als ob dieser als einziger ausgeführt werden würde.**

Bereitzeiten, Ausführungszeiten, Fristen und Prioritäten können der folgenden Tabelle entnommen werden. Zusätzlich fordern die Prozesse die binären Semaphore S_1 und S_2 an. Zu Beginn sind alle Semaphore freigegeben. Der Zeitpunkt für die Anforderung (P) und Freigabe (V) ist in der nachfolgenden Tabelle relativ zur Ausführungszeit des jeweiligen Prozesses angegeben.

	Bereitzeit	Ausführungszeit	Frist	Priorität	Semaphor S_1		Semaphor S_2	
	$r(P_i)$	$e(P_i)$	$d(P_i)$	$Pr(P_i)$	$P(S_1)$	$V(S_1)$	$P(S_2)$	$V(S_2)$
P_3	3	10	20	3 (hoch)	4	10	1	6
P_2	5	7	29	2 (mittel)	-	-	3	6
P_1	0	12	30	1 (niedrig)	2	10	-	-

Verwenden Sie zur Lösung das beigefügte Lösungsblatt. **Beachten Sie bei allen Teilaufgaben die Bereitzeiten der Prozesse.**

a) Ignorieren Sie zunächst die Semaphore. Zeichnen Sie einen Ausführungsplan für das präemptive Schedulingverfahren *least slack time* (Zeitscheiben, $D = 1$).

b) Berücksichtigen Sie jetzt die Semaphore und die angegebenen Prioritäten der Prozesse: $Pr(P_1) = 1 < Pr(P_2) = 2 < Pr(P_3) = 3$. Zeichnen Sie einen Ausführungsplan für ein rein prioritätenbasiertes präemptives Schedulingverfahren. Welches Problem tritt auf und was ist die Ursache (Fachbegriff)?

c) Lösen Sie das Problem aus der letzten Teilaufgabe durch das präemptive Schedulingverfahren Prioritätsvererbung (*priority inheritance*) und zeichnen Sie einen entsprechenden Ausführungsplan.

Aufgabe 3 Nebenläufigkeit **(30 Punkte)**

Gegeben sei das in Abbildung 2 veranschaulichte Szenario:

- Der Zugang zur Mensa erfolgt über die zentrale *Türe*. In der *Essensausgabe* der Mensa wählt man sein Essen am *Menü* aus, holt sein Essen und geht zur *Kasse*, hinter der sich die Tische befinden.
- Einigen Studenten fällt beim Lesen des Menüs ein, dass sie eine wichtige Vorlesung vergessen haben, die sie nicht versäumen wollen und verlassen daher sofort wieder die Mensa.
- Die anderen essen, geben ihr Tablett ab und verlassen dann die Mensa über den Ausgang, der wieder zur zentralen Türe führt.

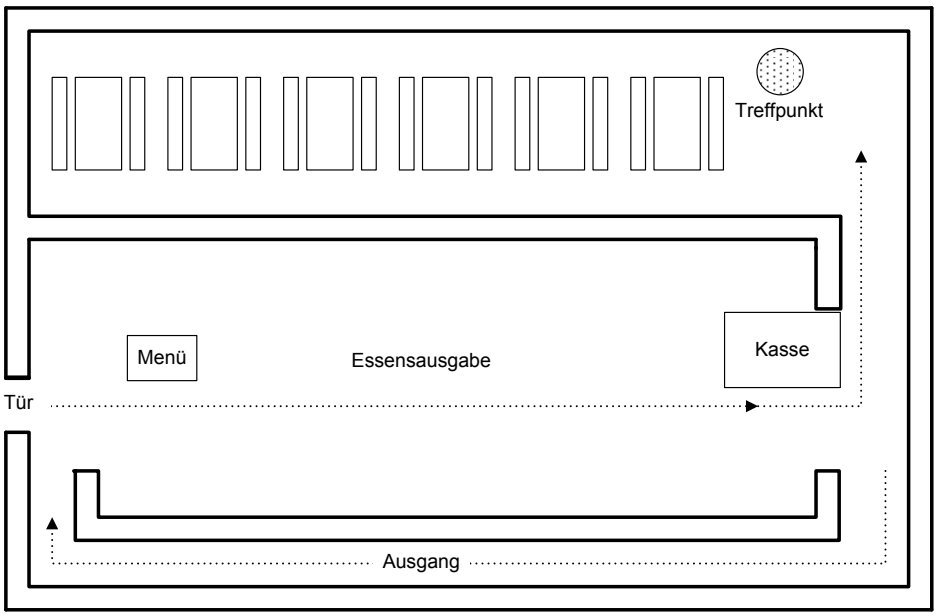

Abbildung 2: Übersicht über Mensa.

Hinweise zur Lösung:

- Achten Sie darauf, ob in der jeweiligen Teilaufgabe der Studentenprozess (Alg. 3) und/oder der Kassierer-Prozess (Alg. 2) erweitert werden sollen. Globale Variablen (Alg. 1) können in jeder der Teilaufgaben verwendet werden. Andere denkbare Prozesse sind **nicht** Bestandteil dieser Aufgabe.
- Kennzeichen Sie immer deutlich, welches Codegerüst Sie jeweils erweitern.
- Achten Sie auf eine effiziente Implementierung (insbesondere kein *busy waiting*).
- Schreiben Sie Ihre Lösungen **nicht** auf diese Angabe.
- Aus Gründen der Übersichtlichkeit ist in dieser Aufgabe nur die männliche Form (Student, Kassierer) angegeben. Selbstverständlich können damit auch Studentinnen bzw. eine Kassiererin gemeint sein.

Verwenden Sie die folgende Notation:

- `int i=0;`, wenn Sie eine ganzzahlige Variable `i` mit Initialisierungswert `0` benutzen wollen.
- `Semaphore semBeispiel(1);`, um einen mit `1` initialisierten Semaphor `semBeispiel` zu deklarieren.
  - `down(semBeispiel);`, um den Semaphor `semBeispiel` anzufordern.
  - `up(semBeispiel);`, um den Semaphor `semBeispiel` freizugeben.
- `Barrier barBeispiel(5);`, um eine mit `5` initialisierte Barriere zu deklarieren.
  - `wait(barBeispiel);`, um den aufrufenden Prozess an der Barriere `barBeispiel` zu synchronisieren.
- `MessageQueue qBeispiel;`, um eine Nachrichten-Warteschlange (*message queue*) `qBeispiel` zu deklarieren.
  - `sendMsg(q, m)`, um die Nachricht `m` über die Warteschlange `q` zu verschicken.
  - `m = recvMsg(q)`, um eine Nachricht von der Warteschlange `q` zu empfangen und diese in `m` zu speichern. Falls `q` leer ist, blockiert der Aufruf von `recvMsg` so lange, bis wieder ein Element in die Warteschlange geschrieben wurde.
- `Signal sigBeispiel;`, um ein pures (nicht-wertbehaftetes) Signal `sigBeispiel` zu deklarieren.
  - `sendSignal(sigBeispiel)`, um das Signal `sigBeispiel` zu verschicken. Die Funktion kehrt nach dem Aufruf sofort zurück, es gehen keine versendeten Signale verloren.
  - `waitForSignal(sigBeispiel)`, um auf das Auftreten des Signals `sigBeispiel` zu warten. Falls das Signal noch nicht angelegen hat, blockiert der Aufruf von `waitForSignal` bis zu dessen Auftreten.
- Kennung, persönlicher Semaphor
  - Jeder Student besitzt eine eindeutige Kennung vom Typ `int`, die in einer **lokalen** Variable (`ID`) gespeichert ist, d.h. diese ist nur innerhalb des Studentenprozesses zugänglich.
  - Jedem Student ist zudem ein Semaphor zugeordnet, auf den über die **global** zugängliche Funktion `Semaphore getSemaphoreById(int)` zugegriffen werden kann. Gehen Sie davon aus, dass die Semaphoren jeweils mit `0` initialisiert wurden.

Algorithmus 1 Codegerüst für Deklaration globaler Variablen

```
1: // Deklaration von globalen Variablen, Semaphoren, etc.
2:
3: Tür tür;
```

Algorithmus 2 Codegerüst für Kassiererprozess

```
1: // Deklaration von lokalen Variablen, Semaphoren, etc. für Kassiererprozess
2:
3: // Code für Kassiererprozess
4:
5: while (!feierabend()) do
6:
7:     kassiere();
8:
9:     schaueAufDieUhr();
10:
11: end while
12:
```

Algorithmus 3 Codegerüst für Studentenprozess

```
1:  // Deklaration von lokalen Variablen, Semaphoren, etc. für Studentenprozess
2:  Essen essen;
3:  int ID; // Eindeutige Kennung für jeden Studentenprozess
4:
5:  // Codegerüst für Studentenprozess
6:
7:  betreteMensa(tür);
8:
9:  essen = liesMenü();
10:
11: if (essen == KeinEssen) then
12:
13:     verlasseMensa(tür);
14:
15:     return
16:
17: end if
18:
19: holeEssen();
20:
21: bezahle();
22:
23: essen();
24:
25: gibTablettZurück();
26:
27: verlasseMensa(tür);
28:
```

Programmieraufgaben:

a) Ergänzen Sie den Studentenprozess (Alg. 3) so, dass sich zu jedem Zeitpunkt maximal ein Student in der zentralen Türe befindet.

b) In der Essensausgabe haben maximal 50 Studenten Platz. Ergänzen Sie den Studentenprozess (Alg. 3) so, dass diese Einschränkung sichergestellt wird, und die Studenten vor der Mensa warten, falls kein Platz in der Essensausgabe ist.

c) Stellen Sie sicher, dass Studenten in der gleichen Reihenfolge ihr Essen bezahlen, in der sie die Auswahl des Essens abgeschlossen haben. Ergänzen Sie hierzu den Studentenprozess (Alg. 3) und/oder den Kassiererprozess (Alg. 2).

d) Die Studenten Tina, Tim und Thomas haben vereinbart, sich hinter der Kasse zu treffen, um gemeinsam einen freien Tisch zu suchen. Erstellen Sie eine effiziente Implementierung im Studentenprozess (Alg. 3), mit der die Verabredung der drei umgesetzt werden kann. Gehen Sie davon aus, dass die Funktion `is3T()` den Wert `true` zurückliefert, falls der Aufrufer einer der drei Studenten ist.

Allgemeine Fragen (**keine** Implementierung nötig!):

e) Dort entdecken die drei eine Gruppe von Philosophen, die jeweils mit einer Gabel in ihrer linken Hand hungrig vor ihren Tabletts sitzen. Erläutern Sie *kurz* vier Bedingungen, die erfüllt sein müssen, damit es zu einer solchen Verklemmung (*deadlock*) kommen kann.

f) Den Studenten wird klar, dass sie soeben ein Beispiel für das Problem der Aussperrung (*starvation*) erlebt haben. Nennen Sie zwei weitere Probleme der nebenläufigen Programmierung und erläutern Sie *kurz* mögliche Lösungen.

Aufgabe 4 Kommunikation (20 Punkte)

Gegeben sei der folgende Kommunikationsablauf:

- Drei Teilnehmer wollen über einen gemeinsamen Bus Nachrichten versenden.
- Der in Abbildung 3 dargestellte Kommunikationsablauf geht davon aus, dass jeder Teilnehmer den Bus für sich **exklusiv** nutzt.

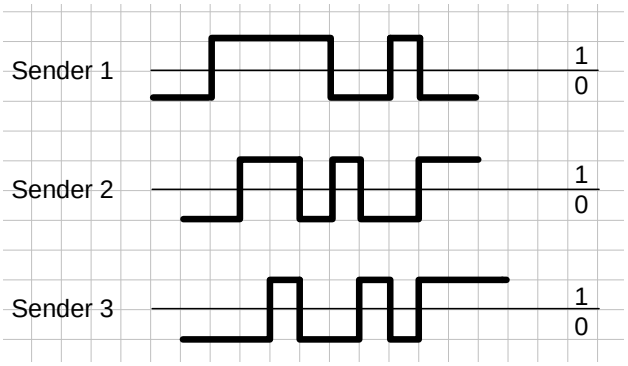

Abbildung 3: Kommunikation

Zeichnen Sie die Lösung der folgenden Aufgaben auf das **seperat ausgeteilte Lösungsblatt**. Beachten Sie dabei folgende Hinweise:

- Gehen Sie davon aus, dass für die Lösung der Aufgabe alle Daten bitsynchron übertragen werden.
- Das JAM-Signal soll aus einer Folge von 6 0-Bits bestehen.
- Das 0-Bit ist dominant.

a) Zeichnen Sie den angegebenen Kommunikationsablauf auf das beigelegte Lösungsblatt für den Fall, dass **CSMA/CA** (wie zum Beispiel im CAN-Protokoll) zum Buszugriff verwendet wird. Zwischen zwei Nachrichten soll es dabei eine Pause (*interframe gap*) von mindestens 3 Bits geben.

b) Zeichnen Sie den angegebenen Kommunikationsablauf auf das beigelegte Lösungsblatt für den Fall, dass **TTP** zum Buszugriff bzw. zur Kommunikation verwendet wird. Zeichnen Sie dabei ebenfalls die Zeitschlitze ein und ordnen Sie diese den Sendern zu. Gehen Sie dabei von folgenden Vereinfachungen aus:

   - Ignorieren Sie die zusätzlichen Daten des Protokoll-Overheads.
   - Nachrichten können fragmentiert werden.
   - Ein Kästchen entspricht 10 μs.
   - Jeder Zeitschlitz ist 30 μs lang, die Reihenfolge der Zeitschlitze ist *S1, S2, S3, S1, ...*.
   - Es gibt kein Interframegap zwischen den einzelnen Zeitschlitzen.

Schreiben Sie Ihre Lösung **nicht** auf diese Angabe.

c) Nennen Sie den signifikanten Unterschied beim Buszugriff zwischen TTP, Token Ring und CAN.