Zentralübung Echtzeitsysteme
Wintersemester 2011/2012

Dr. Christian Buckl

fortiss GmbH – Cyber-Physical Systems

TU München - Lehrstuhl VI Robotics and Embedded Systems

Zentralübung - Hintergrund

- Ziele:
  - Klärung offener Fragen
  - Anwendung der gelernten Inhalte
  - Ansprechpartner für Probleme
- Methodik:
  - Interaktive Gestaltung: Ihre Mitwirkung ist wichtig
  - Hands-On Übungen: Werkzeuge, Programmierung
  - Durchsprache von Klausuraufgaben
    - Anmerkung: die angegebenen Punkte entsprechen auch der vorgesehenen Arbeitszeit (1 Punkt = 1 Minute)

Übung – Aktuelle Informationen

- Wegen der hohen Teilnehmerzahl wird eine weitere Übungsgruppe eingerichtet
- Bitte melden Sie sich spätestens diese Woche über TUM Online an (Eintrag auf Warteliste) oder senden Sie eine Email an Stephan Sommer (sommerst@in.tum.de).
- Herr Sommer wird Ihnen eine Email mit weiteren Informationen zusenden.

Tutorübung – Aktuelle Informationen

- Wegen der hohen Teilnehmerzahl wird eine weitere Übungsgruppe eingerichtet
- Aktuell:
  - Tutorgruppe 1: 3 Personen auf Warteliste
  - Tutorgruppe 2: 3 Personen auf Warteliste
  - Tutorgruppe 4: 4 Personen auf Warteliste
  - Zahlreiche Anmeldungen per E-Mail
- Terminvorschläge von 12 Studenten
- Zusatzübung wird voraussichtlich am Donnerstag 8:30 – 10:00 Uhr stattfinden

Kapitel 1

Einführung Echtzeitsysteme

Klausurfragen

- Klausur WS 06/07
  - Was ist der Unterschied zwischen harten und weichen Echtzeitsystemen? (3 Punkte = 3 min)
  - Wieso sollte Virtual Memory nicht in Echtzeitsystemen verwendet werden? (3 Punkte = 3 min)
- Wiederholungsklausur WS 06/07 (5 Punkte = 5 min)
  - Ordnen Sie folgende Anwendungen in die Kategorien harte bzw. weiche Echtzeitsysteme ein und begründen Sie Ihre Antwort:
    - Ampelsteuerung
    - Flugzeugregelung
    - Internettelefonie

Klausurfragen

- Klausur WS 10/11
- 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.

Kapitel 3

Modellierung von Echtzeitsystemen und Werkzeuge

Fragen zur Vorlesung

- Was ist der Vorteil von ereignisgesteuerten Applikationen gegenüber zeitgesteuerten?

Zeitgesteuerte Applikationen	Ereignisgesteuerte Applikationen
Zeitlicher Systemablauf wird zur Übersetzungszeit festgelegt.	Ausführungen werden durch das Eintreten von Ereignissen angestoßen.
Präzise und globale Uhr erforderlich (inkl. Uhrensynchronisation)	Garantierte Antwortzeiten sind erforderlich.
Einzelberechnungen werden in einem jeweils reservierten Zeitslot durchgeführt. Ableitung der max. Laufzeit notwendig. (worst case execution time)	Das Scheduling erfolgt dynamisch. Keine Aussage über zeitlichen Ablauf zur Übersetzungszeit möglich.
VORTEIL: Statisches Scheduling möglich. Vorhersagbares deterministisches Verhalten.	**VORTEIL:** ???

Häufige Fragen zum Stoff der Vorlesung

- Was sind Aktoren?
  - Unterscheidung zum Begriff Aktoren aus der Mechatronik?
- Wie zuverlässig ist der generierte Code bei modellbasierten Entwicklungswerkzeugen?
- Wie gut ist der generierte Code zu lesen?
- Wann verwendet man synchrone Sprachen, wann synchronen Datenfluss?
    - Unterscheidung zwischen kontrollorientierten (Fragestellung: wie reagiert das System auf Ereignisse) und datenorientierten (Fragestellung: wie werden eingehende Daten verarbeitet) Anwendungen.

Fragen zur letzten Vorlesung

- Wieso wird Rekursion in Esterel nicht unterstützt?
- Wie funktioniert die Umsetzung des Await-Statements?

```
module Temperature:
input IN1,IN2;
output OUT1,OUT2;
    loop
      await IN1; emit OUT1;
    end loop;
    ||
    loop
      await IN2; emit OUT2;
    end loop;
```

Auszug aus Klausur WS 06/07 (7 Punkte = 7 min)

a) Vervollständigen Sie folgende Testfälle, so dass das Modul xxx diese Testfälle erfüllt:

1. T1=({D},___),(___,{F})
2. T2=({D},___),({D},___)
3. T3=(___,{E}),(___,{F}),(___,{})
4. T4=(_____),(___,{N})
5. T5=(_____),(___,{E}),(___,{E})

Zur Erinnerung: ({A},{B}),({C},{D}) bedeutet: im ersten Moment erfolgt das Ereignis A als Eingabe, die Reaktion des Moduls ist B, im zweiten Moment erfolgt das Ereignis C als Eingabe mit der Reaktion D.

```
module xxx:
    input U, D;
    output E,F,N;
    var V=1;
    loop
        await
          case U do
                    if(?V>0)
                              V:=V+1;
                    else
                              V:=V+1;
                              emit F;
          case D do
                    if(?V>0) then
                              V:=V-1;
                              emit E;
                    else
                              emit N;
        end await;
    end loop;
```

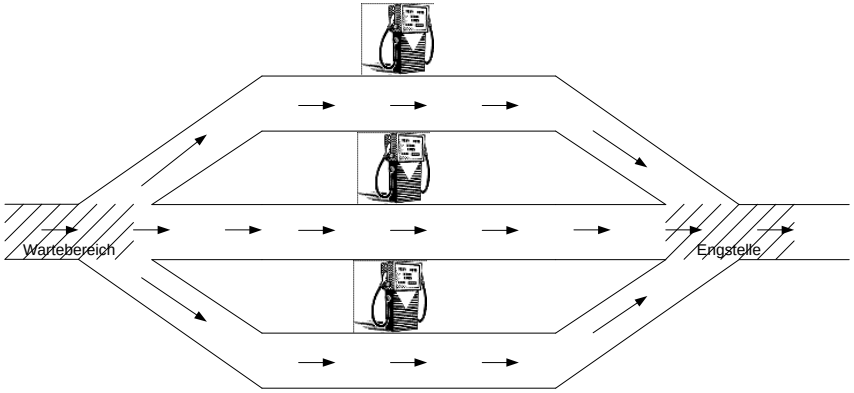
Kapitel 4

Nebenläufigkeit

Klausur WS06/07 – Nebenläufigkeit (15 Punkte = 15min)

Prozess: tankendes Auto

```
fahreInWartebereich();

fahreAnZapfsaeule();

tanke();

bezahle();

fahreInEngstelle2();

verlasseEngstelle2();
```



a) Geben Sie die notwendigen Semaphore (mitsamt Initialisierung) an, um das gegebene Problem zu lösen.
Beispiel: semAuto(1) würde bedeuten, Sie verwenden einen Semaphor semAuto, der mit 1 initialisiert ist.

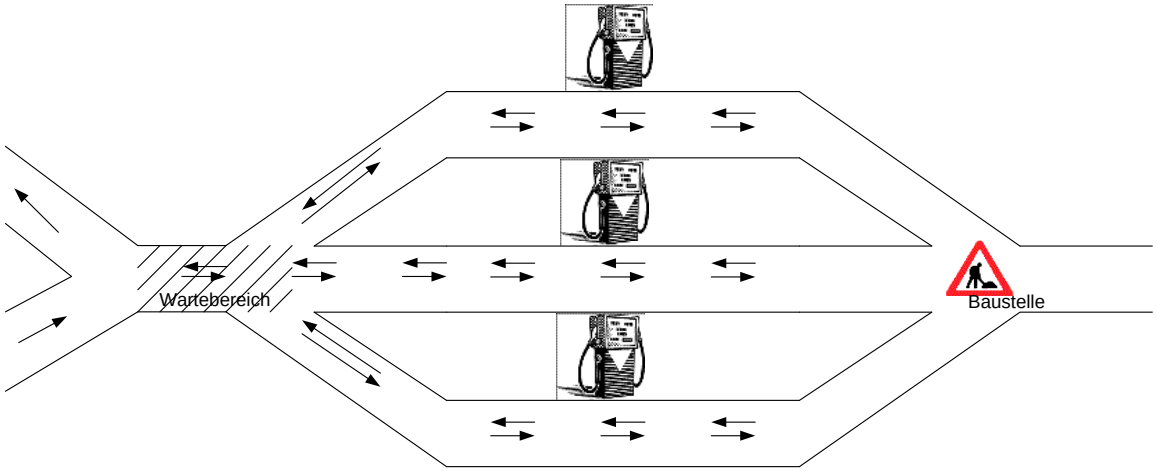
b) Ergänzen Sie den folgenden Autoprozess mit passenden up() und down()-Methoden, um Kollisionen zu vermeiden. Achten Sie darauf, dass es zu keiner Verklemmung kommt. **Anmerkung**: Es muss nicht an jeder freien Stelle Code eingefügt werden.
Beispiel: 1: down(semaAuto); up(semaAuto); bedeutet das Einfügen der beiden Operationen in Zeile 1.

Klausur WS06/07 - Nebenläufigkeit

c) Aufgrund einer Baustelle ist die Ausfahrt blockiert (siehe Abbildung), so dass die Wartebereich sowohl zur Einfahrt, als auch zur Ausfahrt genutzt werden muss. Ergeben sich notwendige Änderungen im Vergleich zur Lösung der Aufgabe b) und wenn ja welche?

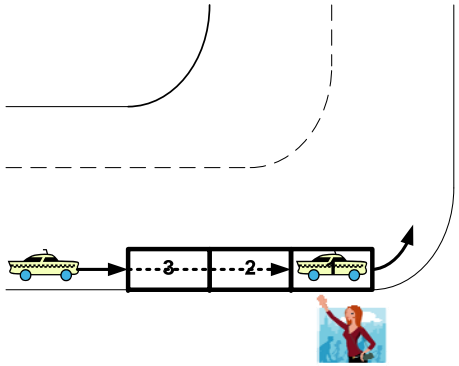
Prozess: tankendes Auto

```
fahreInWartebereich();

fahreAnZapfsaeule();

tanke();

bezahle();

fahreInEngstelle2();

verlasseEngstelle2();
```



Klausur WS07/08 – Nebenläufigkeit (20 Punkte = 20min)

- Gegeben Sie folgendes Szenario: am Münchner Odeonsplatz gibt es eine Wartebucht für Taxis. Zur Vereinfachung gehen wir davon aus, dass die Wartebucht aus drei Plätzen besteht und immer nur ein Passagier gleichzeitig auf ein Taxi wartet. Passagiere steigen an der ersten Wartebucht ein, die Taxis rücken nach, sobald das Taxi vor ihnen losgefahren ist. Implementieren Sie nun schrittweise eine Prozesssynchronisation, so dass es zu keinen Auffahrunfällen kommt, die Taxis in der Ankunftsreihenfolge auch wieder losfahren, Taxis nur mit Passagier losfahren, Passagiere nicht aus Versehen ein nicht-existentes Taxi betreten und es zu keinen Verklemmungen kommt.



a) Notieren Sie die wichtigen Programmabschnitte des Taxiprozesses und des Passagierprozesses. Lassen Sie genügend Platz für spätere Synchronisationsoperationen.
Beispiel: `fahreInErsteWartebucht();`

b) Geben Sie die zur Synchronisation der Taxis und Passagiere benötigten Semaphore, sowie der Initialwerte an. Gehen Sie dabei davon aus, dass zu Beginn kein Taxi in der Wartebucht und keine wartenden Passagiere vorhanden sind.
Beispiel: `semTaxi(1)` würde bedeuten, Sie verwenden einen Semaphor semTaxi, der mit 1 initialisiert ist.
`int i=0;` wenn sie eine ganzzahlige Variable mit Initialisierungswert 1 benutzen wollen.

c) Ergänzen Sie den Taxiprozess und Passagierprozess mit passenden up() und down()-Methoden, um die Aufgabenstellung zu erfüllen.
Beispiel: `down(semTaxi);` bedeutet das Anfordern des Semaphors semTaxi
Beispiel: `up(semTaxi);` bedeutet das Freigeben des Semaphors semTaxi

d) Der Wartebereich am Odeonsplatz ist begrenzt. Stellen Sie sicher, dass maximal 3 Taxis auf Fahrgäste warten und kein Rückstau entsteht. Die Überprüfung ob der Wartebereich belegt ist, soll dabei so schnell wie möglich erfolgen um den Straßenverkehr nicht zu behindern. Andererseits, sollen die Taxifahrer auf jeden Fall in den letzten Warteplatz fahren, falls dieser frei ist.

Klausur WS07/08 – Nebenläufigkeit - Lösung

Algorithm 1 Lösung Nebenläufigkeit: Benötigte Semaphoren

```
1:  semBucht3(1);
2:  semBucht2(1);
3:  semBucht1(1);
4:  semTaxi(0);
5:  semPassagier(0);
6:  semTest(1);
7:  int frei=1;
```

Algorithm 2 Lösung Nebenläufigkeit: Taxiprozess

```
1:  down(semBucht3);
2:  fahreInBucht3();
3:  down(semBucht2);
4:  fahreInBucht2();
5:  up(semBucht3);
6:  down(semBucht1);
7:  fahreInBucht1();
8:  up(semBucht2);
9:  up(semTaxi);
10: down(semPassagier);
11: fahreLos();
12: up(semBucht1);
```

Algorithm 3 Lösung Nebenläufigkeit: Passagierprozess

```
1:  down(semTaxi);
2:  steigeEin();
3:  up(semPassagier);
```

Algorithm 4 Lösung Nebenläufigkeit: Überprüfung

```
1:  down(semTest);
2:  if frei==1 then
3:      frei=0;
4:      up(semTest);
5:      fahreInBucht3();
6:      down(semBucht2);
7:      fahreInBucht2();
8:      down(semTest);
9:      frei=1;
10:     up(semTest);
11:     ...
12: else
13:     up(semTest);
14:     fahreWeiter();
15: end if
```

Klausur WS 10/11 – Nebenläufigkeit (30 P = 30 min)

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.

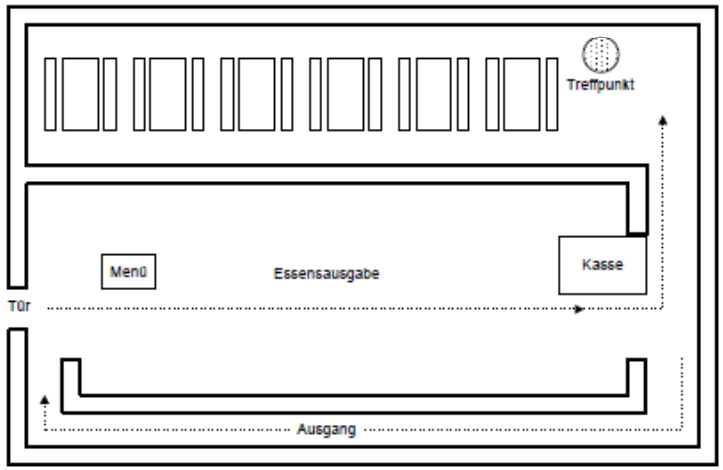


Abbildung 2: Übersicht über Mensa.

Klausur WS 10/11 – Nebenläufigkeit (30 P = 30 min)

- 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.

Klausur WS 10/11 – Nebenläufigkeit (30 P = 30 min)

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.

Klausur WS 10/11 – Nebenläufigkeit (30 P = 30 min)

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:
```

Klausur WS 10/11 – Nebenläufigkeit (30 P = 30 min)

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:
```

Klausur WS 10/11 – Nebenläufigkeit (30 P = 30 min)

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.

Klausur WS 10/11 – Nebenläufigkeit Lösung

Algorithmus 1 Codegerüst für Deklaration globaler Variablen

```
1:  // Deklaration von globalen Variablen, Semaphoren, etc.
2:
3:  Tür tür;
4:
5:  // a)
6:  Semaphore semTür(1);
7:
8:  // b)
9:  Semaphore semAusgabe(50);
10:
11: // c)
12: MessageQueue qKasse;
13:
14: // d)
15: Barrier barTTT(3);
```

Klausur WS 10/11 – Nebenläufigkeit Lösung

Algorithmus 2 Codegerüst für Kassiererprozess

```
1:  // Deklaration von lokalen Variablen, Semaphoren, etc. für Kassiererprozess, c)
2:  int kennungAktuellerStudent; // c)
3:  Semaphore semaphoreAktuellerStudent; // c)
4:
5:  // Code für Kassiererprozess
6:
7:  while (!feierabend()) do
8:      // c)
9:      kennungAktuellerStudent = recvMsg(qKasse);
10:     semaphoreAktuellerStudent = getSemaphoreById(kennungAktuellerStudent);
11:     up(semaphoreAktuellerStudent);
12:
13:     kassiere();
14:
15:     schaueAufDieUhr();
16:
17: end while
18:
```

Klausur WS 10/11 – Nebenläufigkeit Lösung

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:  Semaphore mySemaphore; // c)
5:
6:  // Codegerüst für Studentenprozess
7:  down(semAusgabe); // b)
8:
9:  down(semTür); // a)
10: betreteMensa(tür);
11: up(semTür);
12:
13: essen = liesMenü();
14:
15: if (essen == KeinEssen) then
16:
17:     down(semTür); // a)
18:     verlasseMensa(tür);
19:     up(semTür);
20:
21:     up(semAusgabe); // b)
22:     return
23:
24: end if
25:
26: holeEssen();
27: // c)
28: sendMsg(qKasse, ID);
29: mySemaphore = getSemaphoreById(ID);
30: down(mySemaphore);
31:
32: bezahle();
33:
34: up(semAusgabe); // b)
35: // d)
36: if (is3T()) then
37:     wait(barTTT);
38: end if
39: essen();
40:
41: gibTablettZurück();
42:
43: down(semTür); // a)
44: verlasseMensa(tür);
45: up(semTür);
46:
```

Kapitel 5

Scheduling

Klausur SS 07 – Szenario (20 Punkte = 20 min)

a) Welches Schedulingverfahren wurde verwendet? Welche Änderungen würden sich ergeben, wenn das Verfahren präemptiv wäre?

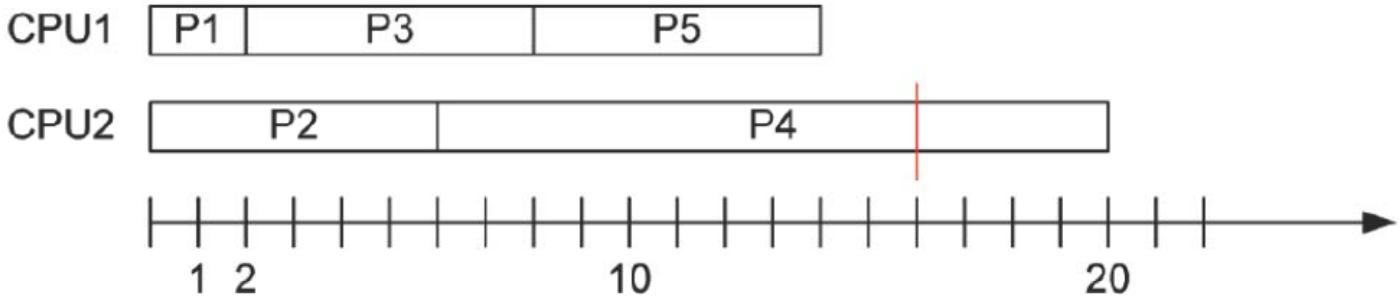
b) Welche optimalen Schedulingverfahren existieren für Mehrprozessorsysteme?

c) Welche Voraussetzungen müssen für ein optimales Schedulingverfahren in Mehrprozessorsystemen erfüllt sein?

d) Zeichnen Sie unter Zuhilfenahme eines optimalen Schedulingverfahrens einen korrekten Ausführungsplan.

e) In der Praxis werden diese Schedulingverfahren nicht angewandt. Was spricht dagegen und welcher Ansatz wird stattdessen gewählt?

Startzeiten s: s(P1)=0; s(P2)=0; s(P3)=0; s(P4)=0; s(P5)=0;
Ausführungszeiten e: e(P1)=2; e(P2)=6; e(P3)=6; e(P4)=14; e(P5)=6;
Deadlines e: e(P1)=4; e(P2)=8; e(P3)=12; e(P4)=16; e(P5)=18;



Klausur SS 07 – Antworten

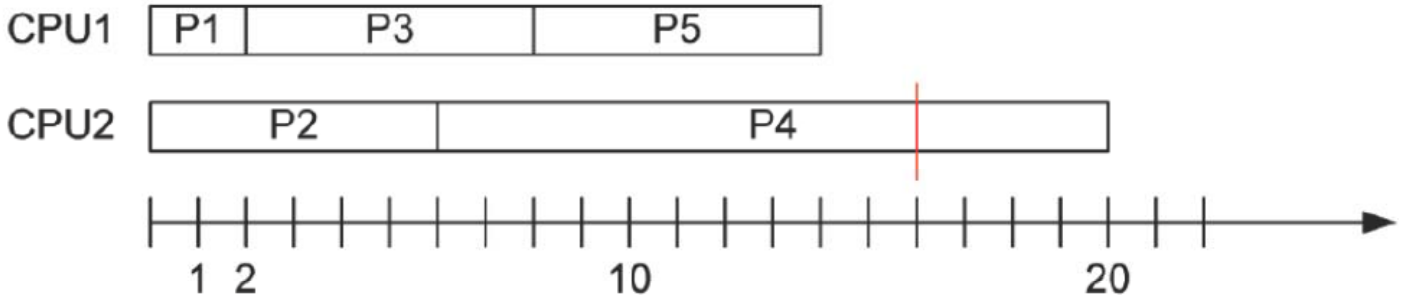
a) EDF, es ergeben sich keine Änderungen, wenn das Verfahren präemptiv ausgeführt wird

b) +c) Planen (wenn alle Prozesse inkl. Startzeit, Laufzeit und Deadline vorab bekannt sind), präemptives Least Slack Time (gleiche Voraussetzungen wie planen, aber zusätzlich müssen alle Prozesse die gleiche Startzeit haben)

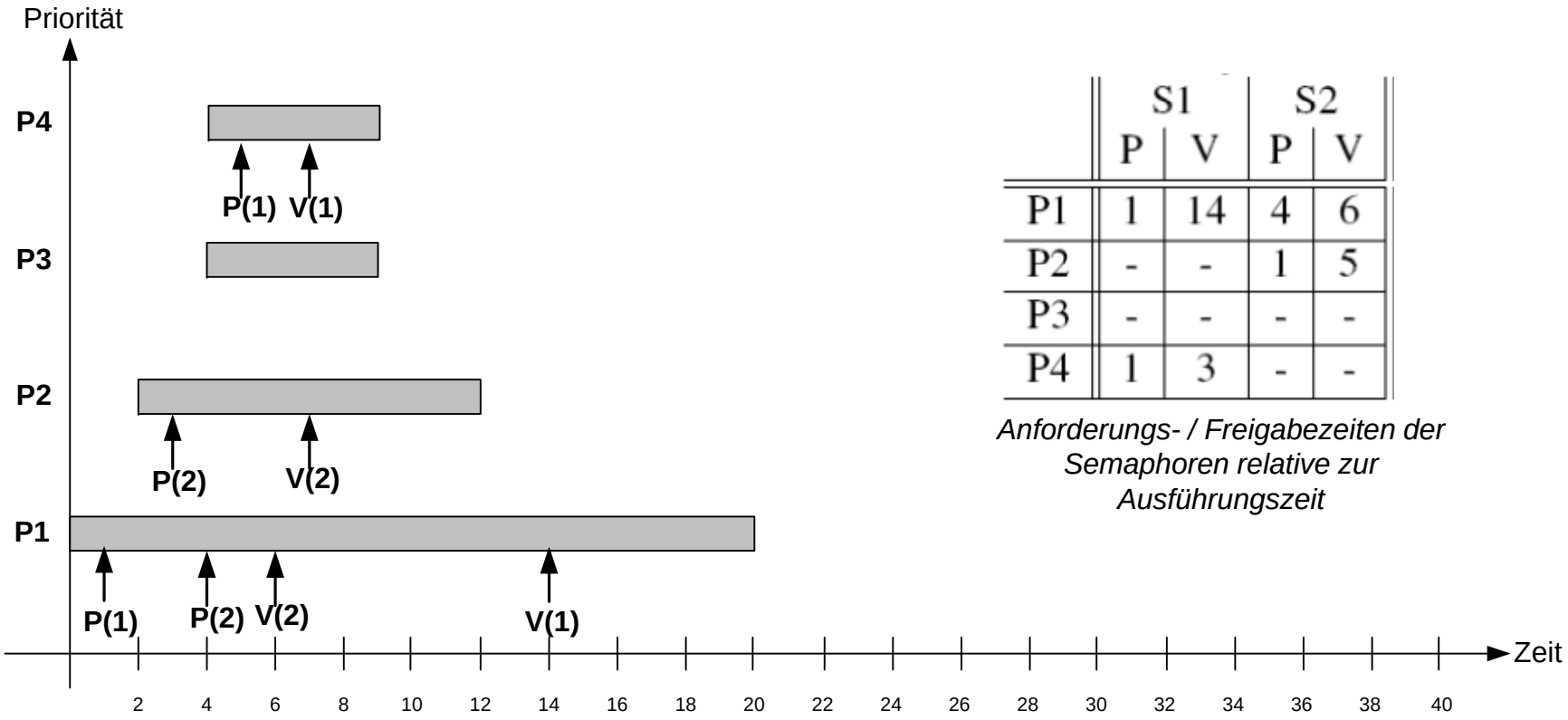
c) Siehe b)

d) CPU 1: P1-> P4; CPU 2: P2->P3->P5

e) Es werden prioritätenbasierte Verfahren angewandt, da diese sehr leicht umzusetzen sind und zur Laufzeit neben der Priorität keine weiteren Informationen verfügbar sein müssen



Klausur WS 06/07 – Szenario (15 Punkte = 15min)



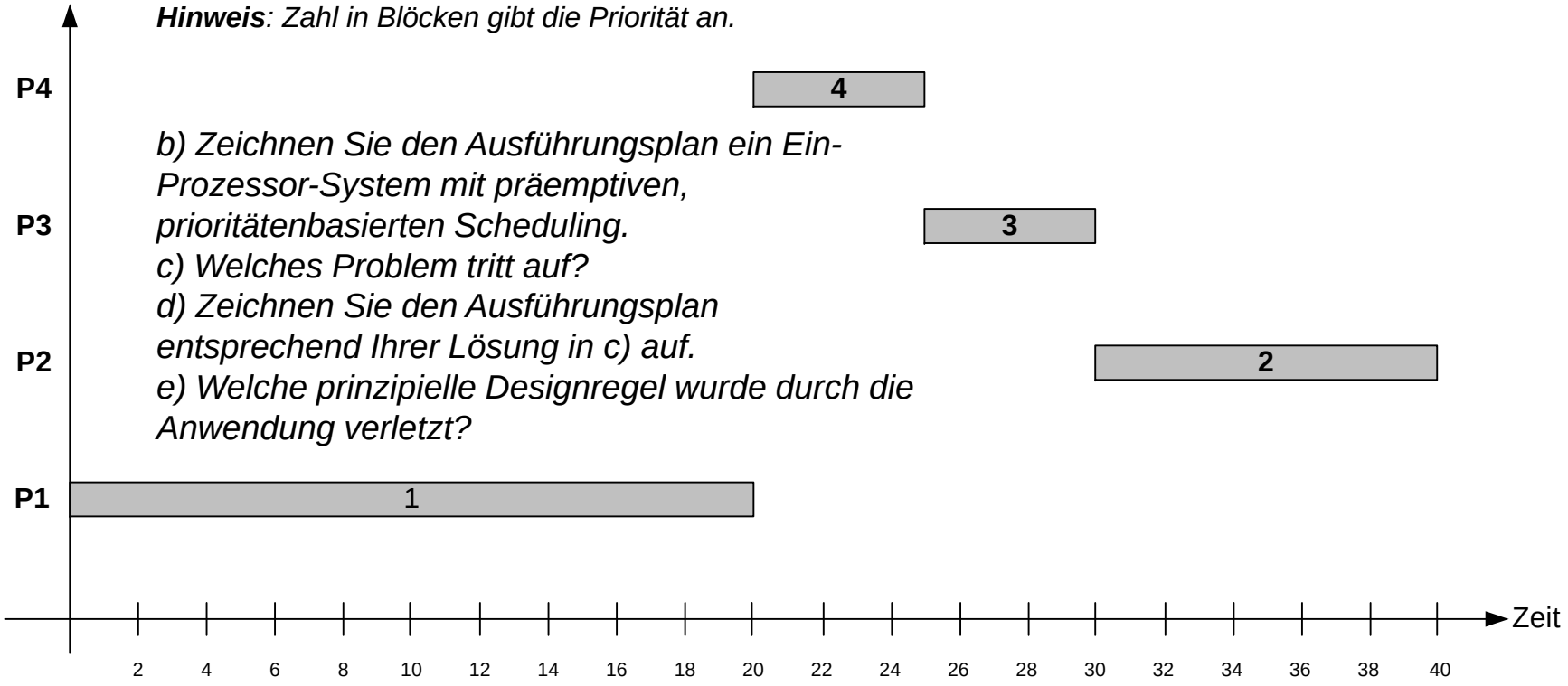
	S1		S2	
	P	V	P	V
P1	1	14	4	6
P2	-	-	1	5
P3	-	-	-	-
P4	1	3	-	-

Anforderungs- / Freigabezeiten der Semaphoren relative zur Ausführungszeit

Fortsetzung – Möglicher Ausführungsplan für ein 1-Prozessor-System

a) Welches Verfahren wird hier angewandt?
***Hinweis**: Zahl in Blöcken gibt die Priorität an.*

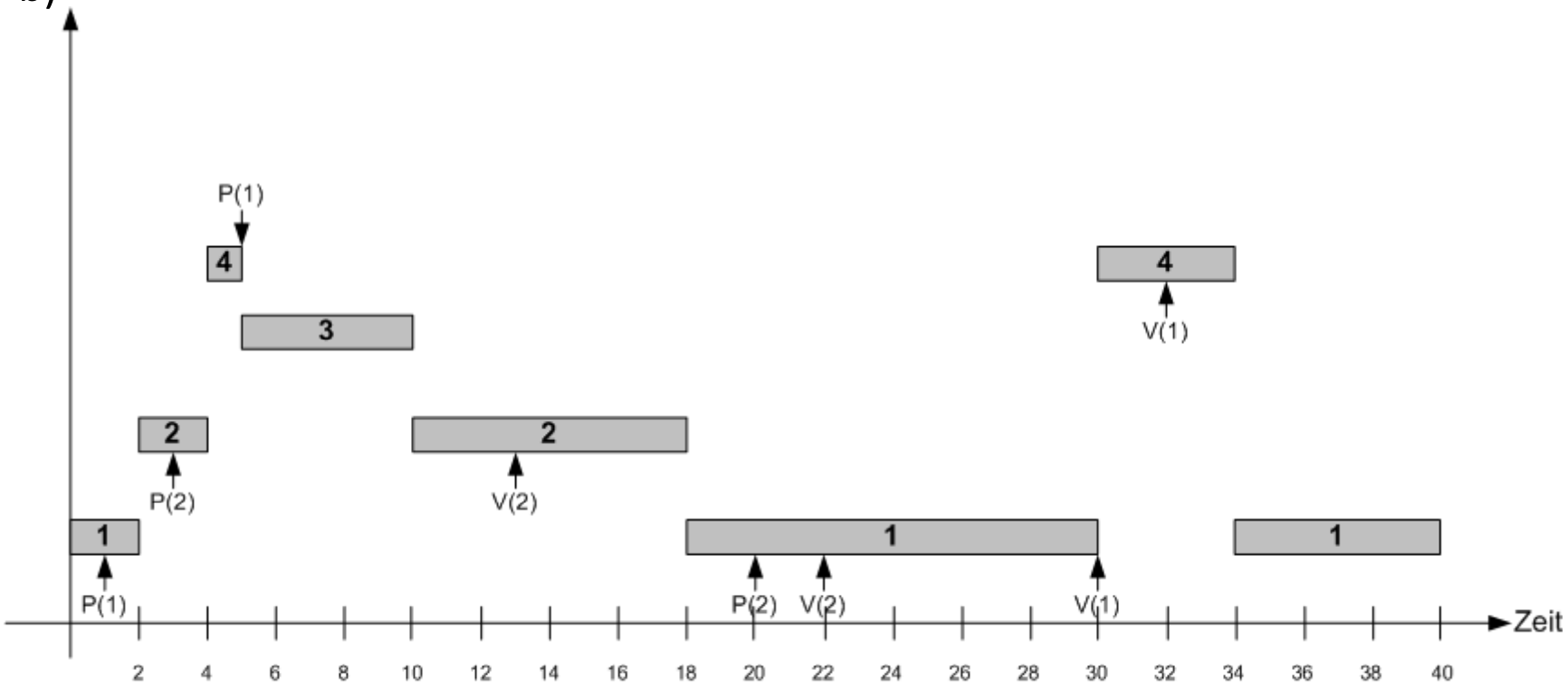
b) Zeichnen Sie den Ausführungsplan ein Ein-Prozessor-System mit präemptiven, prioritätenbasierten Scheduling.
c) Welches Problem tritt auf?
d) Zeichnen Sie den Ausführungsplan entsprechend Ihrer Lösung in c) auf.
e) Welche prinzipielle Designregel wurde durch die Anwendung verletzt?



Lösung Teilaufgabe a) + b)

a) Nicht-präemptives, prioritätenbasiertes Scheduling

b)



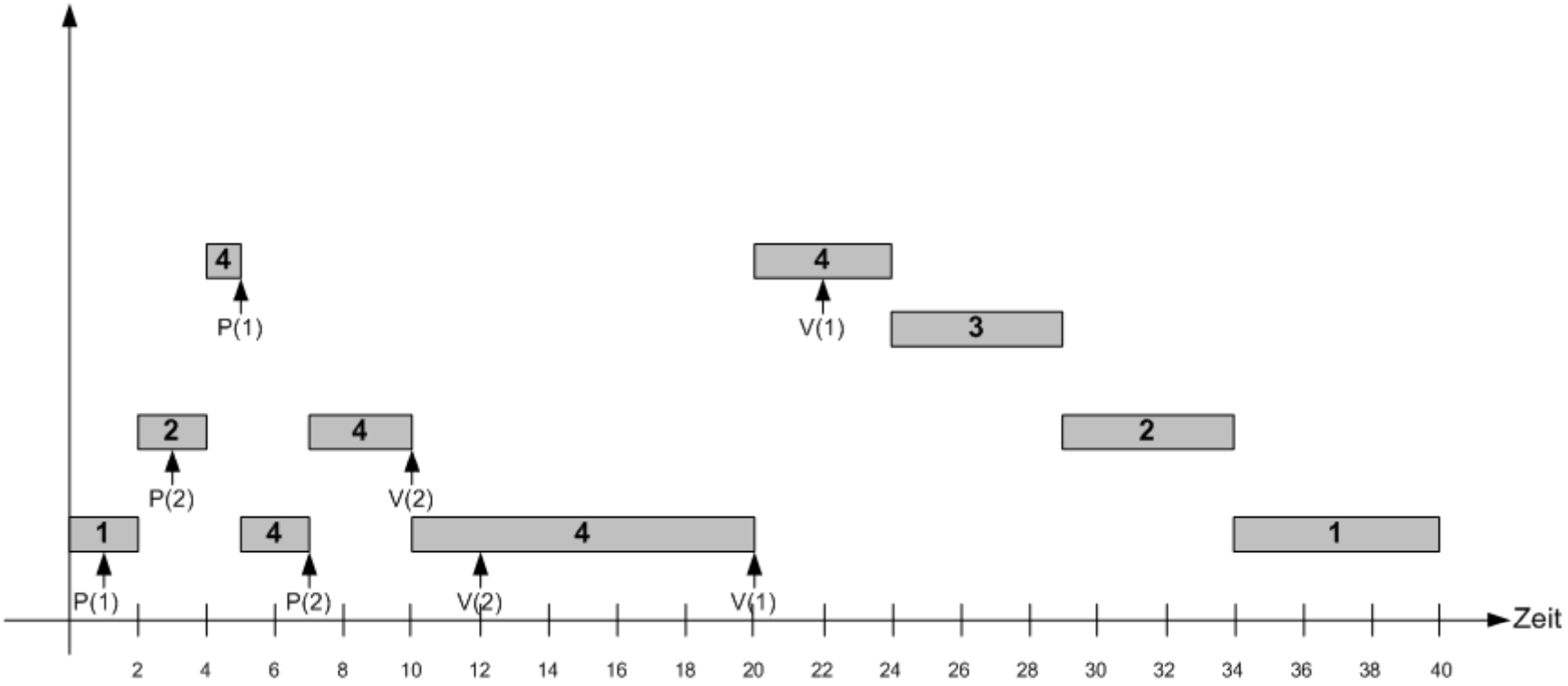
Lösung

c) Prioritätsinversion, Möglichkeiten zur Behebung Prioritätsvererbung, (sofortige) Prioritätsgrenzen

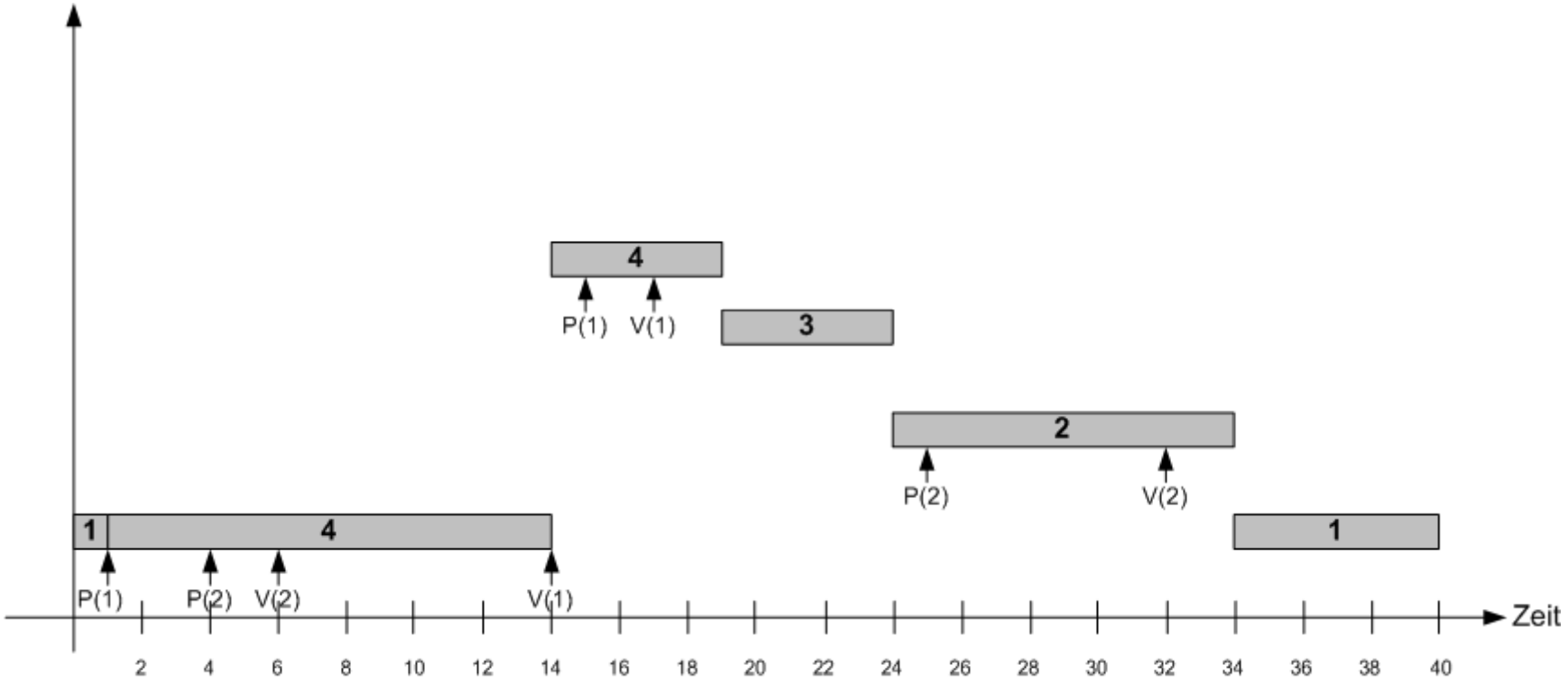
d) Siehe folgende Folien

e) Prozesse mit geringerer Priorität sollten keine Betriebsmittel so lange blockieren, wenn diese auch von Prozessen mit höherer Priorität benötigt werden.

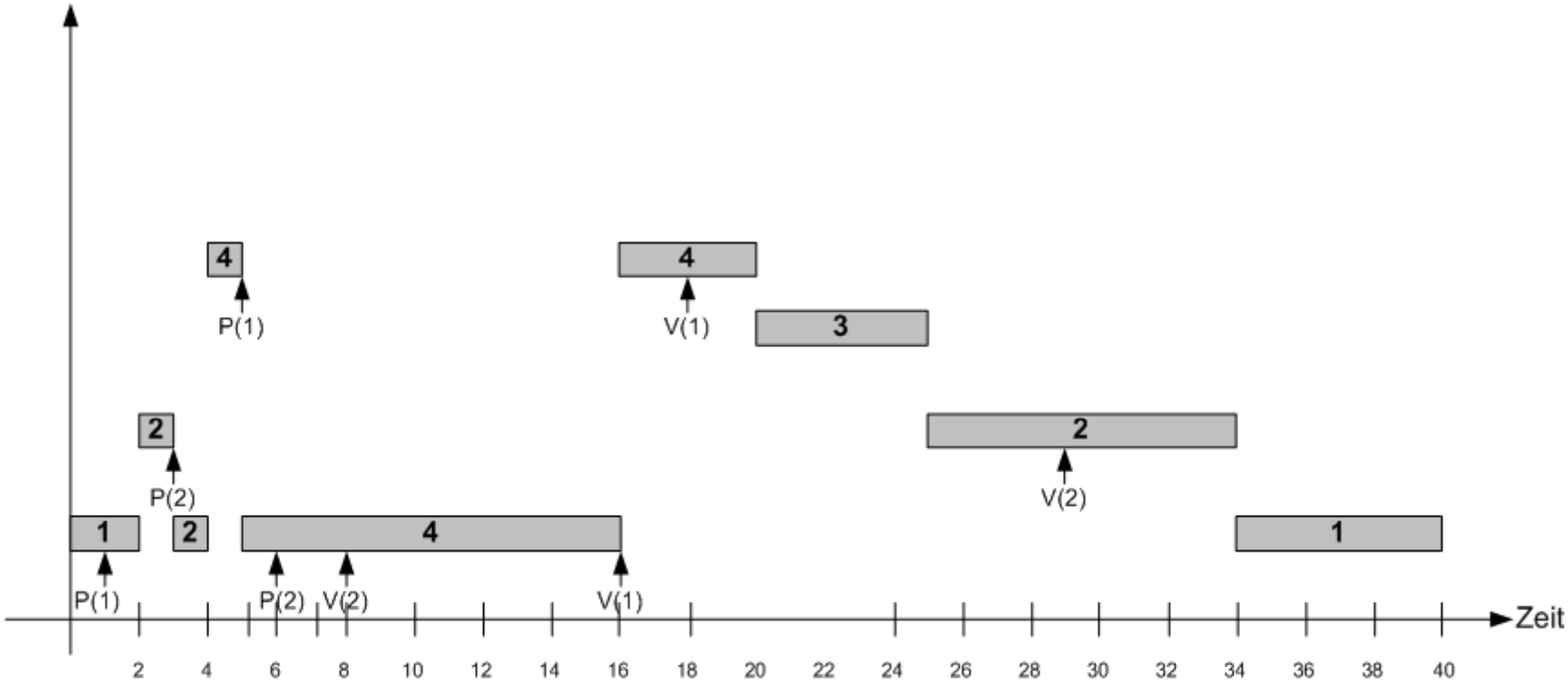
Lösung Teilaufgabe d) mit Prioritätsvererbung



Lösung Teilaufgabe d) mit sofortigen Prioritätsgrenzen



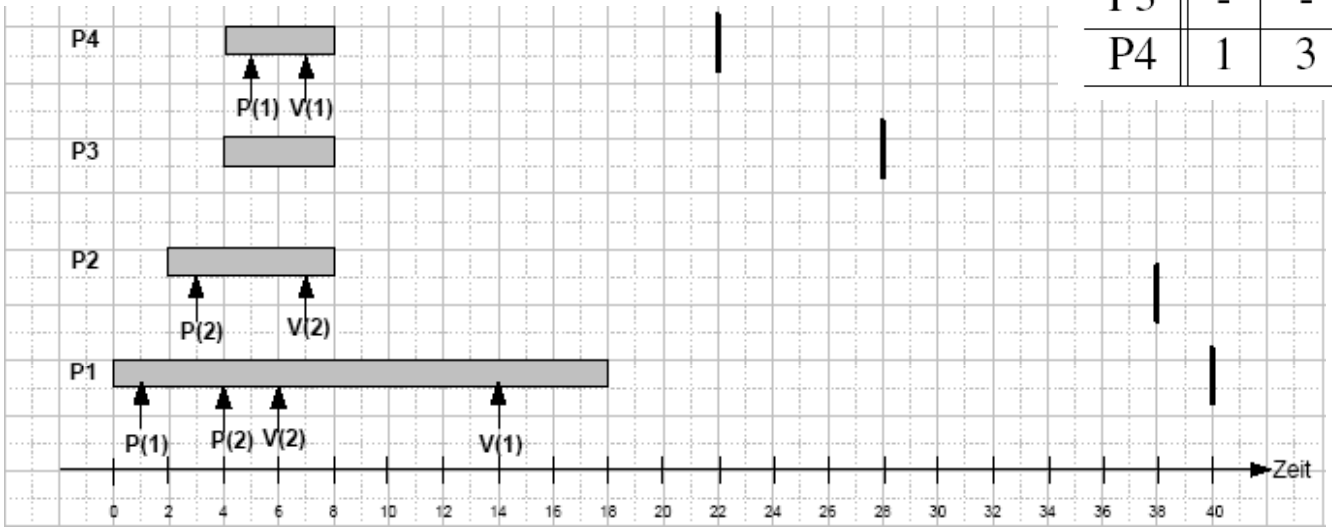
Lösung Teilaufgabe d) mit Prioritätsgrenzen



Klausur WS 07/08 (20 Punkte = 20min)

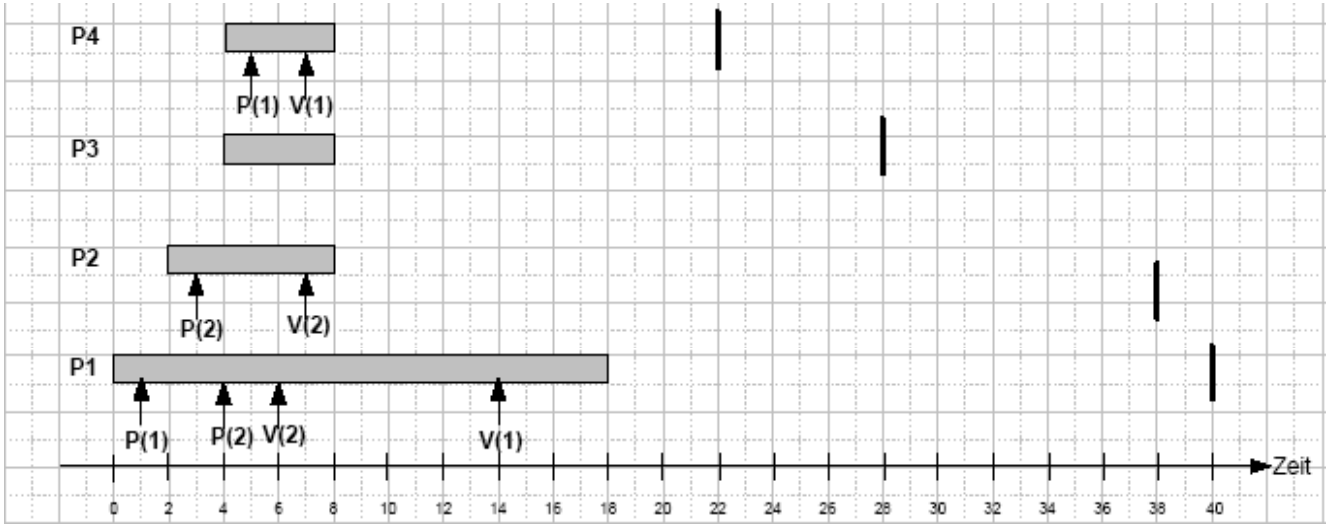
- Startzeiten s: s(P1)=0; s(P2)=2; s(P3)=4; s(P4)=4;
- Ausführungszeiten e: e(P1)=18; e(P2)=6; e(P3)=4; e(P4)=4;
- Fristen d: d(P1)=40; e(P2)=38; e(P3)=28; e(P4)=22;

	S1		S2	
	P	V	P	V
P1	1	14	4	6
P2	-	-	1	5
P3	-	-	-	-
P4	1	3	-	-



Klausur WS 07/08

- Ignorieren Sie zunächst die Semaphore. Zeichnen Sie einen Ausführungsplan für das Schedulingverfahren Earliest-Deadline-First.
- Ignorieren Sie zunächst die Semaphore. Zeichnen Sie einen Ausführungsplan für das Schedulingverfahren Least-Slack-Time (Zeitscheiben: 1).
- Zeichnen Sie nun den Ausführungsplan für das Schedulingverfahren Earliest-Deadline-First unter Berücksichtigung der Semaphoren.
- Wie könnte man das Schedulingverfahren modifizieren, um das in Teilaufgabe c) aufgetretene Problem zu beheben.

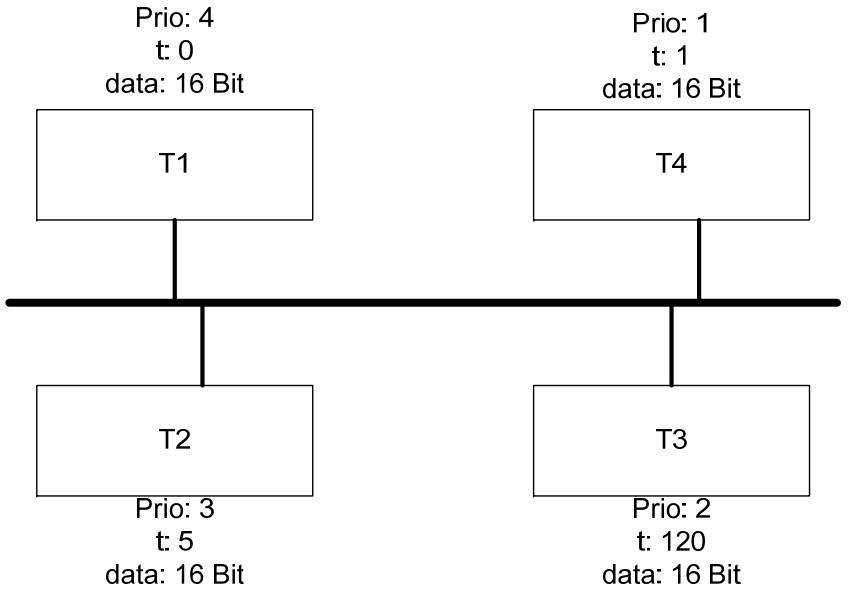


Kapitel 6

Echtzeitfähige Kommunikation

Klausur 06/07 (modifiziert) – CAN (8 Punkte = 8 min)

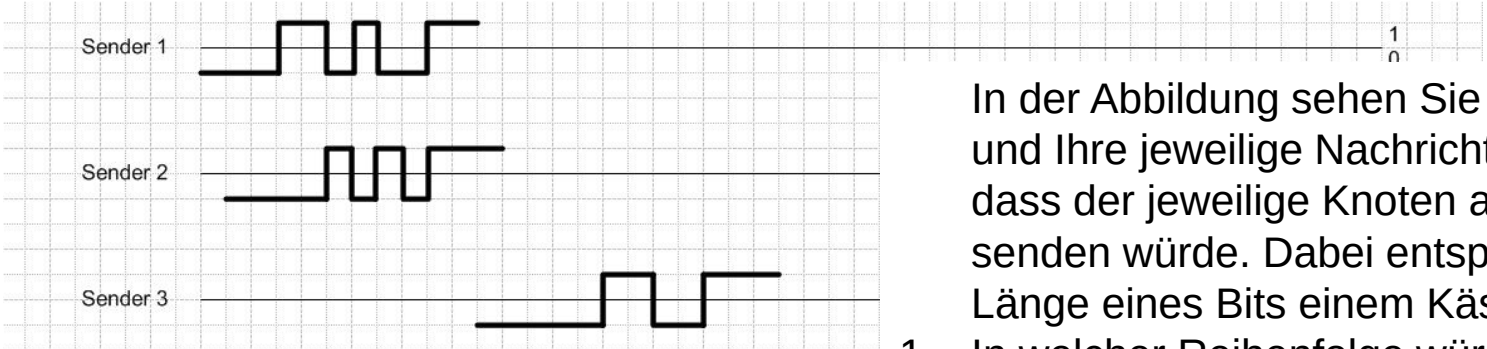
a) Geben Sie die Reihenfolge der Nachrichten an, die im Netzwerk bei Verwendung des CANProtokolls gesendet werden und begründen Sie ihre Antwort. **Zur Erinnerung**: Zusätzlich zu den Nutzdaten sind bei CAN 46 Bit Steuerungsdaten pro Nachricht notwendig. Zwischen den einzelnen Nachrichten ist eine Lücke von mindestens 3 Bit.



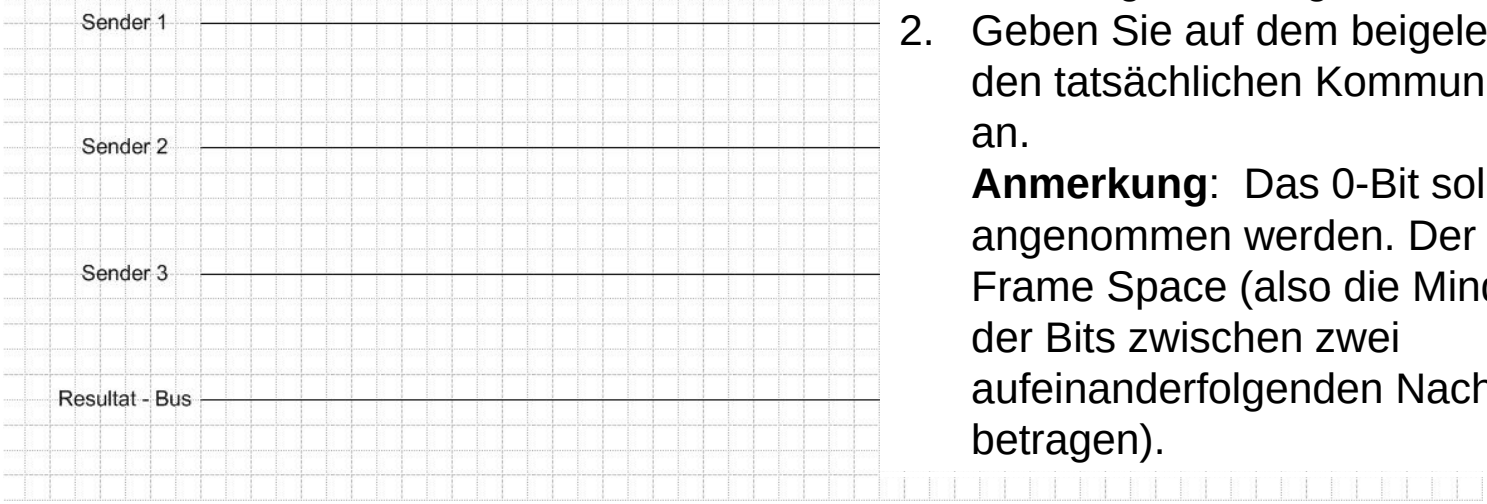
Annahmen: Bitsendedauer 1 Zeiteinheit
Priorität: 1 – hoch, 4 – niedrig

Lösung: Nachricht von T1 (einziger Rechner der zunächst senden will),
Nachricht von T4 (Priorität),
Nachricht von T3 (Priorität),
Nachricht von T1

Wiederholungsklausur SS 07 – CAN-Protokoll (15 Punkte = 15min)



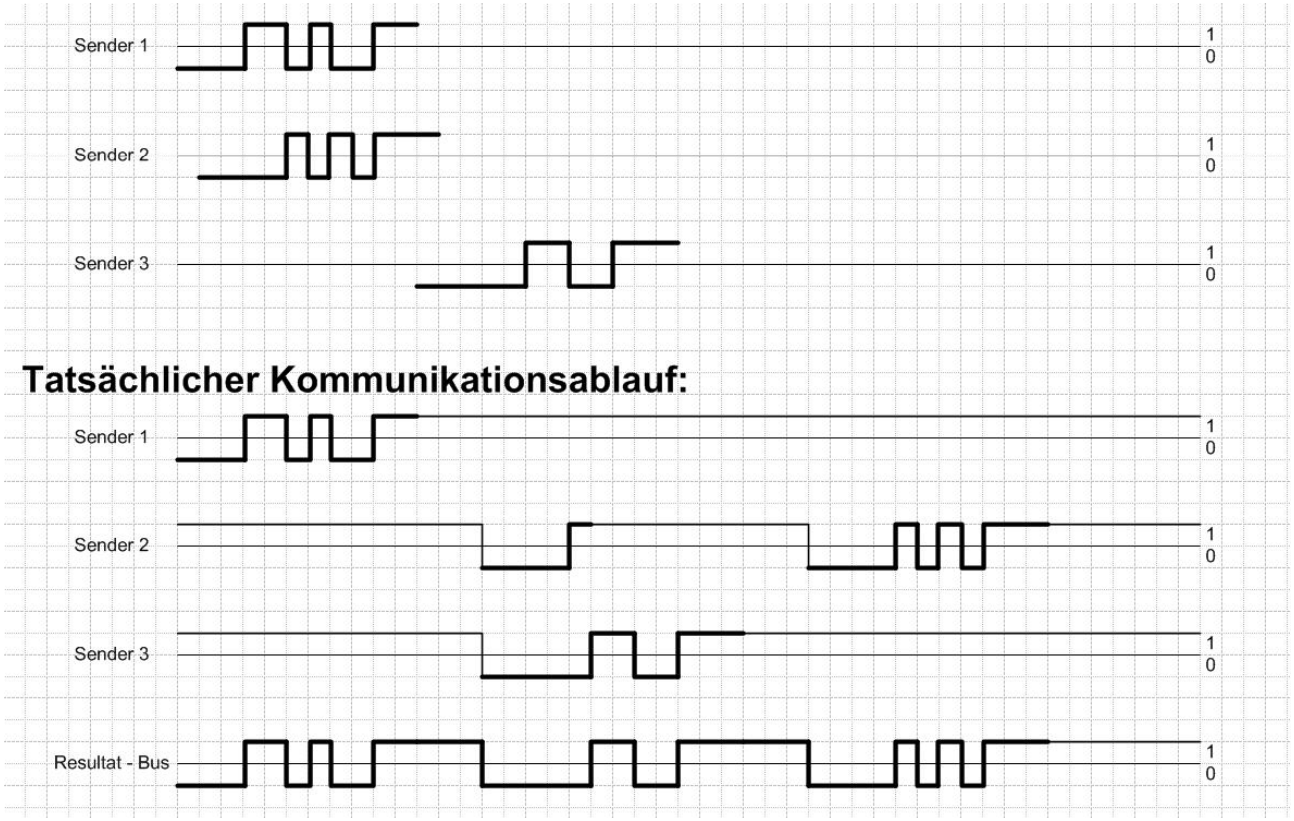
Tatsächlicher Kommunikationsablauf:



In der Abbildung sehen Sie drei Knoten und Ihre jeweilige Nachricht für den Fall, dass der jeweilige Knoten als einziger senden würde. Dabei entspricht die Länge eines Bits einem Kästchen.

1. In welcher Reihenfolge würden die Nachrichten gesendet werden, wenn alle Knoten gleichzeitig senden würden?
2. Geben Sie auf dem beigelegten Blatt den tatsächlichen Kommunikationsablauf an.
   Anmerkung: Das 0-Bit soll als dominant angenommen werden. Der Intermission Frame Space (also die Mindestanzahl der Bits zwischen zwei aufeinanderfolgenden Nachrichten soll 3 betragen).

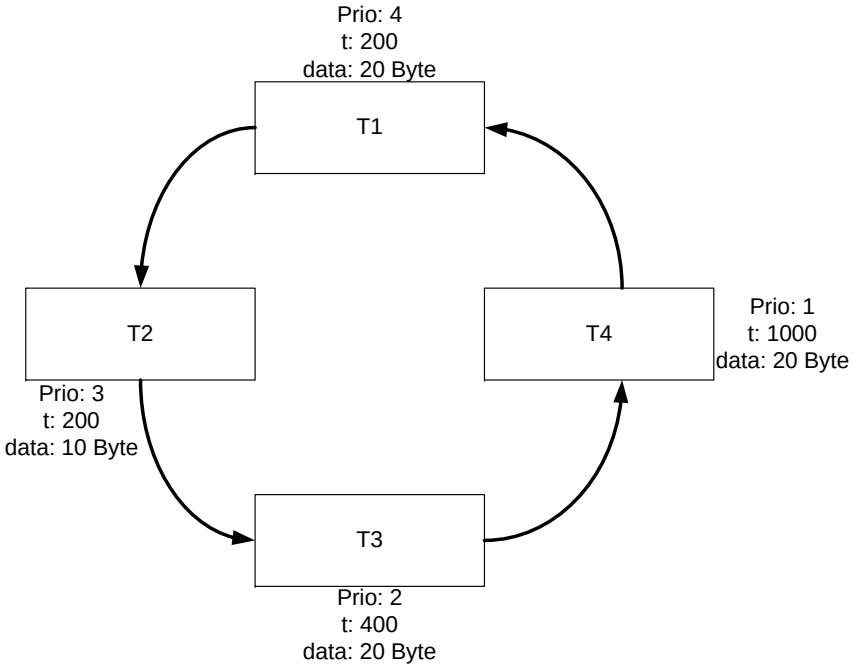
Wiederholungsklausur SS 07 – Lösung



Klausur 06/07 (modifiziert) – TokenRing (8 Punkte = 8 min)

a) Geben Sie die Reihenfolge der Nachrichten an, die im Netzwerk bei Verwendung des TokenRing-Protokolls gesendet werden und begründen Sie ihre Antwort.
Zum Zeitpunkt 0 soll dabei der Teilnehmer T1 im Besitz des Tokens sein.
Zur Erinnerung: Ein Token besteht aus insgesamt 3 Byte (8 Bit Startbegrenzer, 8 Bit Zugriffskontrolle mit Zugriffspriorität und Reservierungspriorität, 8 Bit Endbegrenzer). Der Header für ein Datenpaket besteht aus mindestens 20 Byte.

Lösung: Token, Nachricht von T2, T3 reserviert, T1 kann wegen höherer Priorität von T3 nicht reservieren, Token, Nachricht von T3, T4 reserviert, T1 kann wegen höherer Priorität von T4 nicht reservieren, Token, Nachricht von T4, T1 reserviert, Token, Nachricht von T1



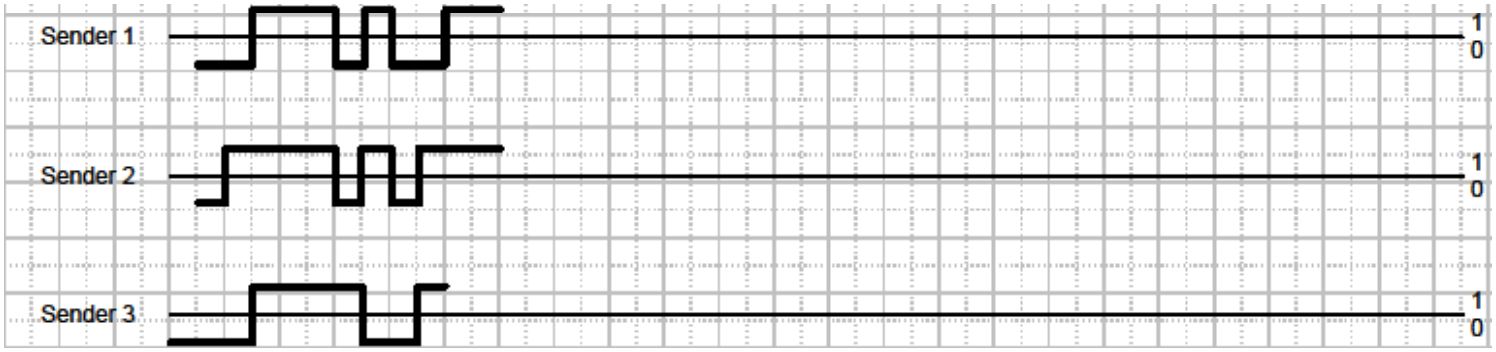
Annahmen: Bitsendedauer 1 Zeiteinheit
Laufzeit zwischen 2 Knoten 200 Zeiteinheiten
Priorität: 1 – hoch, 4 – niedrig

Klausur Wintersemester 07/08 (20 Punkte = 20min)

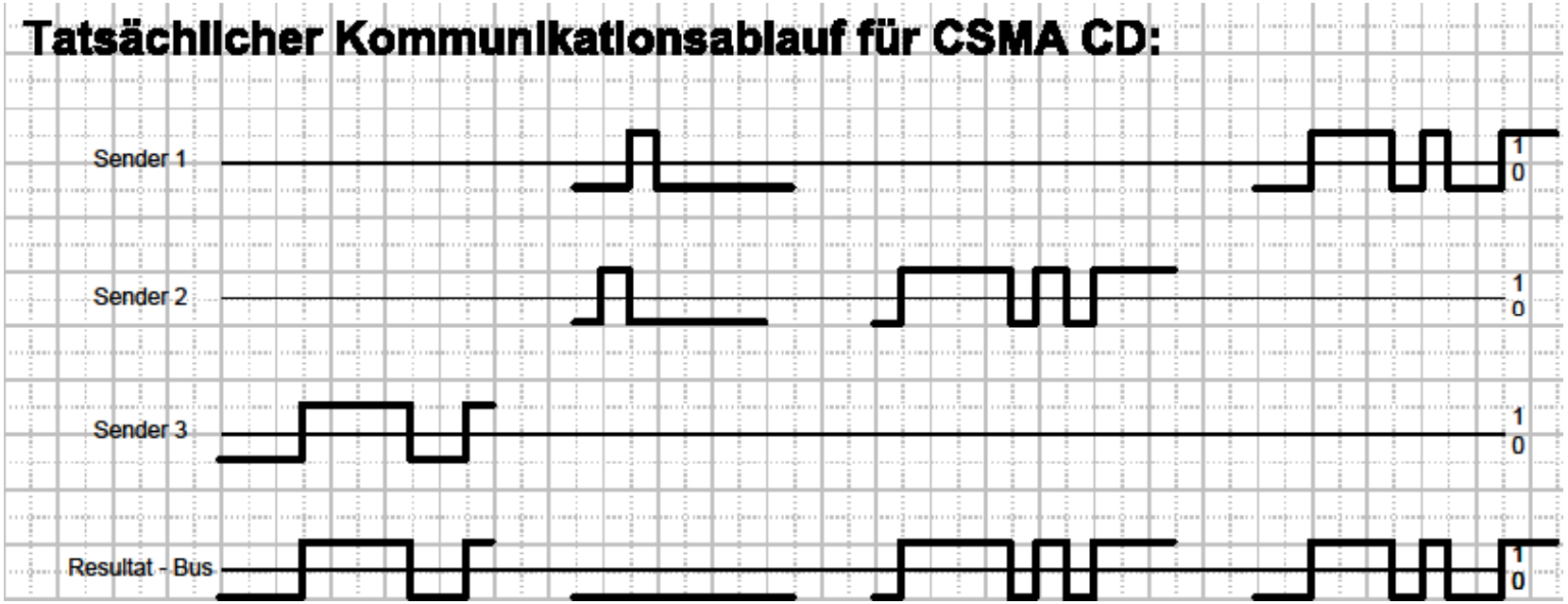
In der Abbildung sehen Sie drei Knoten und Ihre jeweilige Nachricht für den Fall, dass der jeweilige Knoten als einziger senden würde. Dabei entspricht die Länge eines Bits einem Kästchen.

Gehen Sie davon aus, dass für die Lösung der Aufgabe alle Daten bitsynchron übertragen werden. Das JAM-Signal soll aus einer Folge von 5 0-Bits bestehen. Das 0-Bit ist dominant. Zwischen zwei Nachrichten gibt es eine Pause (interframe gap) von mindestens 3 Bits.

- Zeigen Sie für die angegebenen Nachrichten einen möglichen Ablaufplan in CSMA-CD.
- Geben Sie den entsprechenden Plan in CSMA-CA an.
- Für ein konkretes Netzwerk ist die maximale Signallaufzeit mit einer Zeiteinheit angegeben. Welche der angegebenen Bitübertragungsdauern würden Sie für CSMA/CA auswählen. Geben Sie eine knappe Begründung für Ihre Antwort.
  - 0,5 Zeiteinheiten
  - 1 Zeiteinheit
  - 4 Zeiteinheiten
  - 10 Zeiteinheiten



Klausur Wintersemester 07/08 – Lösung CSMA/CD



Grundsätzlich gibt es natürlich sehr viele Lösungen durch die verschiedenen möglichen Wartezeiten aufgrund des Backoff-Verfahrens.

Klausur Wintersemester 07/08 – Lösung CSMA/CA