Beuth Hochschule für Technik Berlin
Fachbereich VI – Informatik und Medien
Dozent: H. Dinse

Juli 2016

1. Klausur
Echtzeitsysteme
Technische Informatik Bachelor

Name : [redacted]

Studien - Nr. : [redacted]

Letzter Versuch : JA ☐ NEIN ☒

Erreichbare Punktzahl : 90

Erreichte Punktzahl : 72

Note : 2,0

Bewertung:	% ab	Note	Punkte ab
	95	1,0	86
	90	1,3	81
	85	1,7	77
	80	2,0	72
	75	2,3	68
	70	2,7	63
	65	3,0	59
	60	3,3	54
	55	3,7	50
	50	4,0	45
	0	5,0	0

Hinweis: Es werden nur Studierende zur Klausur zugelassen, die alle Übungen abgegeben haben und in dieser Lehrveranstaltung eingeschrieben sind.

RTS Klausur – TI

Beachten Sie bitte:

- Als Hilfsmittel ist ein selbständig vorbereitetes und handschriftlich einseitig beschriebenes DIN-A4-Blatt zugelassen.
- Als Hilfsmittel ist ein Taschenrechner zugelassen.
- Mit Bleistift oder Rotstift geschriebene Ergebnisse werden nicht gewertet.
- Die Bearbeitungszeit beträgt 90 Minuten.
- Schalten Sie Ihre Mobiltelefone aus.

1 Welche Eigenschaften muss ein Echtzeitbetriebssystem haben? (min. 3 Eigenschaften) (3 P)

- Rechtzeitigkeit
- Bestimmbarkeit
- Verlässlichkeit
- Sicherheit

3 P

2 Dürfen vorgegebene Deadlines bei harten Echtzeit-systemen überschritten werden? (2 P)

Nein, da harte Echtzeit durch das einhalten von Deadlines definiert ist.

2 P

3 Sind „embedded systems“ immer Echtzeitsysteme? (2 P)

Nein, sie sind es nur wenn sie die Eigenschaften von Echtzeisystemen erfüllen.

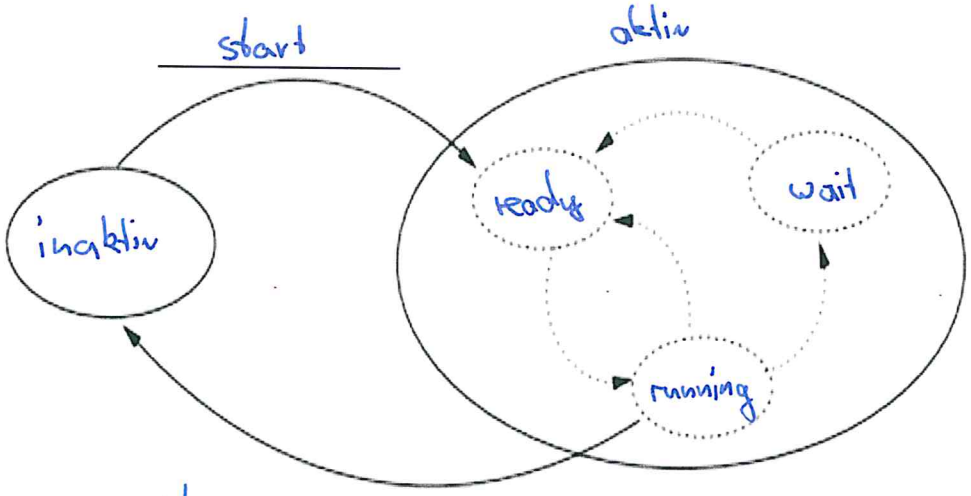
2 P

4 Welche Aufgabe hat der Prozess-Scheduler eines Betriebssystems? (2 P)

Der Prozess-Scheduler steuert Prozesse indem er ihnen Rechen-Zeit auf der CPU zuweist oder entzieht.

2 P

5 Skizzieren Sie die möglichen Zustände und Zustands-übergänge einer Task in einem Echtzeitbetriebssystem. Vereinfachte Darstellung (nutzen Sie die Grafik)(6 P)



6 P

~~Ein prozess ist Grundsätzlich inaktiv wird er gestart geht er in den ready zustand über~~ inaktiv: Task läuft nicht, aktiv: Task läuft, ready: bereit zum ausführen, wait: Task blockiert

6 Ein Steuerungssystem mit einem ein „core“ Prozessor ist durch folgende Daten gekennzeichnet: (12 P)

Task 1	$e_1 = 1$ ms	$p_1 = 4$ ms
Task 2	$e_2 = 1$ ms	$p_2 = 5$ ms
Task 3	$1 \text{ ms} \leq e_3 \leq= 2$ ms	$p_3 = 10$ ms
Task 4	$e_4 = 2$ ms	$20 \text{ ms} \leq p_4 \leq 40$ ms

Task 1 besitzt die höchste Priorität, Task 2 die zweithöchste und Task 4 die niedrigste Priorität.

6.1 Berechnen Sie die Auslastung des Rechners für Task 1 und 2. $\frac{1}{4} + \frac{1}{5} = \frac{9}{20} = 0{,}45 = 45\%$ ✓

6.2 Durch welche Werte ist der „Worst Case“ in Task 3 u. 4 gekennzeichnet?

Task 3 = $e_3 = 1$ ms $p_3 = 10$ ms, $e_4 = 2$ ms $p_4 = 20$ ms

6.3 Geben Sie für den „Worst Case“ Fall die Gesamtauslastung des Systems an.

$\frac{9}{20} + \frac{1}{10} + \frac{2}{20} = \frac{13}{20} = 0{,}65 = 65\%$ ✓

6.4 Erfüllt das System die Auslastungsbedingung?

Ja, da $65\% < 100\%$ ✓

6.5 Zeichnen Sie die Anforderungsfunktion (worst case), d.h. das zeitliche Auftreten der Ereignisse. Nutzen Sie hierfür das untere Diagramm.

6.6 Zeichnen Sie die Rechnerkernbelegung durch die 4 Tasks (worst case).

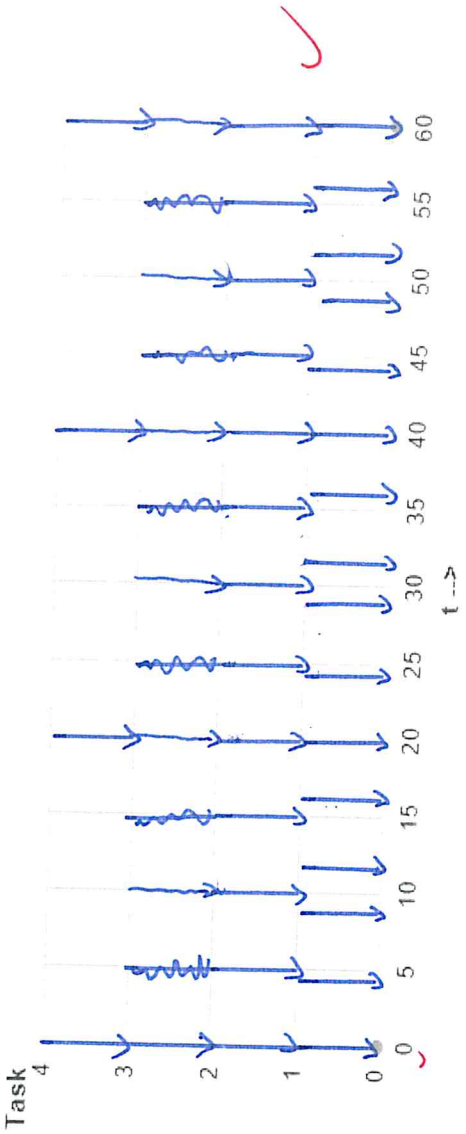
Zu 6.1:

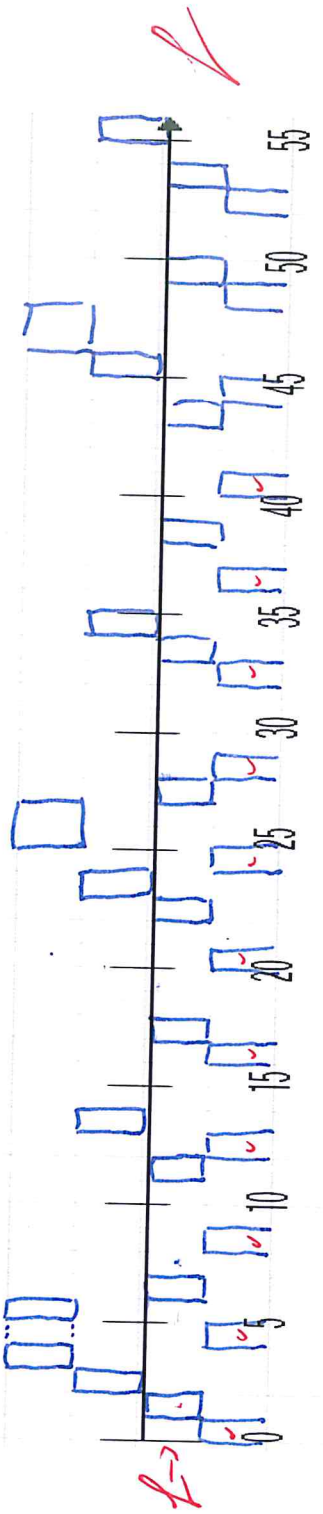
Zu 6.2:

Zu 6.3:

Zu 6.4:

Zu 6.5 und 6.6


Task
4
3
2
1
0
0
5
10
15
20
25
30
35
40
45
50
55
60
t -->


0
5
10
15
20
25
30
35
40
45
50
55
t -->

7 Um ein System als ein Realzeitsystem einsetzen zu können, müssen zwei Bedingungen unbedingt erfüllt sein. Nennen und beschreiben Sie diese beiden Bedingungen. (6 P)

1. Die Gesamtauslastung ~~des Systems~~ für jeden Rechenkern muss $\leq 100\%$ sein, um die Zeitbedingungen zu gewährleisten $\Rightarrow p_{ges} = \sum_{i=0}^{n} \frac{t_{e\,max}}{t_{p\,min}} \leq 100\%$ ✓, dies ist die notwendige Bedingung.

2. Für alle t_p muss gelten, dass die Reaktionszeit des Systems kleiner oder gleich der maximal zulässigen Reaktionszeit und größer oder gleich der minimal zulässigen Reaktionszeit ist $\Rightarrow t_{D\,min} \leq t_{R\,min} \leq t_{R\,max} \leq t_{D\,max}$, dies ist die hinreichende Bedingung.

6 P

8 Zur Synchronisation von gemeinsam verwendeten Ressourcen kann man Semaphore einsetzen. Dijkstra hat zwei Semaphore nach deren Funktionalität beschrieben.
A) Welche beiden Semaphore sind es und welche Aufgabe haben sie?
B) Beschreiben sie die Funktionsweise eines dieser Semaphore. (6 P)

a.) Es gibt die Semaphoren V() und P(), deren Aufgaben, in der gegebenen Reihenfolge sind: aquire und release.
b.) Wenn eine Semaphore initialisiert wurde, wird die Semaphore mit P() verringert und mit V() erhöht. Ist die Semaphore aufgrund der auf ihr ausgeführten Operationen, kleiner 0 wird die geschützte Ressource blockiert, ist sie, größer 0 wird die geschützte Ressource frei gegeben.

5P

9 Was bedeutet bei Semaphor die Formulierung, die Operationen sind „Atomar“? (2 P)

Atomare Operationen sind Aufgaben, die nur als ganzes durchgeführt werden dürfen. Auf die Variablen dieser Operation darf, während ihrer Durchführung, nicht von anderen Prozessen /Tasks aus zugegriffen werden.

2P

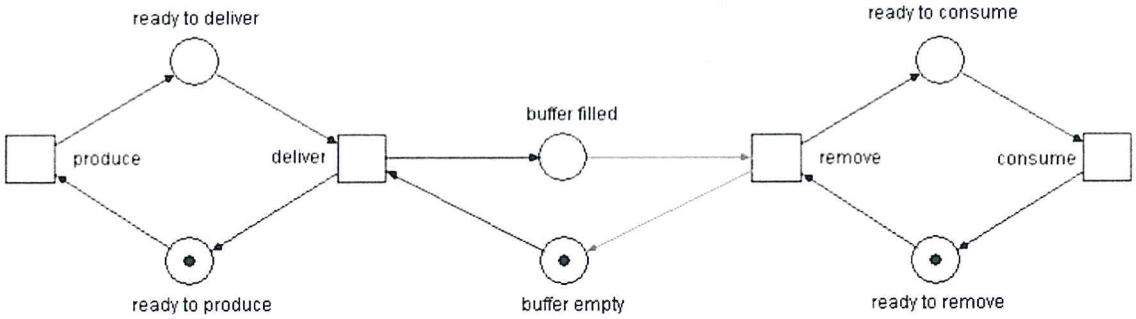
10 A) Was bedeutet Prioritätsinversion bei einem „Präemptivem Prioritäten Scheduling „ und B) welche Auswirkungen kann dieses für ein Echtzeitsystem haben? (6 P)

Präemptives prioritätsscheduling ist ein Verfahren, bei dem Tasks mit niederer Priorität unterbrochen werden wenn Tasks höherer Priorität ausstehen. Die Inversion dieses Verfahrens definiert sich dadurch, dass Tasks niederer Priorität nicht unterbrochen werden, selbst wenn Tasks höherer Priorität anstehen. Eine Folge der Inversion kann unter anderem sein, dass sich das System nicht mehr in der Echtzeit befindet, da Tasks ihre Deadlines nicht mehr einhalten können.

3 P

11 Petri-Netze sind eine der ersten formalen Modellierungs-methoden. (4 P)

Ein Netzgraph N besteht aus den Komponenten P (Menge von Plätzen), T (Menge der Transitionen) und F (Flussrelationen). N = (P, T, F).
Erläutern Sie den nachstehenden PTF Graphen.



Die Marke bei ready produce geht zu produce. Ist dies Task abgeschlossen, befindet sich die Mark im Zustand ready to deliver. Befindet sich nun ein zweite Marke im Zustand Buffer empfg., gehen beide Marken in den ~~Task~~ Transition deliver. Die deliver mark geht zurück zu ihrem Anfangszustand und die Buffermarke ist nun im Zustand Buffer full. Ist nun eine Marke in ready to remove, gehen die Buffer - sowie die remove - Marke in den ~~Task~~ Transition remove. Die buffer - Marke kehrt nun zu ihrem Ausgangszustand zurück und die remove - Marke wechselt zum Zustand ready to consume und schließlich zum Task consume, von dem sie dann, wenn der Task abgeschlossen ist zurück zu ihrem Ausgangszustand wechselt. 4P

12 Welchen Informationen liefert die Funktion „clock_gettime()“ (genau beschreiben)? (2 P)

Die Funktion clock_gettime() bekommt zwei, in der time.h, definierte structs übergeben. Zum einen die timezone zum anderen ein struct in dem die Zeit seit dem 1.1.1970 in Stunden, Sekunden und Mikrosekunden eingetragen wird. Bei sucess wird eine 0 zurück gegeben sonst -1. 0P

13 A) Mit welcher Funktion erzeugt man „shared memory" und wie bindet man diesen Bereich in seiner Prozessumgebung ein?
B) Wie können selbstständige Prozesse, die mit fork und exec aufgerufen wurden, auf den „shared memory" Bereich des Eltern-Prozesses zugreifen? (6 P)

a.) Mit shmget kann shared memory erzeugt werden und mit shmat kann dieser eingebunden werden.

b.) Von geforkten bzw. exec prozessen kann mithilfe von memory keys/ID, sofern ausreichend privilegien bei shmget bzw. shmat gesetzt wurden, zugegriffen werden.

6 P

14 Das nachfolgende Programm steht zur Verfügung.

a) Beschreiben Sie den Programmablauf (nur die freien Abschnitte / Lücken im Code Bereich beschreiben).

b) Welches ist die zu schützende Ressource?

c) Welche Werte gibt a) Thread 1, b) Thread 2 aus? Geben sie diese der Reihenfolge nach an. **(31 P)**

Zu b)

int count 1

Zu c)

Thread 1: 1,2,3,7,8,9,10

Thread 2: 4,5,6,7 3

Programmcode:

```
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <pthread.h>

pthread_mutex_t count_mutex    = PTHREAD_MUTEX_INITIALIZER;
pthread_cond_t  condition_var = PTHREAD_COND_INITIALIZER;
```

deklaration ein pthread mutex und ein condition variable 1

```
void *functionCount1();
void *functionCount2();

int  count = 0;

#define COUNT_DONE  10
#define COUNT_HALT1  3
#define COUNT_HALT2  6

int main(void)
{
   pthread_t thread1, thread2;

   pthread_create( &thread1, NULL, &functionCount1, NULL);
   pthread_create( &thread2, NULL, &functionCount2, NULL);
```

(Parameter mit erläutern)
erzeugen von zwei Threads mit Thread Attribut Null und jeweils einem Funktionspointer ohne Übergabeparameter 3

```
    pthread_join( thread1, NULL);
    pthread_join( thread2, NULL);
```

warte auf das beenden der beiden geöffneten Threads bis fortgefahren wird.

2

```
    printf("Final count: %d\n",count);

    exit(EXIT_SUCCESS);

}

void *functionCount1()
{
    for(;;)
    {
        pthread_mutex_lock( &count_mutex );
```

sperren und betreten des kritischen Bereichs

2

```
        pthread_cond_wait( &condition_var, &count_mutex );
```

warte bis cond-signal mit condition_var aufgerufen wird

2

```
        count++;
        printf("Counter value functionCount1: %d\n",count);

        pthread_mutex_unlock( &count_mutex );
```

entsperre und verlasse kritischen Bereich.

2

```
        if(count >= COUNT_DONE) return(NULL);
```

```
    }
}

void *functionCount2()
{
    for(;;)
    {
       pthread_mutex_lock( &count_mutex );
```

sperre und betrete kritischen Bereich 2

```
       if( count < COUNT_HALT1 || count > COUNT_HALT2 )
```

wenn count kleiner 3 oder größer 6 ist rufe cond_signal auf. 2

```
       {
          pthread_cond_signal( &condition_var );



       }
       else
       {
          count++;
          printf("Counter value functionCount2: %d\n",count);
       }

       pthread_mutex_unlock( &count_mutex );
```

entsperren und verlassen des kritischen Bereichs 2

```
       if(count >= COUNT_DONE) return(NULL);
```

Wenn count größer oder gleich 10 ist, verlasse die Funktion.

2

```
    }
}
```