Realzeit Nachweis

Josef Mueller, Isabella Schoen
Gruppe1

Realzeit Nachweis

Allgemeine Fragen

1. Welche beiden Bedingungen überprüft der Realzeitnachweis?

1.1

> A: Der Realzeitnachweis überprüft die erste und zweite Realzeitbedingung.
> Bei der ersten Bedingung geht es um die Auslastung eines Rechners, die kleiner oder gleich der Anzahl der CPU Kerne sein muss. Die zweite Bedingung beschäftigt sich mit der Reaktionszeit der Tasks. Es muss folgendes gelten: `t_Dmin <= t_Rmin <= t_Rmax <= t_Dmax`. (S. 221f.)

2. Was ist eine Rechenzeitanforderungsfunktion?

> A: Die Rechenzeitanforderungsfunktion gibt an, wie viel Rechenzeit im Worst Case von einer Task in einem Intervall `I` angefordert wird. Sie geht dabei aus der Ereignisfunktion hervor. (S. 230f., http://mediatum.ub.tum.de/doc/603880/document.pdf – S. 43 unten)
>
> Bei Unterteilung des Wortes in *Rechenzeit*, *Anforderung* und *Funktion*, lässt sich Folgendes aussagen: jede Task fordert vom Rechner eine bestimmte Rechenzeit an. Diese Anforderung lässt sich mithilfe einer Funktion beschreiben – der Rechenzeitanforderungsfunktion. Ist die Summe aller Funktionen überhalb der Winkelhalbierenden, welche die maximal zulässige Auslastung des Computers beschreibt, so ist der Rechner ausgelastet. Andernfalls gibt der Abstand zwischen der Summe und der Winkelhalbierenden die Gesamtauslastung des Rechners an, welche durch die Tasks hervorgerufen wurde. (S. 230ff.)

3. Wie kann es bei einem gemeinsamen Zugriff auf Datenbereiche zu inkonsistente Daten kommen?

> A: Durch Raceconditions (dt. Wettlaufsituationen), bei denen mindestens zwei Tasks um dieselbe Ressource konkurrieren, kann es zu inkonsistenten Daten kommen.
> Dabei kann es passieren, dass eine Task die von einer anderen Task in eine Ressource geschriebenen Daten überschreibt oder durch die nicht deterministische Ausführung Daten gelesen werden, bevor sie aktualisiert wurden.

4. Welche zwei Probleme können beim Zugriff auf Ressourcen unter gegenseitigem Ausschluß (mutual exclusion) auftreten?

> A: Livelocks, bei denen der Lock für die Ressource nicht freigegeben wird, und Prioritätsinversion, bei denen eine hochpriore Task auf die Freigabe einer Ressource durch eine niedrigpriore Task warten muss.

1.2

5. Erklären Sie den Begriff Prioritätsinversion. Geben Sie ein Beispiel für eine unkontrollierte Prioritätsinversion an.

> A: Sobald eine hochpriore Task auf die Freigabe einer Ressource durch eine niedrigpriore Task warten muss, spricht man von Prioritätsinversion. Beispiele:

- Wenn ein Privatpatient beim Arzt warten muss, bis der Nicht-Privatpatient behandelt wurde. Der Arzt ist die Ressource.
- Wenn ein Lkw einen anderen Lkw auf einer zweispurigen Straße überholt und ein Krankenwagen dafür warten muss. Die linke Fahrspur ist die Ressource.
- Wenn ein Krankenwagen an einer geschlossenen Schranke auf den Zug warten muss. Die Straße ist die Ressource.

6. Welche Protokolle zur Behandlung der Prioritätsinversion kennen Sie?

- Priority Inheritance Protokoll
- Priority Ceiling Protokoll
- Stack Based Priority Ceiling Protocol

2.1

7. Warum kann bei NPCS (Non Preemptive Critical Section) keine Deadlock Situation auftreten?

> A: NPCS ist eine Unterbrechungssperre, bei der Interrupts und Task-Wechsel gesperrt werden. (S. 28)

2.2

8. Nach welchen Scheduling-, Ressourcen-Zuteilungs- und Prioritäten-Vererbungs-Regeln funktionieren PIP und PCP?

Name	Scheduling	Ressourcen	Prioritäten
PIP	höherpriore Tasks werden bevorzugt, können somit niedrigpriore Tasks unterbrechen. Jedoch musst beachtet werden, wenn eine niedrigpriore Task eine Ressoruce, welche für eine höherpriore Task bestimmt war, blockiert, erbt die Task diese höhere Priorität bis zur Abgabe der Ressource.	1. Ressource frei: angefragte Task besetzt diese bis zur Freigabe; 2. Ressource besetzt: anfragende Ressource wird bis zur Freigabe der Ressource blockiert	eine höherpriore Task vererbt ihre Priorität an eine niedrigpriore Task bis zur Abgabe der Ressoruce, welche für die höherpriore Task bestimmt war
PCP	wie PIP. weitere Info: die Task, die zuletzt nach einer Ressource fragt und ihre höherpriorisierte Task somit abgibt, bekommt auch als Erstes die Ressource.	Tasks, deren Priorität nicht hoch genug für die Ressource sind, erhalten diese nicht.	wie bei PIP

2.3

9

8. Wie wird der Deadlock durch PCP verhindert?

> A: Durch berechnete Blockierzeiten, in denen eine Task die Ressource blockieren kann -> stellt eine höherpriore Task mehrere Anfragen an mehrere Ressourcen, welchen von niedrigpriore Tasks verwendet werden, so wird die Wartezeit bzw. die Blockierzeit gekürtzt, indem die Tasks die blockierten Ressorucen schneller abgeben und die höherpriore Task somit schneller Zugriff auf angefragte Ressorucen hat --> die Task wartet damit maximal auf nur eine Ressource

10

9. Stellen Sie die folgenden Eigenschaften der Protokolle gegenüber

Protokolle	PIP	PCP

Protokolle	PIP	PCP
Blockierarten	Task, welche die Ressource anfordert, erhält diese, sobald sie frei wird und wird solange blockiert - direkt	keine Blockierzeiten, da Ressourcen vorher angemeldet werden müssen und anhand von Prioritätsobergrenzen freigegeben und gesperrt werden
Vorteile	Vorzug von Tasks, die eine Ressource besitzen und dadurch auch Bevorzugung von niedrigen Prioritäten	Verhinderung von Deadlocks und der Gefahr durch Verschachtelungen 3.1
Nachteile	Vernachlässigung von höherpriore Ressourcen 3.2	Programmierer muss Ressourcen zu Beginn anmelden beim System

Prioritätengesteuertes Scheduling

Beschreibung

Ein (Einprozessor-)Steuerungssystem mit vier Tasks ist durch folgende Daten der gekennzeichnet:

Taskname	tP in ms	tE in ms
T1	3	1
T2	5	1,5
T3	7	0,5 ≦ tE(3) ≦ 1,25
T4	9 ≦ tP(4) ≦ 11	0,5

Die Tasks sind voll unterbrechbar. Der Scheduler des Systems arbeitet nach dem Rate-Monotonic Verfahren.

Aufgaben

1. Welche beiden Bedingungen müssen allgemein beim Realzeitnachweis überprüft werden?

> 1. Bedingung: Die Auslastung des Rechners muss kleiner oder gleich der Anzahl der Rechnerkerne sein. (S. 221)
> 2. Bedingung: Die maximale und minimale Reaktionszeit muss innerhalb der **zulässigen** Reaktionszeit (Deadline) sein. (S. 222)

2. Geben Sie das für den Realzeitnachweis relevante Task-Set in Form des periodischen Taskmodells an.

> A: (t_Pmin; t_Emax)
> A: T1: (3;1), T2: (5;1.5), T3: (7;1.25), T4: (9;0.5)

3. Ist der hinreichende Scheduling-Test erfüllt?

$$u = \sum_{j=1}^{n} \frac{t_{Emax,j}}{\min(t_{Dmax,j}, t_{Pmin,j})} \leq n \cdot (2^{1/n} - 1)$$

$$u = \underset{T_1}{\frac{1}{3}} + \underset{T_2}{\frac{1.5}{5}} + \underset{T_3}{\frac{1.25}{7}} + \underset{T_4}{\frac{0.5}{9}}$$

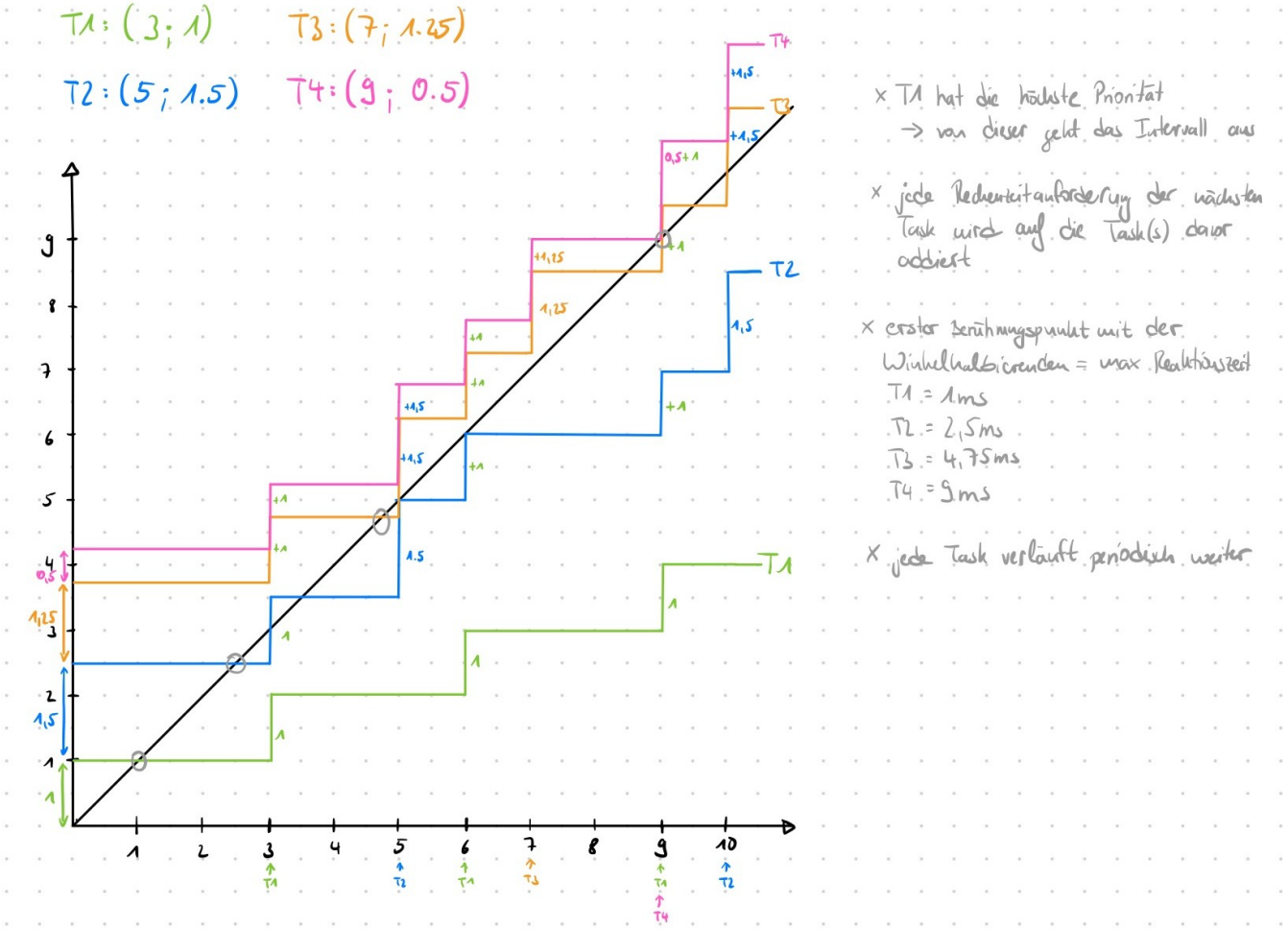
$$= 0,867 = 86,7\% \leq 4 \cdot (2^{1/4} - 1)$$

$$= 86,7\% \not\leq 0,7568 = 75,68\%$$

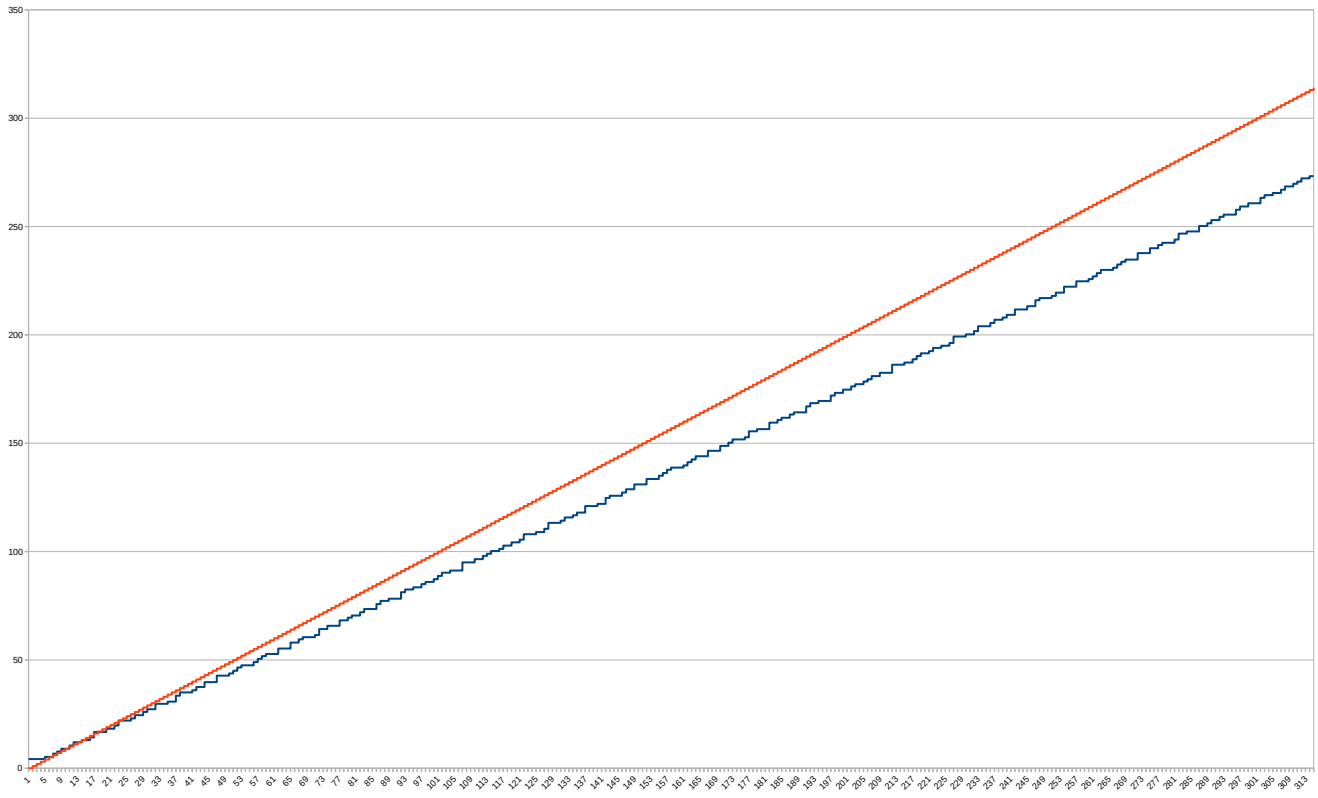
⇒ ein hinreichender Scheduling-Test ist nicht erfüllt

4. Führen Sie einen exakten Realzeitnachweis grafisch durch. Für die Überprüfung der Einhaltung der Deadlines soll die **Time-Demand Analysis** benutzt werden. Geben Sie die maximalen Reaktionszeiten der einzelnen Tasks an.

Im Folgenden ist der grafische Realzeitnachweis für alle Tasks zu sehen. Die maximalen Reaktionszeiten für die einzelnen Tasks sind der ebenfalls der Grafik zu entnehmen.



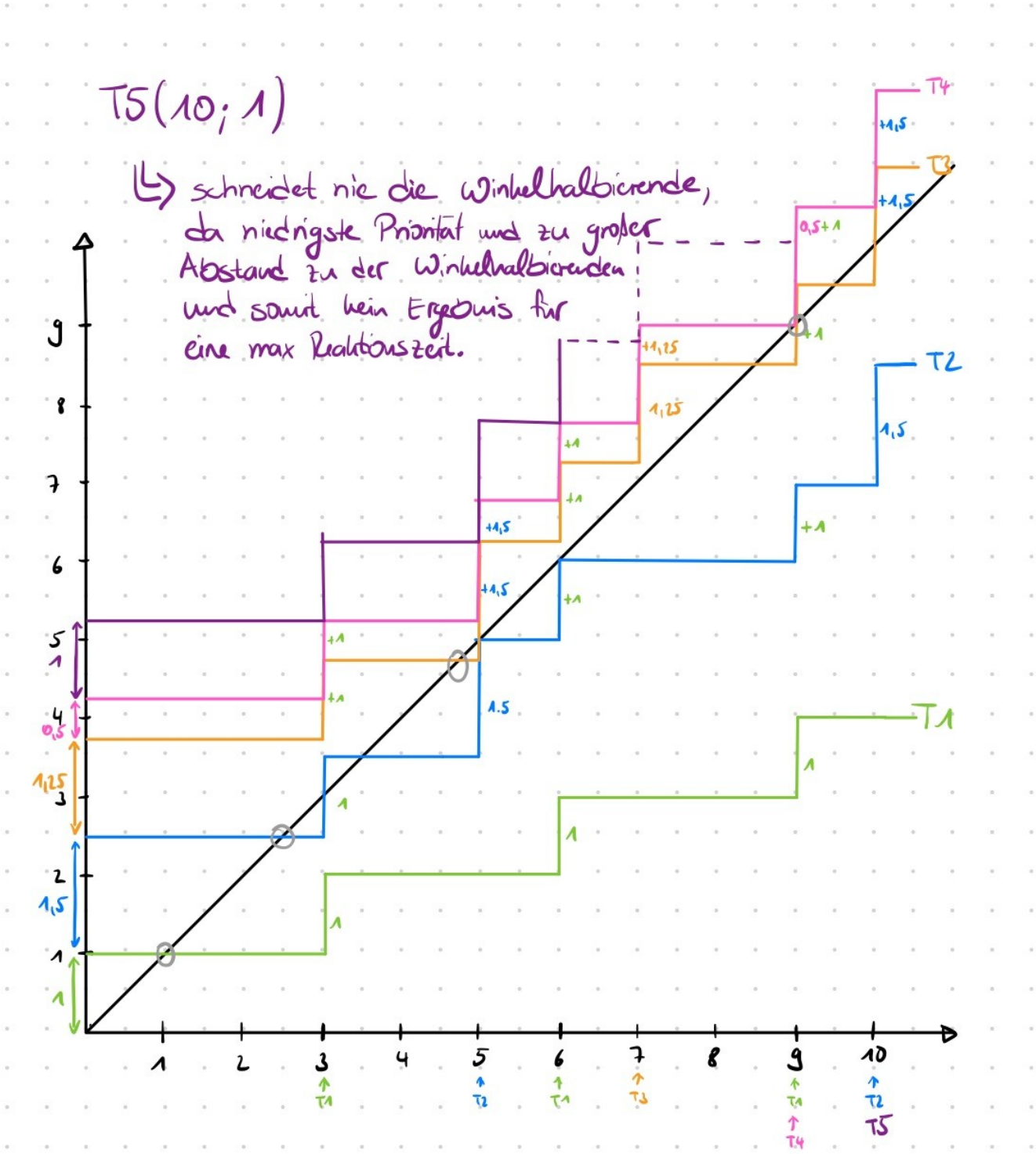
In der folgenden Grafik ist die Entwicklung der zeitlichen Anforderungen zusammen mit der maximal möglichen Auslastung (Winkelhalbierende) zu sehen.



Das Skript berechnet nur die Summe aller zeitlichen Anforderungen. Das Ergebnis ist in der darüber liegenden Grafik zu erkennen.

```
import numpy
import csv

kGV = 315
taskset = [(3, 1), (5, 1.5), (7, 1.25), (9, 0.5)]

intervals = set()
for i in taskset:
    for j in range(1, int(kGV / i[0])):
        intervals.add(j * i[0])

intervals = list(intervals)
intervals.sort()

t = numpy.zeros(kGV)
remainingTasks = taskset.copy()
t_l = sum(map(lambda task: task[1], taskset))

for i in range(kGV):
    t[i] = t_l
    for task in taskset:
        t_Pmin = task[0]
        t_Emax = task[1]
        if i % t_Pmin == 0 and i in intervals:
            t_l += t_Emax

rows = [[str(i), str(value)] for (i, value) in enumerate(t)]
print(rows)

with open('taskset', 'w') as f:
    write = csv.writer(f)
    write.writerows(rows)
```

5. Es soll eine weitere Task T5 mit der Periode 10 ms und der Ausführungszeit 1 ms in das System integriert werden. Erklären Sie anschaulich, warum kein korrekter Scheduling-Plan in dem neuen System (5 Tasks) existiert.



A: Da die Reaktion nicht innerhalb der Periode bzw. maximalen Deadline von 10 ms stattfindet - in der Grafik erkennbar -, kann laut der zweiten Realzeitbedingung kein Schedulingplan aufgestellt werden. (S. 229)

Index der Kommentare

1.1 A.k.a. hinreichende und notwendige Bedingung.

1.2 Nicht ganz inkorrekt ;-)
Erwartet habe ich hier eigentlich Livelock und Deadlock.

2.1 Und NPCS halt, habt ihr ja bei 7. sogar beschrieben

2.2 Und wie verhindern Interruptsperren einen Deadlock genau?

--> Das ist keine Beschreibung sondern eine Begriffserklärung

2.3 Task

3.1 - Meist kürzere Blockierzeiten

3.2 - Keine Verhinderung von Deadlocks
- Benachteiligung von Ressourcen-losen Tasks