jian-jia.chen [☺] tu-dortmund.de
lea.schoenberger [☺] tu-dortmund.de

Übungen zur Vorlesung
Eingebettete Systeme
Wintersemester 17/18

Theoretisches Aufgabenblatt 5

Hinweis: Dieses Aufgabenblatt wird am 20.12.2017 von 10.15 Uhr bis 11.45 Uhr in Raum OH14/E23 besprochen. Sie sind nicht verpflichtet, Ihre Lösungen abzugeben.

1 Schedulability Analysis

Betrachten Sie folgende Parameter für periodische Tasks mit $T_i = D_i$ unter einem *Rate Monotonic Schedule*:

	T_i	C_i
τ_1	5	1
τ_2	8	3
τ_3	9	2

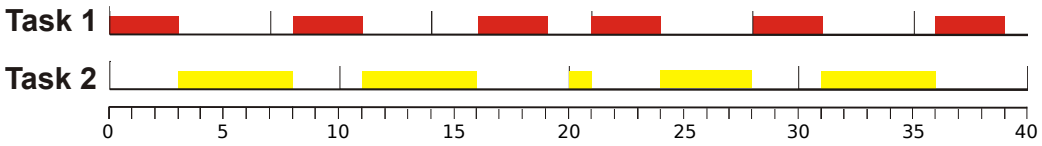
Nehmen Sie an, die Tasks sollen auf einem System mit nur einer CPU ausgeführt werden. Wie ist die Hyperperiode? Bestimmen Sie formal, ob ein gültiger Schedule mit den gegebenen Parametern existiert, sodass alle Deadlines eingehalten werden. Verifizieren Sie Ihr Ergebnis. Was lässt sich feststellen? Begründen Sie.

Lösung:

- Hyperperiod: 360
- Critical instant: schedulable.
- Liu/Layland: $0.7972 \not\leq 0.7797$, möglicherweise nicht schedulable.

2 Scheduling unabhängiger periodischer Tasks

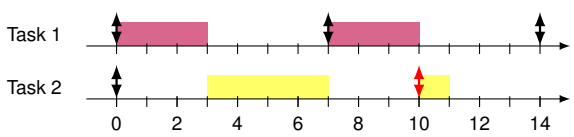
Gegeben sei folgendes Taskdiagramm:



Geben Sie die Parameter der Tasks an (C_i, T_i = D_i). Welches Schedulingverfahren könnte angewandt worden sein? Begründen Sie dies. Zeichnen Sie zum Vergleich das Diagramm, welches entsteht, wenn die Tasks mit einem *Rate Monotonic Schedule* angeordnet werden. Was fällt auf?

Lösung:

- EDF
- RM: Task 2 verpasst seine Deadline.



3 Harmonische Tasksysteme

Betrachten Sie folgende Parameter für periodische Tasks mit $T_i = D_i$:

C_i	0.2	2	2	1.5	1	14	28.8
T_i	2	6	12	24	24	72	288
D_i	2	6	12	24	24	72	288
U_i	0.1	1/3	1/6	0.0625	0.0417	0.195	0.1

Nehmen Sie an, die Tasks sollen auf einem System mit nur einer CPU ausgeführt werden. Bestimmen Sie formal, ob der Rate Monotonic Schedule *feasible* ist. Bestimmen Sie formal, ob der Earliest Deadline First Schedule *feasible* ist.

Lösung:

- RM: $\approx 0.9952 < 1$, feasible, vgl. Theorem von Kuo und Moc, es-chen-4.3.pdf, Folie 29.
- EDF: $\approx 0.9952 < 1$, feasible.

4 Critical Instant Theorem

Erklären Sie das Critical Instant Theorem in Ihren eigenen Worten. Wie in der Vorlesung erwähnt, ist das Critical Instant Theorem für Uniprocessor Fixed-Priority Scheduling sehr fragil, wenn die getroffenen Annahmen nicht zutreffen. Um das Critical Instant Theorem anwenden zu können, müssen verschiedene Bedingungen erfüllt werden. Bitte geben Sie an, welche der folgenden Bedingungen korrekt und welche inkorrekt sind. Wenn eine Bedingung inkorrekt ist, korrigieren Sie sie bitte.

- Das Task Set besteht nur aus voneinander unabhängigen Tasks.
- Das Task Set muss *strikt* periodisch sien.
- Der Scheduling-Algorithmus ist Fixed-Priority Preemptive Scheduling.
- Ein frühzeitiger Abschluss von Jobs ist nicht möglich. Wenn ein Job frühzeitig beendet wird, muss er trotzdem bis Ablauf seiner Worst-Case Execution Time ausgeführt werden (*spinning*).
- Kein Task unterbricht seine Ausführung freiwillig. Das heißt, ein Job kann seine Ausführung nicht selbst unterbrechen.
- Die relative Deadline eines Tasks kann größer sein als seine Periode.
- Scheduling Overheads (Context Switch Overheads) existieren nicht.
- Kein periodischer/sporadischer Task hat einen Release Jitter (der Zeit von der Ankunft eines Tasks bis zum Zeitpunkt, zu dem er bereit zur Ausführung ist).

Lösung:

- Korrekt.
- Inkorrekt.
- Korrekt.
- Inkorrekt.
- Korrekt.
- Inkorrekt.
- Korrekt.
- Korrekt.

5 Periodische Tasks

Ist das folgende Programm ein periodischer Task mit Periode T? Erklären Sie Ihre Antwort.

```
while (true)
   start := get the system tick;
   perform analog-to-digital conversion to get y;
   compute control output u;
   output u and do digital-to-analog conversion;
   end := get the system tick;
   timeToSleep := T-(end-start);
   sleep timeToSleep;
end while
```

Lösung: Nein. Eine Preemption/Unterbrechung des Tasks ist zwischen `end` und `sleep` möglich.

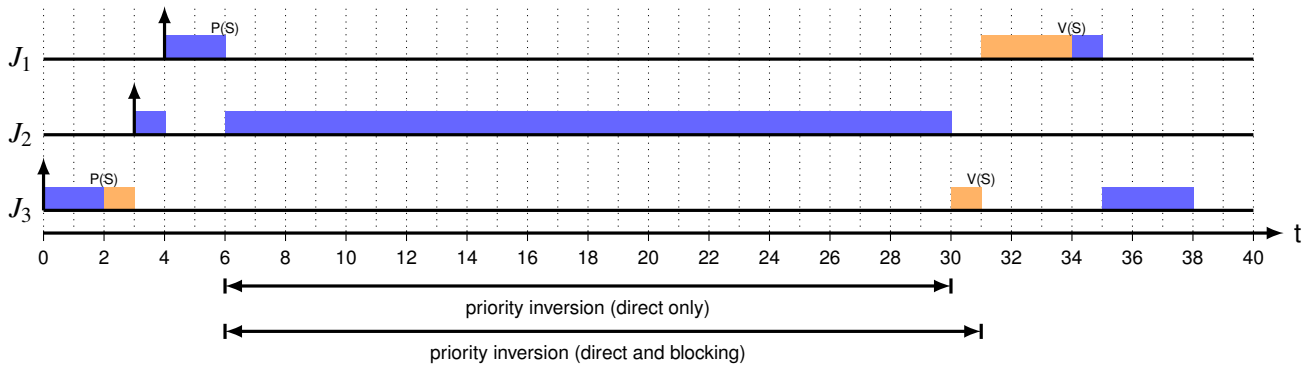
6 Priority Inversion

Wir betrachten ein System mit drei Jobs J_1, J_2 und J_3. Wir nehmen an, dass Job 1 die höchste und Job 3 die niedrigste Priorität besitzt. Diese Jobs verhalten sich wie folgt:

- Job J_3 führt P(S) 2 Zeiteinheiten nach dem Zeitpunkt, zu dem er verfügbar wird, und V(S) nach 4 Zeiteinheiten Ausführungszeit aus. (J_3 wird zunächst 2 Zeiteinheiten lang ausgeführt und betritt dann den kritischen Abschnitt, dessen Länge 2 Zeiteinheiten beträgt.) Seine Gesamtausführungszeit beträgt 8 Zeiteinheiten.
- Job J_2 hat eine Gesamtausführungszeit von 25 Zeiteinheiten.
- Job J_1 führt P(S) nach 2 Zeiteinheiten Ausführungszeit und V(S) nach 5 Zeiteinheiten Ausführungszeit aus. (J_1 wird zunächst 2 Zeiteinheiten lang ausgeführt und betritt dann den kritischen Abschnitt, dessen Länge 3 Zeiteinheiten beträgt.) Seine Gesamtausführungszeit beträgt 6 Zeiteinheiten.

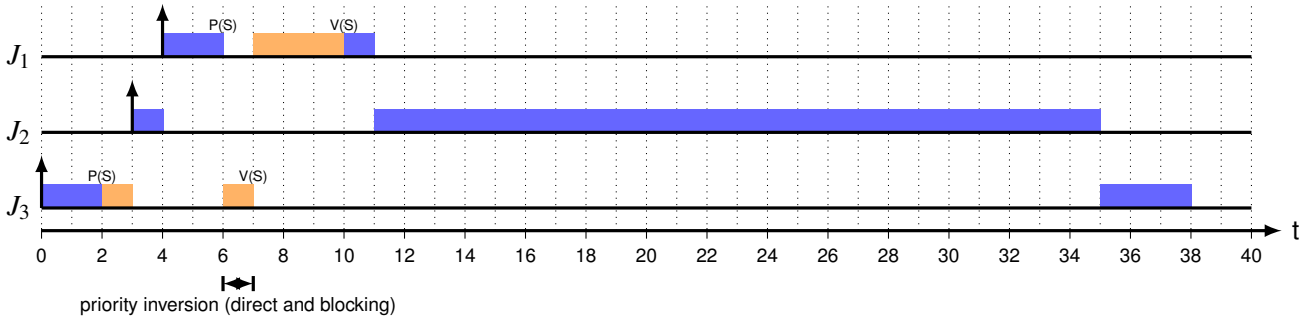
Wie sieht der Schedule des Systems aus, wenn wir Pre-emptive Priority-Based Scheduling verwenden? Berücksichtigen Sie die Aufrufe von P(S) und V(S). Markieren Sie die Zeitintervalle, in denen Priority Inversion stattfindet.

Lösung:



Wie sieht der Schedule des Systems aus, wenn wir das Pre-emptive Priority-Inheritance Protocol (PIP) verwenden? Berücksichtigen Sie die Aufrufe von P(S) und V(S). Markieren Sie die Zeitintervalle, in denen Priority Inversion stattfindet.

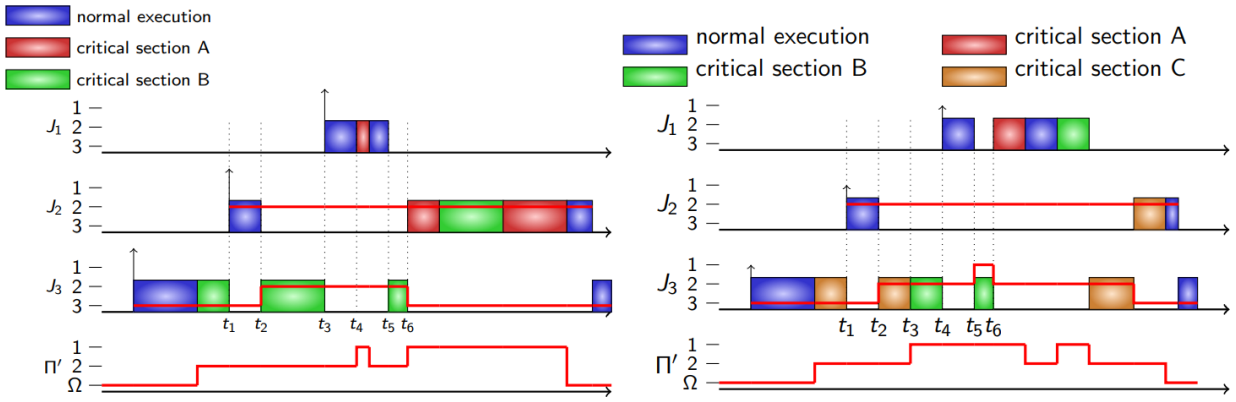
Lösung:



7 PCP-Beispiel

Zeichnen sie das Priority Ceiling $\Pi'(t)$ des Systems sowie die Prioritäten der Jobs für beide Beispiele von PCP, die in der Vorlesung vorgestellt wurden (Folien 19 und 20 in es-chen-4.4.pdf, x-Achse: Zeit, y-Achse: Priority Levels).

Lösung:



8 Distributed PCP (DPCP) (advanced**)

Betrachten Sie ein Multiprozessorsystem. Ein Task ist statisch einem Prozessor zugeordnet. Ein zusätzlicher Prozessor P_0 fungiert als *Synchronization Processor*. Das bedeutet, alle kritischen Abschnitte werden auf Prozessor P_0 ausgeführt, wobei PCP mit Rate-Monotonic Priority Assignment verwendet wird. Dieses Protokoll namens Distributed PCP (DPCP) wurde im Jahre 1990 von Raj Rajkumar entwickelt.

Im folgenden Beispiel betrachten wir drei Tasks, die alle jeweils einem Prozessor zugeordnet sind und deren kritische Abschnitte durch *eine binäre Semaphore* geschützt sind, sodass Multi-Tasking lediglich auf P_0 stattfindet. Task τ_1 befindet sich auf Prozessor P_1, Task τ_2 auf Prozessor P_2 und Task τ_3 auf Prozessor P_3. In der Tabelle 1 wird die Worst-Case Execution Time (inklusive der Länge der kritischen Abschnitte) eines Tasks τ_k mit C_k, die Periode mit T_k, die Anzahl der kritischen Abschnitte pro Job mit N_k und die Worst-Case-Länge eines kritischen Abschnitts (pro kritischem Abschnitt) mit L_k angegeben.

Um zu analysieren, ob ein Task τ_k seine Deadline einhalten kann, müssen wir seine *Remote Blocking Time* B_k auf P_0 analysieren. In dem obigen einfachen Beispiel (nur ein Task pro Prozessor, außer auf P_0) können wir leicht feststellen, ob $B_k + C_k \leq T_k$. Mr. Smart behauptet auf Basis des Critical Instant Theorems, dass die zusätzliche Verzögerung für Task τ_k auf P_0 aufgrund von PCP durch $L_k \cdot (\max_{j>k} L_j) + \sum_{i=1}^{k-1} \left\lceil \frac{T_k}{T_i} \right\rceil L_i N_i = t$ von oben beschränkt ist. Der erste Term $L_k \cdot (\max_{j>k} L_j)$ rührt daher, dass jeder Zugriffsversuch auf einen kritischen Abschnitt durch einen Task mit niedrigerer

Priorität blockiert werden kann. Der zweite Term $\sum_{i=1}^{k-1} \left\lceil \frac{T_k}{T_i} \right\rceil L_i N_i$ resultiert aus der Störung durch Tasks mit höherer Priorität unter Anwendung des Critical Instant Theorems. Daher schlussfolgert er, dass

- B_1 eine obere Schranke von 4 hat,
- B_2 eine obere Schranke von $1+2\cdot 2=5$ hat und
- B_3 eine obere Schranke von $0+2\cdot 2+4\cdot 2=12$ hat.

Da $C_k+B_k \leq T_k \forall k=1,2,3$, schlussfolgert er, dass dieses Task Set unter DPCP feasible ist.

Dennoch gibt es ein konkretes Gegenbeispiel, dargestellt in Abbildung 1, das zeigt, dass Task τ_3 seine Deadline verpasst. Was ist bei der obigen Analyse schief gelaufen?

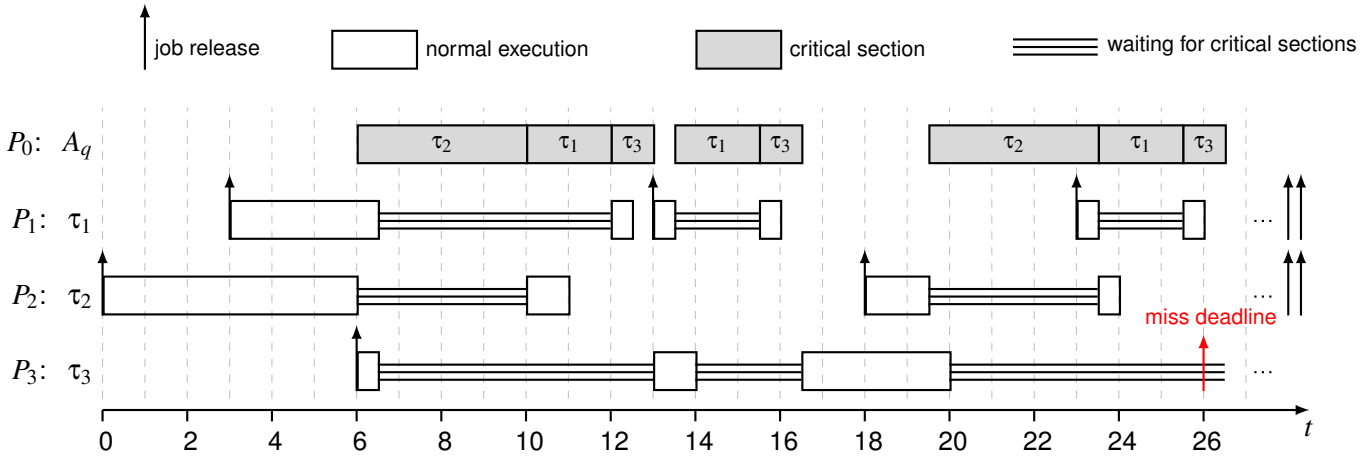


Abbildung 1: Ein Beispielschedule, in dem τ_3 eine Deadline verpasst.

τ_k	$P(\tau_k)$	C_k	$T_k\ (=D_k)$	N_k	L_k
τ_1	P_1	6	10	1	2
τ_2	P_2	11	18	1	4
τ_3	P_3	8	20	3	1

Tabelle 1: Taskparameter für das Gegenbeispiel.

Lösung: Sie können sich vorstellen, dass eine Suspension des Tasks τ_i von P_0 stattfindet, wenn der Task auf P_i ausgeführt wird. Daher basiert die Berechnung $\sum_{i=1}^{k-1} \left\lceil \frac{T_k}{T_i} \right\rceil L_i N_i$ ausschließlich auf dem Critical Instant Theorem, das im Falle einer Self-Suspension eines Tasks nicht gültig ist.