

Fachhochschule Würzburg-Schweinfurt

Wintersemester 2005/2006

Diplomfachprüfung im Fach

Prozessdatenverarbeitung II – Echtzeit-Programmierung

(Prof. Dr.-Ing. Ludwig Eckert)

Datum: 01.02.2006, 14.00 Uhr, Raum 5102

Dauer: 45 Minuten

Erreichte Punktzahl:

Note:

Hilfsmittel: Ohne schriftliche Unterlagen, Taschenrechner erlaubt

Vorname: Name:

Matrikelnr:

Erstprüfer:

Zweitprüfer:

Wichtige Hinweise:

Bitte tragen Sie alle Antworten auf den folgenden Seiten direkt im Anschluss an die Aufgabentexte ein.

Ergebnisse auf Zusatzblättern werden nur in begründeten Ausnahmefällen gewertet.

Aufgabe 1: Begriffe	2
Aufgabe 2: Einsatzgebiete	2
Aufgabe 3: Echtzeit-Betriebssysteme	3
Aufgabe 4: Taskzustände	4
Aufgabe 5: Projektierung	6
Aufgabe 5.1: Projektierung	6
Aufgabe 5.2: Projektierung	7
Aufgabe 5.3: Projektierung	7
Aufgabe 6: Tasksynchronisation	8
Aufgabe 7: Tasksynchronisation	9
Anhang: Befehle zur Tasksynchronisation	10

Aufgabe 1: Begriffe

Was versteht man unter den Begriffen „Harte Echtzeit“, „Weiche Echtzeit“ und „Nicht-Echtzeit“ hinsichtlich der Zeitanforderungen?

Aufgabe 2: Einsatzgebiete

Beschreiben Sie jeweils **ein** Anwendungsszenarium für den Einsatz von Embedded Systems mit „harten“ und „weichen“ Echtzeitanforderungen (Geben Sie jeweils eine Skizze an).

Anwendungsszenario mit „weichen“ Echtzeitanforderungen

Anwendungsszenario mit „harten“ Echtzeitanforderungen

Aufgabe 3: Echtzeit-Betriebssysteme

Zeigen Sie die besonderen Eigenschaften eines Echtzeit-Betriebssystems im Vergleich zu einem Standard-Betriebssystem auf?

Aufgabe 4: Taskzustände

Gegeben ist das nachfolgende Gerüst eines Zustandsmodells für Rechenprozesse in einem Echtzeit-Betriebssystem.

Bitte vervollständigen Sie die Zustände.

Zeichnen Sie die Übergänge durch Pfeile entsprechend ein und erläutern Sie jeweils eine mögliche Übergangsbedingung.

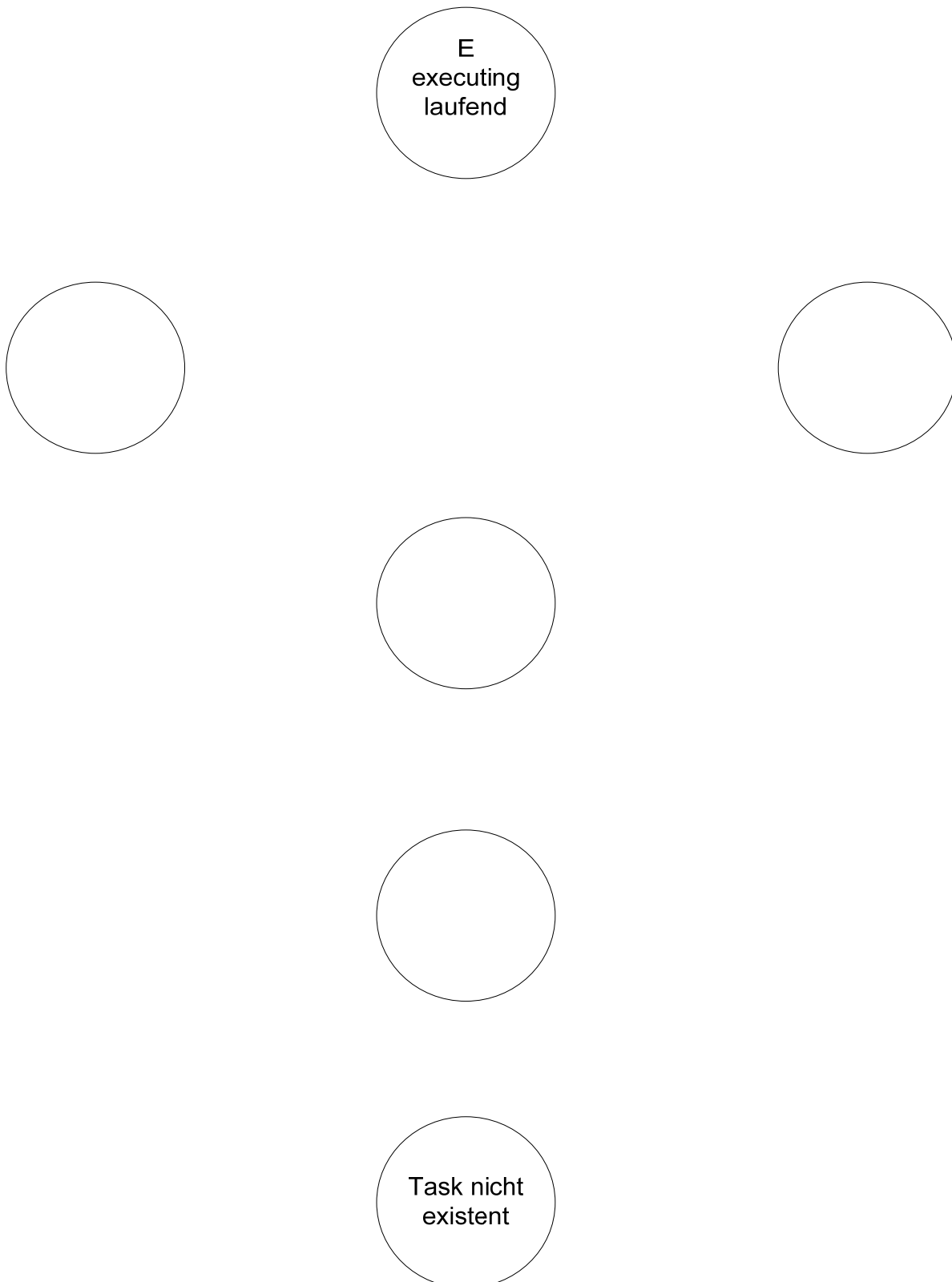


Bild: Zustandsmodell für Rechenprozesse

Aufgabe 5: Projektierung

Nachfolgend ist das zeitliche Sollverhalten von 4 Rechenprozessen (RP1 - 4) dargestellt, wobei der RP4 die niedrigste und RP1 die höchste Priorität aufweist.

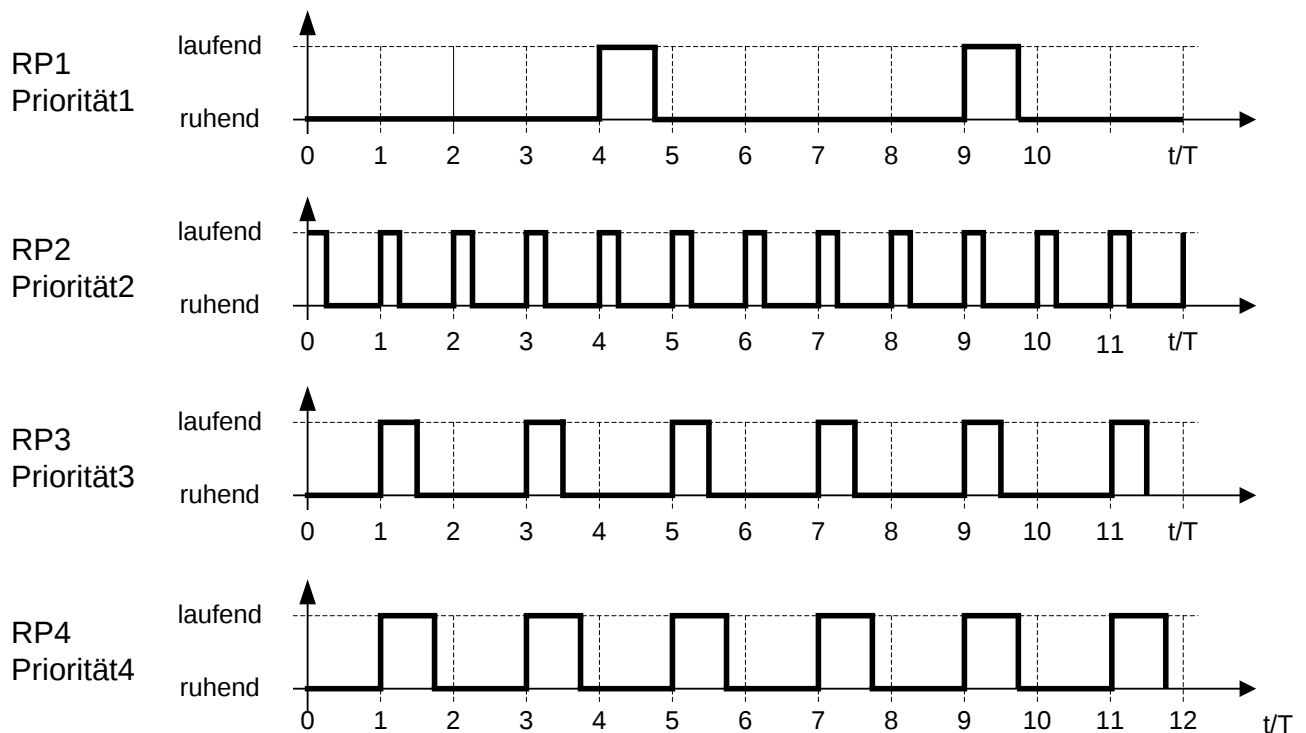


Bild 2: Zeitliches Sollverhalten der Rechenprozesse

Aufgabe 5.1: Projektierung

Tragen Sie in das nachstehende Bild die tatsächliche Abarbeitung der Rechenprozesse ein (bis zum normierten Zeitpunkt $t/T = 10$).

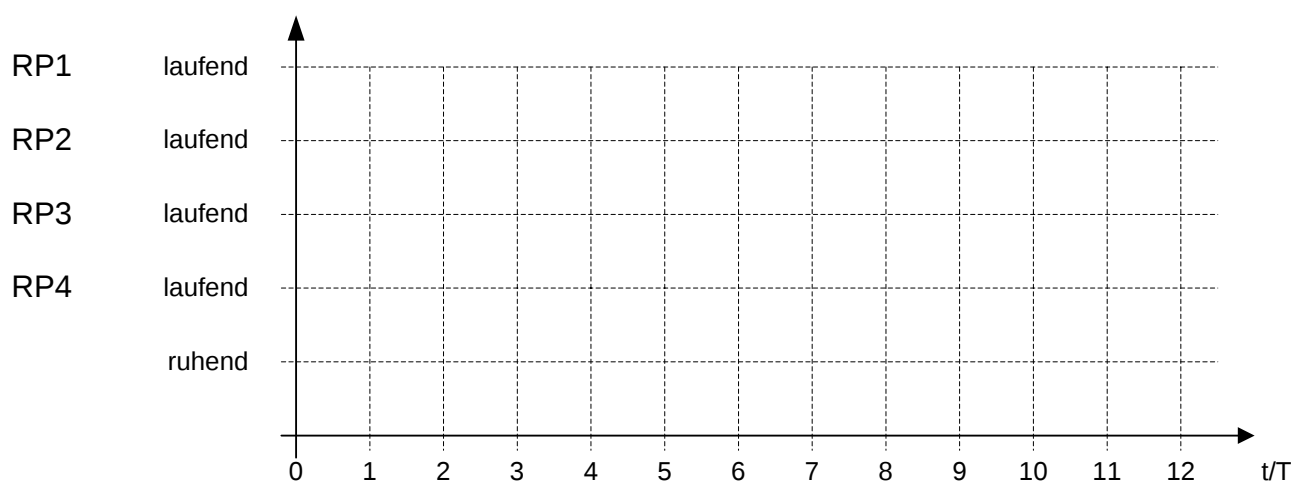


Bild 3: Tatsächlicher zeitliche Ablauf der Rechenprozesse

Aufgabe 5.2: Projektierung

Wie beurteilen Sie die Ausführung der einzelnen Rechenprozesse RP1 bis RP4 unter dem Gesichtspunkt der Rechtzeitigkeit?

Aufgabe 5.3: Projektierung

Tragen Sie den zeitlichen Verlauf der Taskzustände von Rechenprozess RP4 in das nachstehende Bild ein.

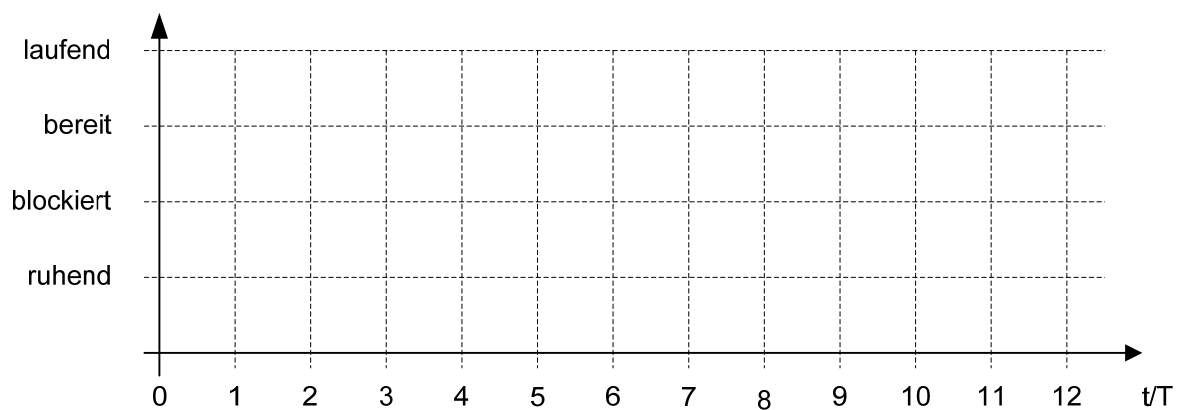


Bild 4: Taskzustände des Rechenprozesses RP4

Aufgabe 6: Tasksynchronisation

Gegeben seien zwei zyklische Tasks. Die eine Task soll Vorgänger (v) und die andere Nachfolger (n) genannt werden. Beide Tasks haben einen Programmabschnitt Kv und Kn. Die Synchronisationsaufgabe lautet:

1. Der Abschnitt Kn des Nachfolgers darf erst dann durchlaufen werden, wenn der Vorgänger den Abschnitt Kv **mindestens einmal** durchlaufen hat.
2. Die Aufgabe soll mit einem binären Semaphor gelöst werden. Das Semaphor soll global vereinbart werden.
3. Die Tasks sollen beliebig lange existieren.

Bitte vervollständigen Sie das Programm durch Eintragen geeigneter Synchronisierungsbefehle.

```
SEM_ID semID;    // Deklaration der Semaphore(n)
INT Tv, Tn;
```

```
void start()
{
    semID = semBCreate(SEM_Q_PRIORITY, SEM_EMPTY);    // Semaphore semID erzeugen

    Tv = taskSpawn ("Task_Tv", 20, 0, 1000, Vorgaenger, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0); // Tasks generieren
    Tn = taskSpawn ("Task_Tn", 20, 0, 1000, Nachfolger, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0);
}
```

```
STATUS Vorgaenger (void)
{
    for ( ;; )
    {

        .....

        // Programmabschnitt Kv

        .....
    }
}
```

```
STATUS Nachfolger (void)
{
    for ( ;; )
    {

        .....

        // Programmabschnitt Kn

        .....
    }
}
```

Aufgabe 7: Tasksynchronisation

Aufbauend auf dem vorherigen Programm sollen nun weitere Bedingungen berücksichtigt werden. Die zusätzlichen Synchronisationsaufgaben lauten:

4. Dem Durchlauf des Vorgängers muss **stets genau ein** Durchlauf des Nachfolgers folgen.
5. Der Nachfolger kann erst wieder ablaufen, wenn der Vorgänger durchlaufen wurde.
6. Der Vorgänger beginnt.

Bitte vervollständigen Sie das Programm durch Eintragen geeigneter Synchronisierungsbefehle.

```
SEM_ID se;    // Deklaration der Semaphore(n)
SEM_ID sv;
INT Tv, Tn;

void start()
{
    se = semBCreate (SEM_Q_PRIORITY, SEM_FULL);    // Semaphore se erzeugen
    sv = semBCreate (SEM_Q_PRIORITY, SEM_EMPTY);    // Semaphore sv erzeugen

    Tv = taskSpawn ("Task_Tv", 20, 0, 1000, Vorgaenger, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0); // Tasks generieren
    Tn = taskSpawn ("Task_Tn", 20, 0, 1000, Nachfolger, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0);
}

STATUS Vorgaenger (void)
{ for ( ; ; )
    {
        .....

        // Programmabschnitt Kv

        .....
    }
}

STATUS Nachfolger (void)
{ for ( ; ; )
    {
        .....

        // Programmabschnitt Kn

        .....
    }
}
```

Anhang: Befehle zur Tasksynchronisation

SEM_ID semBCreate(int *options*, SEM_B_STATE *initialState*)

// Rückgabewert: Semaphor ID oder NULL

// Options: SEM_Q_PRIORITY: Warteschlange nach Priorität und FIFO-Prinzip,
// SEM_Q_FIFO: Warteschlange nur nach FIFO-Prinzip.

// initialState: SEM_EMPTY oder SEM_FULL.

// Hinweis: Der Semaphor ist nach Aufruf arbeitsbereit.

STATUS semDelete(SEM_ID *semId*)

// Semaphoren werden durch semDelete() gelöscht.

// Rückgabewert: OK oder ERROR

// Hinweis: Der Semaphor wird gelöscht.

STATUS semFlush(SEM_ID *semId*)

// Rückgabewert: OK oder ERROR

// Hinweis: Alle Tasks, die auf das Semaphor warten, werden in den Ready-Zustand versetzt

STATUS semGive(SEM_ID *semId*)

// signalisiert die Semaphore *semId*

// Rückgabewert: OK oder ERROR, wenn Fehler bei der Signalisierung

STATUS semTake(SEM_ID *semId*, int *timeout*)

// nimmt die Semaphore *semId*

// Rückgabewert: OK oder ERROR, wenn Semaphore *semId* nicht vorhanden ist oder wenn
// eine Wartezeitüberschreitung aufgetreten ist.

// Timeout: NOWAIT, WAIT_FOREVER oder Anzahl Ticks warten

// Hinweis: Diese Funktion kann nicht in einer ISR aufgerufen werden.

Befehle zur Taskgenerierung

Id = taskSpawn (name, priority, options, stacksize, entrypoint, arg1, .. arg10);

// Task kreieren und aktivieren

name: Name der Task, bestehend aus ASCII Zeichen; "tMeine_Task"

priority: in VxWorks wählbar zwischen 0 (höchste) bis 255 (niedrigste) Priorität

options: Optionen z. B. für das "Task-Debugging"

stacksize: Größe des Stackspeichers für eine Task in Bytes

entrypoint: Hier wird der Name der C-Funktion eingetragen, die im Taskkontext ablaufen soll.

arguments: Diese Argumente sind die C-Funktionsübergabeparameter der entrypoint-Funktion.