BHT Berliner Hochschule für Technik	Matrikelnummer:	
	Punkte:	Note:
Informatik und Medien		
Kurs:	Vertiefung Echtzeitsysteme	**Semester:**	SoSe 2022
Studiengang:	MA Technische Informatik	**Dauer/Anteil:**	90 Minuten
Prüfer:	Heinz Junkes	**Prüfungsdatum**	29.09.2022
Hilfsmittel:	Ein handschriftlich beschriebenes DINA4-Blatt	**Uhrzeit**	14:15
Prüfungsart:	schr. P. 90 Min	**Anzahl d. Seiten**	12

Bevor Sie beginnen:

- Kontrollieren Sie ob Ihr Angabenblatt vollständig ist.
- Lesen Sie sich alle Aufgaben durch und verschaffen Sie sich einen Überblick über den Umfang der Aufgaben.
- Stellen Sie sicher, dass Ihre Antworten vollständig und lesbar sind.
- Schreiben Sie nicht außerhalb der Seitenmarkierungen.

Viel Erfolg!

Frage:	1	2	3	4	5
Punkte:	30	15	4	2	2
Erzielt:					

Frage:	6	7	8	9	10
Punkte:	2	4	4	2	3
Erzielt:					

Frage:	11	12	13	14	Gesamt
Punkte:	4	6	5	17	100
Erzielt:					

BHT Berliner Hochschule für Technik		Matrikelnummer:	
		Name:	
Kurs:	Vertiefung Echtzeitsysteme	**Semester:**	SoSe 2022
Prüfer:	Heinz Junkes		

Frage 1 (30 Punkte):

Gegeben sind drei Echtzeittasks, die nach zwei verschiedenen Schedulingverfahren (*Rate Monotonic Scheduling* bzw. *Earliest Deadline First Scheduling*) abgearbeitet werden sollen. Die Perioden und Ausführungszeiten der drei Tasks sind in der folgenden Tabelle angegeben.

	Periode T_i	Ausführungszeit C_i
Task 1:	4	1
Task 2:	16	1
Task 3:	8	5

(a) (10 Punkte) Führen Sie für das Task Set die notwendigen und hinreichenden *Utilization-Based Schedulability Tests* für (a) *Rate Monotonic Scheduling* und (b) *Earliest Deadline First Scheduling* durch.

Rate Monotonic Scheduling:

Earliest Deadline First Scheduling:

(b) (10 Punkte) Welche Aussagen können Sie aufgrund der Ergebnisse der Schedulability Tests über das gegebene Task Set machen?

Rate Monotonic Scheduling:

Earliest Deadline First Scheduling:

(c) (10 Punkte) Tragen Sie in einem Diagrammen ein, wie das gegebene Task Set nach den beiden Schedulingstrategien abgearbeitet wird. Nehmen Sie dabei an, dass die erste Aktivierung aller drei Tasks zum Zeitpunkt Null erfolgt. Tragen Sie in jedem Diagramm für jede Zeiteinheit den jeweils laufenden Task ein. Sollte es zur Verletzung einer Task-Deadline kommen, dann zeichnen Sie dies deutlich ein. Sie brauchen in diesem Fall das Schedule nicht fortzusetzen.

Rate Monotonic Scheduling:

Earliest Deadline First Scheduling:

Frage 2 (15 Punkte):

Mehrere Threads sollen Zugriff auf eine Thread-sichere Queue haben, in der *Items* eingereiht und entnommen werden können. Für die Implementierung steht eine bereits existierende Queue zur Verfügung, die jedoch nicht Thread-sicher ist. Die zu implementierenden Methoden sind *enqueue* zum Einreihen eines *Items* und *dequeue* zum Entnehmen eines *Items*. Diese Methoden sollen durch Semaphoren für nebenläufigen Zugriff geschützt werden. *dequeue* soll blockieren solange kein *Itemin* der Datenstruktur vorhanden ist. Nehmen Sie an, dass die Queue eine unbegrenzte Anzahl von *Items* aufnehmen kann. Die nicht Thread-sichere Queue hat folgende Methoden:

```
Queue q; /* Anlegen einer nicht Thread-sicheren Queue */
Item x;
q.enqueue( x ); /* mit enqueue wird x in der Queue q eingereiht
Item y <- q.dequeue(); /* mit dequeue wird ein Item entnommen
                                        und hier y zugewiesen */
```

Zusätzlich stehen Ihnen Semaphoren mit den folgenden Methoden zur Verfügung:

```
Semaphore s; /* Deklaration einer Semaphore */
s.init( z ); /*initialisiert die Semaphore s mit z */
s.up(); /* inkrementiert die Semaphore s */
s.down(); /* dekrementiert die Semaphore s und blockiert
                    bis s einen Wert groesser 0 hat */
```

Implementieren Sie die folgenden Methoden für die Klasse ThreadQueue in Pseudocode und nutzen sie die Methoden der angegebenen Queue und Semaphore. Es kann angenommen werden, dass der Konstruktor ThreadQueue vor einem Aufruf von *enqueue* oder *dequeue* einmal aufgerufen worden ist.

void enqeue(Item x)
Item deqeue( )

BHT Berliner Hochschule für Technik		Matrikelnummer:	
		Name:	
Kurs:	Vertiefung Echtzeitsysteme	**Semester:**	SoSe 2022
Prüfer:	Heinz Junkes		

Frage 3 (4 Punkte):
Welche zwei wesentlichen Eigenschaften muss ein Echtzeitsystem aufweisen, um die Bedingungen der Rechtzeitigkeit zu erfüllen?

Frage 4 (2 Punkte):
Was versteht man unter einem optimalen Scheduler?

Frage 5 (2 Punkte):
Wie ist die Prozessorauslastung definiert? Was bedeutet eine Prozessorauslastung größer 100%?

Frage 6 (2 Punkte):
Nennen Sie zwei Unterteilungskriterien für Scheduling-Verfahren.

Frage 7 (4 Punkte):
Wie kommt eine Prioritäteninversion zu Stande. Welche Auswirkungen hat sie auf den Prozess mit höheren Priorität. Nennen Sie ine Möglichkeit, um das Problem zu verhindern?

Frage 8 (4 Punkte):
Welche Funktionen können zu Verklemmungen (Deadlocks) führen?.

- ◯ Wechselseitiger Ausschluss (Mutual exclusion)
- ◯ Hold-and-wait-Bedingung
- ◯ Ununterbrechbarkeit (No preemption)
- ◯ Zyklische Wartebedingung (Circular wait)

Frage 9 (2 Punkte):
Wie funktioniert ”Polling” und warum wird es nur selten verwendet für Abfrage der Daten eines Sensors?

BHT Berliner Hochschule für Technik		Matrikelnummer:	
		Name:	
Kurs:	Vertiefung Echtzeitsysteme	**Semester:**	SoSe 2022
Prüfer:	Heinz Junkes		

Frage 10 (3 Punkte):

Die Implementierung von Echtzeitsystemen als asynchrone Systeme ermöglicht eine deutlich höhere Flexibilität. Welche Flexibilität steht im Vordergrund und auf welcher Basis wird diese Flexibiltät erreicht?

Frage 11 (4 Punkte):

Welche Funktion hinsichtlich Semaphoren dürfen innerhalb von Interrupt-Service-Routinen ausgeführt werden und welche nicht? Begründen Sie Ihre Antwort.

Frage 12 (6 Punkte):

Nennen Sie drei Techniken, die bei Standardcomputersystemen zur Performanzsteigerung eingesetzt werden, die jedoch bei harten Echtzeitsystemen problematisch sein können.

Frage 13 (5 Punkte):

Tasks können mittels Interprozesskommunikation Informationen austauschen. Diskutieren Sie den Informationsaustausch mittels *Shared Memory*, *Message Queues* oder *Linked Lists*.

Frage 14 (17 Punkte):

Der folgende Programmcode (aus capture testsuite) wird zur Ausführung (*void capture_test_1 ()*) gebracht:

```
#ifdef HAVE_CONFIG_H
#include "config.h"
#endif

#include "system.h"
#include <stdio.h>
#include <stdlib.h>
```

```
#include <rtems.h>

static volatile int capture_CT1a_deleted;
static volatile int capture_CT1b_deleted;
static volatile int capture_CT1c_deleted;

static void
capture_wait (uint32_t period)
{
  rtems_task_wake_after (RTEMS_MICROSECONDS_TO_TICKS (period * 1000));
}

static void
capture_CT1a (rtems_task_argument arg)
{
  rtems_id mutex = (rtems_id) arg;
  rtems_status_code sc;

  sc = rtems_semaphore_obtain (mutex, RTEMS_WAIT, 0);

  if (sc != RTEMS_SUCCESSFUL)
    printf ("error: CT1a: mutex obtain: %s\n", rtems_status_text (sc));

  capture_wait (2500);

  sc = rtems_semaphore_release (mutex);

  if (sc != RTEMS_SUCCESSFUL)
    printf ("error: CT1a: mutex release: %s\n", rtems_status_text (sc));

  capture_CT1a_deleted = 1;

  rtems_task_exit();
}

static void
capture_CT1b (rtems_task_argument arg)
{
  volatile int i;

  while (!capture_CT1c_deleted)
    i++;

  capture_CT1b_deleted = 1;

  rtems_task_exit();
}

static void
capture_CT1c (rtems_task_argument arg)
```

```
{
  rtems_id          mutex = (rtems_id) arg;
  rtems_status_code sc;

  sc = rtems_semaphore_obtain (mutex, RTEMS_WAIT, 0);

  if (sc != RTEMS_SUCCESSFUL)
    printf ("error: CT1c: mutex obtain: %s\n", rtems_status_text (sc));

  capture_wait (500);

  sc = rtems_semaphore_release (mutex);

  if (sc != RTEMS_SUCCESSFUL)
    printf ("error: CT1c: mutex release: %s\n", rtems_status_text (sc));

  capture_CT1c_deleted = 1;

  rtems_task_exit();
}

void capture_test_1 ()
{
  rtems_status_code sc;
  rtems_name        name;
  rtems_id          id[3];
  rtems_id          mutex;
  int               loops;

  capture_CT1a_deleted = 0;
  capture_CT1b_deleted = 0;
  capture_CT1c_deleted = 0;

  name = rtems_build_name('C', 'T', 'm', '1');

  sc = rtems_semaphore_create (name, 1,
                               RTEMS_PRIORITY | RTEMS_BINARY_SEMAPHORE |
                               RTEMS_INHERIT_PRIORITY,
                               0, &mutex);

  if (sc != RTEMS_SUCCESSFUL)
  {
    printf ("error: Test 1: cannot mutex: %s\n", rtems_status_text (sc));
    return;
  }

  name = rtems_build_name('C', 'T', '1', 'a');

  sc = rtems_task_create (name, 102, 2 * 1024,
                          RTEMS_NO_FLOATING_POINT | RTEMS_LOCAL,
```

```
                        RTEMS_PREEMPT | RTEMS_TIMESLICE | RTEMS_NO_ASR,
                        &id[0]);

  if (sc != RTEMS_SUCCESSFUL)
  {
    printf ("error: Test 1: cannot create CT1a: %s\n", rtems_status_text (sc));
    rtems_semaphore_delete (mutex);
    return;
  }

  sc = rtems_task_start (id[0], capture_CT1a, (rtems_task_argument) mutex);

  if (sc != RTEMS_SUCCESSFUL)
  {
    printf ("error: Test 1: cannot start CT1a: %s\n", rtems_status_text (sc));
    rtems_task_exit();
    rtems_semaphore_delete (mutex);
    return;
  }

  capture_wait (1000);

  name = rtems_build_name('C', 'T', '1', 'b');

  sc = rtems_task_create (name, 101, 2 * 1024,
                        RTEMS_NO_FLOATING_POINT | RTEMS_LOCAL,
                        RTEMS_PREEMPT | RTEMS_TIMESLICE | RTEMS_NO_ASR,
                        &id[1]);

  if (sc != RTEMS_SUCCESSFUL)
  {
    printf ("error: Test 1: cannot create CT1b: %s\n", rtems_status_text (sc));
    rtems_task_exit();
    rtems_semaphore_delete (mutex);
    return;
  }

  sc = rtems_task_start (id[1], capture_CT1b, 0);

  if (sc != RTEMS_SUCCESSFUL)
  {
    printf ("error: Test 1: cannot start CT1b: %s\n", rtems_status_text (sc));
    rtems_task_exit();
    rtems_task_exit();
    rtems_semaphore_delete (mutex);
    return;
  }

  capture_wait (1000);
```

```
    name = rtems_build_name('C', 'T', '1', 'c');

    sc = rtems_task_create (name, 100, 2 * 1024,
                            RTEMS_NO_FLOATING_POINT | RTEMS_LOCAL,
                            RTEMS_PREEMPT | RTEMS_TIMESLICE | RTEMS_NO_ASR,
                            &id[2]);

    if (sc != RTEMS_SUCCESSFUL)
    {
      printf ("error: Test 1: cannot create CT1c: %s\n", rtems_status_text (sc));
      rtems_task_exit();
      rtems_task_exit();
      rtems_semaphore_delete (mutex);
      return;
    }

    sc = rtems_task_start (id[2], capture_CT1c, (rtems_task_argument) mutex);

    if (sc != RTEMS_SUCCESSFUL)
    {
      printf ("error: Test 1: cannot start CT1c: %s\n", rtems_status_text (sc));
      rtems_task_exit();
      rtems_task_exit();
      rtems_task_exit();
      rtems_semaphore_delete (mutex);
      return;
    }

    loops = 15;

    while (!(capture_CT1a_deleted || capture_CT1b_deleted ||
             capture_CT1c_deleted) && loops)
    {
      loops--;
      capture_wait (1000);
    }

    if (!loops)
    {
      printf ("error: Test 1: test tasks did not delete\n");
      rtems_task_exit();
      rtems_task_exit();
      rtems_task_exit();
    }

    sc = rtems_semaphore_delete (mutex);
    if (sc != RTEMS_SUCCESSFUL)
      printf ("error: Test 1: deleting the mutex: %s\n", rtems_status_text (sc));

  }
```

BHT Berliner Hochschule für Technik		Matrikelnummer:	
		Name:	
Kurs:	Vertiefung Echtzeitsysteme	**Semester:**	SoSe 2022
Prüfer:	Heinz Junkes		

(a) (7 Punkte) In welcher Reihenfolge werden die Tasks (CT1[a-c]) hier abgearbeitet?

(b) (5 Punkte) Nun wird die *semaphore* folgendermassen erzeugt:

```
sc = rtems_semaphore_create (name, 1,
                    RTEMS_PRIORITY | RTEMS_BINARY_SEMAPHORE |
                    RTEMS_PRIORITY_CEILING,
                    0, &mutex);
```

In welcher Reihenfolge werden die Tasks (CT1[a-c]) jetzt abgearbeitet?

(c) (5 Punkte) Nun wird die *semaphore* folgendermassen erzeugt:

```
sc = rtems_semaphore_create (name, 1,
                    RTEMS_PRIORITY | RTEMS_BINARY_SEMAPHORE |
                    RTEMS_NO_INHERIT_PRIORITY | RTEMS_NO_PRIORITY_CEILING,
                     0, &mutex);
```

In welcher Reihenfolge werden die Tasks (CT1[a-c]) jetzt abgearbeitet?
