ingo.korb [☺] tu-dortmund.de
niklas.ueter [☺] tu-dortmund.de
benjamin.glaeser [☺] tu-dortmund.de

Übung zur Vorlesung
Eingebettete Systeme
Wintersemester 16/17

Übungsblatt 9

(10 Punkte)

Präsenzaufgaben zur Woche ab Montag, 9. Januar 2017

Für die Bearbeitung dieses Blatts wird die virtuelle Maschine **RTEMS-de** verwendet.

Hintergründe

Semaphoren sollten als Konzept ja bereits aus der Betriebssysteme-Vorlesung bekannt sein – im Kontext eines Echtzeitbetriebssystems verursachen sie jedoch zusätzliche Probleme: Auch wenn keine Deadlock ensteht, kann die Verwendung von Semaphoren zum Verpassen der Deadline eines Tasks führen. Dies ist in einem Echtzeitsystem natürlich nicht erwünscht, weshalb spezielle Protokolle eingesetzt werden, um dies zu vermeiden. Diese Übung demonstriert dieses Problem und eine richtige sowie eine falsche Lösung an einem synthetischen Beispiel[1]

Vorbereitung

Die Hardware wird wie bereits auf Blatt 8 beschrieben angeschlossen. Um Ihre Kopie von RTEMS auf den Stand von Blatt 9 zu bringen, rufen Sie bitte erneut den Befehl `rtems-setup` auf und führen Sie anschliessend im Verzeichnis `$HOME/rtems/build` den Befehl `make -j4 install` aus.

Das Programm für dieses Blatt befindet sich im Unterverzeichnis `rtems-gpio/testsuites/samples/blatt9` (im Folgenden kurz *blatt9-source*), das zugehörige Verzeichnis mit den compilierten Dateien ist `build/arm-rtems4.11/c/raspberrypi/testsuites/samples/blatt9` (kurz *blatt9-build*). Zum Übertragen des compilierten Programms verwenden Sie bitte den Befehl `raspbootcom /dev/ttyUSB0 blatt9.ralf` im *blatt9-build*-Verzeichnis. Wenn Sie lediglich Änderungen am Programm vorgenommen haben, reicht ein Aufruf von `make` in *blatt9-build*. Bei Änderungen an RTEMS selbst muss dagegen `make install` im Verzeichnis `$HOME/rtems/build` aufgerufen werden, dabei werden eventuelle Änderungen am blatt9-Programm ebenfalls compiliert.

9.1 Beobachten (2 Punkte)

Das heutige Vorgabeprogramm in *blatt9-src* verwendet vier Tasks, die diesesmal der Übersichtlichkeit halber in vier einzelne Quellcode-Dateien (`task1.c` bis `task4.c`) aufgeteilt sind und natürlich wieder eine `init.c` mit der Systeminitialisierung. Zwei der Tasks verwenden eine Semaphore, um den Zugriff auf eine geteilte Resource zu simulieren. Diese Resource wird durch die weisse LED *H* (rechts unten neben dem roten Taster) auf der Pibrella-Platine angedeutet, sie leuchtet während ein Task die Semaphore hält.

[1] eine Beschreibung eines Semaphoren-Problems in einem realen Echtzeitsystem können Sie z.B. unter `http://research.microsoft.com/en-us/um/people/mbj/Mars_Pathfinder/Authoritative_Account.html` finden.

Die Eckdaten der Tasks sind wie folgt:

Task	Periode	Ankunftszeitpunkt	Rechenzeit	Kritischer Abschnitt	
				Anfang	Dauer
τ_1	8	4	1	-	-
τ_2	16	2	4	1	1
τ_3	40	5	3	-	-
τ_4	50	0	12	1	10

Aufgabe: Führen Sie das Programm aus und zeichnen Sie das sich ergebende Scheduling-Diagramm.

9.2 Programm anpassen (3 Punkte)

Ändern Sie nun das *blatt9*-Programm so, dass Priority Inheritance für die Semaphore verwendet wird. Testen Sie das modifizierte Programm und zeichnen Sie das sich ergebende Scheduling-Diagramm.

9.3 Betriebssystem anpassen (5 Punkte)

Man könnte auf die Idee kommen, bei der Implementierung von Priority Inheritance in einem Betriebssystem eine vereinfachte Variante zu verwenden: Versucht ein Task, eine schon belegte Semaphore zu belegen, wird die Priorität des ersten Belegers auf die maximal mögliche Priorität angehoben, statt auf die des zweiten Belegers – im Kontext des Beispielprogramms würde damit Task 4 nicht die Priorität von Task 2 erhalten, sondern eine vordefinierte feste Priorität. Tatsächlich jedoch ist diese Vereinfachung eine schlechte Idee.

Aufgabe: Modifizieren Sie das RTEMS-Betriebssystem so, dass es bei Priority Inheritance die Taskpriorität nicht auf die des zweiten belegenden Tasks anhebt, sondern auf die maximal mögliche Priorität. Prüfen Sie anschliessend das durch diese Änderung verursachte Verhalten des Beispielprogramms und zeichnen Sie das zugehörige Scheduling-Diagramm. Wo tritt ein Problem auf? Wird es vom System erkannt?

Hinweise: Die Änderung der Task-Priorität bei Priority Inheritance findet sich im RTEMS-Quellcode in der Datei `rtems-gpio/cpukit/score/src/coremutexseize.c`. Als fixer maximaler Prioritätswert empfiehlt sich die Zahl “1”. Da Sie das Betriebssystem selbst verändern, müssen Sie nicht nur das Beispielprogramm, sondern das komplette RTEMS compilieren – was aber relativ schnell gehen sollte. Verwenden Sie dazu bitte wieder wie in der Vorbereitung den Befehl `make install` im Verzeichnis `$HOME/rtems/build`. Das Beispielprogramm wird dabei automatisch mitcompiliert.