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

Übung zur Vorlesung
Eingebettete Systeme
Wintersemester 16/17

Übungsblatt 10

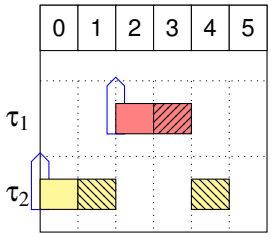
(10 Punkte)

Präsenzaufgaben zur Woche ab Montag, 16. Januar 2017

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

Hintergründe

Das auf dem letzten Blatt behandelte Priority Inversion Protocol ist keine Universallösung gegen Deadlocks – wenn mehrere Semaphoren verwendet werden, um unterschiedliche Resourcen zu schützen, kann es trotz PIP zu Deadlocks kommen. Ein Beispiel dafür ist folgende Situation (eine linksschraffierte und eine rechtsschraffierte Semaphore):



τ_1 reserviert Resource A und blockiert bei Zugriff auf Resource B

τ_2 reserviert Resource B und blockiert bei Zugriff auf Resource A

Es gibt zwei Tasks τ_1 und τ_2 sowie zwei Resourcen A und B. τ_1 belegt zu Zeitpunkt 3 zunächst die Resource A und versucht zu Zeitpunkt 4, die Resource B zu belegen. Dies blockiert jedoch, da τ_2 Resource B schon zu Zeitpunkt 1 belegt hat und zu Zeitpunkt 5 versucht, auch noch Resource A zu belegen. Nun wartet τ_2 darauf, dass τ_1 Resource A freigibt und τ_1 wartet darauf, dass τ_2 Resource B freigibt – ein Deadlock ist die Folge. In diesem Beispiel könnte man ihn vermeiden, indem man die Regel einführt, dass die Resourcen in allen Tasks in der gleichen Reihenfolge (z.B. erst A, dann B) belegt und in umgekehrter Reihenfolge freigegeben werden müssen. Eine solche Einschränkung ist jedoch nicht immer leicht durchsetzbar und erfordert sorgfältige Programmierung.

Als eine mögliche Lösung für dieses Problem wurde das Priority Ceiling Protocol (PCP) entwickelt. Ähnlich wie das Priority Inheritance Protocol verändert es die Priorität eines Tasks bei der Verwendung von Semaphoren, allerdings nach etwas anderen Regeln: Für jede Semaphore im System muss bekannt sein, welches der Task mit der höchsten Priorität ist, der diese Semaphore verwendet. Dieser Wert wird als *Priority Ceiling* bezeichnet und muss bei RTEMS manuell bestimmt werden. Wenn ein Task nun versucht, eine schon belegte Semaphore zu belegen, blockiert er wie üblich – aber es erfolgt im Gegensatz zu Priority Inheritance keine Prioritätsänderung. Stattdessen erhöht PCP die Priorität eines Tasks auf das Priority Ceiling der Semaphore, sobald er erfolgreich diese Semaphore belegt hat. Läuft der Task gerade mit einer höheren Priorität als das Priority Ceiling der Semaphore, so behält er seine aktuelle Priorität.

Im obigen Beispiel hätten beide Semaphoren als Priority Ceiling die Priorität von Task τ_1. Somit wird τ_2 zu Zeitpunkt 1 auf die Priorität von Task τ_1 angehoben und τ_1 kann ohne Unterbrechung seine Berechnungen vollenden.

Vorbereitung

Die Hardware wird wie bereits auf Blatt 8 beschrieben angeschlossen. Um Ihre Kopie von RTEMS auf den Stand von Blatt 10 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/blatt10` (im Folgenden kurz *blatt10-source*), das zugehörige Verzeichnis mit den compilierten Dateien ist `build/arm-rtems4.11/c/raspberrypi/testsuites/samples/blatt10` (kurz *blatt10-build*). Zum Übertragen des compilierten Programms verwenden Sie bitte den Befehl `raspbootcom /dev/ttyUSB0 blatt10.ralf` im *blatt10-build*-Verzeichnis. Wenn Sie lediglich Änderungen am Programm vorgenommen haben, reicht ein Aufruf von `make` in *blatt10-build*. Bei Änderungen an RTEMS selbst muss dagegen `make install` im Verzeichnis `$HOME/rtems/build` aufgerufen werden, und da irgendwo eine Abhängigkeit zu fehlen scheint empfiehlt es sich, zusätzlich eine beliebige Änderung in der `init.c` von blatt10 vorzunehmen und es wie oben beschrieben von Hand neuzucompilieren.

10.1 Deadlock mit Inheritance (4 Punkte)

Die Vorgabe verwendet drei Tasks mit folgenden Eckdaten:

Task	Periode	Ankunftszeitpunkt	Rechenzeit (gesamt)	Semaphore A		Semaphore B	
				Anfang	Dauer	Anfang	Dauer
τ_1	20	2	8	1	6	3	2
τ_2	30	5	3	-	-	-	-
τ_3	40	0	8	3	2	1	6

Stellen Sie sicher, dass das *blatt10*-Programm für Priority Inheritance konfiguriert ist, starten Sie es und untersuchen Sie sein Verhalten durch Konstruktion eines Scheduling-Diagramms.

10.2 Implementierung von Priority Ceiling (6 Punkte)

An sich unterstützt RTEMS von sich aus das Priority Ceiling-Protokoll. Jedoch ist unglücklicherweise in der Ihnen vorliegenden Version ein wenig Code verlorengegangen, so dass PCP nicht funktionsfähig ist.

Aufgabe: Vervollständigen Sie die Implementierung von PCP in RTEMS. Stellen Sie dann sicher, dass das *blatt10*-Programm für PCP konfiguriert ist, compilieren Sie RTEMS und *blatt10* und prüfen Sie, ob PCP das in der vorherigen Teilaufgabe festgestellte Problem löst. Wie verändert sich das Scheduling-Diagramm?

Hinweise:

- Es ist nur Code in zwei Dateien verlorengegangen:
  - `rtems-gpio/cpukit/score/src/coremutexsurrender.c`
  - `rtems-gpio/cpukit/score/include/rtems/score/coremuteximpl.h`
- Die Stellen in den Dateien sind mit “fulfil” markiert.
- `_Thread_Raise_priority(A, B);` hebt die Priorität von Objekt A auf den Wert B an
- `_Thread_queue_Release( queue, lock );` löst einen Lock von der Warteschlage eines Mutexes