Echtzeitbetriebssysteme
Frühlingssemester 2017
Roger Weber
Klassen E3a, E3b, E4p

Übung 3

Semaphoren, MUTEX

Aufgabe 3.1 Die speisenden Philosophen

In der Informatik wird oft das Problem der “speisenden Philosophen“ zitiert, welches ursprünglich vom niederländischen Informatiker Dijkstra gestellt wurde. In diesem Problem geht es um n Philosophen, welche entweder philosophieren, essen oder schlafen (programmiertechnisch in einer Endlosschleife). Bei der Problemstellung geht es weder um das Philosophieren, noch das Schlafen, sondern nur um das Essen. Jeder Philosoph hat am Tisch seinen eigenen Stuhl und seinen eigenen Teller. Zwischen zwei Tellern befindet sich genau eine Gabel (siehe Abbildung 1), somit hat es für n Philosophen auf dem Tisch n Teller und n Gabeln. Um zu essen (ihr Lieblingsmenü sind Spaghetti), brauchen die Philosophen jedoch zwei Gabeln, jeweils eine für die linke und für die rechte Hand. Es kann nun der Fall eintreffen, dass sich alle Philosophen gleichzeitig an den Tisch setzen, jeder die linke Gabel nimmt, und anschliessend darauf wartet, dass die rechte Gabel frei wird (Deadlock). Dijkstra hat gezeigt, wie das Problem gelöst werden kann: Der Zugang zum Tisch wird beschränkt.

Abbildung 1: Die speisenden Philosophen

Ausgangslage für diese Aufgabe:

- Die Zahl der Philosophen ist 5, es hat 5 Teller und 5 Gabeln.
- Eine Gabel entspricht einer Semaphore: Ein Philosoph, der eine Gabel-Semaphore erhält, hat die Gabel.
- Die Zustände der Philosophen (“Denken“, “Warten auf Gabel links“, “Warten auf Gabel rechts“, “Essen“ und allenfalls “Warten auf Zugang zum Tisch“ sollen programmiert und im Display des CARME-M4 Kits dargestellt werden.

- Die Anzahl Portionen Spaghetti, die ein Philosoph schon gegessen hat, soll ebenfalls auf dem Display dargestellt werden.
- Schreiben Sie Ihren Code so, dass Sie über einen Compiler-Switch ein- oder ausschalten können, ob der Zugang zum Tisch mit einer counting Semaphore geschützt wird oder nicht.
- Um die Gefahr eines Deadlocks zu vergrössern soll (ebenfalls über einen Compiler-Switch einschaltbar) ein Delay zwischen “linke Gabel nehmen“ und “rechte Gabel nehmen“ eingefügt werden können.
- FreeRTOS Kernel Awarness: Um beim Debuggen die Semaphoren in einem Debugger-View zu sehen, müssen Sie nach dem Erzeugen der Semaphore diese noch mit Hilfe der Funktion vQueueAddToRegistry() registrieren (siehe FreeRTOS.org).

Aufgabe 3.2 Ausgabe von Text auf die UART

Zwei Tasks sollen je einen Text auf die UART ausgeben:

- Text erster Task: “TASK #1 22222222222222222222222222222222222222222\r\n“
- Text zweiter Task: “TASK #2 THE QUICK BROWN FOX JUMPS OVER THE LAZY DOG!\r\n“

Nach der Ausgabe des Textes warten die Tasks 10 Ticks bevor sie die nächste Nachricht ausgeben.

Mit Hilfe eines Compiler-Switches soll die Verwendung einer MUTEX für den Zugang auf die serielle Schnittstelle aktiviert oder deaktiviert werden können. Gehen Sie folgenden Fragen nach:

- Wie sieht der übertragene Text mit und ohne Verwendung der MUTEX aus?
- Was geschieht wenn die Tasks unterschiedliche oder gleiche Prioritäten haben?

Initialisierung der seriellen Schnittstelle UART0:

```
USART_InitTypeDef USART_InitStruct;
USART_StructInit(&USART_InitStruct);
USART_InitStruct.USART_BaudRate = 115200;
CARME_UART_Init(CARME_UART0, &USART_InitStruct);
```

Senden eines Strings:

```
CARME_UART_SendString(CARME_UART0, "Hello World!\r\n");
```