Berner
Fachhochschule

Echtzeitbetriebssysteme
Frühlingssemester 2017
Roger Weber
Klassen E3a, E3b, E4p

Übung 4

Message Queues

Aufgabe 4.1 Der Spaghetti-Koch und die speisenden Philosophen

Die Aufgabenstellung aus Übung 3-1 mit den speisenden Philosophen soll ausgebaut werden. Damit die Philosophen nicht nur von Wasser und Brot leben müssen, haben sie einen Spaghetti-Koch angestellt. Dieser kann folgende Spaghetti zubereiten: Carbonara, Pesto, Pomodoro und Vesuvio. Der Spaghetti-Koch hat zehn Schüsseln, welche im Programmcode als Message Queue implementiert werden sollen. Jedes Mal wenn der Koch eine neue Portion gekocht hat, richtet er diese in einer Schüssel an (gibt sie in die Message Queue). Die Art der gekochten Spaghetti wird dabei von Mal zu Mal geändert. Die Philosophen nehmen wie bereits bekannt zuerst die linke und danach die rechte Gabel. Anschliessend nehmen sie eine Schüssel mit Spaghetti (d.h. sie nehmen diese aus der Message Queue). Auf diese müssen sie allenfalls warten, was auf dem Display dargestellt werden soll.

Aufgabenstellung:

- Ergänzen sie Übung 3-1 mit dem Spaghetti-Koch.
- Zeigen sie den Zustand des Kochs (Pause oder Typ der Spaghetti die er kocht) auf dem Display an.
- Zeigen Sie die Art der Spaghetti welche ein Philosophe isst an.
- Zeigen sie auch die Anzahl verfügbarer Spaghetti-Portionen im Display an.
- Der Koch darf zwischen dem Kochen von Spaghetti eine Pause machen. Die Länge der Pause soll abhängig davon sein, wie viele Portionen es in der Message Queue hat. Verwenden sie dazu die Funktion uxQueueMessagesWaiting().

Aufgabe 4.2 Log-Meldungen

Schreiben Sie einen UART-Task als Gatekeeper, welcher ankommende Log-Nachrichten von unterschiedlichen Tasks über die serielle Schnittstelle UART0 ausgibt. Folgende Tasks/Software Timer sind zu implementieren:

- SwitchTask: Implementieren Sie einen Switch Task, welcher alle 200ms aufgerufen wird, den Zustand der Schalter überprüft und mit Hilfe der LED’s darstellt. Ändert sich der Zustand eines Schalters, soll eine Log-Meldung abgesetzt werden.
- DummyTask: Implementieren Sie ein Dummy Task, welcher jede zweite Sekunde aufgerufen wird und bei jedem Aufruf eine Dummy Log-Nachricht absetzt.
- Button Software-Timer Callback: Erstellen Sie eine periodische Software-Timer Callback-Funktion, welche alle 250ms aufgerufen wird und den Zustand der Button überprüft. Die Callback-Funktion soll den Zustand der Taster verfolgen und eine Log-Nachricht ausgeben wenn eine Taste gedrückt bzw. wieder losgelassen wird. ACHTUNG: Innerhalb von Software-Timer Callback-Funktionen dürfen keine blockierende API Funktionen aufgerufen werden.
- UartTask: Der UartTask wird als Gatekeeper-Task implementiert und gibt ankommende Log-Nachrichten auf der seriellen Schnittstelle aus. Der Gatekeeper Task bietet eine Zugriffsfunktion um Log-Nachrichten abzusetzen.

Bsp. Struktur einer Log-Nachricht:

```
typedef struct _LogMsg {
  char         cTaskName[configMAX_TASK_NAME_LEN];   // Name des Task
  char         cMsg[64];                             // Nachricht
  portTickType xTimeTag;      // Zeitstempel des Events (xTaskGetTickCount)
}LogMsg;
```

Bsp. Zugriffsfunktion um Log-Nachrichten abzusetzen:

```
void logMsg(char * pcTaskName, char * pcMsg, portTickType xWaitTime) {
  LogMsg  sLogMsg;

  sLogMsg.xTimeTag = xTaskGetTickCount();        // aktueller Zeitstempel
  sprintf(sLogMsg.cTaskName, "%s", pcTaskName); // Task Name eintragen
  sprintf(sLogMsg.cMsg , "%s", pcMsg);           // Log-Nachricht eintragen

  // Nachricht ueber Message Queue an Gatekeeper Task senden
  ...
}
```

Bsp. Ausgabe für Log-Nachrichten auf der UART:

```
> 2000 tick
> DummyTask
> Hello World!
```

Beantworten Sie folgende Fragen:

- Nachrichten werden durch kopieren in die Queue abgelegt oder ausgelesen. Wie gross ist der Overhead bei diesen Log-Nachrichten? Wie könnte dies optimiert werden?
- Überlegen Sie sich wie Sie die Prioritäten an die verschiedenen Tasks verteilen. Welcher Task ist am wenigsten wichtig?