Real-time Concepts for Embedded Systems
WT 18/19

Assignment 5

Submission Deadline: Sunday, 20.01.2019, 23:55

- *Submit the solution in PDF via Ilias (only one solution per team).*
- *Respect the submission guidelines (see Ilias).*

---

1 Intertask Communication with Messages Queues [15 points]

In this question, intertask communication using message queues is addressed. We consider the scenario depicted in Figure 1.

Figure 1: Task communication via a message queue.

A producer task (c.f. P1 in Fig. 1) behave according to the following pseudo-code:

```
while(True) {
  m = generateMessage()  //produce message m of size 1
                         //if not stated differently,
                         //might contain blocking statements
  success = sendMessageToQueue(m) //send message m to the queue
  if (success == false) {
    continue //try with next message.
  }
}
```

Messages are retrieved by one or more consumers (e.g. C1) from the message queue. A consumer task behave according to the following pseudo-code:

```
while(True) {
  receiveMessageFromQueue(m) //receive message m of size 1 from the queue
  if (m!=NULL){ //required if the receive is non-blocking
    consumeMessage(m) //consumes or processes the received message
                      //if not stated differently,
                      //might contain blocking statements
  }
}
```

We assume, the task scheduler supports task priorities and will preempt any lower priority task as soon as a task with higher priority becomes ready for execution. All tasks are ready at the initialization of the system.

a) [3 points] In this question part, there is only one producer task and one consumer task that produce and consume messages as shown above. Both tasks have the same priority. Also assume for this part that the non-blocking mode is disabled.

Explain the relation between the total number of produced messages (n_{produced}) and the total number of consumed messages (n_{consumed}) at any point in time using inequalities or equality equations. Consider the dependency on the queue length (with l_{queue} the number of elements the queue can hold).

b) [3 points] In this question part, there is only one producer task and one consumer task. The producer task has higher priority than the consumer task. Also assume for this part that the non-blocking mode is disabled.

Explain, if the following statement is correct:

"The message queue will never be empty."

If necessary, give additional properties of the consumer task and/or the producer task under which this statement is true.

c) [3 points] In this question part, there is only one producer task and one consumer task. The consumer task has higher priority than the producer task. The consumer task and the producer task use blocking access to the message queue. The method `consumeMessage()` as well as `generateMessage()` do not contain any blocking statements. Explain, if any of the following statements apply:

- The queue will always contain messages.
- The queue will never contain more than one message.
- The queue will never contain any message.

If there is not only one, but multiple producer tasks, each with lower priorities than the consumer task and with non-blocking `generateMessage()`, With respect to the three statements, does this change your answer from the case with a single producer? Justify.

d) [3 points] In this question part, there is only one producer task and one consumer task. The consumer task has higher priority than the producer task. The consumer task and the producer task use non-blocking access to the message queue. The methods `consumeMessage()` as well as `generateMessage()` do not contain any blocking statements.

Explain, if any of the following statements apply:

- The queue will always contain messages.
- The queue will never contain more than one message.
- The queue will never contain any message.

e) [3 points] In this question part, there are n producer tasks $(P_1, P_2, ...P_n)$ and m consumer tasks $(C_1, C_2, ...C_m)$. The message queue can hold at most one message (size of message queue = 1) and a producer can overwrite an existing item in the message queue. All consumer tasks use blocking access to the message queue.

Considering $n + m$ priority levels $(1, 2, ...n + m)$, how do you have to assign priorities to the producer task and the consumer tasks to ensure that the following statements apply:

- consumers will consume all produced messages
- out of all consumer tasks which are ready to consume, always the one consumer task with the lowest index consumes the message. For example, if all consumer tasks are ready, C1 gets to consume the message, if C1 is not ready, C2 gets to consume the message, and so on.

Justify your priority assignment. What is the maximum number of producer tasks that can be accommodated with this priority assignment?

2 Intertask Communication with FreeRTOS [30 points]

In this question, intertask communication in FreeRTOS will be addressed. We consider the scenario depicted in Figure 2.

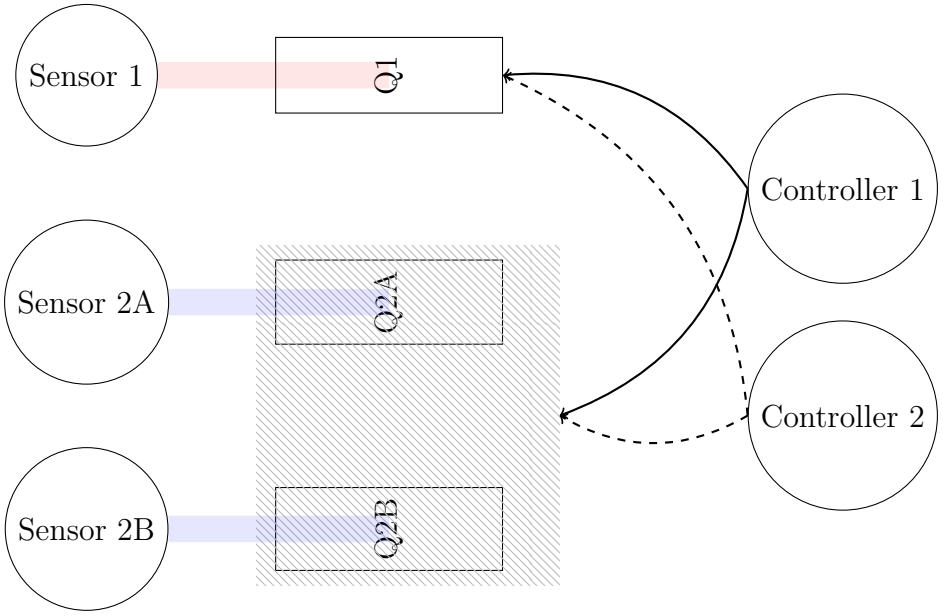

Figure 2: Task communication via message queues.

The scenario consists of the following components:

- There are three producer tasks (one for Sensor 1, one for Sensor 2A and one for Sensor 2B) which put sensor values into messages queues.
- There is one message queue per sensor task and each sensor task writes its sensor values only into its assigned queue.
- Also there are two consumer tasks which can take sensor values from messages queues (one for Controller 1 and one for Controller 2) and then proceed to do some computation with the sensor values.

The implementation shall guarantee the following properties regarding the behavior of the sensor tasks and the controller tasks:

§1 The sensor values shall be generated synthetically within the sensor tasks by cyclic counters. In each execution of a sensor task, the counter is increased by one, if the counter value is within the allowed range, else the counter value is reset (or wraps around) to the start of the range of allowed values. The sensor task then puts the current counter value into the message queue and sleeps for a certain time.

Sensor1 The sensor task for Sensor 1 generates one value in the range of 100 to 199 every 200 ms and puts every newly generated value into Queue 1 (Q1 in Figure 2). Its counter is initially set to 100, and increases every 200 ms by 1 until reaching 199. If the counter has the value 199 and the sensor task for Sensor 1 is executed again, the counter is reset to 100 (wrap-around)

Sensor2A The sensor task for Sensor 2 behaves similar to the sensor task of Sensor 1, but it generates one value in the range of 200 to 249 every 500 ms and puts every newly generated value into Queue 2A (Q2A in Figure 2). The counter of this sensor task initially has value 200.

Sensor2B The sensor task for Sensor 2 behaves similar to the sensor task of the other sensors, but it generates one value in the range of 250 to 299 every 1400 ms and puts every newly generated value into Queue 2B (Q2B in Figure 2). The counter of this sensor task initially has value 250.

Here, the sensor tasks generate values with different frequencies, e.g., Sensor 2A will have produced 2 values before Sensor 2B produces one new value.

§2 The computation of the control action requires a *set of sensor values* which the controller task has to retrieve. The *set of sensor values* has to contain *one* sensor value from Sensor 1, and one sensor value from **either** Sensor 2A **or** Sensor 2B. For example Sensor 2A and Sensor 2B could be different implementations of one type of sensor. In other words, for each computation, a controller task consumes two sensor values out of the three queues, one from Queue 1 (Q1) and one from Queue 2A (Q2A) *or* Queue 2B (Q2B).

Hint: The FreeRTOS-API contains methods that can be used to check if there is something in at least one queue of a set of queues.

§3 A controller task shall be executed as soon as possible, i.e. as soon as there is a set of sensor values ready in the message queues. *Hint: Consider the implications of task priorities.* You can assume, that the execution time of controller task job is less than the waiting time between the generation of sensor values for the sensor task with the highest value generation frequency.

For the practical implementation of this scenario with FreeRTOS, the actual computation on the set of sensor values is nothing but a print-statement which prints information identifying the controller task, the values from the set of sensor values and the current tick count to the console as shown in Figure 3.

```
Controller 1 has received sensor data at 1; Sensor1: 101; Sensor2: 201
Controller 1 has received sensor data at 201; Sensor1: 102; Sensor2: 251
Controller 1 has received sensor data at 501; Sensor1: 103; Sensor2: 202
Controller 1 has received sensor data at 1001; Sensor1: 105; Sensor2: 203
Controller 1 has received sensor data at 1401; Sensor1: 107; Sensor2: 252
Controller 1 has received sensor data at 1501; Sensor1: 108; Sensor2: 204
Controller 1 has had an error at 2001.
Controller 2 has received sensor data at 2002; Sensor1: 110; Sensor2: 205
Controller 2 has received sensor data at 2501; Sensor1: 113; Sensor2: 206
...
```

Figure 3: Sample Output.

§4 For this question, we are always interested in the latest available sensor value for each sensor. This means, the intertask communication shall be implemented such that a controller task will always retrieve the latest available sensor value at the time of the retrieval from the respective sensor queue. *Hint: Which implications does this*

have on the queue length, if we are only interested in the latest sensor values? Also a controller has to avoid retrieving outdated sensor values. This is because, we know that Sensor 1 can produce two values before Sensor 2A generates one value. In such case, the controller has to consider the latest value generated by Sensor 1 together with the value generated from Sensor 2A. In other words, a controller is not allowed to retrieve one sensor value from Sensor 1, store it temporarily while waiting for the value from Sensor 2A or Sensor 2B.

Hint: FreeRTOS provides mechanisms to "signal" with bit flags to one or more tasks that certain events have occurred. It is possible to use these mechanisms to block tasks, e.g., the controller tasks, until a certain signal/bit flag is observed/set, e.g. indicating that there is a set of sensor values (as specified before) in the message queues.

§5 The two controller tasks are used for fault-tolerance purposes as follows: As long as Controller 1 is operational, Controller 1 shall do all the computations (which are mere print-statements as specified in §3 for this question). While Controller 1 is operational, Controller 2 is a "hot reserve" which shall not consume any values from the message queues and is idle. As soon as Controller 1 fails, Controller 2 shall be able to take over, and from this point on Controller 2 consumes the values from the message queues (and prints to the console). Furthermore, in case of failure, at most one set of sensor values in the messages queues is allowed to not be used by the controller task. *Hint: Consider the implications of task priorities.*

For the practical implementation of this scenario using FreeRTOS, Controller 1 and Controller 2 are two instances of the *same* task function which decide according to some information passed during their creation whether they will act as Controller 1 (primary) or Controller 2 (reserve). Emulate a controller failure by deleting the controller task for Controller 1 after 2000 ms as shown in Figure 3.

Since we assume that a controller task may fail at any time, your implementation shall not use any semaphores or mutexes.

Implement the scenario using the FreeRTOS project from the provided zip-file, i.e. implement the sensor task functions, one controller task function and add necessary data structures and definitions in the file `main_exercise.c`. As always, use the `main_exercise()`-method as an entrypoint for the initialization.

Important: Since this question specifies the implementation on a functional level, points will **only** be awarded for code accompanied by comments which clearly state the intention of instructions and justification for the used data structure / FreeRTOS mechanisms (especially regarding properties §1 to §5).