OSES LAB #4 – resources and task organization

Please read the assignment description carefully

Purpose of the lab

In this lab you will implement resource-based synchronization and contemplate the trade-offs of splitting activities into multiple tasks.

Deliverables and deadlines

You shall provide a report (pdf file only) where you describe the oil file and the code you implemented for the exercise we proposed. The output produced by your solutions shall also be included. Insert in the report at least one SimulIDE screenshot and oscilloscope trace that clearly shows the input/output signals and data discussed in the following.

The report shall be uploaded to the portale della didattica using the Elaborati section.
The report shall be provided by December 5, 2025, 18:00.

Exercise #1 (Arduino Uno on SimulIDE)

A periodic task S samples analog input voltage A0 every 100ms. On every activation, S inserts the sampled value X into a queue Q, which can hold at most K elements. S must check for queue overflows and report them by writing a message on the serial console. Moreover, S must set the variable *int error* to *1* if X<10 or X>1013, and to *0* otherwise.

A periodic task B runs every 500ms. On every activation, B offloads all samples currently in Q and calculates their minimum N and maximum M. B must set the variable *int alarm* to *1* if M-N > 500, and to *0* otherwise.

A periodic task V runs every 125ms. It must control the LED connected to GPIO pin 13 according to the value of *error* and *alarm*, as follows:

- If *error* is *1*, the LED must blink “fast” (4Hz).
- If *error* is 0 and *alarm* is 1, the LED must blink “slowly” (1Hz).
- If both *error* and *alarm* are *0*, the LED must be off.

Questions:

- Determine the *minimum* value of K that allows S to run without ever overflowing Q and use this value in your code.
- Implement S, B, V, and Q using *shared memory and resource-based mutual exclusion*. Use the *minimum* number of resources you need to ensure correctness. For the sake of code portability, *do not make any assumptions* about memory access atomicity and memory access ordering.
- Is this software able to detect significant (as implied by M-N > 500) changes in A0 *wherever* they occur? If you think this is not the case, provide an example of input in which the change goes undetected.

Main design points and hints:

- Assign appropriate priorities to tasks S, B, and V, justifying your choices.
- Do not use message-based communication.
- Do not use any external library to implement Q.
- Use only OSEK-provided timing facilities (no delay(), millis(), …).
- Design a set of tests that exercises all possible system behaviors. Perform those tests and include the results in the report.
- Make sure you initialize the analog I/O line and the ADC properly before use.

Exercise #2 (Arduino Uno on SimulIDE)

Start from the code developed for Exercise #1 and merge tasks S and B into one single task W, so that the system now consists of tasks W and V.

Questions:

- What must W's period and priority be?
- What is the effect of merging S and B on the minimum number of resources you need?
- Implement the new code, still using shared memory and resource-based mutual exclusion. Perform the same tests you designed for Exercise #1 on it and make sure its behavior is still the same as before. Include test results in the report.
- Would it be possible to merge tasks W and V into a single task Z, thus doing everything in a single task? If so:
    - What would its period be?
    - How many resources would you still need?
    - What would the effect on code organization and CPU utilization be?