Exercise session 4: software architectures (part 2)

Thomas Braipson

23 October 2026

The Round-Robin with interrupts, and waiting-queue architectures

A few tips:

- Identify which tasks can/must be implemented inside interrupt routines (same criteria as previously)
- Identify communication objects between tasks
- Pay attention to non-atomic operations that may lead to data corruption when manipulating communication objects between main code and interrupts

Exercise 1: Implement a solution to the following problem

2. A marine research buoy contains sensors for measuring water temperature and salinity. It is equipped with an embedded system whose processor performs the following tasks:

   - Starting a new round of measurements every hour. This operation takes negligible time. The measurements are carried out by a separate data acquisition peripheral, and take a few seconds. An interrupt request is send by this peripheral when the results of the measurements are available.
   - Processing the measurements results, which is done each time that they are obtained, and takes up to 500 ms of CPU time. The results are written into flash memory. A write operation takes up to 10 ms, and is performed by a peripheral. This peripheral sends an interrupt request when the write is complete.
   - Sending by radio the last processed measurements. This operation should be performed once every 24 hours, and takes up to 30 s.

   Since the amount of onboard power is limited, it is important to put the processor in a low-power sleep mode whenever possible.

   (a) What is the best software architecture for this system? Justify carefully your answer. [3/20]

Exercise 2: Implement a solution to the following problem

2. A monitoring system for an aircraft engine controller is built around a microcontroller that receives data from a large number of sensors, processes this data, and sends periodic reports as well as alarm signals through a communication channel.

   The sensors operate asynchronously, meaning that they generate data at unpredictable times, independently from each other. When a sensor has new data available, it sends an interrupt request to the microcontroller, which has to fetch this data as soon as possible. This operation can be assumed to take negligible time. Then, the microcontroller processes the sensor data, which takes between 1 and 20 μs. It is known that in the long run, the microcontroller has enough computing power to process all incoming data and perform all communication duties. However, it may happen that rapid bursts of sensor data force the data processing to be sometimes delayed for a short while. The result of data processing can take two forms: nominally, it consists in updating statistics that are (globally for all sensors) sent every 10 ms through the communication channel. Besides that, if the processing detects a problematic situation, then an alarm message must immediately be sent through the communication channel. Transmitting on the communication channel simply amounts to writing appropriate data to the registers of a dedicated device, which takes at most 1 μs.

   Finally, a key feature of the system is that the sensors are classified in decreasing order of importance as being *life-threatening*, *critical*, *essential*, *auxiliary*, *indicative*, ..., the idea being that data issued by more important sensors must as much as possible be processed before that of less important ones.

   (a) What is the best software architecture for this system? Justify carefully your [2/20]