universidade de aveiro theoria poiesis praxis

SOTR 21/22 – Final project

Task Management framework for FreeRTOS

Introduction

FreeRTOS, as other (RT)OSs, provides fixed-priority scheduling. However, some real-time applications have tightly coupled tasks that require specific execution patterns. Such execution patterns include relative phases (e.g. to reduce jitter or guarantee that certain tasks do not compete for a given resource) and precedence constraints (to guarantee that a given task only executes after a set of predecessor tasks completes its execution).

Although FreeRTOS provides the means to implement such execution patterns (e.g. via periodic execution with a user-defined base time, primitives for suspending/resuming tasks, semaphores, queues, etc.), implementing them in an ad-hoc fashion is cumbersome and results in hard to write, hard to read and hard to maintain code.

One alternative is supplementing the basic functionality of FreeRTOS with a framework that allows to have a coordinated activation of tasks, permitting the explicit specification of periods and relative phases, via a suitable API.

This is the challenge proposed in this project.

Specification of the work

The objective of the work is developing a framework (Task Manager – TMan) that allows registering a set of FreeRTOS tasks, associate each task with a set of attributes (e.g. period, deadline, phase, precedence constraints) and activate those tasks at the appropriate instants.
The following methods should be provided:

- TMAN_Init: initialization of the framework
- TMAN_Close: terminate the framework
- TMAN_TaskAdd: Add a task to the framework
- TMAN_TaskRegisterAttributes: Register attributes (e.g. period, phase, deadline, precedence constraints) for a task already added to the framework
- TMAN_TaskWaitPeriod: Called by a task to signal the termination of an instance and wait for the next activation

- TMAN_TaskStats: returns statistical information about a task. Provided information must include at least the number of activations, but additional info (e.g. number of deadline misses) will be valued.

Other methods may be added, if found necessary.

Implementation indications

The implementation of TMAN should be carried out incrementally, as follows:

- Step 1: Basic support to periodic task activation.
  - The body of a task no longer includes FreeRTOS functions such as vTaskDelay/vTaskDelayUntil, but instead a call to TMAN_TaskWaitPeriod.
  - Task periodicity is specified via TMAN_TaskRegisterAttributes.
  - The periodicity is expressed as an integer number of TMAN ticks.
  - TMAN tick period is a configuration parameter that shall be set to an integer multiple of the FreeRTOS ticks (defaults to 1 ms in PIC32)

- Step 2: Add the possibility of assigning a relative phase to a task.
  - With respect to Step 1, the API function TMAN_TaskRegisterAttributes is extended to include the phase parameter.
  - The phase is specified as an integer number of TMAN ticks.

- Step 3: Add the possibility of assigning a relative deadline to a task.
  - With respect to Step 2, the API function TMAN_TaskRegisterAttributes is extended to include the deadline parameter.
  - The deadline is specified as an integer number of TMAN ticks.

- Step 4: Add the possibility of defining precedence constraints
  - With respect to Step 3, API function TMAN_TaskRegisterAttributes is extended to include the definition of precedence constraints.
  - It is only required to support simple precedence. E.g. Task B depends on Task A.

- Step 5: Additional functionalities will be valued and are a requirement for attaining a grade above 16 values.
  - Examples:
    - Compute the feasibility of the task set given the task’s WCET and deadline;
    - Allowing multiple dependencies (e.g. Task C depends both on Task A and B);
    - Allow the dynamic modification of periods, phases or precedence constraints;
    - Detect deadline misses;

Deliverables

- Code (full project, referred to folder "Demo" of a clean FreeRTOS installation) of the more advanced step reached.
  - The code should include a set of 6 tasks (named "A" through "F") that demonstrate the functionalities implemented.
    - When executed, each task should print to the terminal (i.e., via UART) its name followed by the execution instant (obtained using FreeRTOS function xTaskGetTickCount()), in a new line. E.g. :
      ...
      A, 21
      B, 22
      A, 23
      C, 24
      …
    - Each task should include a busy-wait section based on "for" loops that consumes around 0.5 ms when executed in isolation. The base structure (pseudocode) of the expected task body should be :

```
for(;;){
    TMAN_TaskWaitPeriod(args ?);  // Add args if needed
    GET_TICKS
    print "Task Name" and "Ticks"
    for(i=0; i<IMAXCOUNT; i++)
        for(j=0; j<JMAXCOUNT; j++)
            do_some_computation_to_consume_time;
    OTHER_STUFF (if needed)
}
```

- Report
  - "pdf" format, four pages, single column, 11pt font
  - Name and #mec of group members at the first line
  - Contents:
    - Indication of the functionalities implemented (mandatory and optional)
    - Description of the data structures used
    - Description of the implemented API, including method arguments
    - Description of the tests carried out and results obtained

Bibliography:

The framework herein described is a (simplified) version of the PMAN framework developed for Linux. The following book chapter may be helpful:

Paulo Pedreiras and Luis Almeida (2007). Task Management for Soft Real-Time Applications Based on General Purpose Operating Systems, Robotic Soccer, Pedro Lima (Ed.), ISBN: 978-3-902613-21-9, InTech.

Available from:
https://www.intechopen.com/chapters/530