13 STR * Lab project 3: ERIKA RT kernel - Introduction

The goal of this lab exercise is to get familiar with the Erika Enterprise real-time kernel (http://erika.tuxfamily.org/drupal/). Erika Enterprise is a real-time operating system that proposes a free of charge open-source implementation of the OSEK/VDX API (for more details, see the **14TR1rtos.pdf** report that gives an overview of real-time operating systems, inclcding Erika and standards), available under the General Public License (GNU GPL with linking excpetion). Erika Enterprise is supported by RT-Druid, a set of Eclipse plug-ins implementing an OSEK OIL (OSEK Implementation Language) compiler. The kernel configuration is generated by an OIL specification. The kernel has an API that provides support for tasks, events, alarms, resources, application modes, semaphores, and error handling. ERIKA Enterprise gives support for preemptive and non-preemptive multitasking, and implements several scheduling algorithms (in addition to the standard OSEK/VDX conformance classes, Erika provides other customized conformance classes implementing EDF). It also supports stack sharing, allowing all basic tasks in the system to share a single stack, so reducing the overall RAM memory used for this purpose. It also supports a number of third-party libraries, including TCP/IP, 802.15.4, CMOS cameras, analog and digital sensors, encoders, DACs, ADCs, 3-axis accelerometers, DC motors, and many others. Erika provides full support to the Flex board in terms of drivers, libraries, programming facilities, and sample applications.

Scheduling policies in Erika

Erika is a real-time kernel OSEK compliant. Therefore, the scheduling policies implemented in ERIKA follow the OSEK standards. In addition, the use of the single stack approach shared among all tasks mandates the implementation of appropriated resource sharing protocols. This is achieved by the Immediate Priority Ceiling for Fixed Priority (FP)-based systems and Stack Resource Policy for EDF-based systems.

In practice, the implementation of RM is done by specifying the FP scheduling policy and setting PRIORITY values for each task proportional to the task frequency. The EDF implementation, partially reported in "Efficient implementation of an EDF scheduler for small embedded systems" (G Buttazzo, P Gai - Proc. OSPERT, 2006) is done by specifying the EDF scheduling policy and setting PRIORITY values for each task also proportional to the task frequency. For EDF, the PRIORITY assignment is related to stack resource policy used for allowing preemption for EDF tasks sharing the same stack.

Work to be done

1. Getting started: led blinking

1. Execute the "Orange Led Blinking" example program for Erika. To do so, open RT-Druid (Eclipse with Erika plugin) and create a new project

   - File -> New -> Project...

   and select the RTDruid one as

   - Evidence -> RTDruid Oil and C/C++ Project

   Then we must assign a name for this project, e.g. *flex_blink*. The other options are left with the default settings. After clicking next, we are suggested to create a project using a template. Check the box *Create a project using one of these templates*, select *pic30* (it corresponds to our dsPIC33) and select as follows:

   - pic30 -> FLEX -> EDF: Periodic task with period of 0.5s

   As stated in the project description, the example shows a simple periodic task with a frequency of 0.5 s on the FLEX board, including one task, one periodic alarm controlled by timer T1, support for the Flex board and EDF scheduling.

   After clicking *Finish*, the project is created. It has two main files

   - *conf.oil*: it configures the system
   - *code.c*: it contains the main code

   and the required libraries.

   Every time we want to compile the project, it must be firstly **SAVED**, even when no modifications have been done.

   After compiling the project

   - Project -> Build Project

   we should obtain a *Compilation terminated successfully!*. After achieving a successful compilation, a new directory is created, named *Debug*, that containts the *pic30.elf* file, which is the binary to be downloaded to the Flex board. To do so, we open MPLABX, and we create a new project:

   - File -> New Project...

   Then MPLABX let us to choose the type of project we want to create. In our case we will use precompiled file. Therefore, we chose

   - Categories : Microchip Embedded
   - Projects: Prebuilt (Hex, Loadable image) Project

   Afterwards, we need to select our precompiled file. The path probably is

- *home/student/workspace/flex_blink/Debug/pic*30.*elf*

  Then the procedure is as usual. We select the microcontroller

  - Family: DSPIC33
  - Device: dsPIC33FJ256MC710

  and the ICD3 programmer. Finally we have to select a name for the project, and *it is very important* to change the project location folder. We have to set a folder different than the *Debug* one because when we modify the source code within RT-Druid and compile, the *Debug* folder is deleted and created again, and that may cause problems to the MPSLABX project.

  Finally MPLABX will show up all details for the new project. After accepting, to program the FLEX board, we must click the *Make and Program Device* button. Then, the orange led should blink.

2. Identify the main parts of the program just dowloaded. Note that the *Erika Enterprise Reference Manual v. 1.4.5.* is available at

   http://erika.tuxfamily.org/download/manuals/pdf/ee_refman_1_4_5.pdf

   and the *Erika Enterprise Minimal API Reference Manual v. 1.1.3* at

   http://erika.tuxfamily.org/download/manuals/pdf/ee_minimal$_r$*efman*_1_1_3.*pdf*

   (All information can be found at erika.tuxfamily.org/drupal/.)

3. Modify the program to have the led blinking every 1s. To do so, do it in the *code.c* file rather than in the *conf.oil*

2. Basic periodic scheduling using Erika

Let's consider the following task set (with $D_i = T_i$)

Task	T_i	C_i	(P_i)
τ_1	450 ms	150 ms	2
τ_2	600 ms	200 ms	1
τ_3	1000 ms	150 ms	3
τ_4	1800 ms	100 ms	4

The resulting schedules shown in Figure 1 have been obtained using the TrueTime simulator. The work to be done is

1. Program the Erika kernel to execute the previous task set under the same three policies. To do so, use fake but accurate computation times for each task.

2. Use the monitor developed in previous labs to show the resulting schedules. Modify the monitor to show tasks active times (but not running) using another color. Note that these times will include tasks preemption times. This would be similar to the TrueTime visualization, when the task is in the "midle level".

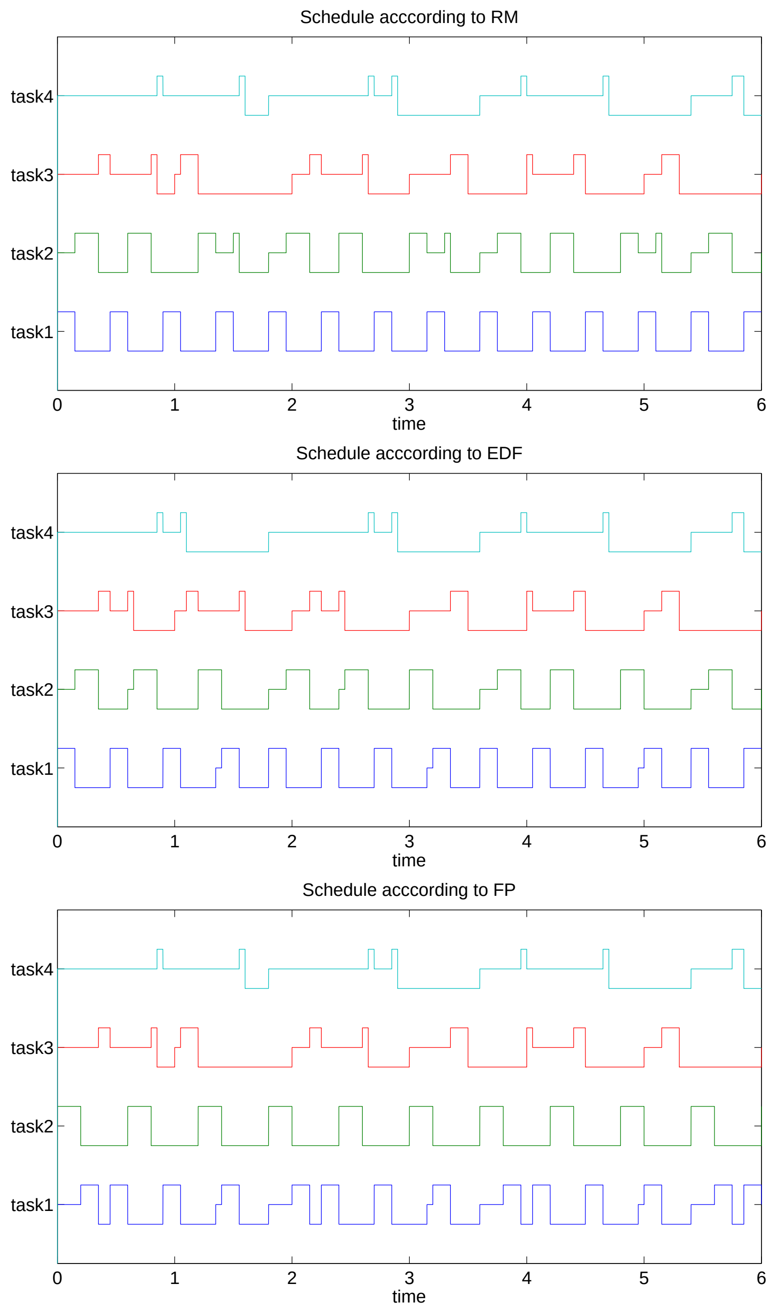

Figure 1: RM, EDF and FP schedule for task set with different periods

Deadline

- Tuesday 22 April 2014, midnight.
- Hand in
  - A short report (in pdf or similar) describing the main points developed for section "2. Basic periodic scheduling using Erika"
  - code
  - everything that can be useful