Introduction to RTEMS

The goal of this lab session is to discover some constructs of the RTEMS API through simple exercises.

Using code for this lab session

All source code is provided in the archive `rtems_lab.tar.gz`. To open this archive, issue the following command: "`tar zxvf rtems_lab.tar.gz`". This will create the directory `rtems_lab` with all source code.
A makefile is provided to compile all source code. Issue "`make help`" for more details.
To run the examples, we will use the `tsim-leon3` simulator from Aeroflex Gaisler.
Note: the code provided for this lab session has been tested on the PRISE platform, using RTEMS 4.8.0. To connect to this platform, you have to use `ssh`.

Assignements

You are tasked to perform all exercises, and write an implementation report. This implementation report is made of two files: a text file describing what you did, and the annotated source code.
Hard deadline: one week after the day of the lab session.

1 `tsim-leon3`

1.1 About `tsim-leon3`

`tsim-leon3` is an instruction-level simulator capable of emulating ERC32- and LEON-based computer systems. From the product page[1], this simulator has the following features:
The simulator is cycle true, i.e a simulator time is maintained and incremented according the IU and FPU instruction timing. The parallel execution between the IU and FPU is modelled, as well as stalls due to operand dependencies. Instruction timing has carefully been modelled after the real devices and average timing accuracy of integer instructions is better than 0.5%. Floating-point instructions have a typical accuracy of 2% due to the somewhat unpredictable operand-dependent timing of the ERC32 and LEON MEIKO FPU. The simulator time is maintained using 64-bit values providing virtually unlimited simulation time. ERC32 emulation
The full functionality of the LEON2 or LEON3 VHDL model is emulated, including cache memories, on-chip peripherals and memory controller. The amount of simulated main memory can be configured at run-time and is only limited by the resources of the host computer. The cache size and organisation can be configured between 1 - 64 kbyte/set, 1 - 4 sets, random/LRR/LRU replacement and 8 - 32 bytes/line. The multiplier latency can be programmed for 1 - 35 cycles.

1.2 Using `tsim-leon3`

`tsim-leon3` is a command-line tool, the default usage is to call it this way:

[1] See `http://www.gaisler.com`

```
$ tsim-leon3 <prog>
```

where ”prog” is an optional parameter: the name of the program to run.
The following command are available:

- `help` to get help;
- `run <function>` to run the program, or run a particular function;
- `perf` to show performance information of the simulator (number of instructions executed, cache hit, ...);
- `reset` to reset performance information;
- `quit` to quit the simulator;
- `load` to load a programm.

`tsim-leon3` also has commands to perform debugging of program.

Exercises around the RTEMS API

The goal of this first lab session is to manipulate the RTEMS concurrent API.

Exercise 1 (About periodic threads) *In this exercise, we want to experiment creation of periodic threads. Source code from* `ex1/init.c` *is an empty skeleton to be completed.*
Review this source code, and complete it to

- *create* 5 *threads. You may use the values provided in the arrays* `Priorities` *and* `Periods`*.*
- *implement a periodic behavior for each thread: it must run periodically for number a of periods equal to* `MAX_TOTAL_PASSES`*. In case a deadline is missed, an error message shall be reported.*
- *report CPU usage using* `rtems_cpu_usage_report()` *and* `rtems_rate_monotonic_report_statistics()`*.*

Note: do not forget to have the `main` *thread waits for the completion of all threads before deleting them ...*

Exercise 2 (Priority inversion and PCP) *In this exercise, we want to illustrate how priority inversion may occur using a typical scenario, and how to solve it using the Priority Ceiling Protocol.*
Here is the detail of this scenario:

- ***Main task***
  - *Create three tasks TA_1 (priority = 21), TA_2 (priority = 22), TA_3 (priority = 23), semaphore SM_1 with a counter of 1;*
  - *when all tasks are created; the main task lowers its priority to 40.*
- TA_1 ***(high priority)***
  - *Waits for some time (see* `rtems_task_wake_after()`*) so that it wakes up after TA_3 is in critical section;*
  - *upon wake up, it tries to seize SM_1;*
  - *waste some time in critical section;*
  - *free SM_1;*
  - *waste some time again;*

  - *complete task*

- TA_2 ***(medium priority)***
  - *Waits for some time (see `rtems_task_wake_after()`) so that it wakes up after TA_1 is in critical section;*
  - *waste some time again;*
  - *complete task*

- TA_3 ***(low priority)***
  - *it tries to seize SM_1;*
  - *waste some time in critical section;*
  - *free SM_1;*
  - *waste some time again;*
  - *complete task*

Source code from `ex2/init.c` *is an empty skeleton to be completed. It provides very minimal information. Review this source code, and complete it to*

- *implement this scenario. Do not try to enforce a PCP mutex at this stage;*
- *analyse the trace, and confirm this is the expected behavior with the current configuration;*
- *correct your implementation using a PCP semaphore. Explain the configuration parameters you used.*