Practice 5. Using the basic primitives of a real-time operating system.

1. Introduction

The aim of this practical is for the student to learn how to use a set of basic primitives of a *Real Time Operating System (*RTOS). The *open source* RTOS RTEMS (http://www.rtems.org/) has been chosen for this practice. This RTOS has numerous implementations, or *ports*, for different architectures. For this practice, the RTEMS *port* for the LEON3 processor will be used, and the programs obtained will be executed in the TSIM2 simulator of the LEON3 processor used in previous practices.

2. Installation of the RCC cross-compiler system

The RTEMS Cross Compiler (RCC) system integrates a cross-compiler for the LEON2 and LEON3 processors, based on GCC, together with a distribution of the RTEMS real-time operating system.

The installation of the integrated compiler in the RCC distribution comprises the following steps.

1. Download RCC distribution sparc-rtems-4.6.6-gcc-3.2.3-1.0.20-linux.tar.bz2

   http://www.gaisler.com/anonftp/rcc/bin/linux/sparc-rtems-4.6.6-gcc-3.2.3-1.0.20-linux.tar.bz2

2. Unzip it in the /opt directory

   ```
   sudo tar -C /opt -xjf sparc-rtems-4.6.6-gcc-3.2.3-1.0.20-linux.tar.bz2
   ```

3. Add **to the end of the** /home/user/. profile **file** (using e.g. gedit or vim) the following PATH modification line that incorporates the directory where the binary files of the RCC distributed compiler are stored

   ```
   PATH="/opt/rtems-4.6/bin:$PATH".
   ```

4. Log out and log back in for the PATH update to take effect (you can also use the `bash -l` command and start eclipse from the console).

3. Basic RTEMS primitives

RTEMS provides its own *Application Program Interface* (API) which can be found at https://docs.rtems.org/releases/rtemsdocs-4.6.2/share/rtems/html/c_user/index.html . It also provides a subset of the standard API for programming applications under Linux called POSIX (*Portable Operating System Interface*). This documentation is available at https://docs.rtems.org/releases/rtemsdocs-4.6.2/share/rtems/html/posix_users/index.html.

For this practice we will use only a very basic set of primitives from the native RTEMS API that will allow us to create multitasking programs and use timing and monotonic clock management services, and mutually exclusive access to shared resources.

4. Creating tasks with RTEMS

The primitives we will use for the creation of tasks are the following:

rtems_task_create

A routine to create a task by initialising the associated data structure. The parameters specify the *name*, *initial_priority*, *stack_size*, *initial_modes, attribute_set*, and a pointer to a variable (*id*) to which RTEMS assigns the identifier after invoking the function. The function returns the status of the operation. The prototype is as follows:

```
rtems_status_code       rtems_task_create (rtems_name name,
                        rtems_task_priority initial_priority,
                        size_t stack_size,
                        rtems_mode initial_modes,
                        rtems_attribute attribute_set,
                        rtems_id *id);
```

rtems_task_start

A routine that starts a task previously created with rtems_task_create. The parameters specify the identifier assigned (*id*) by rtems_task_create, the function code (*entry_point*), and the argument to be passed to the function. The function returns the status of the operation. The prototype is as follows:

```
rtems_status_code rtems_task_start (     rtems_id    id,
                        rtems_task_entry        entry_point,
                        rtems_task_argument     argument);
```

The following code shows an example of using both primitives in order to create and start a task. The task will have the maximum priority assignable to a user task (MAX_USER_TASK_PRIORITY = 1) and the default configuration parameters for stack (RTEMS_MINIMUM_STACK_SIZE), mode (RTEMS_DEFAULT_MODES) and attributes (RTEMS_DEFAULT_ATTRIBUTES) are used. These default parameters are defined by RTEMS itself in order to facilitate the creation of tasks. In addition, an auxiliary function `rtems_build_name` is used to create a variable of type `rtems_name` that can be used as task name.

```
#define MAX_USER_TASK_PRIORITY 1

rtems_task task_code(rtems_task_argument unused){
      int i,j;
      for (i=0; i < 0xFFFF; i++)
           for (j=0; j < 0xFFFF; j++)
                  if ((i==0xFF)&&(j==0xAA))
```

```
                        printf( 'task ' );

}

rtems_name task_name; //variable for the name
rtems_status_code status; //variable for status
rtems_id task_id; //variable for the identifier


//Build the name
task_name = rtems_build_name( 'T', 'A', 'S', 'K');

status = rtems_task_create(task_name,
                           MAX_USER_TASK_PRIORITY,
                           RTEMS_MINIMUM_STACK_SIZE,
                           RTEMS_DEFAULT_MODES,
                           RTEMS_DEFAULT_ATTRIBUTES,
                           & task_id);

status = rtems_task_start( task_id, task_code, 1 );
```

5. Monotonic clock and basic timing services in RTEMS

RTEMS provides the following routines to manage the monotonic system clock, and other basic timing services.

rtems_clock_set

Routine to initialise the universal system time via the *time_of_day (TOD)* parameter. The function returns the status of the operation. Its prototype is as follows:

```
rtems_status_code rtems_clock_set (const rtems_time_of_day
                                   *time_of_day);
```

rtems_clock_get

Routine that allows, on the one hand, to know the system time in different formats, but can also be used to know the number of clock ticks per second, or the number of ticks since the system was started. The function also returns the status of the operation. Its prototype is as follows:

```
rtems_status_code rtems_clock_get (
                        rtems_clock_get_options option,
                        void * time_buffer);
```

With the RTEMS_CLOCK_GET_TOD option, the *time_of_day (TOD)* is obtained, as shown in the following example code:

```
rtems_time_of_day time;

rtems_clock_get( RTEMS_CLOCK_GET_TOD, &time );
```

With the RTEMS_CLOCK_GET_TICKS_PER_SECONDS option, the number of ticks per second is obtained, as shown in the following code:

```
rtems_interval ticks_per_second;

rtems_clock_get( RTEMS_CLOCK_GET_TICKS_PER_SECOND,
            & ticks_per_second );
```

The RTEMS_CLOCK_GET_TICKS_SINCE_BOOT option retrieves the number of ticks since the program was started, as shown in the following code:

```
rtems_interval ticks_since_boot;

rtems_clock_get(RTEMS_CLOCK_GET_TICKS_SINCE_BOOT,
       & ticks_since_boot ); //ticks_since_boot
```

rtems_task_wake_after

`A Delay` type routine that allows a task to sleep for a number of ticks. The function also returns the status of the operation. Its prototype is as follows:

```
rtems_status_code rtems_task_wake_after ( rtems_interval ticks);
```

Although RTEMS does not have a primitive of type `DelayUntil` that works with an absolute time reference, avoiding drift in periodic task scheduling, it is possible to define it from `rtems_task_wake_after` and `rtems_clock_get`. The following code is an example of defining a `task_delay_until` routine

```
rtems_status_code task_delay_until(rtems_interval ticks_from_boot){

      rtems_interval current_time;
      rtems_status_code status=0;
      rtems_clock_get(RTEMS_CLOCK_GET_TICKS_SINCE_BOOT,& current_time);
      if(ticks_from_boot> current_time){
            status=rtems_task_wake_after(ticks_from_boot-current_time);
      }
      return status;
}
```

6. RTEMS configuration parameters

In addition to the API functions, RTEMS has different configuration parameters. The most relevant are the following:

CONFIGURE_MAXIMUM_TASKS

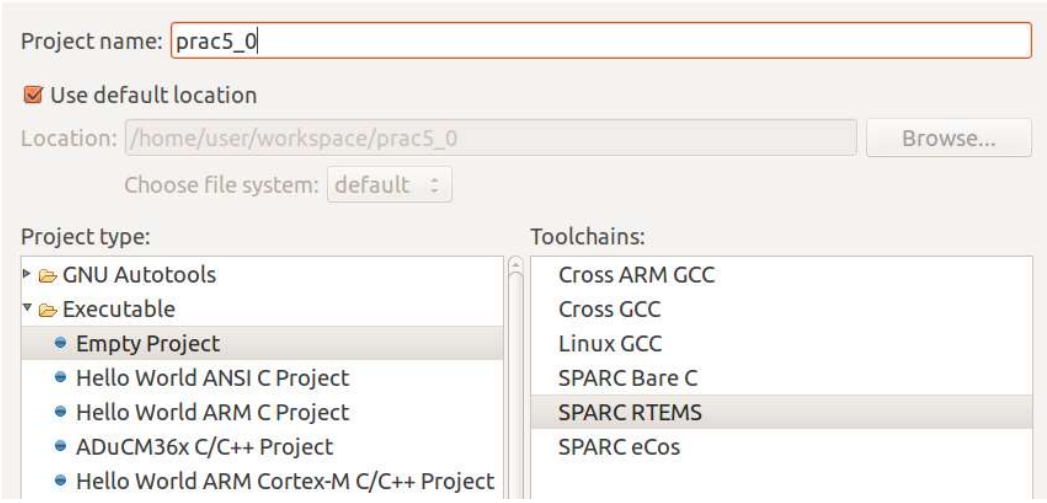
Parameter defining the maximum number of tasks that can be instantiated.

CONFIGURE_EXTRA_TASK_STACKS

Parameter defining an extra stack size for tasks.

7. Practice 5_0: Monotonic clock management under RTEMS and LEON3

Creation of a new C project with the options Project Type: Empty Project and Toolchains: SPARC RTEMS) named prac5_0 whose executable is for the Sparc RTEMS platform, as shown below.



Set the */opt/items-4.6/sparc-rtems/leon3/lib/include* directory as the directory to search for header files.
Add to this project the *Prac5_0.c* file included in the web page. Complete the code marked with /TODO to get a system run like the one below.

```
*** CLOCK TICK TEST ***
TA1 - rtems_clock_get - 00:00:00 22/04/2022
 - rtems_ticks_since_boot - 0
TA2 - rtems_clock_get - 00:00:00 22/04/2022
 - rtems_ticks_since_boot - 0
TA3 - rtems_clock_get - 00:00:00 22/04/2022
 - rtems_ticks_since_boot - 0
TA1 - rtems_clock_get - 00:00:05 22/04/2022
 - rtems_ticks_since_boot - 500
TA2 - rtems_clock_get - 00:00:10 22/04/2022
 - rtems_ticks_since_boot - 1000
TA1 - rtems_clock_get - 00:00:10 22/04/2022
 - rtems_ticks_since_boot - 1000
TA3 - rtems_clock_get - 00:00:15 22/04/2022
 - rtems_ticks_since_boot - 1500
TA1 - rtems_clock_get - 00:00:15 22/04/2022
 - rtems_ticks_since_boot - 1500
```

```
TA2 - rtems_clock_get - 00:00:20 22/04/2022
 - rtems_ticks_since_boot - 2000
TA1 - rtems_clock_get - 00:00:20 22/04/2022
 - rtems_ticks_since_boot - 2000
TA1 - rtems_clock_get - 00:00:25 22/04/2022
 - rtems_ticks_since_boot - 2500
```

8. Practice 5_1: Task creation and basic timing in RTEMS

Create a new C project called prac5_1 whose executable is for the Sparc RTEMS platform. Set the directory */opt/items-4.6/sparc-rtems/leon3/lib/include* as the directory to search for header files. Copy the file *Prac5_0.c* and rename it *Prac5_1.c.* Modify, in addition, the code of *Prac5_1.c* so that the tasks created in the program have the periodicity shown in Table 1. The following functions shall be used to create the tasks (these functions shall therefore replace the Test_task function of *Prac5_0.c***, which shall be removed**). These functions shall be completed using the `rtems_task_wake_after` function so that the period of each one is the one corresponding to the table.

Task	Period (In Ticks)
TAvoidObstacles	10
TPathTracking	15
TSensorFusion	30
TCalculatePath	30

Table 1 Task periods

```
rtems_task TAvoidObstacles (rtems_task_argument unused) {

	// Complete:

	puts("T1 Do Avoid Obstacles");
	printf(" - rtems_ticks_since_boot - %i",get_ticks_since_boot());


}

rtems_task TPathTracking (rtems_task_argument unused) {

	// Complete:

	puts("T2 Do PathTracking");
	printf(" - rtems_ticks_since_boot - %i",get_ticks_since_boot());


}
```

```
rtems_task TSensorFusion (rtems_task_argument unused) {

      // Complete:

      puts("T3 Do Sensor Fusion");
      printf(" - rtems_ticks_since_boot - %i",get_ticks_since_boot());



}


rtems_task TCalculatePath (rtems_task_argument unused) {

      // Complete:

      puts("T4 Do CalculatePath");
      printf(" - rtems_ticks_since_boot - %i",get_ticks_since_boot());

}
```

Once the functions are completed (using rtems_task_wake_after), the program will behave as in practice 4_2, so that TAvoidObstacles is executed periodically every 10 ticks, and TPathTracking every 15 ticks, TSensorFusion and TCalculatePath every 30 ticks. **Check that the console output is as follows**

```
*** PRAC 5_1 ***
T1 Do Avoid Obstacles
 - rtems_ticks_since_boot - 0
T2 Do PathTracking
 - rtems_ticks_since_boot - 0
T3 Do Sensor Fusion
 - rtems_ticks_since_boot - 0
T4 Do CalculatePath
 - rtems_ticks_since_boot - 0
T1 Do Avoid Obstacles
 - rtems_ticks_since_boot - 10
T2 Do PathTracking
 - rtems_ticks_since_boot - 15
T1 Do Avoid Obstacles
 - rtems_ticks_since_boot - 20
T3 Do Sensor Fusion
 - rtems_ticks_since_boot - 30
T1 Do Avoid Obstacles
 - rtems_ticks_since_boot - 30
T2 Do PathTracking
 - rtems_ticks_since_boot - 30
T4 Do CalculatePath
 - rtems_ticks_since_boot - 30
T1 Do Avoid Obstacles
 - rtems_ticks_since_boot - 40
T2 Do PathTracking
 - rtems_ticks_since_boot - 45
T1 Do Avoid Obstacles
 - rtems_ticks_since_boot - 50
T3 Do Sensor Fusion
```

```
 - rtems_ticks_since_boot - 60
T1 Do Avoid Obstacles
 - rtems_ticks_since_boot - 60
T2 Do PathTracking
 - rtems_ticks_since_boot - 60
T4 Do CalculatePath
 - rtems_ticks_since_boot - 60
T1 Do Avoid Obstacles
 - rtems_ticks_since_boot - 70
T2 Do PathTracking
 - rtems_ticks_since_boot - 75
T1 Do Avoid Obstacles
 - rtems_ticks_since_boot - 80
T3 Do Sensor Fusion
 - rtems_ticks_since_boot - 90
T1 Do Avoid Obstacles
 - rtems_ticks_since_boot - 90
T2 Do PathTracking
 - rtems_ticks_since_boot - 90
T4 Do CalculatePath
 - rtems_ticks_since_boot - 90
T1 Do Avoid Obstacles
 - rtems_ticks_since_boot - 100
T2 Do PathTracking
 - rtems_ticks_since_boot - 105
T1 Do Avoid Obstacles
 - rtems_ticks_since_boot - 110
T3 Do Sensor Fusion
 - rtems_ticks_since_boot - 120
T1 Do Avoid Obstacles
 - rtems_ticks_since_boot - 120
T2 Do PathTracking
 - rtems_ticks_since_boot - 120
T4 Do CalculatePath
 - rtems_ticks_since_boot - 120
```

9. Practice 5_2: Task Creation and Periodic Timing without Drift in RTEMS

Create a new C project called prac5_2 whose executable is for the Sparc RTEMS platform. Set the directory */opt/items-4.6/sparc-rtems/leon3/lib/include* as the directory to search for header files. Copy the file *Prac5_1.c* and rename it to *Prac5_2.c.* Modify the program code to add the code for the `task_delay_until` function as defined below

```
rtems_status_code task_delay_until(rtems_interval ticks_from_boot){

	rtems_interval current_time;
	rtems_status_code status=0;
	rtems_clock_get(RTEMS_CLOCK_GET_TICKS_SINCE_BOOT,& current_time);
	if(ticks_from_boot> current_time){
		status=rtems_task_wake_after(ticks_from_boot-current_time);
	}
	return status;
}
```

In addition, modify the four tasks above to replace calls to the `rtems_task_task_wake_after` function with calls to the `task_delay_until` function.

Check that the console output is identical to that obtained in practice 5_1.

10. Critical Section Management in RTEMS

RTEMS allows critical sections to be defined in such a way that access to shared resources is protected, while avoiding the problem of priority inversion. Binary semaphores with priority inheritance or priority ceiling can be used to define these critical sections in RTEMS.

The RTEMS native API provides the following primitives for critical section management.

rtems_semaphore_create

Routine that allows to create a semaphore, initialising the associated data structure. The parameters specify the *name*, the initial value of the semaphore (*count*), used to define counter semaphores, the attributes (*attribute_set*), which allows to define the type of semaphore, the priority ceiling (*priority_ceiling*), only for semaphores with priority ceiling, and a pointer to a variable (*id*) to which RTEMS assigns the identifier of the semaphore after invoking the function. The function returns the status of the operation. The prototype is as follows:

```
rtems_status_code rtems_semaphore_create(
     rtems_name name,
     uint32_t count,
     rtems_attribute attribute_set,
     rtems_task_priority priority_ceiling,
     rtems_id *id
 );
```

The following macros can be used to define the semaphore attributes:

- `RTEMS_FIFO / RTEMS_PRIORITY`
  `(Blocked tasks wait for FIFO/priority)`

- `RTEMS_BINARY_SEMAPHORE/RTEMS_COUNTING_SEMAPHORE`
  `/RTEMS_SIMPLE_BINARY_SEMAPHORE`
  `(Binary semaphore, counter or simple binary)`

- `RTEMS_INHERIT_PRIORITY /RTEMS_NO_INHERIT_PRIORITY`
  `(Implement/Do not implement priority inheritance)`

- `RTEMS_PRIORITY_CEILING / RTEMS_NO_PRIORITY_CEILING`
  `(Implement/Do not implement priority ceiling)`

The following macro combination is used to define a Mutex using the priority inheritance protocol.

```
rtems_attribute mutex_attribute = RTEMS_PRIORITY |
            RTEMS_BINARY_SEMAPHORE | RTEMS_INHERIT_PRIORITY;
```

The following macro combination is used to define a Mutex using the priority ceiling protocol.

```
rtems_attribute mutex_attribute = RTEMS_PRIORITY |
            RTEMS_BINARY_SEMAPHORE | RTEMS_PRIORITY_CEILING;
```

rtems_semaphore_obtain

Routine that allows to capture the semaphore. The parameters specify the identifier assigned to the semaphore (*id*) by rtems_semaphore_create, the capture options (*option_set*) and the maximum waiting interval in ticks (*timeout*) for capturing the resource:

```
rtems_status_code rtems_semaphore_obtain(
     rtems_id id,
     uint32_t option_set,
     rtems_interval timeout
 );
```

The semaphore capture options are: `RTEMS_WAIT` | `RTEMS_NO_WAIT`

- `RTEMS_WAIT` the task is blocked until the semaphore is captured or until the time set by `timeout` is reached.
- `RTEMS_NO_WAIT` the task is not blocked, only the semaphore is captured if it is free.

The `RTEMS_WAIT` option can be combined with the RTEMS_NO_TIMEOUT macro assigned to *timeout* to cause an indefinite timeout

rtems_semaphore_release

Routine that allows the semaphore to be released. Parameter is the identifier assigned (id) by rtems_semaphore_create

```
rtems_status_code rtems_semaphore_release(rtems_id id);
```

11. Practice 5_3: Critical Sections in RTEMS

Create a new C project called prac5_3 whose executable is for the Sparc RTEMS platform. Set the directory */opt/items-4.6/sparc-rtems/leon3/lib/include* as the directory to search for header files. Copy the file *Prac5_2.c* and rename it to *Prac5_3.c.*

Add the declaration of the semaphore identifier and semaphore name as global variables.

```
//Declaration of the Mutex identifier identifier and name as a global variable
rtems_id MutexId;
rtems_name MutexName = rtems_build_name('M', 'T', 'X', ' ');
```

Create a semaphore with priority inheritance within the Init function using the following code:

```
//Creation of the semaphore inside the Init function

rtems_task Init(
  rtems_task_argument argument

{

      rtems_attribute mutex_attribute = RTEMS_PRIORITY |
                    RTEMS_BINARY_SEMAPHORE | RTEMS_INHERIT_PRIORITY;

      rtems_semaphore_create(MutexName,1,mutex_attribute,
                                    RTEMS_NO_PRIORITY, & MutexId);
      ...
```

Modify the program code so that access to standard output via `puts and printf is` always mutually exclusive. To do this, in each task, capture a Mutex before the call to `puts`, and release it after the call to `printf`. **Check that the console output is identical to the one obtained in practice 5_1.**