Práctica 5. Utilización de las primitivas básicas de un sistema operativo de tiempo real.

1. Objetivo

El objetivo de esta práctica es que el alumno aprenda a utilizar un conjunto de primitivas básicas de un sistema operativo de tiempo real (en inglés *Real Time Operating System* o RTOS). Se ha elegido para realizar esta práctica el RTOS de carácter *open source* RTEMS (http://www.rtems.org/). Este RTOS cuenta con numerosas implementaciones, o *ports*, para distintas arquitecturas. Para esta práctica se utilizará el *port* de RTEMS para el procesador LEON3, y se ejecutarán los programas obtenidos en el simulador TSIM2 del procesador LEON3 utilizado en prácticas anteriores.

2. Instalación del sistema de compilación cruzado RCC

El sistema de compilación cruzada RCC (RTEMS Cross Compiler) integra un compilador cruzado para los procesadores LEON2 y LEON3, basado en GCC, junto con una distribución del sistema operativo de tiempo real RTEMS.

La instalación del compilador integrado en la distribución RCC comprende los siguientes pasos.

1. Descargarse la distribución de RCC 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. Descomprimirla en el directorio /opt

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

3. Añadir **al final del archivo** /home/user/.profile (utilizando, por ejemplo, gedit o vim) la siguiente línea de modificación del PATH que incorpora el directorio donde se almacenan los archivos binarios del compilador distribuido con RCC

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

4. Cerrar la sesión y volver a entrar para que se haga efectiva la actualización del PATH (también se puede utilizar la orden `bash -l` y arrancar eclipse desde la consola)

3. Primitivas básicas de RTEMS

RTEMS proporciona su propia interfaz de programación de aplicaciones (*Application Program Interface* o API) que puede consultarse en https://docs.rtems.org/releases/rtemsdocs-4.6.2/share/rtems/html/c_user/index.html . También proporciona un subconjunto de la API estándar para programar aplicaciones

bajo Linux denominada POSIX (*Portable Operating System Interface*). En https://docs.rtems.org/releases/rtemsdocs-4.6.2/share/rtems/html/posix_users/index.html se puede consultar esta documentación.

Para esta práctica utilizaremos solamente con un conjunto muy básico de primitivas de la API nativa de RTEMS que nos van a permitir crear programas multitarea y utilizar servicios de temporización y gestión del reloj monotónico, y acceso en exclusión mutua a recursos compartidos.

4. Creación de tareas con RTEMS

Las primitivas que utilizaremos para la creación de tareas son las siguientes:

rtems_task_create

Rutina que permite crear una tarea, inicializando la estructura de datos asociada. Los parámetros especifican el nombre (*name*), la prioridad (*initial_priority*), el tamaño de la pila de la tarea (*stack_size*), el modo inicial (*initial_modes*), los atributos (*attribute_set*) y un puntero a una variable (*id*) a la que RTEMS asigna el identificador tras invocar la función. La función retorna el estatus de la operación. El prototipo es el siguiente:

```
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

Rutina que pone en marcha una tarea creada previamente con rtems_task_create. Los parámetros especifican el identificador asignado (*id*) por rtems_task_create, el código de la función (*entry_point*), y el argumento que se le pasará a la función. La función retorna el estatus de la operación. El prototipo es el siguiente:

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

El siguientes código muestra un ejemplo de utilización de ambas primitivas con el fin de crear y poner en marcha una tarea. La tarea tendrá la prioridad máxima asignable a una tarea de usuario (MAX_USER_TASK_PRIORITY = 1) y se utilizan los parámetros de configuración por defecto para la pila (RTEMS_MINIMUM_STACK_SIZE), el modo (RTEMS_DEFAULT_MODES) y los atributos (RTEMS_DEFAULT_ATTRIBUTES). Estos parámetros por defecto están definidos por el propio RTEMS con el fin de facilitar la creación de tareas. Además se utiliza una función auxiliar `rtems_build_name` para poder crear una variable de tipo `rtems_name` que pueda ser utilizado como nombre de la tarea.

```
#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( 'tarea \n' );

}

rtems_name task_name; //variable para el nombre
rtems_status_code status; //variable para el status
rtems_id   task_id; //variable para el identificador


//Construir el nombre
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. Reloj monotónico y servicios básicos de temporización en RTEMS

RTEMS proporciona las siguientes rutinas para gestionar el reloj monotónico del sistema, y otros servicios básicos de temporización.

rtems_clock_set

Rutina que permite inicializar el tiempo universal del sistema a través del parámetro *time_of_day (TOD)*. La función retorna el estatus de la operación. Su prototipo es el siguiente:

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

rtems_clock_get

Rutina que permite, por una parte, conocer el tiempo del sistema en distintos formatos, pero también puede ser utilizada para conocer el número de ticks de reloj por segundo, o el número de ticks desde que se inició el sistema. La función retorna además el estatus de la operación. Su prototipo es el siguiente:

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

Con la opción RTEMS_CLOCK_GET_TOD se obtiene el *time_of_day (TOD)*, tal como se muestra en el siguiente código de ejemplo:

```
rtems_time_of_day time;

rtems_clock_get( RTEMS_CLOCK_GET_TOD, &time );
```

Con la opción RTEMS_CLOCK_GET_TICKS_PER_SECONDS se obtiene el número de ticks por segundo, tal como se muestra en el siguiente código:

```
rtems_interval ticks_per_second;

rtems_clock_get( RTEMS_CLOCK_GET_TICKS_PER_SECOND,
                                &ticks_per_second );
```

Con la opción RTEMS_CLOCK_GET_TICKS_SINCE_BOOT se obtiene el número de ticks desde que se inició el programa, tal como se muestra en el siguiente código:

```
rtems_interval ticks_since_boot;

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

rtems_task_wake_after

Rutina del tipo `Delay` que permite dormir una tarea durante un número de ticks. La función retorna además el estatus de la operación. Su prototipo es el siguiente:

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

Si bien RTEMS no cuenta con una primitiva del tipo `DelayUntil` que trabaje con una referencia absoluta de tiempo, evitando la deriva en la programación de tareas periódicas, es posible definirla a partir de `rtems_task_wake_after` y `rtems_clock_get`. El siguiente código es un ejemplo de definición de una rutina task_delay_until

```
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. Parámetros de configuración de RTEMS

Además de las funciones de la API, RTEMS cuenta con diferentes parámetros de configuración. Los más relevantes son los siguientes:

CONFIGURE_MAXIMUM_TASKS

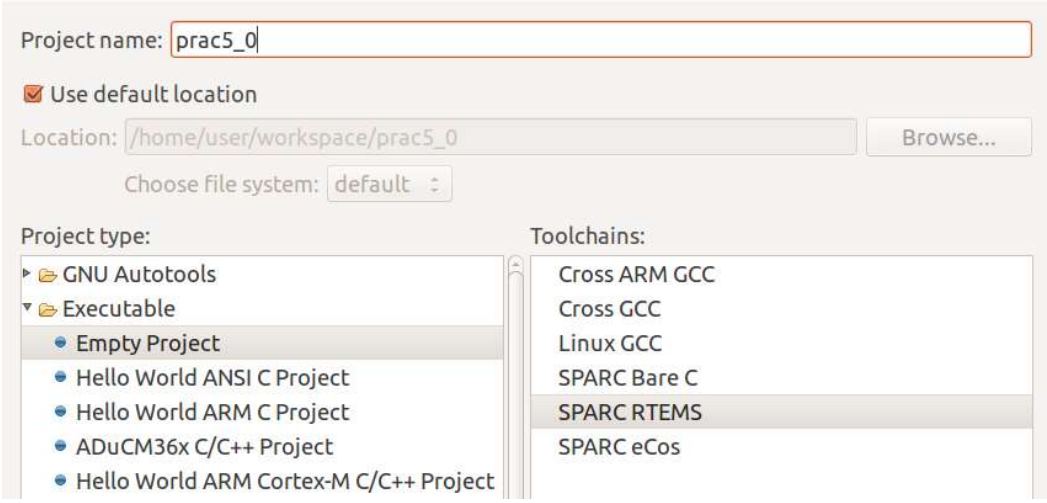
Parámetro que define el número máximo de tareas que se puede instanciar.

CONFIGURE_EXTRA_TASK_STACKS

Parámetro que define un tamaño extra de pila para las tareas.

7. Práctica 5_0: Gestión del reloj monotónico bajo RTEMS y LEON3

Creación de un nuevo proyecto C con las opciónes Project Type: Empty Project y Toolchains: SPARC RTEMS) denominado prac5_0 cuyo ejecutable sea para la plataforma Sparc RTEMS, tal como aparece a continuación.



Fijar el directorio */opt/items-4.6/sparc-rtems/leon3/lib/include* como directorio de búsqueda de archivos de cabecera.
Añadir a ese proyecto el archivo *Prac5_0.c* incluido en la página web. Completar el código marcado con /TODO para conseguir una ejecución del sistema como la que aparece a continuación.

```
*** 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. Práctica 5_1: Creación de tareas y temporización básica en RTEMS

Creación de un nuevo proyecto C denominado prac5_1 cuyo ejecutable sea para la plataforma Sparc RTEMS. Fijar el directorio */opt/items-4.6/sparc-rtems/leon3/lib/include* como directorio de búsqueda de archivos de cabecera. Copiar el archivo *Prac5_0.c* y renombrarlo como *Prac5_1.c.* Modificad, además, el código de *Prac5_1.c* para que las tareas que se crean en el programa tengan la periodicidad que se muestra en la Tabla 1. Para crear las tareas se utilizarán las siguientes funciones (estas funciones sustituirán, por tanto, a la función Test_task de la *Prac5_0.c* **que deberá ser eliminada**). Estas funciones deberán ser completadas utilizando la función `rtems_task_wake_after` para que el periodo de cada una sea el correspondiente a la tabla.

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

Tabla 1 Periodos de las tareas

```
rtems_task TAvoidObstacles (rtems_task_argument unused) {

      //TODO completar.

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


}

rtems_task TPathTracking (rtems_task_argument unused) {

      //TODO completar.

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


}

rtems_task TSensorFusion (rtems_task_argument unused) {

      //TODO completar.

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



}


rtems_task TCalculatePath (rtems_task_argument unused) {

      //TODO completar.

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

}
```

Una vez completadas las funciones (utilizando rtems_task_wake_after), el programa tendrá un comportamiento análogo al de la práctica 4_2, de forma que TAvoidObstacles se ejecute periódicamente cada 10 ticks, y TPathTracking cada 15 ticks, TSensorFuscion y TCalculatePath cada 30 ticks. **Comprobad que la salida de pantalla es la siguiente**

```
*** 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. Práctica 5_2: Creación de tareas y temporización periódica sin Deriva en RTEMS

Creación de un nuevo proyecto C denominado prac5_2 cuyo ejecutable sea para la plataforma Sparc RTEMS. Fijar el directorio */opt/items-4.6/sparc-rtems/leon3/lib/include* como directorio de búsqueda de archivos de cabecera. Copiar el archivo *Prac5_1.c* y renombrarlo como *Prac5_2.c.* Modificar el código del programa para añadir el código de la función `task_delay_until` tal como se define a continuación

```
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;
}
```

Además, modificad las cuatro tareas anteriores para que sustituyan las llamadas a la función `rtems_task_wake_after` por llamadas a la función `task_delay_until`.

Comprobad que la salida de pantalla es idéntica a la obtenida en la práctica 5_1

10. Gestión de Secciones Críticas en RTEMS

RTEMS permite definir secciones críticas de forma que quede protegido el acceso a los recursos compartidos, evitando, además, el problema de la inversión de prioridad. Para definir estas secciones críticas en RTEMS se pueden utilizar semáforos binarios con herencia de prioridad o con techo de prioridad.

La API nativa de RTEMS proporciona las siguientes primitivas para la gestión de secciones críticas.

rtems_semaphore_create

Rutina que permite crear un semáforo, inicializando la estructura de datos asociada. Los parámetros especifican el nombre (*name*), el valor inicial del semáforo (*count*), usado para definir semáforos contadores, los atributos (*attribute_set*), que permite definir el tipo de semáforo, la prioridad techo (*priority_ceiling*),sólo para los semáforos con techo de prioridad, y un puntero a una variable (*id*) a la que RTEMS asigna el identificador del semáforo tras invocar la función. La función retorna el estatus de la operación. El prototipo es el siguiente:

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

Las siguientes macros se pueden usar para definir los atributos del semáforo:

- RTEMS_FIFO / RTEMS_PRIORITY
  (Las tareas bloqueadas esperan por FIFO/prioridad)

- RTEMS_BINARY_SEMAPHORE/RTEMS_COUNTING_SEMAPHORE
  /RTEMS_SIMPLE_BINARY_SEMAPHORE
  (Semáforo binario, contador o binario simple)

- RTEMS_INHERIT_PRIORITY /RTEMS_NO_INHERIT_PRIORITY
  (Implementar/No implementar herencia de prioridad)

- RTEMS_PRIORITY_CEILING / RTEMS_NO_PRIORITY_CEILING
  (Implementar/No implementar techo de prioridad)

Para definir un Mutex que usa el protocolo de herencia de prioridad se utiliza la siguiente combinación de macros.

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

Para definir un Mutex que usa el protocolo de techo de prioridad se utiliza la siguiente combinación de macros.

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

rtems_semaphore_obtain

Rutina que permite capturar el semáforo. Los parámetros especifican el identificador asignado al semáforo (*id*) por rtems_semaphore_create, las opciones de captura (*option_set*) y el intervalo máximo de espera en ticks (*timeout*)para la captura del recurso:

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

Las opciones de captura del semáforo son: `RTEMS_WAIT|RTEMS_NO_WAIT`

- `RTEMS_WAIT` la tarea se bloquea hasta la captura del semáforo o hasta que se alcanza el tiempo fijado por `timeout`
- `RTEMS_NO_WAIT` la tarea no se bloquea, sólo se captura el semáforo si está libre.

La opción `RTEMS_WAIT` se puede combinar con la macro RTEMS_NO_TIMEOUT asignada a *timeou*t para provocar una espera indefinida

rtems_semaphore_release

Rutina que permite liberar el semáforo. El parámetro es el identificador asignado (id) por rtems_semaphore_create

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

11. Práctica 5_3: Secciones críticas en RTEMS

Creación de un nuevo proyecto C denominado prac5_3 cuyo ejecutable sea para la plataforma Sparc RTEMS. Fijar el directorio */opt/items-4.6/sparc-rtems/leon3/lib/include* como directorio de búsqueda de archivos de cabecera. Copiar el archivo *Prac5_2.c* y renombrarlo como *Prac5_3.c.*

Añadir la declaración del identificador y del nombre del semáforo como variables globales.

```
//Declaración del identificador del identificador del Mutex y el nombre como variable global

rtems_id MutexId;

rtems_name MutexName = rtems_build_name('M', 'T', 'X', ' ');
```

Crear un semáforo con herencia de prioridad dentro de la función Init utilizando el siguiente código:

```
//Creación del semáforo dentro de la función Init

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);
      ...
```

Modificar el código del programa para que el acceso a la salida estándar a través de `puts` y `printf` se ejecute siempre en exclusión mutua. Para ello, en cada tarea, capturar un Mutex antes de la llamada a `puts`, y liberarlo después la llamada a `printf`. **Comprobad que la salida de pantalla es idéntica a la obtenida en la práctica 5_1**