Primera práctica de Sistemas Empotrados y Ubicuos
Curso 2018–2019

Juan Zamorano Flores
jzamora@fi.upm.es
Departamento de Arquitectura y Tecnología de Sistemas Informáticos
ETSI Informáticos
Universidad Politécnica de Madrid

24 de septiembre de 2018

1. Descripción

La primera práctica ligera de Sistemas Empotrados y Ubicuos consiste en la utilización de dos sistemas de desarrollo cruzado para computadores empotrados con procesadores SPARC versión 7. Ambos sistemas de desarrollo cruzado se encuentran instalados en el computador `batman` y son:

Sistema basado en RTEMS. RTEMS (http://www.rtems.com) es un sistema operativo abierto para computadores empotrados que proporciona una interfaz C/POSIX y está disponible para una gran cantidad de plataformas de ejecución.

Sistema basado en ORK. ORK (http://www.dit.upm.es/ str/projects/ork/index.html) es un núcleo abierto para computadores empotrados que se usa con el compilador GNAT (http://www.adacore.com) de forma que permite el desarrollo de software en Ada 2005 usando las restricciones del perfil de Ravenscar.

La práctica consiste en la compilación y ejecución de un conjunto de 3 tareas concurrentes independientes entre sí. Las tareas `Task_A`, `Task_B` y `Task_C` son periódicas.

El objetivo es ejecutar las tareas cuyos requisitos temporales que se muestran en la tabla 1.

Tarea	Período	Plazo de respuesta	Tiempo de cómputo
Task_A	14	10	3
Task_B	20	20	6
Task_C	36	36	8

Cuadro 1: requisitos temporales de las tareas

Además, de estas tres tareas existe una tarea `Background` que no posee requisitos de tiempo real y ejecuta cuando ninguna de las tres tareas de tiempo real está lista para ejecutar.

2. Codificación en Ada

La codificación se ha hecho utilizando el subconjunto definido en el perfil de Ravenscar. El fichero de configuración `gnat.adc` reside n el directorio de trabajo para que el sistema de compilación cruzada GNAT/ORK pueda detectar las posibles violaciones a las restricciones.

El material para la realización de esta práctica se puede encontrar en batman en el directorio `/usr/local/openravenscar/SEU/ada`, se incluye el código completo y los ficheros auxiliares necesarios.

Cada tarea se implementa mediante una única tarea Ada independiente en un paquete con la siguiente estructura:

tasks.ads

```
with Ada.Real_Time;
use type Ada.Real_Time.Time_Span;

package Tasks is

   procedure Background;

private

   Time_Unit : constant Ada.Real_Time.Time_Span :=
                          Ada.Real_Time.Milliseconds (100);

   -- 500 Milliseconds is the initial offset for the tasks
   -- It is enough time to elaborate the program

   Offset : constant Ada.Real_Time.Time_Span :=
              Ada.Real_Time.Milliseconds (500);
```

```
   Time_Zero : constant Ada.Real_Time.Time :=
               Ada.Real_Time.Time_of (0, Ada.Real_Time.Time_Span_Zero) +
               Offset;

end Tasks;
```

En la parte privada del paquete `tasks` se define la unidad de tiempo y el tiempo a partir del cual empezarán a ejecutar las tareas de tiempo real. En el cuerpo de dicho paquete se declaran todas las tareas. para cada tarea consuma su tiempo de cómputo correspondiente se ejecuta un procedimiento cuyo tiempo de cómputo es equivalente. Dicho procedimiento es `Small_Whetstone (numero_iteraciones)` del paquete `workload`, que consiste en una mezcla de instrucciones de coma flotante.

tasks.adb

```
with Kernel.Serial_Output;

with System;
with Workload;

package body Tasks is

   -- This constant was meausured with sis -freq 10
   -- A program for measuring this constant can be built with
   -- make -f Makefile.measure

   Time_per_Kwhetstones : constant Ada.Real_Time.Time_Span :=
                             Ada.Real_Time.Nanoseconds (476_120);

   procedure Execution_Time (Time : Ada.Real_Time.Time_Span) is

   begin
      Workload.Small_Whetstone (Time / Time_per_Kwhetstones);
   end Execution_Time;

   -- This procedure prints Real_Time.Clock - Time_Zero

   procedure Print_RTClok is
      Seconds_Count_From_Time_Zero : Ada.Real_Time.Seconds_Count;
      Time_Span_From_Time_Zero : Ada.Real_Time.Time_Span;
      Duration_From_Time_Zero : Duration;

   begin

      Ada.Real_Time.Split (Ada.Real_Time.Clock - Offset,
```

```
                                Seconds_Count_From_Time_Zero,
                                Time_Span_From_Time_Zero);
   Duration_From_Time_Zero := Duration (Seconds_Count_From_Time_Zero) +
                        Ada.Real_Time.To_Duration (Time_Span_From_Time_Zero);
   Kernel.Serial_Output.Put (" RT.Clock = ");
   Kernel.Serial_Output.Put (DurationImage(duration_From_Time_Zero));

end Print_RTClok;

-- Temporal parameters of Tasks

subtype Tasks is character range A .. C;

WCET_A : constant Ada.Real_Time.Time_Span := 3 * Time_Unit;
Period_A : constant Ada.Real_Time.Time_Span := 14 * Time_Unit;

WCET_B : constant Ada.Real_Time.Time_Span := 6 * Time_Unit;
Period_B : constant Ada.Real_Time.Time_Span := 20 * Time_Unit;

WCET_C : constant Ada.Real_Time.Time_Span := 8 * Time_Unit;
Period_C : constant Ada.Real_Time.Time_Span := 36 * Time_Unit;

procedure Background is

begin
   loop
      null;
   end loop;
end Background;

task A is
   pragma Priority (System.PriorityLast);
end A;

task B is
   pragma Priority (System.PriorityLast - 1);
end B;

task C is
   pragma Priority (System.PriorityLast - 2);
end C;

task body A is
   Next_Time : Ada.Real_Time.Time := Time_Zero + Period_A;
begin
   loop
      delay until Next_Time;
      Kernel.Serial_Output.Put ("Task A running  ");
      Print_RTClok;
```

```
            Kernel.Serial_Output.New_Line;
            Execution_Time (WCET_A);
            Next_Time := Next_Time + Period_A;
            Kernel.Serial_Output.Put ("Task A finishing");
            Print_RTClok;
            Kernel.Serial_Output.New_Line;
        end loop;
    end A;

    task body B is
        Next_Time : Ada.Real_Time.Time := Time_Zero + Period_B;
    begin
        loop
            delay until Next_Time;
            Kernel.Serial_Output.Put ("Task B running  ");
            Print_RTClok;
            Kernel.Serial_Output.New_Line;
            Execution_Time (WCET_B);
            Next_Time := Next_Time + Period_B;
            Kernel.Serial_Output.Put ("Task B finishing");
            Print_RTClok;
            Kernel.Serial_Output.New_Line;
        end loop;
    end B;

    task body C is
        Next_Time : Ada.Real_Time.Time := Time_Zero + Period_C;
    begin
        loop
            delay until Next_Time;
            Kernel.Serial_Output.Put ("Task C running  ");
            Print_RTClok;
            Kernel.Serial_Output.New_Line;
            Execution_Time (WCET_C);
            Next_Time := Next_Time + Period_C;
            Kernel.Serial_Output.Put ("Task C finishing");
            Print_RTClok;
            Kernel.Serial_Output.New_Line;
        end loop;
    end C;

end Tasks;
```

El planificador del *run-time system* de Ada se encarga de arrancar todas las tareas, y el programa principal tiene la siguiente estructura:

main.adb

```
with Tasks;
with System;

procedure Main is

   pragma Priority (System.PriorityFirst);

begin

   Tasks.Background;

end Main;
```

Al ejecutar el programa se puede observar las tareas se activan periódicamente tras el *offset* inicial. Además, notifican sus instantes de activación y tiempos de respuesta relativos a su activación inicial de la siguiente manera:

```
Task A running   RT.Clock =  1.407423700
Task A finishing RT.Clock =  1.727690000
Task B running   RT.Clock =  2.007419300
Task B finishing RT.Clock =  2.627659000
Task A running   RT.Clock =  2.807433500
Task A finishing RT.Clock =  3.127548600
Task C running   RT.Clock =  3.607494500
Task B running   RT.Clock =  4.007529400
Task A running   RT.Clock =  4.207448200
Task A finishing RT.Clock =  4.527642200
Task B finishing RT.Clock =  4.968035400
Task C finishing RT.Clock =  5.408362500
```

Las desviaciones de los tiempos de activación y respuesta de las tareas se deben a la sobrecarga que introduce el sistema de ejecución y a otras actividades de las tareas cuyos tiempos de cómputo no se han tenido en cuenta.

3. Sistema de desarrollo cruzado para Ada

Una vez codificados el programa principal y el paquete tareas, para compilar, enlazar y montar se utilizará el sistema de compilación GNAT/ORK.

Este sistema de compilación permite desarrollar software en Ada utilizando las restricciones del perfil de Ravenscar en computadores i386 GNU/Linux y SPARC/Solaris. El código generado se ejecuta en computadores basados en el ERC32, que es un procesador resistente a la radiación desarrollado por la ESA (European Space Agency).

El sistema se encuentra instalado en batman en el directorio `/usr/local/openravenscar` y, para poder utilizarlo, el directorio `/usr/local/openravenscar/bin` debe de estar incluido en la variable de entorno `PATH`.

Por otra parte, como no se dispone de computadores reales se utilizará el simulador TSIM (http://www.gaisler.com) que se encuentra instalado en mismo directorio de batman.

Para compilar, enlazar y montar el programa se puede utilizar el siguiente `Makefile`, que se encuentra en batman en el directorio `/usr/local/openravenscar/SEU/ada`.

Makefile

```
#-----------------------------------------------------------------------
#                              EXAMPLES Makefile
#-----------------------------------------------------------------------

# the Main procedure                                                    5
MAIN = main

# the gnatmake
GNATMAKE = sparc-ork-gnatmake
                                                                        10
# the size binutil
SIZE = sparc-ork-size

# the nm binutil
NM = sparc-ork-nm                                                       15

# Gnat1 compilation flags
#GF = -g
# Do not use optimization for debugging
GF = -O2                                                                20

# Gnatbind flags
BF =

# Gnatlink flags                                                        25
LF = -k -specs ork_specs -mcpu=cypress
# Use the following to obtain a diagnostic link file
LF_LINK_FILE = -Xlinker -Map -Xlinker $(MAIN).map

#-----------------------------------------------------------------------
# Main rule

all :   $(MAIN).adb
        $(GNATMAKE) $(MAIN) -cargs $(GF) -bargs $(BF) -largs $(LF) $(LF_LINK_FILE)
        $(NM) $(MAIN) > $(MAIN).nm                                      35
        $(SIZE) $(MAIN)
```

```
clean : force
        @/bin/rm -f *.o *.nm *.ali b~*.* *.s *~ $(MAIN) *.map
                                                                    40
force :
```

Si se ordena:

`make`

La salida mostrada será:

```
sparc-ork-gnatmake main -cargs -O2 -bargs  -largs -k
-specs ork_specs -mcpu=cypress -Xlinker -Map -Xlinker main.map
sparc-ork-gcc -c -O2 main.adb
sparc-ork-gcc -c -O2 tasks.adb
sparc-ork-gcc -c -O2 workload.adb
sparc-ork-gnatbind -x main.ali
sparc-ork-gnatlink -k -specs ork_specs -mcpu=cypress -Xlinker
-Map -Xlinker main.map main.ali
sparc-ork-nm main > main.nm
sparc-ork-size main
   text    data     bss     dec     hex filename
 172368    5828 1926076 2104272  201bd0 main
```

En cuyo caso se puede proceder a ejecutar el programa. Para lo cual se carga en el simulador:

```
tsim main

 TSIM/ERC32 - remote SPARC simulator, version 1.0.2 (evaluation version)

 Copyright (C) 2001, Gaisler Research - all rights reserved.
 For latest updates, go to http://www.gaisler.com/
 Comments or bug-reports to tsim@gaisler.com

serial port A on stdin/stdout
allocated 4096 K RAM memory
allocated 2048 K ROM memory
section: .text at 0x2000000, size 172368 bytes
section: .data at 0x202a150, size 5828 bytes
```

y posteriormente se ejecuta ordenando `go`:

```
tsim>go
resuming at 0x02000000
Task A running   RT.Clock =  1.407423700
Task A finishing RT.Clock =  1.727690000
Task B running   RT.Clock =  2.007419300
```

Dicho programa, si no se producen errores, no acaba nunca. Para suspender su ejecución basta con teclear control-C, y para salir del simulador `quit`.

3.1. Depuración

Para depurar los programas, se simula una conexión entre el computador simulado y batman. Para ello, se arranca el TSIM en otro terminal con la opción `-gdb`:

```
xterm -exec tsim -gdb &
```

Aparecerá un terminal con el siguiente mensaje:

```
 TSIM/ERC32 - remote SPARC simulator, version 1.0.2 (evaluation version)

 Copyright (C) 2001, Gaisler Research - all rights reserved.
 For latest updates, go to http://www.gaisler.com/
 Comments or bug-reports to tsim@gaisler.com

serial port A on stdin/stdout
allocated 4096 K RAM memory
allocated 2048 K ROM memory
gdb interface: using port 1234
```

Este mensaje indica que el simulador está conectado a través del puerto 1234. Ahora podemos utilizar el depurador GDB personalizado para GNAT/ORK, decirle que se conecte a través del puerto 1234 del computador local, cargar el programa en el simulador y empezar la sesión de depuración como si se tratase del GDB convencional:

```
sparc-ork-gdb main
GNU gdb 4.17.gnat.3.13a
Copyright 1998 Free Software Foundation, Inc.
GDB is free software, covered by the GNU General Public License, and you
are welcome to change it and/or distribute copies of it under certain
conditions.
Type "show copying" to see the conditions.
There is absolutely no warranty for GDB.  Type "show warranty" for
details.
This GDB was configured as "--host=sparc-sun-solaris2.8
--target=sparc-ork-elf"...
(gdb) target extended-remote localhost:1234
Remote debugging using localhost:1234
0x0 in ?? ()
(gdb) load
Loading section .text, size 0x1f180 lma 0x2000000
Loading section .data, size 0x15c4 lma 0x201f180
Start address 0x2000000
Transfer rate: 354485 bits/sec.
(gdb) continue
Continuing.
```

4. Codificación en C/POSIX

En esta segunda parte de la práctica se utilizarán las llamadas POSIX y el lenguaje C para la implementación de las tareas periódicas. La codificación de las tareas en C/POSIX se muestra a continuación.

tasks.c

```
#include "tasks.h"

/* task A  */

void * task_a (void *arg)
{
    struct    timespec my_period;
    int       my_sig, received_sig;
    struct    itimerspec timerdata;
    timer_t   timer_id;
    struct    sigevent event;
    sigset_t  set;
    struct timespec time;

    my_period = ((struct periodic_params*) arg)->period;
    my_sig    = ((struct periodic_params*) arg)->signo;

    /* timer create */
    event.sigev_notify = SIGEV_SIGNAL;
    event.sigev_signo = my_sig;
    if (timer_create (CLOCK_REALTIME, &event, &timer_id) == -1) {
        perror ("Error in timer creation\n");
        pthread_exit ((void *) -1);
     }

    /* block the timer signal */
    sigemptyset (&set);
    sigaddset (&set,my_sig);
    pthread_sigmask(SIG_BLOCK,&set,NULL);

    /* set the timer in periodic mode */
    timerdata.it_interval = my_period;
    timerdata.it_value    = my_period;
    if (timer_settime(timer_id, 0, &timerdata, NULL) == -1) {
      perror ("Error in timer setting\n");
      pthread_exit ((void *) -1);
    }

    /* periodic activity */
    while(1) {
```

```
      if (sigwait(&set,&received_sig) == -1) {
        perror ("Error in sigwait\n");
        pthread_exit ((void *) -1);
      }
      clock_gettime (CLOCK_REALTIME, &time);
      printf("Task A starting %ld%ld\n",
               time.tv_sec - starttime.tv_sec,
               time.tv_nsec - starttime.tv_nsec);
      small_whetstone (300/MILLISEC_PER_WHETSTONES);
      clock_gettime (CLOCK_REALTIME, &time);
      printf("Task A finishing%ld%ld\n",
               time.tv_sec - starttime.tv_sec,
               time.tv_nsec - starttime.tv_nsec);
   }
 }

/* task B    */

void * task_b (void *arg)
{
    struct     timespec my_period;
    int        my_sig, received_sig;
    struct     itimerspec timerdata;
    timer_t    timer_id;
    struct     sigevent event;
    sigset_t   set;
    struct timespec time;

    my_period = ((struct periodic_params*) arg)->period;
    my_sig    = ((struct periodic_params*) arg)->signo;

    /* timer create */
    event.sigev_notify = SIGEV_SIGNAL;
    event.sigev_signo = my_sig;
    if (timer_create (CLOCK_REALTIME, &event, &timer_id) == -1) {
        perror ("Error in timer creation\n");
        pthread_exit ((void *) -1);
     }

    /* block the timer signal */
    sigemptyset (&set);
    sigaddset (&set,my_sig);
    pthread_sigmask(SIG_BLOCK,&set,NULL);

    /* set the timer in periodic mode */
    timerdata.it_interval = my_period;
    timerdata.it_value    = my_period;
    if (timer_settime(timer_id, 0, &timerdata, NULL) == -1) {
      perror ("Error in timer setting\n");
```

```
      pthread_exit ((void *) -1);
    }

    /* periodic activity */
    while(1) {
      if (sigwait(&set,&received_sig) == -1) {
        perror ("Error in sigwait\n");
        pthread_exit ((void *) -1);
      }
      clock_gettime (CLOCK_REALTIME, &time);
      printf("Task B starting %ld%ld\n",
               time.tv_sec - starttime.tv_sec,
               time.tv_nsec - starttime.tv_nsec);
      small_whetstone (600/MILLISEC_PER_WHETSTONES);
      clock_gettime (CLOCK_REALTIME, &time);
      printf("Task B finishing%ld%ld\n",
               time.tv_sec - starttime.tv_sec,
               time.tv_nsec - starttime.tv_nsec);
    }
}

/* task C */

void * task_c (void *arg)
{
    struct     timespec my_period;
    int        my_sig, received_sig;
    struct     itimerspec timerdata;
    timer_t    timer_id;
    struct     sigevent event;
    sigset_t set;
    struct timespec time;

    my_period = ((struct periodic_params*) arg)->period;
    my_sig    = ((struct periodic_params*) arg)->signo;

    /* timer create */
    event.sigev_notify = SIGEV_SIGNAL;
    event.sigev_signo = my_sig;
    if (timer_create (CLOCK_REALTIME, &event, &timer_id) == -1) {
        perror ("Error in timer creation\n");
        pthread_exit ((void *) -1);
     }

    /* block the timer signal */
    sigemptyset (&set);
    sigaddset (&set,my_sig);
    pthread_sigmask(SIG_BLOCK,&set,NULL);
```

```
    /* set the timer in periodic mode */
    timerdata.it_interval = my_period;
    timerdata.it_value    = my_period;
    if (timer_settime(timer_id, 0, &timerdata, NULL) == -1) {
      perror ("Error in timer setting\n");
      pthread_exit ((void *) -1);
    }

    /* periodic activity */
    while(1) {
      if (sigwait(&set,&received_sig) == -1) {
        perror ("Error in sigwait\n");
        pthread_exit ((void *) -1);
      }
      clock_gettime (CLOCK_REALTIME, &time);
      printf("Task C starting %ld%ld\n",
             time.tv_sec - starttime.tv_sec,
             time.tv_nsec - starttime.tv_nsec);
      small_whetstone (800/MILLISEC_PER_WHETSTONES);
      clock_gettime (CLOCK_REALTIME, &time);
      printf("Task C finishing%ld%ld\n",
             time.tv_sec - starttime.tv_sec,
             time.tv_nsec - starttime.tv_nsec);
    }
  }
```

---

tasks.h

```
#include "system.h"
#include "workload.h"


#include <pthread.h>   /* thread facilities */
#include <signal.h>    /* signal facilities */
#include <unistd.h>    /* sleep facilities */
#include <sched.h>     /* schedule facilities */
#include <time.h>      /* time facilities */
#include <stdio.h>     /* console facilities */

#define MILLISEC_PER_WHETSTONES 27.7

/* temporal parameters of a task */

struct periodic_params {
    struct timespec period;
    int signo;          /* signal number */
```

```
    int id;                 /* task identification */
};

struct timespec starttime;

/* task A    */

void *task_a (void *arg);

/* task B    */

void *task_b (void *arg);

/* task C */

void *task_c (void *arg);
```

En este caso las tareas periódicas se activan utilizando los temporizadores de POSIX, aunque existen nuevos mecanismos que permiten realizar un retardo en tiempo absoluto más fáciles de manejar.

Por último el programa principal tiene que crear e inicializar los tres threads. Debe tener por lo tanto la siguiente estructura:

main.c

```
#define CONFIGURE_INIT
#include "tasks.h"

void *POSIX_Init (
  void *argument
)

{
    pthread_attr_t attr;                    /* task attributes */
    pthread_t ta,tb,tc;                   /* threads */
    struct sched_param sch_param;           /* schedule parameters */
    struct periodic_params params_a, params_b, params_c;

    clock_gettime (CLOCK_REALTIME, &starttime);

    /* init task attributes */
    if (pthread_attr_init(&attr) != 0) {
      perror ("Error in attribute init\n");
    }

    /* set explicit schedule for every task */
    if (pthread_attr_setinheritsched (&attr,
```

```
    PTHREAD_EXPLICIT_SCHED) != 0) {
     perror("Error in attribute inheritsched\n");
  }

  /* set task independent (join will not use) */
  if (pthread_attr_setdetachstate (&attr,
    PTHREAD_CREATE_DETACHED) != 0) {
     perror ("Error in attribute detachstate\n");
  }

  /* schedule policy POSIX_FIFO (priority preemtive and FIFO within the same
     priority) */
  if (pthread_attr_setschedpolicy (&attr, SCHED_FIFO) != 0) {
     perror ("Error in attribute setschedpolicy\n");
   }

  /* set and create thread Interrupt with priority 31 */

  sch_param.sched_priority = 31;
  if (pthread_attr_setschedparam(&attr, &sch_param) != 0) {
     perror ("Error in attribute schedparam\n");
   }

  /* Temporal parameters (1'4 sec. periodicity) */

  params_a.period.tv_sec  = 1;          /* seconds */
  params_a.period.tv_nsec = 400000000; /* nanoseconds */
  params_a.signo = SIGALRM;
  if (pthread_create (&ta, &attr, task_a, &params_a) != 0 ) {
    perror ("Error in thread create for task a\n");
}

  /* set and create thread B with priority 15 */

  sch_param.sched_priority = 15;
  if (pthread_attr_setschedparam(&attr, &sch_param) != 0) {
     perror ("Error in attribute schedparam");
   }

  /* Temporal parameters (2 sec. periodicity) */
  params_b.period.tv_sec  = 2;          /* seconds */
  params_b.period.tv_nsec = 000000000; /* nanoseconds */
  params_b.signo = SIGALRM;
  if (pthread_create (&tb, &attr, task_b, &params_b) != 0) {
    perror ("Error in thread create for task b\n");
  }

  /* set and create thread C with priority 14 */
```

```
    sch_param.sched_priority = 14;
    if (pthread_attr_setschedparam(&attr, &sch_param) != 0 ) {
        perror ("Error in attribute schedparam\n");
     }

    /* Temporal parameters (3'6 sec. periodicity) */
    params_c.period.tv_sec      = 3;          /* seconds */
    params_c.period.tv_nsec = 600000000; /* nanoseconds */
    params_c.signo = SIGALRM;
    if (pthread_create (&tc, &attr, task_c, &params_c) != 0) {
      perror ("Error in trhead create for task c\n");
    }

    while (TRUE) {};

}
```

Nótese que no se puede simular el tiempo de cómputo de las tareas mediante una llamada `sleep`, ya que la ejecución de ésta supone el desalojo de la tarea de la CPU. Por lo tanto, para que cada tarea consuma el tiempo de cómputo correspondiente de CPU y hay que ejecutar un procedimiento cuyo tiempo de cómputo sea equivalente.

El tiempo de cómputo de un whetstone en la versión C puede ser diferente de la versión Ada y, por lo tanto, se deberá volver a medir dicho tiempo de cómputo. El programa `measure.c`, que se muestra a continuación, realiza dicha medida y, además, sirve para ilustrar el manejo de fechas en POSIX.

measure.c

```
#define CONFIGURE_INIT
#include "system.h"
#include "workload.h"

void *POSIX_Init (void *argument)
{
      struct timespec timebefore, timeafter;
      long elapsed_nsec, elapsed_msec;
      time_t elapsed_sec;

      clock_gettime (CLOCK_REALTIME, &timebefore);
      small_whetstone (100);
      clock_gettime (CLOCK_REALTIME, &timeafter);
      elapsed_nsec = timeafter.tv_nsec - timebefore.tv_nsec;
      elapsed_sec = timeafter.tv_sec - timebefore.tv_sec;
      elapsed_msec = elapsed_nsec/1000000 + (elapsed_sec * 1000);
      printf("Tiempo por whetstones en milisegundos %ld\n", elapsed_msec/100);
      exit (0);
```

```
}
```

Como en la versión Ada, las tareas notifican sus instantes de activación y tiempos de respuesta relativos a su activación inicial de la siguiente manera:

```
Task A starting  1 400000000
Task A finishing 1 680000000
Task B starting  2 0
Task B finishing 2 580000000
Task A starting  2 800000000
Task A finishing 3 80000000
Task C starting  3 600000000
Task B starting  4 0
Task A starting  4 200000000
Task A finishing 4 480000000
Task B finishing 4 870000000
Task C finishing 5 250000000
```

Nótese, que en este caso, los tiempos resultantes no contienen tantas cifras significativas porque la resolución del reloj de RTEMS es mucho menor que la de GNAT/ORK. El material de apoyo para la realización de esta práctica se puede encontrar en batman en el directorio `/usr/local/openravenscar/SEU/posix`.

5. Sistema de desarrollo cruzado C/POSIX

Para esta parte de la práctica se utilizará el interfaz POSIX del sistema operativo RTEMS (http://www.rtems.com). RTEMS es un sistema operativo de tiempo real que proporciona una interfaz POSIX muy completa y que está disponible para una gran cantidad de plataformas de ejecución.

Entre estas plataformas se encuentran los computadores basados en el ERC32. Por lo tanto, los ejecutables se pueden ejecutar y depurar como los generados con GNAT/ORK.

El sistema operativo RTEMS y sus herramientas asociadas, se encuentra instalado en batman en el directorio `/opt/rtems`. Para poder utilizarlo, el directorio `/opt/rtems/bin` debe de estar incluido en la variable de entorno `PATH`. Además, se debe definir la variable de entorno `RTEMS_MAKEFILE_PATH` como `/opt/rtems/erc32`.

A diferencia de un sistema operativo convencional, para el sistema de compilación de RTEMS es necesario que se ubique el fichero de configuración (`system.h`) en el directorio de trabajo. En este fichero se configuran los recursos necesarios para la correcta ejecución de la aplicación, tales como número de *threads*, número de *mutex*, etc.

El fichero de configuración necesario para esta parte de la práctica es el siguiente:

system.h

```
/*  system.h
 *
 *  This include file contains information that is included in every
 *  function in the test set.
 *
 *  The license and distribution terms for this file may be
 *  found in the file LICENSE in this distribution or at
 *  http://www.OARcorp.com/rtems/license.html.
 *
 *  system.h,v 1.9 1996/08/09 18:48:33 joel Exp
 */

/* functions */

#include <tmacros.h>
#include <unistd.h>
#include <errno.h>
#include <sched.h>

void *POSIX_Init (
  void *arg
);

void *task_a(
  void *arg
);

void *task_b(
  void *arg
);

void *task_c(
  void *arg
);

/* configuration information */

#define CONFIGURE_APPLICATION_NEEDS_CONSOLE_DRIVER
#define CONFIGURE_APPLICATION_NEEDS_CLOCK_DRIVER
/* #define CONFIGURE_MICROSECONDS_PER_TICK 1000 */

#define CONFIGURE_POSIX_INIT_THREAD_TABLE
```

```
#define CONFIGURE_MAXIMUM_POSIX_THREADS    4
#define CONFIGURE_MAXIMUM_POSIX_TIMERS     3
#define CONFIGURE_MAXIMUM_TIMERS           3

#include <confdefs.h>

/* global variables */

TEST_EXTERN pthread_t       Init_id;
TEST_EXTERN pthread_t       Task_id;

/* end of include file */
```

Este fichero junto con el Makefile necesario, se puede encontrar en batman en el directorio `/usr/local/openravenscar/SEU/posix`.

De este modo, para compilar y montar tan solo hay que ordenar `make` en el directorio de trabajo y se mostrará la salida siguiente:

```
test -d o-optimize || mkdir o-optimize
/opt/rtems/bin/sparc-rtems-gcc --pipe -B/opt/rtems//erc32/lib/
-specs bsp_specs -qrtems -g -Wall -ansi -fasm  -O4 -mcpu=cypress
-c   -o o-optimize/main.o main.c
/opt/rtems/bin/sparc-rtems-gcc --pipe -B/opt/rtems//erc32/lib/
-specs bsp_specs -qrtems -g -Wall -ansi -fasm  -O4 -mcpu=cypress
-c   -o o-optimize/tasks.o tasks.c
/opt/rtems/bin/sparc-rtems-gcc --pipe -B/opt/rtems//erc32/lib/
-specs bsp_specs -qrtems -g -Wall -ansi -fasm  -O4 -mcpu=cypress
-c   -o o-optimize/workload.o workload.c
/opt/rtems/bin/sparc-rtems-gcc --pipe -B/opt/rtems//erc32/lib/
-specs bsp_specs -qrtems -g -Wall -ansi -fasm  -O4 -mcpu=cypress
-L /opt/rtems//erc32/lib    -o o-optimize/main.exe  o-optimize/main.o
o-optimize/tasks.o o-optimize/workload.o    -lm
/opt/rtems/sparc-rtems/bin/nm -g -n o-optimize/main.exe > o-optimize/main.num
/opt/rtems/bin/sparc-rtems-size o-optimize/main.exe
   text    data     bss     dec     hex filename
 121504    2240   22592  146336   23ba0 o-optimize/main.exe
```

Se crea un subdirectorio llamado `o-optimize` en el que se encuentra el ejecutable (`ciclico.exe`). Este ejecutable se puede cargar, ejecutar y depurar con el simulador de la misma forma que los ejecutables generados con GNAT/ORK:

```
tsim o-optimize/main.exe

 TSIM/ERC32 - remote SPARC simulator, version 1.0.2 (evaluation version)

 Copyright (C) 2001, Gaisler Research - all rights reserved.
 For latest updates, go to http://www.gaisler.com/
 Comments or bug-reports to tsim@gaisler.com
```

```
serial port A on stdin/stdout
allocated 4096 K RAM memory
allocated 2048 K ROM memory
section: .text at 0x2000000, size 121504 bytes
section: .data at 0x201daa0, size 2240 bytes
tsim>
```

y posteriormente se ejecuta ordenando `go`:

```
tsim> go
resuming at 0x02000000
Task A starting  1 400000000
Task A finishing 1 680000000
```

6. Trabajo práctico

El trabajo práctico consiste en:

1. Compilar montar y ejecutar estos dos programas. Para ello se recomienda copiar el directorio `/usr/local/openravenscar/SEU` en el directorio raíz de cada cuenta y seguir las instrucciones.

2. Entregar una pequeña memoria de unas dos páginas sobre el trabajo realizado con un pequeño análisis de los pros y los contras de un sistema de desarrollo cruzado frente a uno convencional y de los dos sistemas cruzados ejercitados en esta práctica.