

Práctica 6

Planificación

Esta práctica se compone de tres partes:

1. medida del tiempo de cómputo de un conjunto de procedimientos,
2. cálculo de la planificabilidad de un conjunto de tareas independientes que utilizan esos procedimientos y
3. construcción de un programa de tiempo real crítico donde se compruebe que el conjunto de tareas es realmente planificable.

6.1 Tareas a realizar

6.1.1 Medida de tiempos de cómputo

En el paquete Proc hay cuatro procedimientos cuyos tiempos de cómputo hay que medir. La especificación de este paquete es:

Fichero 6.1: Especificación del paquete Proc (Proc.ads)

```
1 package Proc is
2   procedure P1;
3   procedure P2;
4   procedure P3;
5   procedure P4;
6 end Proc;
```

Los procedimientos de este paquete los escribirá el alumno y deben simular procedimientos de cálculo con un tiempo de cómputo cada uno de:

| Procedimiento | C (ms) |
|---------------|--------|
| P1 | 400 |
| P2 | 600 |
| P3 | 800 |
| P4 | 800 |

Estos tiempos se pueden simular con bucles de espera activa. Para realizar estas medidas el alumno escribirá un paquete de nombre Plan que exporte la operación Medir acorde con las declaraciones siguientes:

```
type ref_Procedimiento_t is access procedure;
type array_ref_Procedimiento_t is array (Positive range <>)
    of ref_Procedimiento_t;
type array_Tiempos_t is array (Positive range <>) of Natural;
procedure Medir (Procedimientos: array_ref_Procedimiento_t;
    Tiempos : out array_Tiempos_t);
```

La operación **Medir** recibe un array de punteros a procedimientos y devuelve un array con los tiempos medidos. Los tiempos se expresan en *ms* y se representan como números naturales.

La única novedad sintáctica que aparece en estas declaraciones es la línea

```
type ref_Procedimiento_t is access procedure;
```

Con esta línea se está declarando que **ref_Procedimiento** es un tipo puntero para almacenar la dirección de un procedimiento sin parámetros. La forma de usar este procedimiento se muestra a continuación:

```
declare
  procedure P is ... begin ... end P;
  procedure Q is ... begin ... end Q;
  ptr_Proc: ref_Procedimiento_t := P'Access;
  -- Access es un atributo que representa la dirección del
  -- procedimiento "P"
begin
  P; -- Esta es una forma de ejecutar el procedimiento "P"
  ptr_Proc.all; -- Esta es otra forma de ejecutar el procedimiento "P"
  ptr_Proc := Q'Access; -- "ptr_Proc" apunta al procedimiento "Q"
  ptr_Proc.all; -- Ejecuta el procedimiento "Q"
end;
```

A continuación podemos ver una sección de código que muestra cómo usar la operación **Medir**:

```
Procedimientos: array_ref_Procedimiento_t := (P1'Access, P2'Access,
                                              P3'Access, P4'Access);
Tiempos: array_Tiempos_t (Procedimientos'Range);
...
Medir (Procedimientos, Tiempos);
Put_line ("-----+");
Put_Line ("| Procedimiento T.computo |");
Put_line ("|-----|");
for I in Tiempos'Range loop
  Put ("| "); Put (I, Width=>13); Put (" ");
  Put (Tiempos(I), Width=> 9); Put (" |");
  New_Line;
end loop;
Put_line ("-----+");
```

6.1.2 Construcción de un planificador

El paquete **Plan** debe completarse con un procedimiento de nombre **Planificar** que sirva para: asignar prioridades a las tareas según el criterio *DMS* (*deadline monotonic scheduling*), calcular los tiempos de respuesta de cada tarea y determinar si las tareas son planificables o no. En "Unix:Programación Avanzada (Francisco M. Márquez) 3ª Ed. Ra-Ma" [págs. 173–181] podemos encontrar un resumen de la teoría de planificación de sistemas con requisitos de tiempo real y un ejemplo de implementación en lenguaje C de la operación **Planificar**. Las declaraciones para construir esta operación en lenguaje Ada son:

```
type reg_Planificacion_t is record
  Nombre : Positive; -- Número de la tarea
  T : Natural; -- Período
  D : Natural; -- Plazo
  C : Natural; -- Tiempo de cómputo
  P : Positive; -- Prioridad
  R : Natural; -- Tiempo de respuesta
  Planificable: Boolean;
```

```

end record;
type array_reg_Planificacion_t is array (Positive range <>)
    of reg_Planificacion_t;
procedure Planificar (Tareas: in out array_reg_Planificacion_t);

```

El array **Tareas** contiene un registro **reg_Planificacion_t** por cada tarea. Los campos **Nombre**, **T**, **D** y **C** son datos para el algoritmo de planificación; los campos **P**, **R** y **Planificable** los calculará el algoritmo.

Recordamos que los pasos necesarios para calcular estos campos son:

1. Ordenar la tareas según el plazo de respuesta D y asignar las prioridades según el criterio DMS ($D_i < D_j \rightarrow P_i > P_j$).
2. Calcular los tiempos de respuesta con la expresión:

$$R_i = C_i + \sum_{j \in pM(i)} \left\lceil \frac{R_i}{T_j} \right\rceil \cdot C_j \quad (6.1)$$

3. La tarea τ_i será planificable si $R_i \leq D_i$

Con el procedimiento siguiente se puede comprobar el correcto funcionamiento de la operación **Planificar**:

Programa 6.2: Prueba del procedimiento **Planificar** (Prueba.adb)

```

1 with Ada.Text_IO, Plan;
2 use Ada.Text_IO, Plan;
3 procedure Prueba is
4     package Integer_Es is new Integer_IO (Integer);
5     use Integer_Es;
6     Tareas: array_reg_Planificacion_t := (
7         -- -----
8         -- Tarea T D C P R Planificable
9         -- -----
10        ( 1, 20, 20, 3, 1, 0, False ),
11        ( 2, 20, 5, 3, 1, 0, False ),
12        ( 3, 15, 7, 3, 1, 0, False ),
13        ( 4, 10, 10, 4, 1, 0, False )
14        -- -----
15        );
16 begin
17     Planificar (Tareas);
18     Put_line (
19         "+-----+");
20     Put_Line (
21         "| Tarea T D C P R Planificable |");
22     Put_line (
23         "|-----|");
24     for I in Tareas'Range loop
25         Put ("| ");
26         Put (Tareas(I).Nombre, Width=>5); Put (" ");
27         Put (Tareas(I).T, Width=>4); Put (" ");
28         Put (Tareas(I).D, Width=>4); Put (" ");
29         Put (Tareas(I).C, Width=>4); Put (" ");
30         Put (Tareas(I).P, Width=>4); Put (" ");
31         Put (Tareas(I).R, Width=>4); Put (" ");
32         if Tareas(I).Planificable then

```

```

33     Put_Line (" SI |");
34   else
35     Put_Line (" NO |");
36   end if;
37 end loop;
38 Put_line (
39   "+-----+");
40 end Prueba;

```

El resultado que se obtiene tras ejecutar este programa es:

```

+-----+
| Tarea      T      D      C      P      R      Planificable |
+-----+
|      1     20     20      3      1     20              SI |
|      4     10     10      4      2     10              SI |
|      3     15      7      3      3      6              SI |
|      2     20      5      3      4      3              SI |
+-----+

```

Utilizando los procedimientos **Medir** y **Planificar** del paquete **Plan**, escribir un procedimiento de nombre **Medir** que mida los tiempos de cómputo de los procedimientos que hay en el paquete **Proc** y construya una tabla de planificación como la anterior. El programa imprimirá un resultado parecido al siguiente:

```

+-----+
| Procedimiento  T.computo |
+-----+
|           1           ? |
|           2           ? |
|           3           ? |
|           4           ? |
+-----+
+-----+
| Tarea      T      D      C      P      R      Planificable |
+-----+
|      1    2400     600      ?      ?      ?              ? |
|      2    3200    1200      ?      ?      ?              ? |
|      3    3600    2000      ?      ?      ?              ? |
|      4    4000    3200      ?      ?      ?              ? |
+-----+

```

Los campos que aparecen marcados con ? tienen que ser calculados con los procedimientos del paquete **Plan**.

6.1.3 Construcción de un simulador

Escribir un programa de nombre **Simular** donde se definan 4 tareas periódicas con los atributos temporales obtenidos en las tablas anteriores. La forma de definir estas tareas puede ser la siguiente:

```

type ref_Procedimiento_t is access procedure;
task type Tarea_t (Nombre : Natural;
  T : Natural;
  D : Natural;
  C : Natural;
  P : Natural;
  Codigo_Ciclico: ref_Procedimiento) is

```

```
    pragma Priority (P);  
end Tarea_t;  
Tarea1: Tarea_t (1, 2400, 600, ?, ?, P1'Access);  
Tarea2: Tarea_t (2, 3200, 1200, ?, ?, P2'Access);  
Tarea3: Tarea_t (3, 3600, 2000, ?, ?, P3'Access);  
Tarea4: Tarea_t (4, 4000, 3200, ?, ?, P4'Access);
```

Los campos marcador con ? se inicializan con los resultados del §6.1.2 pág. 2. Cada una de las tareas anteriores comprobará que no se supera su plazo de respuesta, en caso contrario se emitirá un mensaje indicándolo. La simulación durará 50 segundos.