Selección de algunos Ejercicios aparecidos en los exámenes de Sistemas en Tiempo Real de las Escuelas de Ingeniería técnica Informática de Oviedo y Gijón

Ejercicio

a). Con unas buenas técnicas de prevención de fallos, ¿son necesarias las técnicas de tolerancia de fallos? Justificar la respuesta.
b). Explicar brevemente la diferencia entre recuperación de fallos directa e inversa

Ejercicio

Se tiene el siguiente conjunto de tareas con las operaciones que realizan:

TAREA	TIPO	PERIODO	TIEMPO EJECUCIÓN	PLAZO
A	Periódica	20	10	20
B	Periódica	50	10	50
C	Periódica	40	15	40
D	Esporádica	200	10	15

Además hay dos objetos protegidos, X e Y. Las duraciones máximas de las operaciones protegidas que ejecutan las tareas con estos objetos son las siguientes:

TAREA	X	Y
A	1	1
B	-	2
C	-	-
D	1	-

Las tareas se ejecutan con planificación con prioridades fijas y desalojo, y los accesos a los objetos compartidos se efectúan mediante el protocolo del techo de prioridad inmediato. Se pide:

a). Asignar prioridades a las tareas y objetos protegidos.
b). Calcular los tiempos de bloqueo máximo de las tareas.
c). Calcular el tiempo de respuesta máximo de la tarea C.

Ejercicio

Tres procesos P, Q y S, tienen las siguientes características:
P. Periodo:3 Tiempo de ejecución máximo: 1
Q. Periodo:6 Tiempo de ejecución máximo: 2
S. Periodo:18 Tiempo de ejecución máximo: 5

Suponiendo el modelo simple, y un método de planificación con prioridades y desalojo (preemptive),
a). Utilizar un algoritmo de planificación y analizar la planificabilidad.
b). Dibujar el cronograma del comportamiento temporal.

Ejercicio

El siguiente texto es la especificación de un tipo protegido Mensaje. Una tarea puede enviar a otra un mensaje consistente en un solo valor entero mediante un objeto de este tipo. La tarea receptora se bloquea siempre al hacer Get y se desbloquea cuando la tarea emisora hace un Send.

```
protected type Mensaje is
   entry Get (Item: out Integer);
   procedure Send (Item: in Integer);
private
   Valor: Integer;
   Recibido: Boolean := false;
end Mensaje;
```

a). Escribir una posible realización del cuerpo del objeto.
b). Escribir un ejemplo de una tarea emisora periódica que almacene un valor cada 15 segundos.

Ejercicio

Definir una clase PERSONA que tenga como estructura interna: SEXO (HOMBRE,MUJER), NOMBRE y FECHA_NAC, y como métodos las siguientes funciones y procedures:
Funciones
QUE_NOMBRE(PERSONA) retorna NOMBRE,
QUE_GENERO(PERSONA) retorna SEXO,
QUE_FECHA(PERSONA) retorna FECHA_NAC.
Procedures
DEFINE_PERSONA(PERSONA),
PUT(PERSONA)

Asegurar el encapsulamiento y la posibilidad de herencia de los objetos. No es necesario incluir la implementación de los métodos.

a). Definir una clase EMPLEADO descendiente de PERSONA que añada los datos FECHA_CONTRATO, NUM_REGISTRO y añada o modifique los métodos PUT, DEFINE_EMPLEADO, QUE_FECHA_CON, QUE_NUM_REG.
b). Escribir la implementación de los métodos PUT y DEFINE_EMPLEADO de la clase EMPLEADO, aprovechando los métodos heredados.

Ejercicio

Mediante un menú por pantalla se nos insta a escoger entre las opciones del 1 al 6. Diseñar un fragmento de código que empleando excepciones, nos pida repetir la selección si la opción elegida no está en ese rango, y que siga adelante en caso contrario.

Realizad un subprograma para ir añadiendo componentes a una lista enlazada de enteros. Solo será preciso conocer donde se coloca el último elemento ya que la lista se recorrerá de manera LIFO. Realizad otro subprograma para que la lista sea FIFO, aquí se puede tener un puntero a cabeza y otro a cola.

Ejercicio

Realizad un fragmento de código con una función que, solo con vectores y sin sentencias *if* ni *case,* devuelva el número de días del mes.

Ejercicio

Se desea realizar el producto de una matriz N*M de floats por una matriz columna M*1. El resultado será un vector columna N*1. Realizad dicho programa de manera que cada componente del vector se calcule *en paralelo* y que los índices N y M puedan tomar cualquier valor positivo.

Para ahorrar trabajo se supone que contamos con un procedure Leer_Matriz a la que se le facilita donde queramos que nos sea entregada la matriz de floats. De la misma manera contamos con otro procedimiento Leer_Vector.

Indicar cuales serían las especificaciones de tales procedimientos, mostrando los tipos de argumentos que manejan.

Ejercicio

Sea un conjunto de tareas N constante que deben depositar un dato de tipo Item en un array compartido de N elementos. Cada depósito debe hacerse de manera exclusiva. Cada tarea cuando deposita el dato indica el elemento del array donde debe dejarlo, se supone que cada tarea lo deja en un sitio diferente por lo que no habrá conflictos. Se pretende que una vez que cada tarea deposite su dato se detenga. La tarea estará suspendida hasta que todas las restantes hayan depositado su elemento en el array. A partir de ese instante cada tarea se podrá desbloquear y podrá seguir con su ejecución. Esta secuencia de accesos podría repetirse varias veces durante el programa.

Hacer un fragmento de código donde se muestren los accesos de las tareas al recurso y como se gestiona éste.

Sugerencia: Después de depositar el dato desde cada tarea hacer una llamada *wait* (que deberíais implementar) para que en el caso de que sea la última despierte a todas las demás para que puedan continuar con la ejecución.
Aviso: no es preciso que sea una unidad genérica, se puede considerar que N e item son conocidos y declarados en algún lugar previo.

Ejercicio

Sea una tarea servidor que en su cuerpo tiene el siguiente fragmento de código:

```
loop
   select
accept A do
               Accion_A ;          -- ejecución de un procedure
end A;
or
when E => accept B do
               Accion_B ;          -- ejecución de un procedure
end B;
or delay 3.0
               Otra_Cosa;          -- ejecución de un procedure
   end select;
end loop;
```

Analizar que sucede con la tarea Servidor, cual será la secuencia de operaciones que ejecutará, en los siguientes casos.

Servidor llega al select y se encuentra con la expresión E cierta, las tareas T1, T2, T3 están esperando en la entrada A y la tarea T4 está en la entrada B.
Servidor llega al select y se encuentra con la expresión E falsa, pasado 1 s llega T3 a la entrada B.
c). Servidor llega al select y se encuentra con la expresión E falsa, T2 está esperando en la entrada B.

Pasados 2 s E se convierte en cierto, pasados 2.5 s llega la tarea T3 a la cita en A.
Servidor llega al select y se encuentra con la expresión E falsa, pasado 1 s E se vuelve cierta, pasados 2 s llega T1 a citarse en B, pasados 2.2 s llega T3 para citarse en A.

Ejercicio

Se tiene el siguiente conjunto de tareas con las operaciones que realizan:

TAREA	PRIORIDAD	T. LANZAMIENTO	OPERACIONES
T1	1	0	VVCQQQC
T2	2	1	CVVCC
T3	3	1	CCVVC
T4	4	2	CQQC
T5	5	4	CVVQC

Donde C indica la ejecución del programa durante una unidad de tiempo empleando solamente la CPU, mientras que V y Q señalan el uso exclusivo de un recurso compartido durante una unidad de tiempo.
Dibujar el cronograma correspondiente y calcular los tiempos de respuesta de cada tarea bajo los siguientes supuestos:

No hay herencia de prioridades
Se emplea el protocolo de herencia ICPP (ceiling inmediato).

Se empleará el siguiente convenio:

C	Q	V		Bloq_Q	Bloq_V
Ejecutando en C	Ejecutando Q	Ejecutando V	Pasada por alto	Bloqueada en Q	Bloqueada en V

Ejercicio

En el siguiente código hay 2 sentencias precedidas de comentarios. Indicar justificadamente cual será el resultado de la ejecución de ese programa para los 4 casos posibles de combinaciones de comentar o dejar sin comentar esas líneas.

```
with text_io; use text_io;

procedure Prueba is
Error: exception;

task servidor1 is
  entry peticion(dato : out integer);
end servidor1;
```

```
task body servidor1 is
-- i: Positive := -1;   -- LINEA 1  bit 0, Comentada=0
begin
Put_Line("El servidor1 ha arrancado");
select
accept peticion(dato: out integer) do
    dato:=1;
    raise ERROR;
end peticion;
 or
   delay 0.3;
    null;
end select;
put_line("fin de la tarea servidor1");
exception
  when Tasking_Error =>
    put_line("Servidor1: Tarea concluida o mal inicializada");
end servidor1;


j: integer;
begin
--  delay 2.0;  -- LINEA 2 bit 1, Comentada=0
  put_line("dentro de Prueba");
  servidor1.peticion(j);
exception
  when Tasking_Error =>
     put_line("Prueba: Tarea concluida o mal inicializada");
  when others => put_line("Prueba: Excepcion manejada");
end Prueba;
```

Solución

Hay dos tareas, servidor1 y prueba, que se citan en peticion. La línea 1, si se elimina el comentario (1) elevará una excepción del tipo TASKING_ERROR por producirse durante la inicialización de una tarea. La tarea servidor1 en ese caso ni siquiera comenzará a ejecutarse. La excepción se propagará a la tarea padre. Que ejecutará el manejador correspondiente y se acabará. Prueba solo escribirá el mensaje del manejador, no ejecutará nada más, ya que no comienza su ejecución hasta que acaben de activarse sus tareas dependientes.

El segundo comentario está en la tarea Prueba y enmascara un delay 2.0. Esta sentencia en si no genera excepción alguna, pero dependiendo del retardo el resultado sobre la cita será distinto. Así, si no hay retardo, Prueba se cita inmediatamente con Servidor1, se ejecuta el cuerpo del accept y dentro de él se eleva ERROR. El accept no contiene manejador, por estar la excepción en una cita se propaga en el receptor y en el emisor. En el receptor no hay manejador para ERROR en ninguna parte y por tanto finaliza sin acabar de ejecutar todas sus sentencias. Dentro del emisor hay un manejador destinado a others que puede tratar de manera anónima a ERROR.

Si se suprime el comentario del delay la tarea prueba esperará 2 s antes de ir a la cita, mientras que Servidor1 solo está dispuesta a esperar 0.3. Por tanto pasados 0.3 s la tarea servidor1 finaliza, esto implica que cuando Prueba acuda a la cita pretenderá citarse con una tarea ya terminada, por tanto nuevamente se eleva TASKING_ERROR en Prueba. La combinación 10 será la única que permita a Servidor1 ejecutar todas sus sentencias y escribir por pantalla el mensaje de finalización

La salida para cada uno de las 4 combinaciones es:

00
El servidor1 ha arrancado
dentro de prueba
Prueba: Excepcion manejada

01
Prueba: Tarea concluida o mal inicializada

10
El servidor1 ha arrancado
fin de la tarea servidor1
dentro de prueba
Prueba: Tarea concluida o mal inicializada

11
Prueba: Tarea concluida o mal inicializada

Ejercicio

Se desea crear un paquete para que las tareas se puedan asignar recursos de manera exclusiva. Para ello se cuenta con un procedimiento:

Procedure Asignacion(Recurso : Tipo_Recurso; Cantidad: Rango_Recursos);

Al cual se le indica qué recurso se desea y en que cantidad. El problema es que este procedimiento no es capaz de comprobar si los recursos están disponibles en la cantidad deseada.

Programar un paquete con lo necesario para que una tarea se pueda asignar y liberar tantos recursos como precise. Si los recursos no están disponibles en cantidad suficiente la tarea debe bloquearse. Cuando acabe de usar los recursos, en la cantidad que sea, debe indicarlo de alguna manera.

Nota, si pretendéis emplear objetos protegidos sabed que en una barrera no puede aparecer un parámetro de llamada a la entry.

Solución

Una posible solución es la presentada a continuación.

```
Package Recursos is
type Rango_Recursos is range 1....Max;
Procedure Asignacion(Recurso : Tipo_Recurso; Cantidad: Rango_Recursos);
end Recursos;

Package body Recursos is

protected Controlador_Recursos is
   entry Solicitar (R: out Recurso; Cantidad : Rango_Recursos);
   procedure Liberar (R : Recurso ; Cantidad : Rango_Recursos);
private
   entry Asignar (R: out Recurso; Cantidad : Rango_Recursos);
```

```
   Liberados : Rango_Solicitud := Rango_Solicitud ´Last;
   Nuevos_Recursos_Liberados : Boolean := False;
   Intentar : Natural := 0;
end Controlador_Recusro;


protected body Controlador_Recurso is
 entry Solicitar (R:out Recurso;Cantidad :Rango_Recursos)  when Liberados > 0 is
begin
   if Cantidad <= Liberados then
      Liberados := Liberados – Cantidad;  -- Asignar
   else
       requeue Asignar;
   end if;
end Solicitar;

entry Asignar (R : out Recurso; Cantidad : Rango__Recursos)
     when New_Recurso_Released is
begin
   Intentar := Intentar – 1;
   If Intentar = 0 then
       Nuevos_Recursos_Liberados := False;
   end if;
   if Cantidad < = Liberados then
       Liberados := Liberados – Cantidad; -- asignar
     else
       requeue Asignar;
     end if;
end Asignar

procedure Liberar(R: Recurso; Cantidad : Rango_Recursos) is
begin
   Liberados := Liberados + Cantidad;     -- liberar recursos
   if Asignar ´Count  > 0 then
      Intentar := Asignar'Count;
      Nuevos_Recursos_Liberados := True;
       end if;
    end Liberar;
end Controlador_Recursos;

procedure Asignacion(Recurso : Tipo_Recurso; Cantidad: Rango_Recursos) is
begin
          Controlador_Recursos.Solicitar(Recurso,Cantidad)
end;

procedure Liberacion (R : Recurso ; Cantidad : Rango_Recursos);
begin
          Controlador_Recursos.Liberación(Recurso,Cantidad)
end;

end Recursos;
```

La esencia del problema estriba en que si quiero J recursos y tengo disponibles K no puedo evaluar en la barrera de una entry si J < K. Si fuera así podría asignar los recursos, en caso contrario la tarea debe bloquearse. Si se bloquea debe permanecer así hasta que los recursos disponibles sean >= que J. Vayamos por partes.
En primer lugar la entry Solicitar examina en su interior si Cantidad <= Liberados. De ser así se asignan los recursos y no hay ningún problema. En caso contrario hace una llamada a la entry Asignar del mismo objeto protegido. La barrera asociada es una variable booleana Nuevos_Recursos_Liberados, que será cierta cuando

alguna tarea libere recursos y ejecute el procedure Liberar. Antes de analizar esta entry veamos el procedimiento Liberar. En él se incrementa la cantidad Libre que indica los recursos disponibles y si hay tareas esperando en Asignar actualiza una variable A_Intentar, privada del objeto pero global para todas las subrutinas del mismo. Seguidamente hace cierta la variable que indica que se han liberado recursos.
Veamos ahora la entry Asignar que se ejecuta cuando se liberan recursos. En primer lugar se decrementa A_Intentar, ya que una tarea va a probar a asignarse recursos. Si A_Intentar=0 entonces hacer false la variable booleana, esto significaría que nadie más liberará recursos. Ahora se verá si los recursos solicitados son menos que los disponibles, de ser así apropiárselos, en caso contrario volver a reencolarse llamado nuevamente a la entry Asignar.
Este esquema permite aprovechar mejor los recursos. Supongamos que hay una cola de tareas esperando por recursos y hay, por ejemplo, 4 recursos libres, si la tarea que está en primer lugar demanda 9 recursos no se podrá ejecutar, pero al intentarlo y volver a reencolarse permite que las siguientes de la cola lo intenten.

Ejercicio

Contestar brevemente a las siguientes cuestiones:

a) ¿Cuáles son las tres características principales que se pueden utilizar para hacer una clasificación de sistemas de tiempo real?. Si sólo hubiera 2 posibles valores para cada característica, ¿cuáles serían los más adecuados para cada una?
b) ¿Qué se entiende por tipología fuerte (respecto a un lenguaje de programación)?
c) ¿Qué se entiende por *programación con N versiones*?.¿Para qué se utiliza?
d) Si trabajamos con un único procesador, ¿qué ventajas puede tener el construir un programa concurrente en lugar de escribir un programa 'secuencial' (un solo proceso) que realice las mismas operaciones?

Ejercicio

¿Cómo definirías un tipo UPDOWN si quieres que su estructura no sea conocida por el usuario, y que con los datos de ese tipo sólo se puedan hacer dos operaciones, llamadas UP y DOWN? Añadir lo necesario al siguiente programa para que pueda utilizar el tipo UPDOWN:

```
procedure Main is
   U:UPDOWN;
begin
   UP(U);
   -- no nos interesa el resto del código
   DOWN(U);
end;
```

Ejercicio

Dada la siguiente especificación de package:

```
with Ada.Calendar;
package Fechas is
     type Mes is (ENE,FEB,MAR,ABR,MAY,JUN,JUL,AGO,SEP,OCT,NOV,DIC);
     type Formato is
          (Completo,     -- 7 Febrero 1991
           Numerico);    -- 7/2/91
     type Fecha is private;
     Error_Fecha: exception;
     procedure Get(Item: out Fecha);
     -- Lee una fecha en la forma DIA (1..31), MES(ENE..DIC), AÑO(1901..2099)
```

```
     -- y devuelve el valor de tipo Fecha correspondiente.
     -- Alza la excepción Error_Fecha si los datos de entrada no son del tipo o
     -- no están dentro de los rangos dados, o si la fecha no es correcta
     -- (ej: 30 FEB 1990).
     -- Para comprobar si la fecha es o no correcta, se puede utilizar la
     -- función Time_Of del package Ada.Calendar, que alza la excepción
     -- Time_Error en caso de fecha errónea.
     Function Hoy return Fecha;
     -- Retorna la Fecha de hoy. Utiliza para ello el package Ada.Calendar.
     procedure Put(Item:in Fecha; F: in Formato);
     -- Escribe una Fecha en el formato dado.
Private
     type Fecha is
     record
        D:Ada.Calendar.Day_Number := Ada.Calendar.Day_Number'First;
   M:Mes := Mes'First;
   A: Ada.Calendar.Year_Number := Ada.Calendar.Year_Number'First;
end record;
end Fechas;
```

Se pide:

a). Escribir el cuerpo correspondiente.

b). Escribir un procedure de prueba que haga lo siguiente:

- Escribir la fecha de hoy en formato completo.
- Leer una fecha utilizando un bloque de recuperación ante el error Error_Fecha, de modo que repita la lectura hasta que demos una fecha correcta.
- Escribir la fecha leída con el formato numérico.

```
package Ada.Calendar is
   type Time is private;
   subtype Year_Number  is Integer range 1901 .. 2099;
   subtype Month_Number is Integer range 1 .. 12;
   subtype Day_Number   is Integer range 1 .. 31;
   subtype Day_Duration is Duration range 0.0 .. 86_400.0;

   function Clock return Time;
   function Year    (Date : Time) return Year_Number;
   function Month   (Date : Time) return Month_Number;
   function Day     (Date : Time) return Day_Number;
   function Seconds (Date : Time) return Day_Duration;
   procedure Split
     (Date    : Time;
      Year    : out Year_Number;
      Month   : out Month_Number;
      Day     : out Day_Number;
      Seconds : out Day_Duration);
   function Time_Of
     (Year    : Year_Number;
      Month   : Month_Number;
      Day     : Day_Number;
      Seconds : Day_Duration := 0.0)
      return    Time;
-- Alza la excepción Time_Error si los parámetros no forman un valor de tiempo
-- válido

   function "+" (Left : Time;     Right : Duration) return Time;
   function "+" (Left : Duration; Right : Time)     return Time;
   function "-" (Left : Time;     Right : Duration) return Time;
```

```
   function "-" (Left : Time;     Right : Time)     return Duration;
   function "<"  (Left, Right : Time) return Boolean;
   function "<=" (Left, Right : Time) return Boolean;
   function ">"  (Left, Right : Time) return Boolean;
   function ">=" (Left, Right : Time) return Boolean;
   Time_Error : exception;
private
   ...
end CALENDAR;
```

Ejercicio

Supongamos que el siguiente procedure es utilizado concurrentemente por varias tareas

```
procedure PRUEBA(I:integer) is
-- cada tarea, al utilizar el procedure PRUEBA, le pasa su nº de identificación
begin
  put_line("PRUEBA está siendo utilizado por la tarea " & integer'image(I));
  delay 3.0;
  put_line("La tarea " & integer'image(I) & " acaba de ejecutar PRUEBA");
end PRUEBA;
```

¿Cómo podríamos conseguir que el procedure PRUEBA se ejecute como si fuera una operación atómica?. Escribir el código correspondiente y las llamadas de las tareas al procedure.

Ejercicio

Decir si las siguientes afirmaciones son verdaderas o falsas, justificando las respuestas:

a). Se entiende por sistema en tiempo real rígido (hard) aquel que es robusto, es decir, que presenta alta fiabilidad.
b). No pueden declararse arrays de tareas por ser éstas de tipo anónimo.
c). Decimos que la planificación de procesos es pre-emptive cuando cada proceso tiene asignado un 'quanto' (intervalo) de tiempo de ejecución.
d). Un semáforo de tres valores (S=0,1,2) permite programar el mecanismo de exclusión mutua.

Ejercicio

Definición de los términos región crítica y exclusión mutua. Explica cómo un objeto protegido puede proporcionar exclusión mutua en el acceso para leer y escribir un dato compartido por varias tareas. Escribe el código del tipo protegido correspondiente.

Ejercicio

Dado el bloque siguiente:

```
declare
     V: Integer;
begin
     PUT("Dime un número:");
     GET(V);
     LLAMADA_A_PROCEDURE(V);
end;
```

Añadir lo necesario para:

a). Declarar la excepción NUMERO_IMPAR.

b). Si el usuario da un valor impar, que se alce la excepción NUMERO_IMPAR.

c). Incluir un tratamiento de excepciones que, en caso de que se produzca la excepción NUMERO_IMPAR, que le pase al procedure el número par siguiente al valor dado por el usuario.

Ejercicio

¿Qué diferencias puede haber en la ejecución de estos segmentos de código?

```
 if A then
   accept B;
end if;
```

```
select
   when A=> accept B;
else
   null;
end select;
```

```
select
   when A=> accept B;
end select;
```

(Comparar lo que ocurre cuando A es TRUE y cuando A es FALSE)

Ejercicio

Dada la siguiente especificación:

```
generic
     type ELEMENT is private;
package PILA is
     type STACK is limited private;
     procedure PUSH(P:in out STACK; X: in ELEMENT);
     procedure POP(P:in out STACK; X: out ELEMENT);
     function VACIA(P: STACK) return BOOLEAN;
     function LLENA(P: STACK) return BOOLEAN;
     procedure PUT(P:in STACK);
     function LONG(P: STACK) return INTEGER;
     private
          MAX: constant:= 100;
          type VECTOR is array (INTEGER range <>) of ELEMENT;
          type STACK is
             record
               S: VECTOR(1..MAX);
               TOP: INTEGER range 0..MAX:= 0;
             end record;
end PILA;
```

Añadirle una función "=" que compare 2 variables STACK devuelva TRUE si ambas son iguales (elementos iguales) y FALSE en caso contrario. Escribir el cuerpo de la función.

Ejercicio

Dada la siguiente declaración

```
     package FORMAS is
        type OBJECT is tagged private;
        function AREA(O:OBJECT) return FLOAT;
        procedure GET(O: in out OBJECT);
     private
        type OBJECT is tagged
        record
```

```
      X,Y:FLOAT;
   end record;
end FORMAS;
```

Reformular el tipo OBJECT como un tipo abstracto sin componentes, y definir como abstractas las operaciones.

- Definir la clase PUNTO como descendiente de OBJECT.
- Definir la clase CIRCULO como descendiente de PUNTO.
- Definir la clase RECTANGULO como descendiente de PUNTO. El rectángulo vendrá dado por los vértices opuestos (0,0) y (X,Y). Implementar las operaciones de la clase RECTANGULO, utilizando lo más posible las operaciones heredadas.
- Definir una función MAYOR que tome como parámetros dos objetos de tipos descendientes de PUNTO y retorne el objeto de mayor área.

Ejercicio

Se dispone de un esquema cliente servidor, de manera que el cliente se encuentra en un bucle infinito donde primero ejecuta PROCEDURE1 (que no hace falta implementar) y después se cita con el servidor transfiriéndole un valor entero y recibiendo un float. Solo esperará como máximo 0.5 s para obtener la cita, si en ese tiempo el servidor no le ha atendido intentará conectarse con un segundo servidor para realizar la misma transferencia de información. Si no consigue comunicar en 1s con el segundo servidor elevará una excepción TIME_OUT que mostrará un mensaje por pantalla.

La tarea servidor (tanto el 1 como el 2) está también ejecutando un bucle infinito, en él ejecuta en primer lugar el PROCEDURE2 y después examina si hay tareas esperando por una cita, si es así la atiende, en caso contrario sale de la cita y repite el bucle.

Realizar un programa que cumpla con las especificaciones anteriores.

Ejercicio

Se pretende disponer de un programa de control con varias tareas. En el mismo se debe hacer una lectura de un dispositivo por el mecanismo de muestreo, para ello se cuenta con las funciones que responden a los siguientes prototipos

function ver_si hay_dato return booelan;
function leer_dato return integer;

La aparición de este dato se considera un evento de alta prioridad y debe ser notificado inmediatamente para su procesamiento con la subrutina. El muestreo se ejecutará cada 5 ms.

procedure tratar_evento(evento : integer);

Siendo evento el dato leído por la función leer_dato.

Además del tratamiento de este evento el programa ejecuta un lazo de control cada 50 ms llamando a :

procedure ejecutar_control;

La ejecución de esta rutina consume unos 15 ms, el tiempo libre restante lo dedica a la monitorización del sistema mediante

procedure monitoriza;

Realizar un programa con las correspondientes tareas de forma que se comuniquen convenientemente y se cumplan los requisitos de planificación que se deducen del enunciado.

Nota: Se considera que todas las subrutinas anteriores están disponibles en el paquete CONTROL.

Ejercicio

Realizar un paquete para trabajar con pilas, cuyos elementos pueden ser de cualquier tipo, que incorporen las operaciones push, pop y comparación entre pilas. Añadir el tratamiento para las excepciones de pila llena y pila vacía, elevándolas cuando sea preciso.

Ejercicio

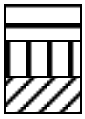
El mecanismo de Ceiling Locking del ADA es una forma de herencia de prioridades que se basa en los siguientes puntos:
Cada proceso tiene una prioridad base, asignada según algún esquema de planificación.
Cada recurso tiene un valor de tope (*ceiling*) definido, que el máxima de la de los procesos que lo usan.
Un proceso tiene una prioridad dinámica que es la máxima de su prioridad base y el valor tope de cualquiera de los procesos que tiene asignados.

Ahora vamos a suponer que un conjunto de procesos comparten recursos mediante mecanismos de exclusión mutua (un objeto protegido) Sean 4 tareas T1 a T4 que pueden acceder a dos recursos Q y V de manera exclusiva según el orden que se detalla en la siguiente tabla, donde E representa la ejecución durante un tick y Q y V representa la ejecución durante un tick utilizando los respectivos recursos.

Procesos	Prioridad	Secuencia de ejecución	Tiempo de lanzamiento
T4	4	EEQVE	4
T3	3	EVVE	2
T2	2	EE	2
T1	1	EQQQQE	0

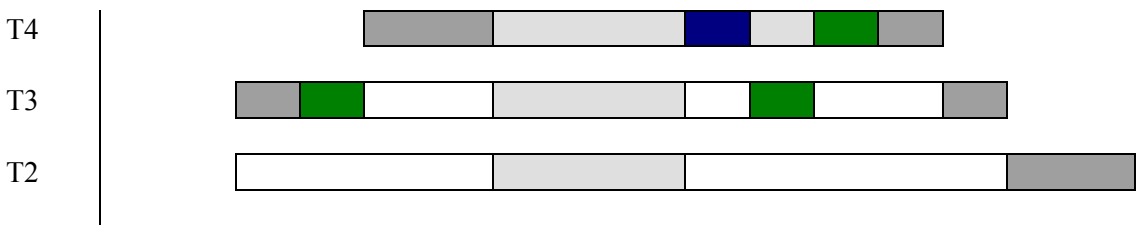
Establecer un diagrama temporal de barras que muestre la ejecución de las tareas suponiendo:
Que no hay herencia de prioridades.
Que hay herencia según el mecanismo de ceiling locking descrito anteriormente.

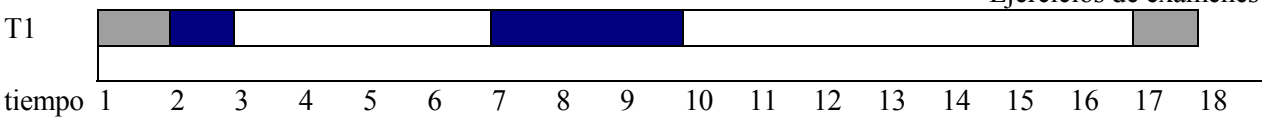
Para mostrar los estados en que puede estar una tarea se pueden usar las siguientes convenciones:


En ejecución con Q cerrado (*locked)*
En ejecución con V cerrado (*locked)*
Bloqueado


Pasado por alto (preparado)
En ejecución

Este podría ser un ejemplo del tipo de diagrama que debes realizar





Ejercicio

Desde un programa principal cliente se desea establecer comunicación con una tarea servidor, la especificación de dicha tarea es:

```
task servidor1 is
  entry  peticion(dato : out  integer);
end servidor1;
```

Existe también una tarea servidor2 con análoga especificación. Se desea que desde la tarea cliente se establezca una comunicación con servidor1, si dicha petición no es atendida en 20 s se intentará establecer la comunicación con servidor2, si pasados 10 s no se establece este segundo enlace deberá elevarse una excepción Fallo_Comunicacion. En respuesta a la excepción se escribirá un mensaje por pantalla, se generará un valor devuelto por defecto (valor -1) y se continuará con el programa como si la comunicación hubiera funcionado correctamente.

Realizar un programa que responda a tales especificaciones de dos maneras distintas:

Haciendo que tras el primer fallo de comunicación con el servidor1 aparezca una excepción, y en el manejador de la misma esté la llamada a sevidor2. Si la cita con servidor2 no se establece en los 10 s previstos se alzará la excepción Fallo_Comunicación como se detalló antes.
Sin empleo de la excepción intermedia.

Solución

```
with text_io; use text_io;

procedure cliente1 is
time_out,fallo_comunicacion: exception;
j: integer;

task servidor1 is                       --dos tareas servidores, no hacen nada
  entry peticion(dato : out integer);  --es para comprobar que salta la excepcion
end servidor1;
task body servidor1 is
begin
delay 35.0;
end servidor1;
task servidor2 is
  entry peticion(dato : out integer);
end servidor2;
task body servidor2 is
begin
delay 35.0;
end servidor2;

-- en esta solución he decidido diferenciar las excepciones que se producen con
cada
-- servidor, hay otras alternativas.
```

```
begin          -- del procedimiento
-- aquí podrían venir muchas cosas
begin          -- abriendo un bloque para acotar una excepción
put_line("dentro del cliente");

begin          -- abriendo un bloque para acotar una excepción
  select
     servidor1.peticion(j);
  or
     delay 2.0;
        raise time_out;
  end select;

exception
when time_out =>             --manejando la excepcion primera
        put_line("se levanto la excepcion time_out");
        select     -- intentar cita con segundo servidor
          servidor2.peticion(j);
          or
        delay 2.0;
          raise fallo_comunicacion;    -- elevar segunda excepción
     end select;
end;                     --del bloque1
exception                --manejando la segunda excepcion
when fallo_comunicacion =>
        j:= -1 ;  -- valor por defecto
        put_line("se levanto la excepcion");
end ;                        --del bloque2
-- aquí seguiria el programa
end cliente1;
```

```
-- segunda solución, hasta aquí todo igual.
begin
put_line("dentro del cliente");
begin
  select
     servidor1.peticion(j);
  or
     delay 2.0;
     select
        servidor2.peticion(j);
     or
        delay 2.0;
        raise fallo_comunicacion;
     end select;
  end select;
exception
when fallo_comunicacion =>
        j:= -1 ;  -- valor por defecto
        put_line("se levanto la excepcion");
end ; -- del bloque
end cliente;
```

Ejercicio

Se desea mantener un registro de las horas trabajadas a lo largo del año por los empleados de una oficina. Para cada uno de ellos se va a elaborar un array de flotas que indicará las horas trabajadas cada día de los laborables. Dicho array tendrá dos índices, uno de ellos el nombre del mes (un tipo enumerado que debéis definir) y otro el día del mes (hasta 31 días).

Habrá un subprograma para rellenar las horas trabajadas entre días, pasados como parámetros.
También deberá haber un subprograma para introducir las horas trabajadas en el día actual.

Realizar un programa que cumpla con tales especificaciones

Nota, los subprogramas no deberán permitir al usuario la escritura en fechas no existentes, por ejemplo 31 de Junio, rellenando estos campos con ceros. Pudiera darse la circunstancia de trabajar Sábados y festivos, por lo que estos no requerirán un tratamiento especial. A continuación se recuerdan datos y prototipos que se encuentran dentro del paquete calendar que os pueden resultar de utilidad.

```
type Time is private;
subtype Month_Number is Integer range 1 .. 12;
subtype Day_Number   is Integer range 1 .. 31;

function Clock return Time;
function Month   (Date : Time) return Month_Number;
function Day     (Date : Time) return Day_Number;
```

Solución

```
with Calendar; use Calendar;
with Ada.Float_Text_IO; use Ada.Float_Text_IO;
with Text_IO;use Text_IO;

-- en esta solucion la entrada/salida por pantalla resulta pobre
-- se la podria mejorar bastante mostrando el dia del mes que se va a rrellenar
-- esto obligaria a hacer entrada/salida con los tipos enumerados
-- Por simplicidad se supone que siempre se introducen datos correctos
procedure Trabajos is
-- definicion de tipos
type Meses is( ene, feb, mar, abr, may , jun, jul, ago, sep , oct, nov, dic);
type Horas_Trabajadas is array (Meses, integer range 1..31 ) of float;
type Dia_Mes is
record
  Num_Dia : integer range 1..31;
  Nom_Mes : Meses;
end record;
pp : Horas_Trabajadas;  -- esto seria el array para un empleado
Dia1, Dia2 : Dia_Mes;   -- un par de variables para un ejemplo

procedure Hoy(Quien: in out Horas_Trabajadas) is
Hoy : time;             -- para averiguar la fecha
Horas: float;
begin
  Hoy:=clock;
  Put("Cuantas horas has trabajado ? ");
  Get(Horas);
  -- emplear funciones de calendar para obtener dia y mes de la fecha
```

```
  Quien(Meses'val(Month(Hoy)),Day(Hoy)):=Horas;
end hoy;

procedure Desde_El_Dia(Dd1, Dd2: Dia_mes; Quien : in out Horas_Trabajadas) is
Dia: integer range 1..31;
Mes: Meses;
Horas: float;

begin
  Dia:= Dd1.Num_dia;     -- calcular fecha inicio
  Mes:= Dd1.Nom_mes;
-- establecer un bucle desde comienzo al final
while ((Meses'pos(Mes)*31+Dia)<=(Meses'Pos(Dd2.Nom_Mes)*31+Dd2.Num_Dia)) loop
    case Mes is         -- ver cuantos dias tiene el mes
    when Abr | Jun | Sep | Nov =>
      if dia =31 then   -- no hay dia 31, ponerlo a cero y saltar al siguiente
        Quien(Mes,Dia):=0.0;
        Dia :=1;
        Mes := Meses'succ(Mes); -- pasar a siguiente mes
      else              -- leer horas, cargar valor e incrementar dia
        Put("Cuantas horas has trabajado ? ");
        Get(Horas);
        Quien(Mes,Dia):=Horas;
        Dia:= Dia+1;
      end if;
    when feb =>         -- como en le caso anterior
      if 28< dia then
        Quien(mes,dia):=0.0;
        Dia :=1;
        Mes := Meses'succ(Mes);
      else
      put("Cuantas horas has trabajado ? ");
      get(Horas);
        Quien(Mes,Dia):=Horas;
        Dia:=Dia+1;
      end if;
    when others =>
      Put("Cuantas horas has trabajado ? ");
      Get(horas);
      Quien(Mes,Dia):=Horas;
      Dia:= Dia+1;
      if Dia = 32 then
        Dia:=1;
        Mes:= Meses'succ(Mes);
      end if;
    end case;
end loop;
end Desde_El_Dia;

begin
  Hoy(pp);
  Dia1.Num_Dia:=29;
  Dia1.Nom_Mes:=Abr;
  Dia2.Num_Dia:=2;
  Dia2.Nom_Mes:=May;
  Desde_El_Dia(Dia1,Dia2,Pp);
end Trabajos;
```

Ejercicio

Realizar un bloque de código para que un procedure Accion esté preparado para ejecutarse cada 0.7 s.

Realizar un bloque de código para que una tarea intente establecer la comunicación con la tarea servidor1, en caso de no lograrlo en 0.3s lo intentará con servidor2. Si en 0.2s tampoco lo logra elevará una excepción Fallo_De_Comunicación.

Ejercicio)

Un STR consta de 3 tareas, A, B, y C con los siguiente parámetros temporales:

Tarea	Periodo T	Tiempo de cálculo C
A	4	1
B	6	2
C	3	1

Las tareas se ejecutan con planificación por prioridades y desalojo. Se pide: asignar prioridades a las tareas empleando la planificación por frecuencias, determinar si el sistema es planificable, calcular el tiempo máximo de respuesta para cada tarea C. Hacer un cronograma con la ejecución de las tareas.

$$wi^{n+1} = Ci + \sum_{j \in hp(i)} \left\lceil \frac{wi^n}{Tj} \right\rceil Cj$$

$$\sum_{i=1}^{N} \frac{C_i}{Ti} \leq n(2^{1/n} - 1)$$

Ejercicio

Implementar un paquete donde se pueda declarar un objeto de tipo semáforo y se implementen las operaciones up y down para subir y bajar el semáforo.

Se harán dos versiones, una empleando tareas y otra con objetos protegidos.
El semáforo se puede subir o bajar de uno en uno.
Cuando se cree un objeto del tipo semáforo se le deberá facilitar un valor inicial.
La interfaz que ambas implementaciones presenten al usuario debe ser exactamente el mismo. Es decir, un programa que emplee una versión u otra necesitaría modificar ni una línea de código.

Ejercicio

Un programa que divide 2 vectores de enteros elemento a elemento (A(i)/B(j)), para cada par i,j. La división de ambos vectores se realizará implementando bucles. Se redefine la función de dividir para elevar una excepción cuando aparezca una división por cero. Se desea que cuando aparezca tal excepción se escribe el mensaje:

División por Cero: A(i)/B(i).

Siendo A(i) y B(i) el par de números con los que se ha generado la excepción. Tras el tratamiento de la excepción se continuará con el siguiente par de números hasta que se hayan dividido todos los pares.

Si la función de dividir es la que viene a continuación completar el programa para cumplir con las especificaciones antes citadas

```
FUNCTION ”/” (IZQ, DCHA: in INTEGER) return INTEGER is
begin
  if DCHA = 0 then
    raise DIVISION_POR_CERO;
  end if;
  return Standard.”/”(IZQ,DCHA);
end ”/”;
```

Solución

```
with text_io; use text_io;
with iio; use iio;

procedure main is
type INT_ARRAY is array ( NATURAL range <> ) of INTEGER;
VECTOR1: constant int_array := (6,3,4);
VECTOR2: constant int_array := (3,0,2,0);
COCIENTE: INTEGER;
DIVISION_POR_CERO: exception;

function "/"(IZQ,DCHA: INTEGER) return INTEGER is
begin
  if ( DCHA = 0 ) then
    raise DIVISION_POR_CERO;
  end if;
  return standard."/"(IZQ,DCHA);
end;

begin
  for J in VECTOR2'range loop
    for I in VECTOR1'range loop
      begin
      COCIENTE:= VECTOR1(I)/VECTOR2(J);
      put("El cociente de "); put(VECTOR1(I), width=>2) ;put(" / ");
      put(VECTOR2(J), width=>2);put(" es "); put(COCIENTE,width=>2); new_line;
      exception
        when DIVISION_POR_CERO =>
          put("Error DIVISON POR CERO en el par de numeros ");
          put(VECTOR1(I),width=>2);put(" / ); put(VECTOR2(J),width=>2);
          new_line;
      end;
    end loop;
  end loop;
end;
```

Ejercicio

Se desea realizar un PAQUETE GENERICO para el manejo de un buffer circular. Las especificaciones son las siguientes.
El tamaño del buffer y el tipo de elementos que maneja serán pasados como parámetros cuando se realice la creación genérica del paquete.
El buffer estará implementado como un tipo privado.
Se deben suministrar dos subprogramas PONER y SACAR para el manejo del buffer circular.
Si el buffer está lleno y se intenta realizar una operación PONER se elevará la excepción BUFFER_LLENO.
Si el buffer está vacío y se intenta realizar una operación SACAR se elevará la excepción BUFFER_VACIO.
Las excepciones no serán manejadas ni en los procedimientos ni en el paquete, serían manejadas desde el programa en que se haga uso del paquete, no es preciso realizar tal programa.

```
generic
  SIZE : POSITIVE;
  type ITEM is private;
package BUFFER_CIRCULAR is
  type BUFFER is private;
  BUFFER_LLENO: exception;
  BUFFER_VACIO: exception;
  procedure PONER( B: in out BUFFER; X : in ITEM );
  procedure SACAR( B: in out BUFFER; X : out ITEM );
private
  type BUFFER is array(1..SIZE) of ITEM;
  CABEZA: POSITIVE := 1;
  COLA: POSITIVE := 1;
  VACIO: BOOLEAN := TRUE;
  LLENO: BOOLEAN := FALSE;
end;
package body BUFFER_CIRCULAR is
  procedure PONER( B: in out BUFFER; X : in ITEM ) is
  begin
    if LLENO then
      raise BUFFER_LLENO;
    end if;
    B(CABEZA):= X;
    if ( CABEZA = SIZE ) then
      CABEZA := 1;
    else
      CABEZA := CABEZA +1;
    end if;
    if (CABEZA = COLA) then
      LLENO := TRUE ;
    end if;
  end PONER;
  procedure SACAR( B: in out BUFFER; X : out ITEM ) is
  begin
    if VACIO then
      raise BUFFER_VACIO;
    end if;
    X := B(COLA);
    if ( COLA = SIZE ) then
       COLA := 1;
```

```
    else
      COLA := COLA +1;
   end if;
   if (CABEZA = COLA) then
     VACIO := TRUE ;
   end if;
 end SACAR;
end BUFFER_CIRCULAR;
```

Ejercicio

Se va a implementar un sistema de riego con la ayuda de un computador. El jardín a regar está dividido en 8 puntos de riego, con 8 líneas de agua independientes. Cada línea tiene una válvula que abre/cierra el paso de agua. Estas válvulas van a ser abiertas o cerradas según la humedad del suelo. La humedad se determina mediante unos elementos sensores continuos, que producen una salida proporcional al % de humedad. Cada 30 minutos, el sensor mide la humedad. Si ésta es <10%, se abre el paso de agua durante 5 minutos. Si 10%<=humedad<20%, se abre el paso de agua durante 3 minutos.
Los puntos de riego se pueden representar mediante tareas.
El algoritmo de implementación de cada punto de riego se puede describir como:

```
esperar llamada INICIO para comenzar el funcionamiento
loop
        Si hay una llamada a FIN, salir del bucle
        Esperar 30 minutos
        Leer sensor
        Si humedad >10%
                abrir la válvula durante 5 minutos
        Sino, si 10%<=humedad<20%
                abrir la válvula durante 3 minutos
        fin si
fin loop
```

Para simular cada punto de riego, suponemos que disponemos de las siguientes unidades de compilación:

- Un procedure LEE_SENSOR, que devuelve el % de humedad medido por el sensor.
- Un procedure ABRIR, que abre la válvula.
- Un procedure CERRAR, que cierra la válvula.

Se pide:

a). Escribir el cuerpo de un procedure ABRE_VALVULA, que recibe el número de minutos a regar, y se encarga de abrir, esperar y cerrar la válvula.

b). Escribir el package RIEGO, que define el tipo task PUNTO_RIEGO.

Solución

El procedure Abre_Valvula sería semejante al siguiente

```
procedure Abre_valvula(Minutos :in integer; Numero : valvula) is
begin
  Abrir(N_valvula);          -- procedure ya facilitado que abre la válvula
  delay minutos*60.0;                -- mantener válvula abierta
  Cerrar(N_valvula);                 -- cerrar válvula pasado el tiempo;
end Abre_Valvula;
```

Este procedure se va a usar dentro de la tarea Punto_Riego, que estará incluida en el paquete Riego, por tanto también se debería incluir en el paquete o en la tarea. A su vez también se incluirán los que según el enunciada están ya disponibles.

```
With ada.calendar; use ada.calendar;
Package Riego is
   function Lee_Sensor(Numero : integer) return float;
   procedure Abrir(Numero: integer);
   procedure Cerrar(Numero : integer);
 -- La manera más cómoda de identificar sensores y válvulas sería mediante
 -- números enteros, ya que así podrían pasarse como parámetros a las tareas
   task type Punto_Riego(N_Sensor, N_valvula :integer) is
      entry Parar;
      entry Arrancar;
   end PuntoRiego;
end Riego;
package body Riego (N_Sensor, N_valvula :integer) is
 --Aquí podría ir Abre_Valvula
   task body Punto_Riego is
   Humedad : float;
   Salir: Boolean :=False;
   begin
     accept Arrancar do                -- esperar a que se active
       null;
     end Arrancar;
     loop
       select
          accept Parar do
   salir=True;;
          end Parar;
        or
          delay 30.0;
          Humedad:= Lee_Sensor(N_Sensor);
          if ( Humedad < 0.1 ) then
            Abre_Valvula(5,N_Valvula);
          elsif (0.1 <= humedad ) and (humedad < 0.2) then
            Abre_Valvula(3,N_Valvula);
          end if;
       end select;
       exit when salir;
     end loop;
   end Punto_Riego;
end Riego;
```

Ejercicio

Completar el siguiente programa.

```
with TEXT_IO; USE TEXT_IO;
with CALENDAR ; use CALENDAR;
with IIO; use IIO;

procedure tasking is
```

```
  package DURATION_IO is new fixed_io(duration);   -- para hacer E/S con tipos
Duration (coma fija)
  use duration_io;

  NUM:INTEGER;
  INTER: DURATION;
  TIEMPO :TIME;

task type TICK is
   entry ESCRIBE;
   entry ACABA;
end TICK;
T: TICK;
```

Solución

```
task body TICK is
-- cada vez que hay una llamada a ESCRIBE deberá escribir TICK!!
-- si hay una llamada a ACABA la tarea finaliza.
  salir: boolean:=false;     -- para salir de la tarea

  begin
    while (salir= false)loop
     select
       accept escribe do
         put("TICK !!"); NEW_LINE;
       end escribe;
       or
       accept acaba do
         salir:=true;
       end acaba;
     end select;
    end loop;
  end tick;

begin --TASKING
   PUT("N° de llamadas:  ");
   GET(NUM); NEW_LINE;
   PUT("Intervalo entre llamadas  ");
   GET(INTER); NEW_LINE;
-- Debe hacer NUM llamadas a ESCRIBE exactamente cada INTER segundos
-- Considérese que entre cada llamada a ESCRIBE se debe ejecutar un procedure
-- ZZ que no necesitamos reflejar en el problema, que necesita X < T segundos
-- para ejecutarse (X desconocido). Despues se hará una llamada a ACABA.

  TIEMPO :=clock;
  for i in 1..num loop
    delay C+INTER - clock;
    TIEMPO :=clock;
    ZZ;
    T.ESCRIBE;
  end loop;
  T.ACABA;

end TASKING;
```

Ejercicio

Realizar el cuerpo de un tipo task llamado SENSORES. Las tares de este tipo se encargarán de monitorizar la temperatura de diversas habitaciones. La especificación del tipo es:

```
task type SENSORES is
   entry START(T: in DURATION);        --Inicia la actividad del sensor. Se indica
el tiempo entre medidas
   entry SET_LIMIT(VALUE: in INTEGER); -- Establece un valor límite en la T
                        --A partir de este valor debe activar una alarma.
   entry GET_STATUS(VALUE : out INTEGER);  --devuelve el último valor de T medido
   entry SHUT_DOWN;                    -- acaba la actividad del sensor
end SENSORES;
```

La tarea deberá realizar lo siguiente:

Iniciar la actividad y fijar el periódo T de medidas
Cada T segundos debe medir la Tª ambiente, mediante una llamada a la función medir, que suponemos ya implementada. Si la Tª es mayor que el límite, llamada al procedure SOUND_ALARM, que también suponemos escrito.
Entre medidas de Tª puede recibir llamadas a SHUT_DOWN, a GET_STATUS o a SET_LIMIT.

Solución

```
task body SENSORES is
SALIR: BOOLEAN := FALSE;-- para salir del bucle
PERIODO: DURATION;
TEMPERATURA : INTEGER ;
-- valor por defecto del limite, seria equivalente a no tener limite, ya que el
enunciado no especifica el valor inicial
-- serían válidas varias alternativas, como por ejemplo aceptar cita en SET_LIMIT
entes del bucle
LIMITE_TEMPERATURA : INTEGER := INTEGER'LAST;

begin
  accept START ( T: in DURATION) do
    PERIODO := T;
  end START;
  while (SALIR = FALSE) loop
    TEMPERATURA := MEDIR;
    if ( TEMPERATURA > LIMITE_TEMPERATURA ) then
      SOUND_ALARM;
    end if;
    select
      accept SET_LIMIT(VALUE : in INTEGER)  do
        LIMITE_TEMPERATURA := VALUE;
      end SET_LIMIT;
      or
      accept SHUT_DOWN do
        SALIR:=TRUE;
      end SHUT_DOWN ;
      or
      accept GET_STATUS(VALUE : out INTEGER)  do
        VALUE := TEMPERATURA;
      end GET_STATUS;
      or
```

```
        delay PERIODO;
    end select;
  end loop;
end SENSORES;
```

Ejercicio

Programar un paquete para almacenar los volúmenes de una biblioteca particular. Para cada libro se deberá almacenar, en el formato que se considere adecuado:

Título.
Autor
Editorial
ISBN (código de letras y dígitos, hasta 12 caracteres)

Cada vez que se incorpore un nuevo libro a la lista de los existentes se hará por orden alfabético del autor.
Se desea tener una opción para que se muestren todos los libros.
Se desea tener una opción para que se muestren los libros de un autor.

Realizar tal paquete de manera que los datos de la biblioteca no se puedan manipular desde otros módulos si no es con los subprogramas que facilite el paquete.

Ejercicio

Una computadora se puede comunicar con otro dispositivo a través de un periférico según dos modos de operación.

Enviando una trama de ordenes y esperando la correspondiente contestación.
Recibiendo una señal de alarma, que puede llegar en cualquier momento.

Si se recibe una señal de alarma habrá que transmitirla inmediatamente a una tarea Tratar_Alarmas que debe ser de máxima prioridad. No se dispone programación de interrupciones por tanto la consulta del puerto para detectar las alarmas se realiza por polling. Además es aconsejable realizarla cada poco tiempo, por ejemplo 0.1 s

Tanto las tramas normales, como las de alarmas comparten el mismo puerto de comunicaciones, por lo que antes de iniciar una operación de envío de tramas hay que verificar que no haya un dato esperando ser leído.

Se disponen de los siguientes subprogramas
Function Hay_dato return boolean; -- indica si hay algún dato en el puerto
Function Leer_Alarma return Trama_Alarma;
Procedure Enviar_Orden(Orden : in Trama_Orden);
Function Leer_Contestacion return Trama_Datos;

Se desea construir el armazón de un programa que permita enviar ordenes a través del puerto y recibir los correspondientes resultados. Este programa correrá un bucle infinito de forma que entre envío y envío de ordenes se ejecuta Haz_algo que consume un tiempo sustancial y que no está determinado.

Utilizad un objeto protegido con un tipo puerto definido también en el paquete, podéis incluir en ese objeto los subprogramas del paquete.

Se supone que los correspondientes cuerpos junto con los tipos de datos involucrados están en un paquete Comunicaciones. Igualmente Tratar_alarmas es un tipo tarea definido en el paquete anterior.

Solución

Este ejercicio es bastante abierto en cuanto a posibilidades, aquí se apunta una de ellas.
Consideraciones generales:

Las operaciones con el puerto se deben incluir dentro de un objeto protegido

```
Protected type Puerto is
   function Hay_dato return boolean;
   function Leer_Alarma return Trama_Alarma;
   procedure Enviar_Orden(Orden: in Trama_Orden);
-- otra posibilidad es que enviar_orden sea una entry cuyo barrier
-- sea el resultado de Hay_dato
   function Leer_contestacion return Trama_Datos;    private
   tipo_puerto   -- Aquí los detalles de la implementación del puerto
end Puerto.

Protected body Puerto is
......--detalles de los cuerpos de los subprogramas
end  Puerto.
```

Si se detecta una alarma se debe pasar la misma a la tarea del tipo TRATAR_ALARMAS que podemos suponer que su cuerpo tiene una estructura semejante a la siguiente:

```
Task type Tratar_Alarmas;
   pragma priority(5);
   Entry Hay_Alarma(Alarma: in Trama_Alarma);
end Tratar_Alarmas;
task body Tratar_Alarmas is
...
begin
loop
   .....
   accept Hay_alarma(Alarma: Trama_Alarma) do
      ......       -- aquí tratar la alarma
   end accept;
   ....
end loop;
end Tratar_Alarmas.
```

Al no haber interrupciones la detección de datos en el puerto se deberá realizar muestreándolo, contamos con una función que indica si se ha recibido algún dato. Para hacer que el muestreo sea periódico e independiente de lo que esté haciendo el programa se puede incluir dentro de una tarea POLLING. Esta tarea Polling se puede incluir dentro del main.

```
with calendar; use Calendar;
with Comunicaciones; use Comunicaciones;

procedure main is

Gestor_Alarmas : Tratar_Alarmas;      --declaración de una tarea del tipo
anterior
Mi_puerto: Puerto;          --declaración del objeto protegido;
```

```
Task Polling is
   pragma priority(6);           -- tarea que realiza el muestreo para ver si hay
alarmas.
end Polling;

task body Polling is             -- va a ser la tarea más prioritaria
ahora:Time;

begin
loop
   ...
   ahora := clock;
   delay until (ahora +0.1);
   aquí acceso al puerto
   if  Mi_Puerto.Hay_Dato then        -- si hay un dato esperando tiene que ser
una alarma
       Gestor_Alarmas.Hay_Alarma(Leer_alarma);       --citarse con tarea que trata
alarma
   end if;

end loop;
end Polling

Mis_Datos : Trama_Datos;
Mi_Orden: Trama_Orden;


begin                            --aquí comienzo del programa principal
loop
   ...
   while Mi_Puerto.Hay_dato loop          -- si se accede al puerto inicialmente
y hay dato significa que
       null;                     --  hay  una  alarma  pendiente,  en  tal  caso  no
hacer nada hasta que
   end loop;                     -- sea retirada por POLLING.
   Mi_Puerto.Enviar_Orden(Mi_Orden);
   Mis_Datos:=Mi_Puerto.Leer_Contestacion;
   ...
end loop;
end main;
```

Ejercicio

¿Qué salida producirá por pantalla el procedure servicio cuando las teclas pulsadas son A B C D J.?

```
-- Server.ADS
package Server is
   A,B,C,D:exception;
   procedure Servicio;  --puede alzar A,B,C o D
end Server;

--Server.ADB
with Ada.Text_IO; use Ada.Text_IO;
package body Server is
   procedure Servicio is
      Ch:Character;
   begin
```

```
      put_line("A: Alzar excepción A");
      put_line("B: Alzar excepción B");
      put_line("C: Alzar excepción C");
      put_line("D: Alzar excepción D");
      put_line("Otro carácter: No alzar excepción");
      new_line;
      put("A,B,C,D,N: ");
      get(Ch);
      case Ch is
         when 'A'|'a' => raise A;
         when 'B'|'b' => raise B;
         when 'C'|'c' => raise C;
         when 'D'|'d' => raise D;
         when others => null;
      end case;
   end Servicio;
end Server;

-- Usa_Server.ADB
with Server; use Server;
with Ada.Text_IO; use Ada.Text_IO;
procedure Main is
   task Uno;
   task Dos is
      entry Sinc;
   end Dos;

   task body Uno is
   begin
      Dos.Sinc;
   exception
      when A=>
         put_line("A tratada en Uno");
         raise;
      when B=>
         put_line("B tratada en Uno");
         raise C;
      when C=>
         put_line("C tratada en Uno");
      when D=>
         put_line("D tratada en Uno");
   end Uno;

   task body Dos is
   begin  --bloque X
      begin  --bloque Y
         begin  --bloque Z
            accept Sinc do
               begin
                  Servicio;
              exception
                  when A=>
                     put_line("A tratada en Sinc");
                  when B=>
                     put_line("B tratada en Sinc");
                     raise;
                  when C=>
                     put_line("C tratada en Sinc");
                     raise D;
```

```
               end;
            end Sinc;
         exception
            when A=>
               put_line("A tratada en bloque Z");
            when B=>
               put_line("B tratada en bloque Z");
               raise C;
            when others=>
               put_line("OTHERS tratada en bloque Z");
               raise C;
         end; --bloque Z
      exception
         when C=>
            put_line("C tratada en bloque Y");
         when others=>
            put_line("OTHERS tratada en bloque Y");
            raise C;
      end; --bloque Y
   exception
      when A=>
         put_line("A tratada en bloque X");
      when others=>
         put_line("OTHERS tratada en bloque X");
   end Dos; --bloque X y tarea Dos

   begin  --Main
      null;
   exception
      when A=>
         put_line("A tratada en Main");
      when B=>
         put_line("B tratada en Main");
      when C=>
         put_line("C tratada en Main");
      when D=>
         put_line("D tratada en Main");
end Main;
```

Solución

El procedure Servicio lee tecla pulsada y en caso de que sea A,B,C o D eleva la excepción del mismo nombre, pero no la trata, por lo que se propagará a la unidad que haya llamado a Servicio, esta es la tarea Dos. A partir de este momento la excepción se propagará o no a otras unidades dependiendo de:
Que encuentre el manejador.
Que no se lance una nueva excepción en el manejador.
Que la excepción se eleve dentro del accept Sinc y no sea tratada

Tecla pulsada	Salida por Pantalla	Causa
A	A tratada en Sinc	La excepción A se trata en el accept Sinc de la tarea Dos, por lo que no se propaga
B	B tratada en Sinc	La excepción B se trata en el accept Sinc de la tarea Dos, pero se relanza
	B tratada en bloque Z	La excepción se propaga al bloque Z que contiene al accept. Aquí se trata pero se lanza C.
	C tratada en bloque Y	La excepción se propaga al bloque Y que contiene al Z Aquí

	B tratada en uno	se trata y no se transmite más. Al relanzarse B, dentro de una cita y no tratarse, se transmite a la tarea emisora que la trata en su único bloque.
C	C tratada en Sinc Others tratada en bloque Z C tratada en bloque Y D tratada en uno	La excepción C se trata en el accept Sinc de la tarea Dos, pero se lanza D. La excepción se propaga al bloque Z que contiene al accept. Aquí se trata anónimamente como others y se lanza C. La excepción se propaga al bloque Y que contiene al Z Aquí se trata y no se transmite más. Al lanzarse D, dentro de una cita y no tratarse, se transmite a la tarea emisora que la trata en su único bloque.
D	Others tratada en bloque Z C tratada en bloque Y D tratada en uno	D no se trata en la cita, por lo que se propaga al bloque Z que contiene al accept. Aquí se trata anónimamente como others y se lanza C. La excepción se propaga al bloque Y que contiene al Z Aquí se trata y no se transmite más. Al lanzarse D, dentro de una cita y no tratarse, se transmite a la tarea emisora que la trata en su único bloque.
Otra tecla	No aparece nada en pantalla	

Nota: Se puede admitir que el tratamiento de la excepción en la tarea Uno no sea precisamente la última, tal como aparece en esta solución.

Ejercicio

Escribir la especificación y cuerpo de un procedimiento genérico cuya función sea llamar periódicamente a otro procedure. Los parámetros del procedure a escribir serán:

El procedure a llamar (parámetro genérico).
El tiempo de la primera llamada (fecha y hora, puede ser en formato Time)
El intervalo entre llamadas en segundos.
El número de llamadas.
Si el tiempo de la primera llamada ya ha pasado se tomará la fecha y hora actual como inicio.

Ejercicio

Escribir un programa de prueba del procedure genérico, incluyendo todas las cláusulas with y las declaraciones necesarias. El procedimiento a llamar será:

```
procedure Pru is
begin
Put_Line("Hola");
delay 1.0;
Put_Line("Paso un segundo, adios");
new_line;
end Pru;
```

Ejercicio

Como es sabido el modelo de comunicaciones en Ada es síncrono y no buffereado. Debéis realizar un programa donde haya una tarea que funcione de manera semejante a un mailbox. Como respuesta a una orden send(…) recibirá mensajes de un tipo correo, que podemos considerar predefinido y los depositará en un buzón (zona de memoria). Cuando reciba una orden receive irá extrayendo los mensajes según un orden FIFO. Si el buffer se

llena no se podrá enviar nuevos mensajes hasta que quede espacio libre, quedando bloqueada la tarea emisora. Si no hay mensajes disponibles la tarea receptora se bloquea esperando algún mensaje.

Ejercicio

El siguiente programa se ejecuta tras haber sido compilado con la política Ceiling Priority que afecta al funcionamiento de los objetos protegidos. Se desea determinar cual será la salida del programa al ejecutarse.

Acompañar también un gráfico a la manera de cronograma donde se muestre para cada una de las tareas si se encuentra bloqueada, en espera, en ejecución o ejecutando el procedure protegido write. Para mayor claridad usad las siguientes convenciones:

— · — · — · — · —	Bloqueado	———————	Ejecutándose
═══════	En espera	.....	Ejecutando en el objeto

```
with Text_IO; use Text_IO;
with Ada.Integer_Text_IO; use Ada.Integer_Text_IO;
with calendar; use calendar;
with System;

procedure ceiling is

protected type ENTERO_COMPARTIDO(VALOR_INICIAL:Integer) is
   function READ return Integer;
procedure WRITE(NUEVO_VALOR:Integer);

private
  DATO:Integer:=VALOR_INICIAL;
end ENTERO_COMPARTIDO;

protected body ENTERO_COMPARTIDO is
function READ return Integer is
begin
  return DATO;
end READ;
procedure WRITE(NUEVO_VALOR:Integer) is
now: time;
begin
  now:=Clock;
  loop
    exit when (clock - now) > 4.0;
  end loop;
  DATO:=NUEVO_VALOR;
  put_line("WRITE,salgo del procedure protegido");
end WRITE;
end ENTERO_COMPARTIDO;

Inicio:Time;
```

```
Dato_Compartido : Entero_Compartido(10);

task Tarea_A is
pragma priority(System.Priority'last);
end Tarea_A;

task body Tarea_A is
begin
  Inicio:=clock;
  delay 7.4;
  put(integer(clock-inicio));
  put(Ascii.ht&Ascii.ht&"Desde la Tarea_A leo el entero y vale  ");
  put(Dato_Compartido.Read);new_line;
  put(integer(clock-inicio));
  put_line(Ascii.ht&Ascii.ht&"Desde la Tarea_A escribo el valor 20");
  Dato_Compartido.Write(20);
end Tarea_A;

task Tarea_M is
pragma priority(System.Priority'last-1);
end Tarea_M;

task body Tarea_M is
now: Time;
begin
  delay 0.8;
  put(integer(clock-inicio));
  put_Line(Ascii.ht&"Desde la Tarea_M escribo cosas");
  for i in 1..4 loop
    now:=clock;
    loop
      exit when (clock - now) > 2.0;
    end loop;
    put(integer(clock-inicio));
    put_line(Ascii.ht&"Desde Tarea_M escribo el valor de i "&Integer'image(i));
    Dato_Compartido.Write(i);
  end loop;
end Tarea_M;

now:time;
begin
  put(integer(clock-inicio));
  put("Desde el principal, tarea_B, leo el dato ");
  put(Dato_Compartido.Read); new_line;
  put(integer(clock-inicio));
  put_line("Desde el principal,tarea_B, escribo el valor 30");
  Dato_Compartido.Write(30);
  now:=clock;
  loop
    exit when (clock - now) > 1.0;
  end loop;
  put(integer(clock-inicio));
  put("Desde el principal, tarea_B, leo el dato ");
```

```
  put(Dato_Compartido.Read);
end ceiling;
```