Seminario de Programación en Ada

UNIVERSIDAD DE CANTABRIA

Bloque II

- **Programación concurrente**

Concurrencia en Ada

UNIVERSIDAD DE CANTABRIA

El Ada soporta la programación de actividades concurrentes mediante las tareas ("*tasks*").

Las tareas constan de dos partes: especificación y cuerpo

- **la especificación de una tarea declara su interfaz visible (puntos de entrada)**
- **el cuerpo define una actividad que se ejecuta independientemente; tiene declaraciones e instrucciones**

Las tareas van en la parte declarativa de un módulo

- **se arrancan nada más ser visibles**
- **el módulo que las declara no termina si no han terminado todas sus tareas**

Programación concurrente

UNIVERSIDAD DE CANTABRIA

```
procedure Control_del_Automóvil is

   task Medida_Velocidad; -- Especificacion de tarea anonima

   task Control_ABS;

   task Control_Inyeccion;

   task body Medida_Velocidad is -- Cuerpo
   --  Parte declarativa
   begin
      --  Instrucciones
      loop
         Acciones de Medida_de_Velocidad;
         Esperar al próximo periodo (20 ms);
      end loop;
   end Medida_Velocidad;
```

Programación concurrente (cont.)

UNIVERSIDAD DE CANTABRIA

```
   task body Control_ABS is  is
   begin
      loop
         Acciones de Medida_de_Velocidad;
         Esperar al próximo periodo (40 ms);
      end loop;
   end Control_ABS;

   task body Control_Inyección is
   begin
      loop
         Acciones de Control de Inyección;
         Esperar al próximo periodo (80 ms);
      end loop;
   end Control_Inyección;

begin  --  Activacion de tareas definidas
   null; --  El programa principal es otra tarea
end Control_del_Automovil;
```

Sincronización entre Tareas

UNIVERSIDAD DE CANTABRIA

Ada soporta dos mecanismos de interacción entre tareas

- **Explícito**
  - Mediante el intercambio de mensajes (***rendezvous***)
- **Implícito**
  - Mediante el acceso a datos compartidos (***objetos protegidos***)

Sincronización de espera

UNIVERSIDAD DE CANTABRIA

La sincronización de espera entre tareas en Ada se realiza mediante el mecanismo del "*rendezvous*" o punto de entrada:

- **Hay una tarea que llama, y otra que acepta el encuentro**
- **En la llamada se pueden pasar parámetros, como en un procedimiento**
- **La tarea que "llama" se queda esperando hasta que la tarea que acepta la llamada termine de ejecutarla**

Los puntos de entrada se declaran en la especificación

Tareas en Ada

UNIVERSIDAD DE CANTABRIA

Especificación

```
task NombreTarea is
   ...
   entry PtoEntrada(Params);
   ...
end NombreTarea;
```

Cuerpo

```
task body NombreTarea is
begin
   ...
   accept PtoEntrada(Params) do
   --  Instrucciones
   end PtoEntrada;
   ...
end NombreTarea;
```

Instrucción accept

- La tarea invocante pasa sus parámetros (**in/out**) a la tarea llamada y se suspende hasta la finalización de la cita.
- La tarea llamada ejecuta las instrucciones de la clausula **accept**.
- Los parámetros de salida (**out**) se devuelven a la tarea invocante.
- La cita está completa y ambas tareas pueden continuar su ejecución.

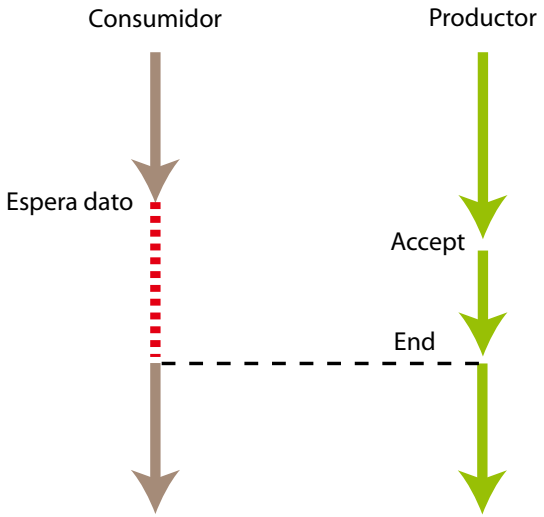
Ejemplo de sincronización de espera

UNIVERSIDAD DE CANTABRIA

Este ejemplo muestra a una tarea (consumidora) que debe esperar a un dato que le suministra la otra tarea (productora)

```
task Consumidor;

task Productor is
    entry Espera_Dato(El_dato : out Dato);
end Productor;
```



Ejemplo de sincronización de espera

UNIVERSIDAD DE CANTABRIA

```
task body Consumidor is
    Copia_del_dato : Dato;
begin
    -- Instrucciones
    Productor.Espera_Dato(Copia_Del_Dato);
    -- Mas instrucciones
end Consumidor;

task body Productor is
   Dato_Producido : Dato;
begin
   -- Instrucciones para calcular el valor de Dato_Producido
   accept Espera_Dato (El_Dato : out Dato) do
       El_Dato := Dato_Producido;
   end Espera_Dato;
   -- Otras instrucciones
end Productor;
```

Sincronización de datos

UNIVERSIDAD DE CANTABRIA

La sincronización de datos, para compartir información de manera mutuamente exclusiva, se realiza mediante *objetos protegidos*

El objeto protegido tiene una especificación con:

- ***parte visible*: tiene funciones, procedimientos y puntos de entrada**
- ***parte privada*: contiene los datos protegidos**

Con las funciones y procedimientos se garantiza un acceso mutuamente exclusivo a la información protegida.

Objetos protegidos

UNIVERSIDAD DE CANTABRIA

Especificación

```
protected ObjProtegido is
   function Fun (Params_1)
        return tipo;
   procedure Proc (Params_2);
   entry Ent (Params_3);

private
   --  Datos privados
   --  Procedimientos privados
end ObjProtegido;
```

Cuerpo

```
protected body ObjProtegido is
   -- Implementación
   -- NO se pueden declarar
   -- datos (parte privada)
end ObjProtegido;
```

Operaciones

- Procedure (procedimiento protegido)
  - Acceso exclusivo de lectura-escritura al objeto
- Function (función protegida)
  - Acceso concurrente de sólo lectura al objeto
  - Excluye a cualquier escritor
- Entry (entrada)
  - Como el procedure, pero además tiene asociada una barrera de entrada (condición **when**)
  - Si la barrera es falsa la tarea que llama se suspende
  - Cuando la barrera se hace verdadera se reanuda una de las tareas suspendidas

Ejemplo de sincronización de datos

UNIVERSIDAD DE CANTABRIA

```
type Coordenadas is record
   X,Y,Z : Float;
end record;


protected Datos_Avion is
   function Posicion return Coordenadas;
   procedure Cambia_Posicion (La_Posicion : Coordenadas);
private
   Posicion_Avion : Coordenadas;
end Datos_Avion;
```

Ejemplo de sincronización de datos

UNIVERSIDAD DE CANTABRIA

```
protected body Datos_Avion is

   function Posicion return Coordenadas is
   begin
      return Posicion_Avion;
   end Posicion;

   procedure Cambia_Posicion (La_Posicion : Coordenadas) is
   begin
      Posicion_Avion := La_Posicion;
   end Cambia_Posicion;

end Datos_Avion;
```

Puntos de entrada en objetos protegidos

UNIVERSIDAD DE CANTABRIA

Los *puntos de entrada* (entry) proporcionan la misma protección que los procedimientos protegidos, pero además:

- permiten a una tarea *esperar*, hasta que se cumpla una determinada condición
- la evaluación de esta condición también está *protegida*

El ejemplo que se muestra a continuación implementa una cola con datos:

- utiliza el tipo abstracto de datos “Cola”
- las operaciones de la cola (`Insertar`, `Extraer`, etc.) están protegidas
- la operación de `Extraer` hace que la tarea espere hasta que haya datos disponibles

Ejemplo con punto de entrada

UNIVERSIDAD DE CANTABRIA

```
with Colas, Elementos;
package Cola_Protegida is

   subtype Dato is Elementos.Elemento;

   protected Cola is
      procedure Inserta (El_Dato : Dato);
      entry Extrae (El_Dato : out Dato); -- si no hay datos, espera
      procedure Haz_Nula;
   private
      La_Cola : Colas.Cola;
   end Cola;

end Cola_Protegida;
```

Ejemplo con punto de entrada (cont.)

UNIVERSIDAD DE CANTABRIA

```
package body Cola_Protegida is
    protected body Cola is
       procedure Inserta (El_Dato : Dato) is
       begin
            Colas.Inserta (El_Dato,La_Cola);
       end Inserta;

       entry Extrae (El_Dato : out Dato)
           when not Colas.Esta_Vacia(La_Cola) is
       begin
            Colas.Extrae (El_Dato,La_Cola);
       end Extrae;

       procedure Haz_Nula is
       begin
            Colas.Haz_Nula (La_Cola);
       end Haz_Nula;
    end Cola;
end Cola_Protegida;
```

Tipos de datos concurrentes

UNIVERSIDAD DE CANTABRIA

Podemos crear tipos tareas u objetos protegidos, añadiendo la palabra reservada **type** a su especificación

```
task type NombreTarea is
   ...
end NombreTarea;

protected type Objeto_Protegido is
   ...
end Objeto_Protegido;
```

La declaración se realizará como cualquier otro objeto

```
Tarea_1, Tarea_2 : NombreTarea;
PV               : Objeto_Protegido;
```

También pueden crearse dinámicamente a través de los tipos acceso

Tipos de datos concurrentes (cont'd)

UNIVERSIDAD DE CANTABRIA

Se pueden emplear discriminantes para especializar nuestro tipo de datos concurrente

- Parte_discriminante: Parámetros que se pasan en la instanciación del tipo tarea en el instante de su creación. (Solo pueden ser parámetros de tipo discreto y tipo accesso)

```
task type NombreTarea (Identificador : Natural := 1) is
   ...
end NombreTarea;

Tarea_1 : NombreTarea (1);
Tarea_2 : NombreTarea (2);
```