

TEMAS 4 Y 5

- Comunicación: transferencia de información de un proceso a otro
- Sincronización: cumplir las restricciones de orden en el que se ejecutan las acciones de distintos procesos.

Formas de conseguir la comunicación:

- Memoria Compartida: Datos a los que pueden acceder más de un proceso
- Paso de mensajes: Intercambio explícito de datos entre dos procesos.

● Memoria Compartida:

Problemas:

- Uso de restricciones de variables compartidas presenta problemas de actualización
- Dos procesos que actualizan una variable compartida X mediante la instrucción.
- Uso de instrucciones no atómicas.

Exclusión Mutua:

- Sección crítica: Secuencia de sentencias que debe ser ejecutada de forma que presente un comportamiento atómico.
- Exclusión mutua: Sincronización necesaria para proteger una sección crítica. No puede haber más de un proceso simultáneamente en secciones críticas mutuamente excluyentes

● Objetos Protegidos:

Controlan el acceso a datos compartidos entre múltiples procesos, garantizan la exclusión mutua, proporciona sincronización condicional, mediante expresiones booleanas (barreras en el entry).

Declaración:

Se pueden declarar tipos de objetos protegidos, instancias únicas de objetos protegidos. Se debe de indicar la especificación y el cuerpo. Además son unidades de programa y no de compilación.

Métodos de acceso: Función protegida, procedimiento protegido, entrada (entry) protegida.

```
protected type NombreOP (Discriminante) is
    function NombreFunc(Parametros) return NombreTipo;
    procedure NombreProc(Parametros);
    entry Nombre_Ent1(Parametros);
```

Sólo métodos de acceso

```
private
    NombreObj : NombreTipo;
    entry Nombre_Ent2(Parametros);
```

Declaraciones de variables y métodos de acceso (opcional)

```
end NombreOP;
```

```
entry Nombre_Ent (Parametros) when condicion is
begin
    -- Acciones
end Nombre_Ent;
```

```

protected type Entero_Compartido(Valor_Inicial: Integer) is
    function Valor return Integer;
    procedure Incrementar;
    procedure Decrementar;

private
    Dato: Integer := Valor_Inicial;
end Entero_Compartido;

```

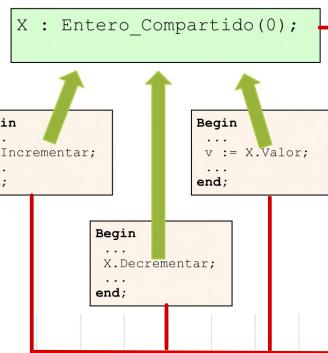
```

protected body Entero_Compartido is
    function Valor return Integer is
        begin
            return Dato;
        end;

    procedure Incrementar is
        begin
            Dato := Dato + 1;
        end;

    procedure Decrementar is
        begin
            Dato := Dato - 1;
        end;
end Entero_Compartido;

```



Sección crítica de nuestro objeto protegido.

Task o procesos que desean entrar en nuestra sección crítica.

Bloqueo de lectura de un objeto protegido:

Las funciones protegidas proporcionan acceso concurrente de lectura, una o más tareas están ejecutándose en funciones protegidas, las tareas que requieren "escribir" deben esperar.

Bloques de lectura + escritura de un objeto protegido:

Los procedimientos y entrys protegidos proporcionan acceso mutuamente exclusivo de lectura+escritura.

Sincronización Condicional: Entry protegido

Presentan interfaces muy similares a los procedimientos:

- Proporcionan acceso exclusivo de lectura-escritura a datos
- Protegidos por una barrera condicional

```
entry Nombre (parámetros) when Barrera is
```

- Si la barrera es falsa el proceso se encola
- Cuando la barrera sea cierta se puede ejecutar el cuerpo del entry.

- Las barreras se evalúan cuando se llama al entry
- La barrera se evalúa cuando se termina la ejecución.

```

Max_Elementos : constant Integer := 5;
type Index is mod Max_Elementos;
type Buffer is array (Index) of Elemento;

protected type ColaCircular is
    entry Quitar (Item : out Elemento);
    entry Añadir (Item : in Elemento);
private
    Primero : Index := Index'First;
    Ultimo : Index := Index'Last;
    Contador : Integer := 0;
    Buf : Buffer;
end ColaCircular;

```

```

protected body ColaCircular is
    entry Quitar (Item : out Elemento) when Contador /= 0 is
    begin
        Item := Buf(Primero);
        Primero := Primero + 1;
        Contador := Contador - 1;
    end Quitar;
    entry Añadir (Item : in Data_Item)
        when Contador /= Max_Elementos is
    begin
        Ultimo := Ultimo + 1;
        Buf(Ultimo) := Item;
        Contador := Contador + 1;
    end Añadir;
end ColaCircular;

```



Acceso a escritura: Las tareas esperando en barreras tienen preferencia sobre las que esperan acceder al objeto protegido.

Atributo Count: indica el número de tareas en la cola de un entry.

Excepciones y objetos protegidos: cualquier excepción provocada y no controlada en una tarea protegida se propaga a la tarea invocante.

Una excepción que se provoque durante la evaluación de una barrera se convierte en un Program-error lanzado a todas las tareas encoladas.

Llamada Condicional a un entry: Sentencia Select con la alternativa else

```
task body Tarea is
begin
...
select
  OP.Entradal(...) -- llamada al entry Entradal del objeto protegido OP
  -- sentencias opcionales
else
  -- sentencias
end select;
...
end Tarea;
```

La tarea no espera en la cola del entry si su barrera es falsa

Llamada temporizada a un entry: Sentencia select con alternativa or delay

```
task body Tarea is
begin
...
select
  OP.Entradal(...)
  -- sentencias opcionales
or
  delay 10.0; -- también puede utilizarse delay until
  -- sentencias opcionales
end select;
...
end Tarea;
```

La tarea estará encolada en el entry como máximo durante 10 segundos

Transferencia Asíncrona de Control: Sentencia Select con la alternativa then abort.

```
task body Tarea is
begin
...
select
  OP.Entradal(...)
  -- sentencias opcionales
then abort
  -- sentencias abortables y que se ejecutan durante la espera.
end select;
...
end Tarea;
```

Se realiza la llamada al entry del espacio protegido

En el caso de que se encole la tarea, las sentencias de debajo del then abort se seguirán ejecutando hasta que se entre al espacio protegido

Enviar y Recibir mensajes:

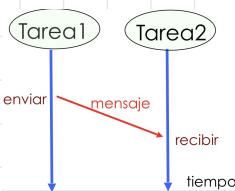
Los procesos se pueden comunicar y sincronizar mediante mensajes, un process envía mensajes y otro espera a recibirlas.

Método de Sincronización.

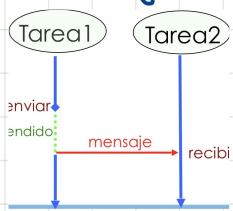
- Sincronización implícita: el receptor no puede obtener un mensaje antes de que este se haya enviado

- Sincronización Asíncrona: el emisor continua su ejecución independientemente de si se ha recibido el mensaje.

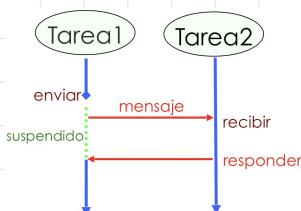
Se requiere un búfer para almacenar los mensajes, si es limitado puede bloquearse el emisor.



- **Comunicación Síncrona:** el emisor continua solo cuando se ha recibido el mensaje, no hace falta almacenarlo



- **Invocación Remota:** el emisor continua solo cuando se ha devuelto una respuesta desde el receptor. Se llama Cita Extendida.



Identificación de Procesos

- **Identificación Directa:** el emisor identifica explícitamente el receptor

`send mensaje to nombre_proceso`

- **Identificación Indirecta:** Se utiliza una entidad intermedia, facilita la descomposición del software

`send mensaje to buzón_correo`

- **Identificación Simétrica:** el emisor y el receptor se pueden identificar entre sí.

`send mensaje to nombre_proceso`
`receive mensaje from nombre_proceso`
`send mensaje to buzón_correo`
`receive mensaje from buzón_correo`

- **Identificación Asimétrica:** El receptor no identifica un origen específico, sino que acepta mensajes de cualquier emisor.

Paso de mensajes en Ada

- **Modelo de sincronización de cita extendida:** la tarea que llega primero espera a la otra, las tareas esperan respuesta.
- **Intercambio de información mediante parámetros**
- **Se proporciona exclusión mutua:** Una tarea no puede mantener varias citas a la vez.
- **Cita basada en modelo cliente/servidor:** el servidor ofrece un servicio

Entry:

- Para que una tarea pueda recibir un mensaje debe tener definido en un entry.
- Cada entry indica el nombre del servicio, los parámetros necesarios para la petición y los resultados que devolverá.
- La tarea que realiza la llamada "conoce" a la tarea que llama.

Accept:

- Para que una tarea pueda servir un mensaje debe de tener definido un accept.
- La sentencia accept permite al servidor atender una petición.
- La tarea que sirve la petición no conoce a la tarea que manda el mensaje.

Tarea servidora	Tarea cliente
<pre>task Op_Telef is entry Consulta_Listin (Persona : in Nombre; Num : out Numero); end Op_Telef; task body Op_Telef is begin ... loop -- preparada para aceptar la siguiente -- llamada accept Consulta_Listin (Persona : in Nombre; Num : out Numero) do -- buscar el numero de telefono end Consulta_Listin; ... end loop; ... end Op_Telef;</pre>	<pre>task type Abonado; task body Abonado is personra : Nombre; Num : Numero; begin ... -- invocación a un entry de una tarea Op_Telef.Consultar_Listin(personra,num); ... end Abonado;</pre>

Atributo Count de un Entrg:

Nombre_entry'Count devuelve el número de tareas de la cola del entry, solo es accesible desde el cuerpo de la tarea propietaria del entry, se incrementa con nuevos llamados y decremente cuando: se atiende a una petición, expiración de una llamada, aborto de una tarea o ATC en llamada al entry.

Restricciones del Accept:

- Se puede colocar únicamente en el cuerpo de una tarea.
- Se puede anidar, aunque no para el mismo entry.
- Debería haber un accept para cada entry.

Manejo de Excepciones.

```
accept Calcular(x, y: in Float; res: out Float) do
    -- realizar cálculos
    declare
        r: Float;
    begin
    ...
    res := ...;
exception
    when Numeric_Error => ... ;
    end;
exception
    when Constraint_Error => ... ;
end Calcular;
```

Si no se maneja la excepción dentro del accept la excepción se lanza en la tarea invocadora y en la receptora.

La excepción Tasking_Error se puede generar cuando se realiza una llamada a un entry de una tarea que no está activa, o se está encolando en un entry de una tarea que termina.

Espesa Selectiva en el Servidor

La sentencia Select... accept permite al servidor:

- Esperar más de una cita a la vez (or accept)
- Retira la oferta de comunicación si ninguna cita está disponible inmediatamente (else)
- Detener la espera si ninguna cita ha llegado en un tiempo especificado (or delay)
- Terminar si no hay clientes que puedan llamar a sus entradas (or terminate)

Servidor

```
task Servidor is
  entry S1(...);
  entry S2(...);
end Servidor;
```

Alternativa Or Accept

```
task body Servidor is
  ...
  begin
    loop
      select
        accept S1(...) do
          -- código para el servicio S1
        end S1;
        -- secuencia de sentencias opcionales
      or
        accept S2(...) do
          -- código para el servicio S2
        end S2;
        -- secuencia de sentencias opcionales
      or
        accept
        ...
      end select;
    end loop;
  end Servidor;
```

en cada iteración del bucle se establece una cita

Alternativa Else

```
task body Servidor is
begin
  loop
    select
      accept S1(...) do
        -- código para el servicio S1
      end S1;
      -- secuencia de sentencias opcionales
    or
      accept S2(...) do
        -- código para el servicio S2
      end S2;
      -- secuencia de sentencias opcionales
    else
      -- código ejecutado si no se establece ninguna cita inmediatamente
    end select;
  ...
end loop;
end Servidor;
```

no hay ninguna tarea encolada en algún entry cuando se ejecuta la sentencia select

Alternativa Delay

```
task body Servidor is
begin
  loop
    select
      accept S1(...) do
        -- código para el servicio S1
      end S1;
      -- secuencia de sentencias opcionales
    or
      delay 5.0;
      -- código ejecutado si no se establece la cita en 5 segundos
    end select;
  ...
end loop;
end Servidor;
```

no hay ninguna tarea encolada en el entry cuando se ejecuta la select ni es invocado en los siguientes 5 segundos

Alternativa Terminate

```
task body Servidor is
begin
  loop
    select
      accept S1(...) do
        -- código para el servicio S1
      end S1;
      -- secuencia de sentencias opcionales
    or
      accept S2(...) do
        -- código para el servicio S2
      end S2;
      -- secuencia de sentencias opcionales
    or
      terminate; -- la tarea termina
    end select;
  ...
end loop;
end Servidor;
```

El servidor termina si no hay ninguna tarea que pueda solicitar sus servicios

Espesa Selectiva en el cliente:

La sentencia Select... llamada entry permite al cliente:

- No esperar si el servidor no está disponible inmediatamente para establecer la cita (else)
- Dejar de esperar si el servidor no ha llegado a la cita en un tiempo especificado (or delay)
- Abortar la ejecución de acciones que se han estado realizando durante la espera (then abort)

Alternativa Else Cliente

```
task body Cliente is
begin
loop
...
select
  -- llamada al entry S1 de la tarea Servidor
  Servidor.S1(...);
  -- secuencia de sentencias opcionales
else
  -- código ejecutado si la cita no se establece inmediatamente
end select;
...
end loop;
end Servidor;
```

el cliente no se encola en el entry si el servidor no le puede atender inmediatamente

Alternativa delay Cliente

```
task body Cliente is
begin
loop
...
select
  -- llamada al entry S1 de la tarea Servidor
  Servidor.S1(...);
  -- sentencias opcionales
or
  delay 5.0;
  -- código ejecutado si no se establece la cita en 5 segundos
end select;
...
end loop;
end Servidor;
```

El cliente estará encolado en el entry como máximo durante 5 segundos

Alternativa then Abort cliente

```
task body Cliente is
begin
loop
...
select
  -- llamada al entry S1 de la tarea Servidor
  Servidor.S1(...);
  -- sentencias opcionales
then abort
  -- sentencias abortables
end select;
...
end loop;
end Servidor;
```

El cliente mientras está encolado en el entry puede estar ejecutando sentencias

Práctica 4 explicación. En la práctica 4 hay que evitar que los aviones colisionen al generarse en las aerovías, para ello hay que generar un espacio protegido, lo que quiere decir una sección crítica.

1º Paso creamos el paquete protegido:

pkg-protected.ads

package pkg-protected is

subtype contador is Integer range 0...MAX_Aviones_Aerovia; Subtype para el máximo de aviones por aerovía

function Posicion_Rejilla (pos_x : T_Coordenada) return T_Rango_Rejilla_x;

protected type espacio_Aereo_Protegido is

function esLibre (pos_x : T_Rango_Rejilla_x) return Boolean; → Dice si un avión cabe
procedure ocupar (pos_x : T_Rango_Rejilla_x); ocupa una zona de la aerovía
procedure liberar (pos_x : T_Rango_Rejilla_x); libera una zona de la aerovía.
procedure cambiar (bloq : T_Rango_Rejilla_x; lib : T_Rango_Rejilla_x); cambia una posición bloqueado por un avión a libre.

function Cantidad return contador; → Devuelve la cantidad de aviones en la aerovía

procedure incrementar_Aviones;

procedure decrementar_Aviones;

entry Aerovia_Disponible (posx : T_Rango_Rejilla_x);

private

aire : T_Rejilla_Ocupacion := (others => False);

C : contador := 0;

end espacio_Aereo_Protegido;

array de aerovías para colocar a los aviones

espacioAereo : array (T_Rango_Aerovia) of espacio_Aereo_Protegido;

end pkg_protected;

→ Entry para entrar a la sección crítica

pkg_protegidos.adb

package body pkg_protegidos is

function Posicion_Rejilla (posx : T_coordenado_x) return T_Rango_Rejilla_x is
begin
return T_Rango_Rejilla_x (posx * TAM_X_Rejilla / (X_INICIO_DER_AVION + 1));
end Posicion_Rejilla;

To para digir
un espacio entre
aviones.

protected body espacioAereoProtegido is
function esLibre (posx : T_Rango_Rejilla_x) return Boolean is
begin
return not aire (posx);
end esLibre;
procedure ocupar (posx : T_Rango_Rejilla_x) is
begin
aire (Posx) := True;
end ocupar;
procedure liberar (posx : T_Rango_Rejilla_x) is
begin
aire (Posx) := False;
end liberar;
procedure cambiar (blog : T_Rango_Rejilla_x; lib : T_Rango_Rejilla_x) is
begin
liberar (lib);
ocupar (blog);
end Cambiar;

function cantidad return contador is
begin
return C;
end cantidad;

function incrementarAviones is
begin
C := C + 1;
end incrementarAviones;

function incrementarAviones is
begin
C := C + 1;
end incrementarAviones;

Entry AeroviaDisponible (posx : T_Rango_Rejilla_x) when

not aire (T_Rango_Rejilla_x'First)
and not aire (T_Rango_Rejilla_x'Last)
and not aire (T_Rango_Rejilla_x'First + 1)
and not aire (T_Rango_Rejilla_x'Last - 1)
and C <= MAX_AVIONES_AEROVIA is

Comprobamos que las celdas
de aire estan disponibles y que
no superamos el maximo de aviones.

begin
if posx = T_Rango_Rejilla_x'First then aire (T_Rango_Rejilla_x'First) := TRUE;
else aire (T_Rango_Rejilla_x'Last) := TRUE;
end AeroviaDisponible;

Acceso al espacio protegido por parte de los aviones

Task body TareaAvion is

rejillaSigiente : T_Rango_Rejilla_x;

rejillaActual : T_Rango_Rejilla_x

avion : T_Record_Avion

procedure Avanza is

begin

 rejillaSigiente := Posicion_Rejilla (nueva_Posicionx (avion.Pos.x, avion.velocidad.x));
 rejillaActual .= Posicion_Rejilla (avion.Pos.x);

 if rejillaActual /= rejillaSigiente then

 'espacioAereo(Avion.aerovia).cambiar(rejillaSigiente, rejillaActual);

 end if

 Actualiza_Movimiento (avion);

 delay Duration (Retardo_Movimiento);

end Avanza,

begin

 avion := ptc_avion.all

 Comprobamos la disponibilidad de aparición haciendo la llamada al entry
 espacioAereo(Avion.aerovia).Aerovia_Disponible (Posicion_Rejilla(avion.Pos.X));

(D En cuanto la condición del entry se cumpla se procede a la Aparición.

 Aparece (Avion);

 loop

 Avanza

 end loop

exception

 When event: Detectada_Colision =>

 Desaparece (Avion);

 When event: Other=>

Le solicitamos al espacio protegido
el cambiar las celdas ocupadas

TEMA 5

Problema del control de recursos:

- Se requiere sincronización entre procesos que comparten acceso a recursos limitados
- La asignación de recursos entre procesos competitivos puede afectar a la fiabilidad del sistema
 - Fallo de un proceso con un recurso asignado
 - Monopolizar un recurso
 - Interbloqueo
- El control de recursos se puede llevar a cabo mediante
 - Un recurso protegido
 - Un servidor

Implementar la gestión de recursos

Los recursos deben encapsularse y ser accedidos a través de una interfaz procedimental de alto nivel, siguiendo los principios de modularidad y ocultación de información.

Primitivas de Sincronización

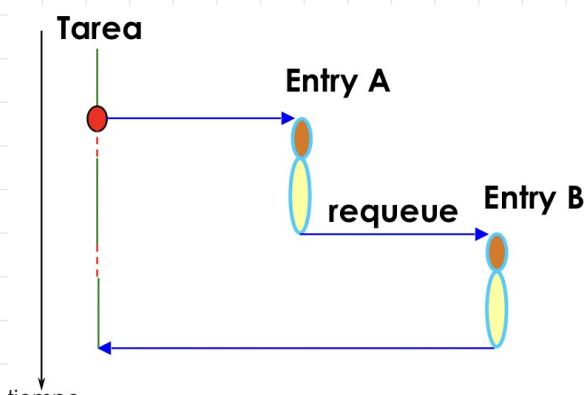
- Se requiere evaluar las primitivas de sincronización de un lenguaje
- ADA permite implementar servidores, con una interfaz de paso de mensajes
- Recursos protegidos: mediante objetos protegidos.

Funcionalidad de reencolado en ADA.

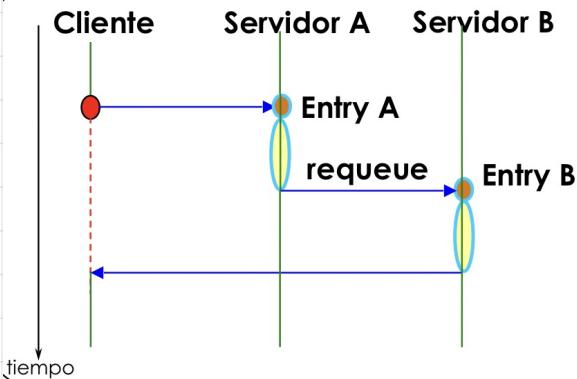
requeue Nombre_Entry; El reencolado consiste en mover una tarea X que estaba encolada en un entry A a la cola de otro entry B cuando se ejecuta el entry A. Es decir, desde el cuerpo de un entry se puede "redirigir" la tarea cliente a la cola de otro entry (o al mismo_si $A=B$);

- El reencolado no es una simple llamada
 - Cuando el cuerpo del entry A ejecuta un requeue, ese cuerpo se completa.
 - La tarea X continua suspendida hasta que el cuerpo del entry B se ejecute.

Sincronización en el reencolado desde un objeto protegido.



Sincronización en el reencolado desde una cita extendida.



• Paso de parámetros en el requeue

- No se proporcionan parámetros actuales en la llamada al entry en la sentencia Requeue
- El entry nombrado en la sentencia Requeue (entry objetivo): o no tiene parámetros, o si los tiene, debe ser el mismo número y tipo de parámetros que el entry desde el que se hace el reencolado.
- Los valores de los parámetros de salida pueden cambiarse antes del requeue.
- En la barrera de un entry no pueden utilizarse los parámetros de entrada.

• Uso de la sentencia requeue:

- Se permiten reencolados entre entries de tareas y sus objetos protegidos.
- El reencolado puede ser al mismo entry a otro entry en la misma unidad, o un entry en otra unidad.

Requeue en objetos protegidos:

- Si se realiza un reencolado de un objeto protegido a otro, la exclusión mutua sobre el objeto original se abandona.
- Si se realiza un reencolado sobre el mismo objeto protegido, mantendrá el bloqueo de exclusión mutua (si el entry objetivo está abierto).

Llamadas temporizadas: Cláusula With Abort

requeue Nombre_Entry **with abort;**

- Mantien cualquier timeout
- Permite que en la tarea reencolada pueda cancelarse la llamada (se desencole si se alcanza el timeout)

requeue Nombre_Entry ;

- Cancela cualquier timeout
- En la tarea reencolada no puede cancelarse la llamada (No se desencola si alcanza timeout).

Ejemplo con cláusula with abort:

Tarea1	Tarea2	Tarea3
-- ejecución en -- instante t_0 select Tarea2.Servicio; or delay 5.0; end select; ...	-- ejecución en -- instante $t_1 = t_0 + 3.0$ Accept Servicio do requeue Tarea3.Servicio with abort; end Servicio; ...	-- ejecución en -- instante $t_1 + 4.0$ Accept Servicio do Procedimiento_A; end Servicio; ...

Asignación de N recursos

```

type Rango_Peticiones is range 1..Max;
type Recurso ...;

protected Controlador_Recursos is
    entry Solicitar(R: out Recurso; cantidad: Rango_Peticiones);
    procedure Liberar(R: Recurso; cantidad: Rango_Peticiones);

private
    entry Asignar(R: out Recurso; cantidad: Rango_Peticiones);
    liberados: Rango_Peticiones := Rango_Peticiones'Last;
    nuevos_recursos_liberados : Boolean := False;
    a_intentar: Natural := 0;
    ...
end Controlador_Recursos;

```

```

...
procedure Liberar(R: Recurso; cantidad: Rango_Peticiones) is
begin
    liberados := liberados + cantidad;
    -- liberar recursos
    if Asignar'Count > 0 then
        a_intentar := Asignar'Count;
        nuevos_recursos_liberados := True;
    end if;
end Liberar;
end Controlador Recursos;

```

Asignación de N recursos

```

protected body Controlador_Recursos is
    entry Solicitar(R: out Recurso; cantidad: Rango_Peticiones) when liberados > 0 is
    begin
        if cantidad <= liberados then
            liberados := liberados - cantidad;
            -- asignar recursos
        else
            requeue Asignar;
        end if;
    end Solicitar;

    entry Asignar(R: out Recurso; cantidad: Rango_Peticiones) when nuevos_recursos_liberados is
    Begin
        a_intentar := a_intentar - 1;
        if a_intentar = 0 then
            nuevos_recursos_liberados := False;
        end if;
        if cantidad <= liberados then
            liberados := liberados - cantidad;
            -- asignar recursos
        else
            requeue Asignar;
        end if;
    ... end Asignar;

```

Temporizar el tiempo de respuesta del servidor

```

Task Cliente;
Task Servidor is
    entry Servicio(res: out Float);
private
    entry Servicio_Realizado(res: out Float);
End Servidor;

```

```

task body Cliente is
    respuesta : Float;
    begin
        select
            Servidor.Servicio(respuesta);
        or
            delay 10.0;
        end select;
        ...
    end Cliente;

```

```

task body Servidor is
    x : Float;
    begin
        accept Servicio(res:out Float) do
            requeue Servicio_Realizado with abort;
        end Servicio;
        Calcular_Resultado(x);

        select
            accept Servicio_Realizado(res:out Float) do
                res := x;
            end Servicio_Realizado;
        else
            null;
        end select;
        ...
    end Servidor;

```



Práctica 5: para la práctica 5 debemos generar la torre de control la cual permitirá a los aviones descender entre las aerovías y aterrizar. Para ello generamos un nuevo paquete con una Task anónima

```
package pkg_torre_control is
    task TorreDeControl is
        entry SolicitarDescenso (avion : in TRecordAvion; concedido : out Boolean;
            pistaAvion : out TPistaAterrizaje);
    end TorreDeControl;
end pkg_Torre_Control;
```

```
package body pkg_Torre_Control is
    task body TorreDeControl is
        pista : TPistaAterrizaje := PISTA1
        begin
            delay 20.0;
            loop
                accept SolicitarDescenso (avion: in TRecordAvion; concedido :out Boolean;
                    pistaAvion : out TPistaAterrizaje)
                do
                    if espacioAereo(avion.aerovia+1).cantidad /= MAX_AVIONES_AEROVIA
                    then
                        concedido := True;
                        if avion.aerovia + 1 /= TRange_Aerovia'Last then
                            espacioAereo(avion.aerovia+1).incrementarAviones;
                        end if;
                        if avion.pista = SIN_Pista then
                            pistaAvion := pista;
                            pista := (if pista = PISTA1 then PISTA2 else PISTA1);
                        end if;
                    else
                        concedido := False;
                    end if;
                end SolicitarDescenso;
                delay 7.0
            end loop;
        end TorreDeControl;
    end pkg_Torre_Control;
```

task body TareaAvion is

rejillaSigiente := T_Rango_Rejilla_x;
rejillaActual := T_Rango_Rejilla_x
avion := T_Record_Avion

procedure Avanza is

begin

rejillaSigiente := Posicion_Rejilla (nueva_Posicionx (avion.Pos.x, avion.velocidad.x));
rejillaActual := Posicion_Rejilla (avion.Pos.x);

if rejillaActual /= rejillaSigiente then
espacioAereo (avion.aerovia). cambiar (rejillaSigiente, rejillaActual);

end if

Actualiza_Movimiento (avion);

delay Duration (Retardo_Movimiento);

end Avanza,

procedure Baja is

begin

rejillaActual := Posicion_Rejilla (avion.Pos.x);

espacioAereo (avion.aerovia). liberar (rejillaActual);

espacioAereo (avion.aerovia + 1). ocupar (rejillaActual);

Desaparece (avion);

avion.aerovia := avion.aerovia + 1;

avion.velocidad.x := (if avion.aerovia rem 2 = 0 then VELOCIDAD_VUELO else -VELOCIDAD_VUELO)

avion.pos = Pos_Inicio (avion.pos.x, avion.aerovia)

IF avion.pista = SIN_Pista then

avion.pista := PISTA;

avion.color := (if Pista = the Green else Red);

end if;

Aparece (avion);

espacioAereo (avion.aerovia - 1). decrementar_Aviones;

end Baja;