

Tema 5

Control de Recursos



Objetivos

1. Comprender el problema de control de recursos en la programación concurrente
2. Conocer y saber utilizar la sentencia *requeue* de Ada



Índice

1. **Sincronización en el control de recursos**
2. La funcionalidad de reencolado en Ada
3. Ejemplos de aplicación de reencolado



El problema de 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 principios de modularidad y ocultación de información
- En Ada se utilizan los paquetes



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

El paquete debería contener

- una **tarea** (servidor)
- o bien un **objeto protegido** (recurso protegido)



Índice

1. Sincronización en el control de recursos
2. **La funcionalidad de reencolado en Ada**
3. Ejemplos de aplicación de reencolado



Semántica de requeue

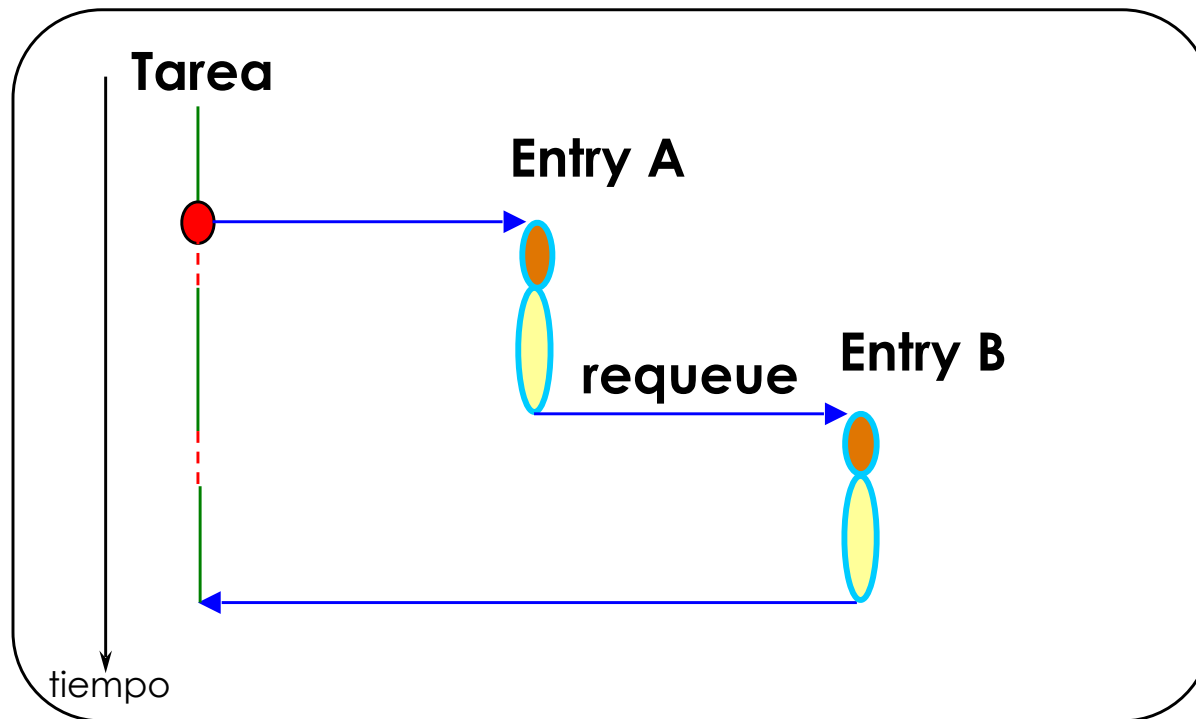
requeue *Nombre_Entry* ;

- El reencolado consiste en mover una tarea X que estaba en la cola de 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 **continúa suspendida** hasta que el cuerpo del entry B se ejecute



2. La funcionalidad de reencolado en Ada

Sincronización en el reencolado desde un objeto protegido

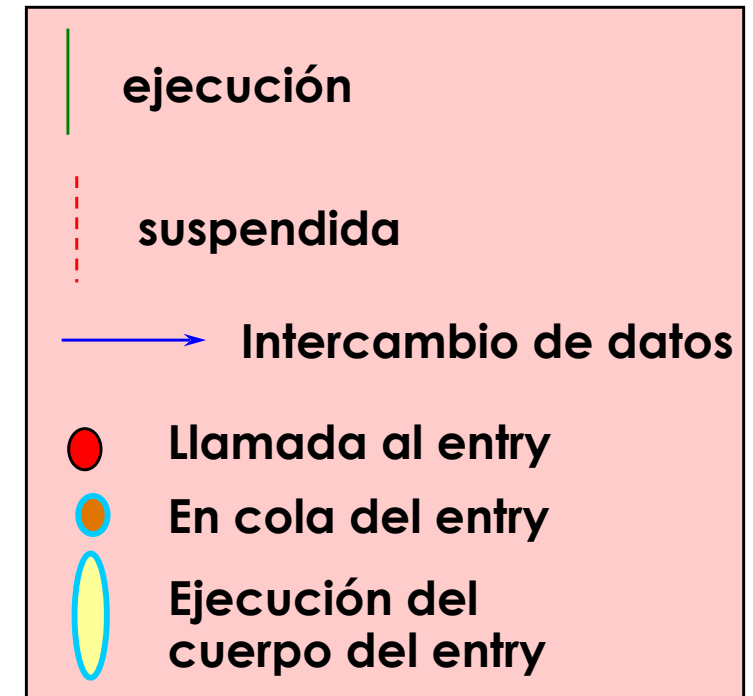
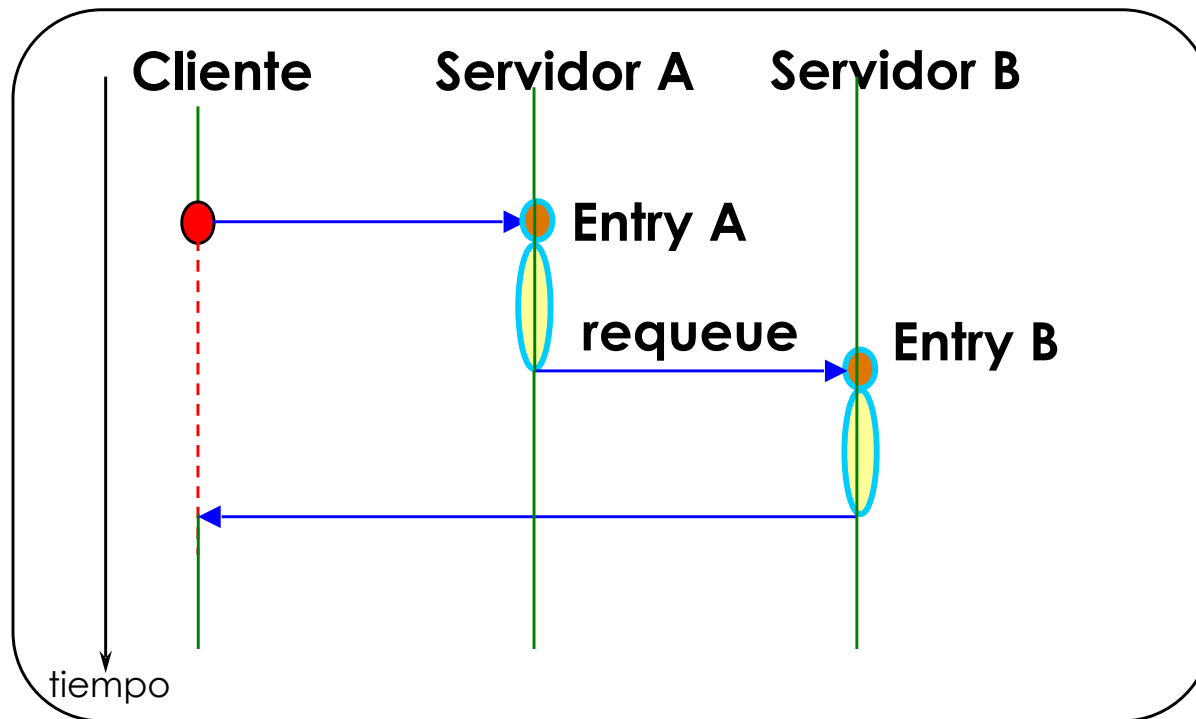


La tarea estará suspendida mientras no pueda ejecutar el cuerpo de un entry



2. La funcionalidad de reencolado en Ada

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 de objetos protegidos
- El reencolado puede ser para:
 - el mismo entry
 - otro entry en la misma unidad
 - un entry de 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;
```

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

```
requeue Nombre_Entry ; -- sin cláusula with abort
```

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



Ejemplo con cláusula with abort

Tarea1

```
-- ejecución en
-- instante  $t_0$ 
select
  Tarea2.Servicio;
or
  delay 5.0;
end select;
...
```

Tarea2

```
-- ejecución en
-- instante  $t_1 = t_0 + 3.0$ 
Accept Servicio do
  requeue Tarea3.Servicio
  with abort;
end Servicio;
...
```

Tarea3

```
-- ejecución en
-- instante  $t_1 + 4.0$ 
Accept Servicio do
  Procedimiento_A;
end Servicio;
...
```

¿ Se invoca a Procedimiento_A ?



Índice

1. Sincronización en el control de recursos
2. La funcionalidad de reencolado en Ada
3. **Ejemplos de aplicación de reencolado**



Asignación de N recursos

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

protected Controlador_Recurso 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_Recurso;
```



3. Ejemplos de aplicación de reencolado

Asignación de N recursos

```
protected body Controlador_Recurso 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;
```



Asignación de N 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;
```



3. Ejemplos de aplicación de reencolado

Enrutamiento



3. Ejemplos de aplicación de reencolado

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;
```



Conclusiones

- Generalmente las tareas concurrentes necesitan sincronizarse para acceder a recursos compartidos
- Ada dispone de citas extendidas y objetos protegidos para proporcionar mecanismos de sincronización
- La sentencia requeue de Ada permite expresar sincronización basada en parámetros de la llamada




Bibliografía Recomendada

Sistemas de tiempo real y lenguajes de programación (**3ª edición**)

Alan Burns and Andy Wellings

Addison Wesley (2002)

 **Capítulo 11** (Apartados 2, 4 y 7, excepto todo lo referente a otros lenguajes todo lo referente a otros lenguajes)

Concurrency in Ada (**2nd edition**)

Alan Burns and Andy Wellings

Cambridge University Press (1998)

 **Capítulos 8** (Completo)

