

PROGRAMACION CONCURRENTE

**Sincronización basada en memoria
compartida:**

Regiones críticas Condicionada

Regiones críticas

- Son bloques de código que al ser declarados como regiones críticas respecto de una variable, el compilador introduce en ella los mecanismos de sincronización necesarios para que su ejecución se realice en régimen de exclusión mutua respecto de otras regiones críticas declaradas respecto de la misma variable.
- Las regiones críticas se definen para garantizar la actualización segura a una variable compartida. Si una variable se declara como variables compartida (Shared), su acceso solo se puede realizar en regiones críticas y por tanto, todos los accesos son realizador en régimen de exclusión mutua.
- El concepto fue introducida por Brinch Hansen en 1972

Declaración de una región crítica

- La definición de una región crítica implica la declaración de dos
- elementos:
 - La variable compartida respecto de la que se define la zona crítica: **var** registro: **shared** Tipo_Registro;
 - Cada bloque de código que se requiere que se ejecute en régimen de exclusión mutua.

region registro **do** Bloque_de_código;

- Un proceso que trata de ejecutar una región crítica, compite con otros procesos que también lo intentan:
 - Si gana el acceso ejecuta su bloque en régimen exclusivo.
 - Si pierde el acceso se encola en la lista asociada a la variable compartida y se suspende a la espera de que la región crítica quede libre.
- Regiones críticas relativas a variable compartidas diferentes se pueden ejecutar concurrentemente.

Regiones crítica condicionales

- Las **regiones críticas condicionales** se definen en función de dos componentes:
 - [?] Declarar una variable compartida
var variable **shared** TipoVariable;
 - [?] Declaración del bloque crítico
region variable **when** expresión_Booleana **do** Bloque_de_Sentencias;
- **Semántica:**
 - El proceso que ejecuta la región crítica debe obtener el acceso exclusivo a la variable compartida.
 - Sin liberar el acceso exclusivo se evalúa la expresión booleana:
 - Si resulta True: Se ejecuta el código del bloque.
 - Si resulta False: El proceso libera la exclusividad y queda suspendido a la espera del acceso.
 - Cuando un proceso concluye la ejecución del bloque protegido, libera el acceso a la variable y se da acceso a otro proceso que se encuentre en la lista de espera de la variable.

Regiones crítica condicionales

- Las regiones críticas tal como se han definido en el apartado anterior son muy útiles y simples, pero no proporcionan solución a todos los requerimientos de interacción entre procesos concurrentes. En especial no son capaces de simular un semáforo o establecer ciertos mecanismos de sincronización. Por ello se ha extendido su definición al nuevo concepto de regiones críticas condicionales que sí que constituyen una solución completa.
- Las regiones **críticas condicionales (CCR)**, se definen en función de dos componentes:
 - Una variable compartida (shared) definida a través de una sentencia declarativa, tal como,

```
var Variable : shared Tipo_variable;
```

 - equivalente a la definida para la región crítica.
- Un bloque de sentencias que se ejecuta en régimen de exclusión mutua respecto de otras regiones declaradas respecto a la misma variable compartida, y a la que se accede solo si tras evaluar una expresión booleana de guarda, esta proporciona el valor "true".
 - El formato que utilizaremos para esta sentencia es:

```
region Variable when Expresión_booleana do  
Bloque_de_sentencias;
```
- La Expresión_booleana de guarda se refiere habitualmente a la variable compartida, y se evalúa dentro del régimen de exclusión mutua.

Ejemplo: Productor-Consumidor con buffer finito

```
program Productor_consumidor;  
  const LONG_BUFFER = 5;  
  type TipoDato = .....  
    TipoBuffer = record dato: array [1..LONG_BUFFER] of TipoDato;  
      nextIn, nextOut, cuenta: Integer; end;  
  var buffer : shared TipoBuffer;  
  process Productor begin ... end;  
  process Consumidor; begin ... end;  
  begin  
    region buffer do begin  
      buffer.nextIn:=1; buffer.nextOut:=1; buffer.cuenta:=0; end;  
      cobegin Productor; Consumidor; coend;  
  end.
```

Ejemplo: Productor-Consumidor con buffer finito

```
process Productor;  
  var dato: TipoDato;  
  begin  
    repeat  
      .... -- Produce el dato  
      region Buffer when (buffer.cuenta < LONG_BUFFER) do begin  
        buffer.datos[nextIn] := dato;  
        buffer.nextIn := (buffer.nextIn mod LONG_BUFFER) + 1;  
        buffer.cuenta := buffer.cuenta + 1;  
      end;  
    forever;  
  end;
```

Ejemplo: Productor-Consumidor con buffer finito

```
process Consumidor;  
  var dato: Tipo_dato;  
  begin  
    repeat  
      region buffer when (buffer.cuenta>0) do begin  
        buffer.datos[nextIn]:= dato;  
        buffer.nextOut := (buffer.nextOut mod LONG_BUFFER)  
        +1;  
        buffer.cuenta:= buffer.cuenta - 1;  
      end;  
    forever  
  end;
```


Crítica de las regiones críticas

- Las regiones críticas presenta un nivel de abstracción muy superior al semáforo, su uso es menos proclive al error.
- Sin embargo presenta los siguientes problemas:
 - Las sentencias relativas a una misma variable compartida está dispersa por todo el código de la aplicación, esto dificulta su mantenimiento.
 - La integridad de las variables compartidas está comprometida, ya que no hay ninguna protección sobre lo que el programador de cualquier módulo puede realizar sobre ella.
 - Existen dificultades para implementar eficientemente las regiones críticas. Requieren asociar a cada variable compartida dos lista, la “lista principal” en la que esperan los procesos que tratan de acceder y la “lista de eventos” en la que se encolan los procesos que tras acceder no han satisfecho la condición de guarda

Problemas de la región crítica condicional

- Las regiones críticas condicionales constituyen por su mayor nivel de abstracción un avance respecto de los semáforos, pero presentan aún un **conjunto de problemas**:
 - • **Las sentencias** relativas a una misma región crítica condicional, se encuentran **dispersas por todo el código** del programa. Lo que hace que el mantenimiento de los aspectos relativos a una variable compartida sea problemático.
 - • La **integridad de las variable compartidas** pueden ser fácilmente dañadas, como consecuencia de que no hay ninguna protección sobre lo que el programador de cualquier módulo del programa puede realizar sobre la variable.
 - • Existen dificultades para implementar eficientemente el mecanismo de región crítica. Se necesitan diseñar **una estructura con dos listas**: la "**lista principal**" en la que se incluyen los **procesos a la espera de disponer de la autorización de acceso** exclusivo sobre la variable, y la "**lista de eventos**" en la que se encuentra los procesos que tras haber accedido a la variable compartida, **han evaluado la expresión booleana como false**.