**MONITORES**



# 1. INTRODUCCIÓN

Un monitor es una estructura software que consta de

- datos locales
- uno o mas procedimientos
- un código de inicialización

Tiene además las siguientes características:

- ü las variables de datos locales **sólo** pueden ser accedidas por los procedimientos del monitor
- ü la implementación del monitor garantiza que sólo un proceso puede estar ejecutando código del monitor en un instante dado (es decir, los procedimientos del monitor se ejecutan en exclusión mutua)
- ü existe un tipo de variables, denominadas variables *condition*, para la sincronización de los procesos. Sobre una variable *condition* solo se pueden hacer las operaciones wait y signal
- ü la operación wait sobre una variable condition x deja al proceso que la hace en espera. La denotaremos por x.wait (puede encontrarse también wait(x) cwait(x))
- ü la operación signal sobre una variable condition x saca de la espera a uno de los procesos que han hecho x.wait; si no hay procesos en espera no tiene efecto. La denotaremos por x.signal (signal (x), csignal(x))

Un monitor se describe de la siguiente forma:

```
type nombre_monitor = monitor

var
   declaración de variables

procedure entry P1 (…)
begin
   …
end;

procedure entry P2 (…)
begin
   …
end;

procedure entry Pn (…)
begin
   …
end;

begin
   código de inicialización
end;
```

## 1.1. Implementación

Para su implementación hay que conseguir:

- exclusión mutua entre los procedimientos del monitor
- implementar correctamente las operaciones sobre las variables condition

Considérese la siguiente cadena de sucesos:

1. Un proceso A está detenido porque ha hecho wait sobre una variable condition x (x.wait).
2. Un proceso B hace x.signal

La definición de monitor indica que A ha de salir de la espera, pero teniendo en cuenta que dos procesos no pueden ejecutar simultáneamente código del monitor hay varios posibles planteamientos:

a) La instrucción signal solo puede ocurrir como última instrucción dentro de un procedimiento del monitor (o lo que es análogo, equivale a hacer signal y salir del procedimiento del monitor)
b) A espera a que B salga del monitor o haga wait sobre una variable condition
c) B espera a que A salga del monitor o haga wait sobre una variable condition

Implementamos el planteamiento c (Hoare) por ser el más general

Esta implementación tiene los siguientes elementos:

- Código para cada procedimiento del monitor
- Código para la operación wait
- Código para la operación signal

***Código del procedimiento monitor***

## Variables compartidas

| | |
|---|---|
| ***mutex*** | Semáforo de exclusión mutua dentro del monitor |
| ***sig*** | Semáforo donde esperan los que han hecho signal |
| ***contsig*** | Contador de los procesos que esperan en sig |

## Cada procedimiento del monitor

```
begin
   P (mutex);
      …..
          /* código del procedimiento  */
      …..
   if (contsig > 0) then
        V (sig);
   else
        V (mutex);
end;
```

## ***Código de las operaciones wait y signal***

### Variables para cada variable condition k

| | |
|---|---|
| ***k_sem*** | Semáforo donde esperan los que han hecho k.wait |
| ***k_cont*** | Contador de procesos en k_sem |

***Operación k.wait***

```
k_cont := k_cont +1;
if (contsig > 0) then
    V (sig);
else
    V (mutex);
P(k_sem);
k_cont := k_cont -1;
```

***Operación k.signal***

```
if (k_cont > 0) then
    begin
        contsig := contsig + 1;
        V(k_sem);
        P(sig);
        contsig := contsig – 1;
    end;
```

# 2. SOLUCIONES A LOS PROBLEMAS CLÁSICOS DE SINCRONIZACIÓN

Se presentan a continuación algunas soluciones con monitores a problemas típicos de sincronización. La descripción del problema se encuentra en el apartado de semáforos.

## 2.1. Productores-Consumidores

Variables compartidas

```
type
        productor_consumidor = monitor;
    var
        in, out, cont: integer;
        buff = array[0..N-1] of TIPOITEM;
        ok_productor, ok_consumidor: condition;
```

Añadir elementos al buffer

```
Procedure entry añadir(it: tipoitem);
begin
   if (cont = N) then ok_productor.wait;
```

```
   buff[in] := it;
   in := (in + 1) mod N;
   con := cont + 1;
   ok_consumidor.signal;
end;
```

Sacar elementos del buffer

```
Procedure entry obtener(var it: tipoitem);
begin
   if (cont = 0) then ok_consumidor.wait
   it := buff[out];
   out := (out +1) mod N;
   cont := cont – 1;
   ok_productor.signal
end;
```

Inicialización

```
begin
   in: = 0;
   out: = 0;
   cont: = 0;
end;
```

## 2.2. **Lectores-Escritores** (prioridad lectores)

Variables compartidas

```
type
        lectores_escritores = monitor;
    var
        n_lectores: integer;
        ok_lectura, ok_escritura: condition;
        ocupado: boolean;
```

Comienzo de lectura

```
Procedure entry Comenzar_Lectura;
begin
   n_lectores := n_lectores + 1;
   if (ocupado) then ok_lectura.wait;
   ok_lectura.signal;
end;
```

Fin de lectura

```
Procedure entry Terminar_Lectura;
begin
   n_lectores := n_lectores - 1;
   if (n_lectores = 0) then ok_escritura.signal;
end;
```

Comienzo de escritura

```
Procedure entry Comenzar_Escritura;
begin
   if (n_lectores > 0 OR ocupado) then ok_escritura.wait;
   ocupado := TRUE;
end;
```

Fin de escritura

```
Procedure entry Terminar_Escritura;
begin
   ocupado := FALSE;
   if (n_lectores > 0) then ok_lectura.signal;
   else ok_escritura.signal;
end;
```

Inicialización

```
begin
   ocupado := FALSE;
   n_lectores := 0;
end;
```

## 2.3. Lectores-Escritores (prioridad escritores)

Variables compartidas

```
type
        lectores_escritores = monitor;
var
        n_lectores, n_escritores: integer;
        ok_lectura, ok_escritura: condition;
        ocupado: boolean;
```

Comienzo de lectura

```
Procedure entry Comenzar_Lectura;
begin
   if (n_escritores > 0) then ok_lectura.wait;
   n_lectores := n_lectores + 1;
   ok_lectura.signal;
end;
```

Fin de lectura

```
Procedure entry Terminar_Lectura;
begin
   n_lectores := n_lectores - 1;
   if (n_lectores = 0) then ok_escritura.signal;
end;
```

Comienzo de escritura

```
Procedure entry Comenzar_Escritura;
begin
   n_escritores := n_escritores + 1;
   if (n_lectores > 0 OR ocupado) then ok_escritura.wait;
   ocupado := TRUE;
end;
```

Fin de escritura

```
Procedure entry Terminar_Escritura;
begin
   ocupado := FALSE;
   n_escritores := n_escritores – 1;
   if (n_escritores > 0) then ok_escritura.signal;
   else ok_lectura.signal;
end;
```

Inicialización

```
begín
   ocupado = FALSE;
   n_lectores := 0;
   n_escritores := 0;
end;
```

## 2.4. Filósofos cenando

Variables compartidas

```
type     filosofos_cenando = monitor;

var
         estado: array [0..N-1] of (pensando, hambriento, comiendo);
         filos: array [0..N-1] of condition;
```

Coger cubiertos

```
Procedure entry Toma_Cubiertos(i: integer);
begin
   estado[i] := hambriento;
   test(i);
   if (estado[i] <> comiendo) then filos[i].wait;
end;
```

Dejar cubiertos

```
Procedure entry Deja_Cubiertos(i: integer);
begin
   estado[i] := pensando;
   test(izq(i));
   test(der(i));
end;
```

Chequeo

```
Procedure test(i: integer);
begin
   if (estado[izq(i)] <> comiendo AND estado[i] = hambriento AND
      estado[der(i)] <> comiendo) then
         begin
            estado[i] := comiendo;
            filos[i].signal;
         end;
end;
```

Inicialización

```
begin
   for i:= 0 to N-1
      estado[i] := pensando;
end;
```

## 2.5. Operaciones wait con prioridades.

En la definición original de monitor, al hacer signal sobre una variable condition en la que se han detenido varios procesos, no se especifica cual reanuda su ejecución.
La operación wait con prioridad, asigna a cada proceso que se detiene en una variable condition una prioridad, de manera que al hacer signal sobre esa variable condition reanudará su ejecución el proceso de mas prioridad

### *Ejemplo de uso*

Variables compartidas

```
type  asignacion_recurso = monitor;

var
        s: condition;
        ocupado: boolean;
```

```
Procedure entry Adquirir (pri: integer);
begin
   if ocupado then s.wait (pri);
   ocupado := true;
end;
```

```
Procedure entry Liberar;
begin
   ocupado := false;
   s.signal
end;
```

Inicialización

```
begin
   ocupado := false;
end;
```