Aunque los semaforos proporcionan un mecanismo adecuado y efectivo para el proceso de sincronización, un uso incorrecto de los mismos puede dar lugares a errores de temporización que son difíciles de detectar.
Para abordar tales errores, los investigadores han desarrollado estructuras de lenguaje de alto nivel. Una estructura fundamental de sincronización de alto nivel, el tipo monitor.

MONITORES

Utilización

Un tipo monitor tiene un conjunto de operaciones definidas por el programador que gozan de la característica de exclusión mutua dentro del monitor. El tipo monitor también contiene la declaración de una serie de variables cuyos valores definen el estado de una instancia de dicho tipo, junto con los cuerpos de los procedimientos o funciones que operan sobre dichas variables.

```
monitor nombre del monitor
{
    // declaraciones de variables compartidas
    procedimiento P1 ( . . . )  {
      . . .
    }
    procedimiento P2 ( . . . )  {
      . . .
    }
        .
        .
        .
    procedimiento Pn ( . . . )  {
      . . .
    }
    código de inicialización ( . . . )  {
      . . .
    }
}
```

Figura 6.16 Sintaxis de un monitor.

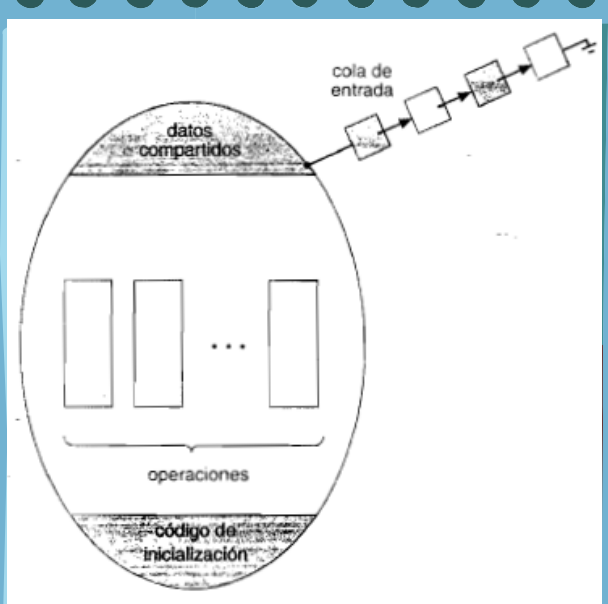


Figura 6.17 Vista esquemática de un monitor.

La estructura del monitor asegura que sólo un proceso esté activo cada vez dentro del monitor. En consecuencia, el programador no tiene que codificar explícitamente esta restricción de sincronización.

Un programador que necesite escribir un esquema de sincronización a medida puede definir una o más variables de tipo *condition:*
Condition x, y;
Las únicas operaciónes que se pueden invocar en una variable de condición son ***wait()*** y ***signal()***.
La operacion ***x.wait()***; indica el proceso que invoca esta operación queda suspendido hasta que otro proceso invoque la operación.
La operación ***x.signal()*** hace que se reanude exactamente uno de los procesos suspendidos. Si no había ningún proceso suspendido, entonces la operación *signal()* no tiene efecto, es decir el estado de x será el mismo que si la operación nunca se hubiera ejecutado.

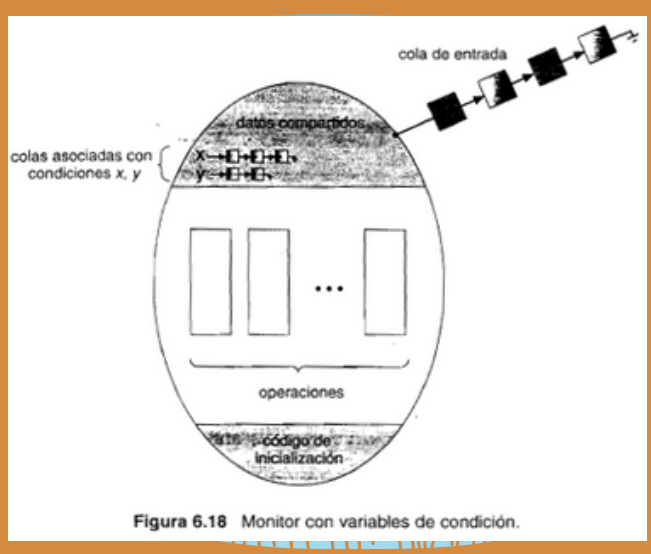


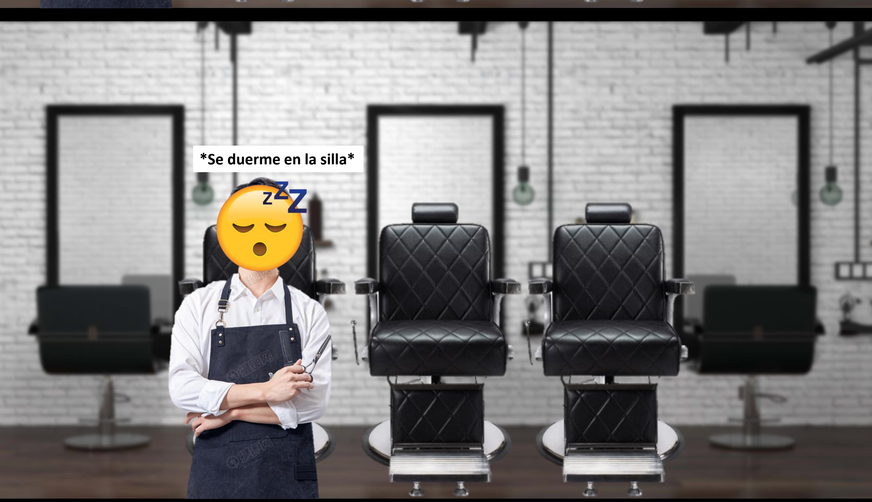
Figura 6.18 Monitor con variables de condición.

```
monitor ProducerConsumer
condition full, empty;
integer count;

      procedure enter;
      begin
            if count = N then wait(full);
            enter _item;
            count:= count + 1;
            if count = 1 then signal(empty)
      end;

      procedure remove;
      begin
            if count = 0 then wait(empty);
            remove_item;
            count:= count - 1;
            if count = N - 1 then signal(full)
      end;

      count := 0;
end monitor;

procedure producer;
begin
      while true do
      begin
            produce_item;
            ProducerConsumer.enter
      end
end;

procedure consumer;
begin
      while true do
      begin
            ProducerConsumer.remove;
            consumer_item
      end
end;
```

Programa 2.8: Bosquejo del problema de **productor-consumidor** con **monitores**. Sólo un procedimiento de monitor está activo a la vez. El buffer tiene **N ranuras**.

Al hacer automática la exclución mutua de las regiones criticas, los monitores hacen a la programación en paralelo mucho menos propensa a errores que cuando se usan semáforos. No obstante, tienen algunas desventajas. No es por capricho que el Programa 2.8 esté escrito en un lenguaje ficticio y no en C. Los monitores son un concepto de lenguajes de programación. El compilador debe reconocerlos y lograr de alguna manera la exclusión mutua. C, Pascal y casi todos los demás lenguajes carecen de monitores.

PROBLEMA DEL PELUQUERO DORMIDO



FIN

EQUIPO 6, 4CM1