

# Monitores

Los monitores son módulos de programa con más estructura, y que pueden ser implementados tan eficientemente como los semáforos.

En algo que nos ayudan los monitores es que nos libera de tener que resolver de forma explícita uno de los dos tipos de sincronización.

- Exclusión mutua : Implícita asegurando que los procedimientos en el mismo monitor no ejecutan concurrentemente (Los procedimientos dentro de un mismo monitor no se pueden ejecutar concurrentemente).
- Sincronización por condición : Explícita con variables condición (Nos permite dormir un proceso para liberar el monitor y de esa manera empezar a ejecutar un procedimiento.....545.....

.

## Notación

# Notación

---

```
monitor NombreMonitor {  
  declaraciones de variables permanentes;  
  código de inicialización  
  
  procedure  $op_1$  (par. formales1)  
  { cuerpo de  $op_1$   
  }  
  
  .....  
  procedure  $op_n$  (par. formalesn)  
  { cuerpo de  $op_n$   
  }  
}
```

Hasta que no finalice el código de inicialización del monitor, el mismo no aceptara ningún llamado a ningún procedimiento.

## Ejemplo de uso de monitores

# Ejemplo de uso de monitores

Tenemos 5 procesos empleados que continuamente hacen algún producto. Hay un proceso coordinador que cada cierto tiempo debe ver la cantidad total de productos hechos.

```
process empleado{id: 0..4} {  
  while (true)  
  { .....  
    TOTAL.incrementar();  
    .....  
  }  
}
```

```
process coordinador{  
  int c;  
  while (true)  
  { .....  
    TOTAL.verificar(c);  
    .....  
  }  
}
```

```
monitor TOTAL {  
  int cant = 0;  
  
  procedure incrementar ()  
  { cant = cant+1;  
  }  
  
  procedure verificar (R: out int)  
  { R = cant;  
  }  
}
```

## Operaciones usadas en la practica

La sincronización por condición es programada explícitamente con variables condición.

cond cv;

El valor asociado a **cv** es una cola de procesos demorados, no visibles directamente al programador. Operaciones sobre las variables condición :

1. wait(cv) - El proceso se demora al final de la cola de cv y deja el acceso exclusivo al monitor (Libera el monitor para que otro proceso pueda comenzar su ejecución).
2. signal(cv) - Despierta al proceso que esta al frente de la cola (Si hay alguno) y lo saca de ella. El proceso despertado recién podrá ejecutar cuando readquiera el acceso exclusivo al monitor.
3. signal\_all(cv) - Despierta todos los procesos demorados en cv, quedando vacía la cola asociada a cv.

# Operaciones adicionales

Operaciones adicionales que NO SON USADAS EN LA PRÁCTICA sobre las *variables condición*:

- **empty(cv)** → retorna *true* si la cola controlada por *cv* está vacía.
- **wait(cv, rank)** → el proceso se demora en la cola de *cv* en orden ascendente de acuerdo al parámetro *rank* y deja el acceso exclusivo al monitor.
- **minrank(cv)** → función que retorna el mínimo ranking de demora.

Operaciones utilizadas en la teoría pero no aplicables a la practica.

## Diferencia entre las disciplinas de señalización

En la materia hacemos uso del "Signal and continue"

Signal and continued : El proceso que hace que el signal continua usando el monitor, y el proceso despertado pasa a competir por acceder nuevamente al monitor para continuar con su ejecución (En la instrucción que lógicamente le sigue al wait).

Signal and wait : El proceso que hace el signal pasa a competir por acceder nuevamente al monitor, mientras que el proceso despertado pasa a ejecutar dentro del monitor a partir de la instrucción que lógicamente le sigue al wait.

## Ejemplos utilizando las operaciones

Tenemos dos procesos A y B, donde A le debe comunicar un valor a B (múltiples veces).

```
process A {  
  int aux;  
  while (true)  
  { --Genera valor a enviar en aux  
    Buffer.Enviar (aux);  
    .....  
  }  
}
```

```
process B {  
  int aux;  
  while (true)  
  { .....  
    Buffer.Recibir (aux);  
    --Trabaja con el valor aux recibido  
  }  
}
```

```
monitor Buffer {  
  int dato;  
  bool hayDato = false;  
  cond P, C;  
  
  procedure Enviar (D: in int)  
  { if (hayDato) → wait (P);  
    dato = D;  
    hayDato = true;  
    signal (C);  
  }  
  
  procedure Recibir (R: out int)  
  { if (not hayDato) → wait (C);  
    R = dato;  
    hayDato = false;  
    signal (P);  
  }  
}
```