

# Tema 6

**Procesos concurrentes:** En el mismo instante de tiempo

**Procesos cooperativos:** Comparten datos. { Memoria compartida  
Ficheros.

El acceso concurrente a datos modificables puede causar problemas  
protocolos.

→ Consideramos algoritmos de planificación  
sobre todo los explícitos.

Antecedentes

Notación para concurrencia:

{ Parbegin: Inicio concurrencia  
Parend: Fin concurrencia.

**Condición de carrera:** Comportamiento de software en el que la solución depende de un orden de ejecución de eventos no controlado y por el que se puede producir un error en el resultado.

**Sección crítica:** Sección del código en la cual los recursos son compartidos y modificables.

Tiene que haber exclusión mutua (que solo un proceso esté utilizando la sección crítica)

. Para proporcionar esta exclusión la mejor opción es a través de un protocolo antes y después de la SC

Soluciones a problemas de la SC.

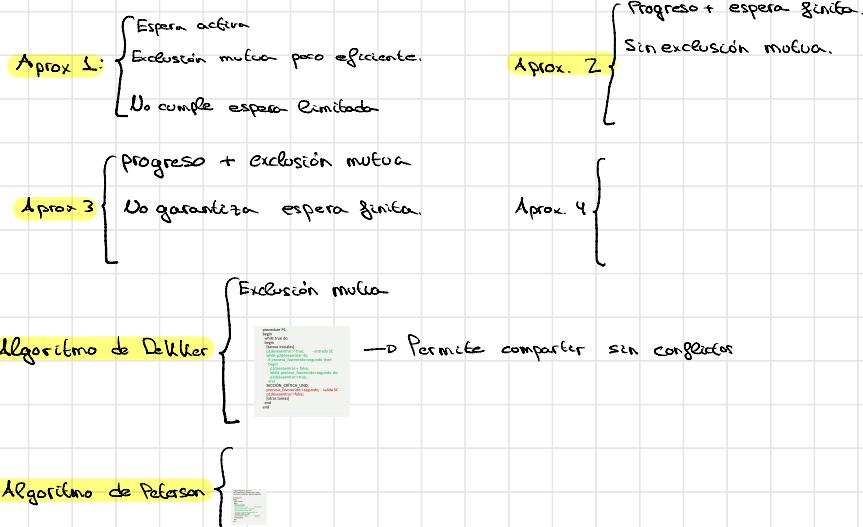
{ Exclusión mutua  
Progreso: Si la SC está libre hay que ocuparla y esto depende de los procesos que están en la cola de espera.  
Espera limitada: Si un proceso quiere acceder, hay que dedicar un tiempo límite.  
Los procesos se ejecutan a velocidad mayor o "0".

Las soluciones pueden ser:

{ Dependiente del hardware: Instrucciones de una CPU.  
Independiente del hardware: Algoritmos.

Las instrucciones y accesos a memoria son **atomic@s**.

### Soluciones al problema del S.C.



### Soluciones para dos procesos

Comentarios generales: Uso de un bucle infinito para → En cada paso se aumenta el número de informaciones compartidas y el grado de conflicto

los

Termina avisando y rediendo el turno al otro proceso.

Son todas soluciones software:

Necesitan de memoria compartida y acceso atómico

El hardware es el que impide que entran dos procesos a la vez

Problemas:

Hay espera activa

Solo sirven para 2 procesos.

## Hardware de sincronización.

Problemas de exclusión mutua

→ Habilitar y deshabilitar interrupciones:

- Solo lo puede hacer el S.O.

- Puede quedar colgado en la S.C.

→ Solo se puede en monoprocesadores.

Además el software de sincronización se hace mucho mejor con un buen hardware de sincronización.

→ Instrucciones que apoyan la sincronización: Simples atomicas, ingreso y salida de la S.C. muy sencillas.

→ Test and Set:

```
boolean Tas (int i){  
    If (i == 0){  
        i = 1;  
        return true;  
    }  
    else  
        return false;  
}  
  
ocupado = 0;  
while (!Tas (ocupado)){  
    //hacer nada  
    SC;  
    ocupado = 0;
```

```
TsS (boolean a, boolean b){  
    puts ("a");  
    b=ocupado;  
    a=0;  
    activa=false; //var compartida  
  
    //procedimiento 1  
    if (a==1){  
        ocupado=true;  
        while (Proc_1_no_puede);  
        tas (Proc_1_no_puede, activa);  
        SC;  
        activa=false;  
    }
```

Swap:

```
swap (int a, int b){  
    int temp;  
    temp = a;  
    a = b;  
    b = temp;  
  
    int libre = k; /* compartido */  
    while (k <= n / local /)  
        while (libre > 0);  
        swap (k, libre);  
    libre=1;
```

Consideraciones generales:

Espesa activa → No satisface espera límite

## SEMAFOROS

Es una variable entera compartida entre los procesos que sincronizan sobre la que solo se puede inicializar, esperar (P) y señalar (V).

El objetivo es sincronizar procesos mediante señales.

!! Valor de inicialización !! → Número máximo de procesos que puede utilizar el recurso simultáneamente.

```
Función P - Esperar - Wait  
P(S){  
    While (S<0){  
        // hacer nada  
    }  
    S = S - 1;  
}  
  
Función V - Señalar - Signal  
V(S){  
    S = S + 1;  
}
```

Atómicas sobre "S"

## Implementación de semáforos.

Sustituye la espera activa por la espera en estado de bloqueado.

```
typedef Semáforo{  
    int valor; // valor del semáforo  
    Lista L; // lista procs. bloqueados  
    // esperando pasar el sem.  
};  
  
P(S){  
    S.valor = S.valor - 1;  
    If (S.valor < 0){  
        Agregar Proceso a cola S.L;  
        Bloquear el Proceso  
    }  
}  
  
V(S){  
    S.Vvalor = S.Vvalor + 1;  
    If (S.Vvalor <= 0){  
        Extraer un proceso P de S.L;  
        Despertar el proceso P;  
    }  
}
```

Cola de espera

L

↳ Se hace usar cola de espera con los procesos que no pueden pasar el semáforo.



Pueden tener valores negativos

Todas las operaciones son llamadas al SO.

- FIFO → Robustos, sin inanición
- Sin FIFO → No Robustos

## Problemas al usar semáforos

**Bloqueo mutuo:** Se produce cuando dos procesos se quedan esperando a la ocurrencia de un evento que está en espera.

**Inanición:** Los procesos esperan indefinidamente en la cola mientras otros entran y salen.

## Semáforos binarios

**Semáforos contadores o generales:** Los anteriores, si toman un valor positivo son los procesos que pueden pasar al semáforo sin ser bloqueados y si es negativo, son los procesos que hay en cola.

**Semáforos binarios:** Solo pueden tomar "0" o "1":

**P:** Si el semáforo vale "1", el proceso pasa y se pone a "0". Si el semáforo vale "0", el proceso se bloquea.

**V:** Si hay algo esperando lo desbloquea. Si no hay nadie esperando, pone el semáforo a "1".

## Atomicidad en semáforos

**Monoprocesador:** Inhiben las interrupciones cuando entra algo en el semáforo y al contrario.

**Multiprocesador:** Usa alguna solución de hardware o software. (aún que tengan espera activa).

# Problema básico de sincronización

Solo después de que un proceso termine sus sentencias, otro puede empezar las suyas.

## El problema del buffer emitido

El caso más sencillo es comunicar dos procesos: uno que produce datos y otro que los consume a través de una zona de memoria compartida denominada buffer.

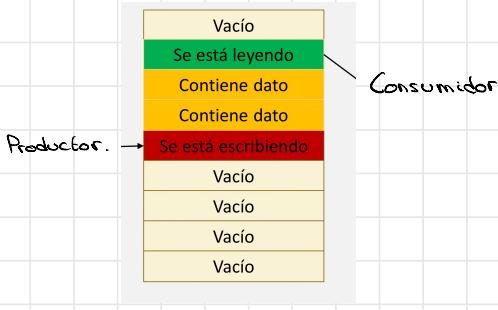
- Permite desacoplar la velocidad de producción de la velocidad de consumición de los datos

Buffer

Registros ( $B[N]$ ): En cada registro se puede almacenar un ejemplar de dato (lleno).

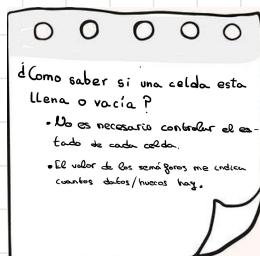
A fin de reutilizar los registros se usa una estructura circular.

### ~ Buffer circular ~



#### Se debe cumplir:

- Orden de llenado = orden de vaciado
- Sin pérdida de datos.
- Si un registro no se ha leído, no se puede escribir sobre él.
- No se puede leer un dato varias veces.
- Exclusión mutua (celdas).
- Pueden usar el buffer a la vez siempre y cuando no sea en la misma celda.



## ~ Generalización ~

- Un dato producido puede ser consumido por cualquier consumidor
- Hay que evitar que:
  - Se lea a la vez
  - Se escriba a la vez.

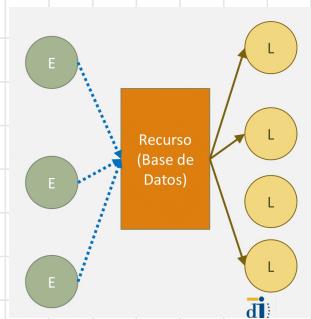
## Problema de Los Lectores/Escritores

Varios procesos intentando leer/escribir sobre un recurso:

- Se permiten las **lecturas simultáneas**
- No se permite la escritura simultánea.

**Variante:** Se da prioridad a la **lectura** o a la **escritura**

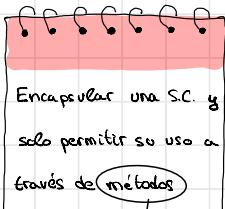
Si hay lectores, los escritores **esperan** hasta que no haya lectores (riesgo de **inanción**).



## ~ Problema de la cena de filósofos ~

<https://pacoportillo.es/informatica-avanzada/programacion-multiproceso/la-cena-de-los-filosofos/>

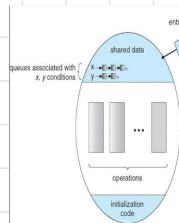
# Monitores



Encapsular una SC y solo permitir su uso a través de métodos

Con exclusión mutua

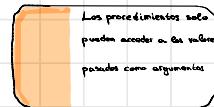
Representación de un tipo monitor:



shared data  
queues associated with:  
x conditions  
y conditions  
operations  
initialization code

Solo puede estar activo a la vez un proceso del monitor

- **Variables** que representan el estado del monitor
- **Procedimientos** que manipulan ese estado.



Los procedimientos solo pueden acceder a su valor pasándolo como argumento

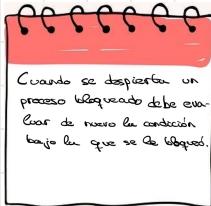
## ~ Variables "Condition" ~

Operaciones:

- x.espera(): El proceso se suspende hasta que otro invoca x.señal().
  - x.señal(): Reanuda un proceso sus pendido.
- !! Aun que lo parezca no son como la P y la V de los semáforos !!

Función x.señal(): Al ejecutarla se despierta un proceso que estaba dentro del monitor, mientras que el proceso despertador también.

Se viola la exclusión mutua.



Cuando se despierta un proceso bloqueado debe encargar de nuevo la condición bajo la que se le bloqueó.

→ Soluciones

- {
- El despertador se bloquea
  - El despertador espera a que salga el despertador.

## Sincronización con envío de mensajes

En los sistemas distribuidos no hay memoria compartida;

La solución para el paso de mensajes:

- Tuberías: Permiten comunicar procesos.

Tuberías  $\cong$  ficheros