Tema 3. Herramientas de Sincronización

Jorge García Duque
http://www.det.uvigo.es/~jgd
Depto. Enxeñería Telemática
Universidad de Vigo

Semáforos (I). Introducción

- Objetivo: solución al problema de la exclusión mutua evitando la espera activa *(Dijkstra 1965).*
- Un semáforo `sem` consta de tres partes:
  - Una variable entera interna (`s`) con un valor máximo N (no accesible para los procesos).
  - Una cola de procesos (no accesible para los procesos).
  - Dos funciones básicas de acceso:
    - *`wait(sem):`* si la variable interna es cero, el proceso se suspende en la cola asociada al semáforo, en caso contrario se decreincrementa en una unidad el valor de la variable interna.
    - *`signal(sem):`* si existe algún proceso suspendido en la cola asociada al semáforo, se despierta al más prioritatio; en caso contrario se incrementa en una unidad el valor de la variable interna (sin superar el valor máximo N).
- El valor inicial y máximo de la variable interna determina la funcionalidad del semáforo.

Semáforos (II). Exclusión Mutua

- Semáforos Binarios: la variable interna sólo puede tomar los valores `0` y `1`.
- Solución al problema de la exclusión mutua:
  - Un semáforo binario cuya variable interna esté inicializada a `1` (*Semáforo de Exclusión Mutua)*.
  - Antes de acceder a la sección crítica el proceso ejecuta la sentencia `wait(sem)`.
  - Al finalizar la sección crítica el proceso ejecuta la sentencia `signal(sem)`.

Semáforos (III). Exclusión Mutua

```
                    semaforo sem(1);

Process P                          Process Q
  while (1) {                        while (1) {
p1: ……;                            q1: ……;
p2: wait(sem);                     q2: wait(sem);
p3: SECCIÓN CRÍTICA;               q3: SECCIÓN CRÍTICA;
p4: signal(sem);                   q4: signal(sem);
}                                  }
```

Semáforos (IV). De Paso

- Grafos de Precedencia:
  - Para cada punto de sincronización entre dos procesos: semáforo binario cuya variable interna esté inicializada a `0` *(Semáforos de Paso).*
  - El proceso que debe esperar ejecuta la sentencia `wait(sem)`.
  - Al alcanzar un punto de sincronización el otro proceso ejecuta la sentencia `signal(sem)`.

Semáforos (V). Enteros

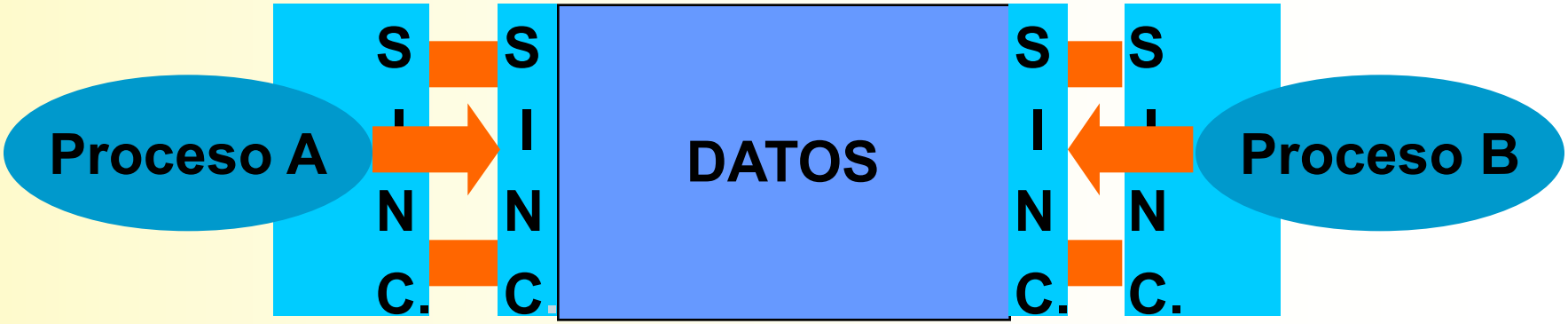
- `N` procesos accediendo a la sección crítica:
  - Semáforo cuya variable interna esté inicializada a `N`, siendo `N` el valor máximo (*Semáforo Entero).*
  - Antes de acceder a la sección crítica el proceso ejecuta la sentencia `wait(sem)`.
  - Al finalizar la sección crítica el proceso ejecuta la sentencia `signal(sem)`.
- Sincronización entre procesos en función de una variable entera:
  - Semáforo cuya variable interna esté inicializada a `N` ó `0`, siendo `N` el valor máximo *(Semáforo de Condición).*
  - El proceso que debe esperar ejecuta la sentencia `wait(sem)`.
  - Al alcanzar un punto de sincronización el otro proceso ejecuta la sentencia `signal(sem)`.

Semáforos (VI). Conclusiones

- Solución al problema de la exclusión mutua evitando la espera activa.
- Además, semáforos *de paso, enteros y de condición*.
- Inconvenientes:
  - La inicialización es crítica, así como confundir un `wait` con un `signal` u omitir alguno de ellos.
  - Las soluciones son difíciles de depurar y validar.

Monitores(I). Introducción

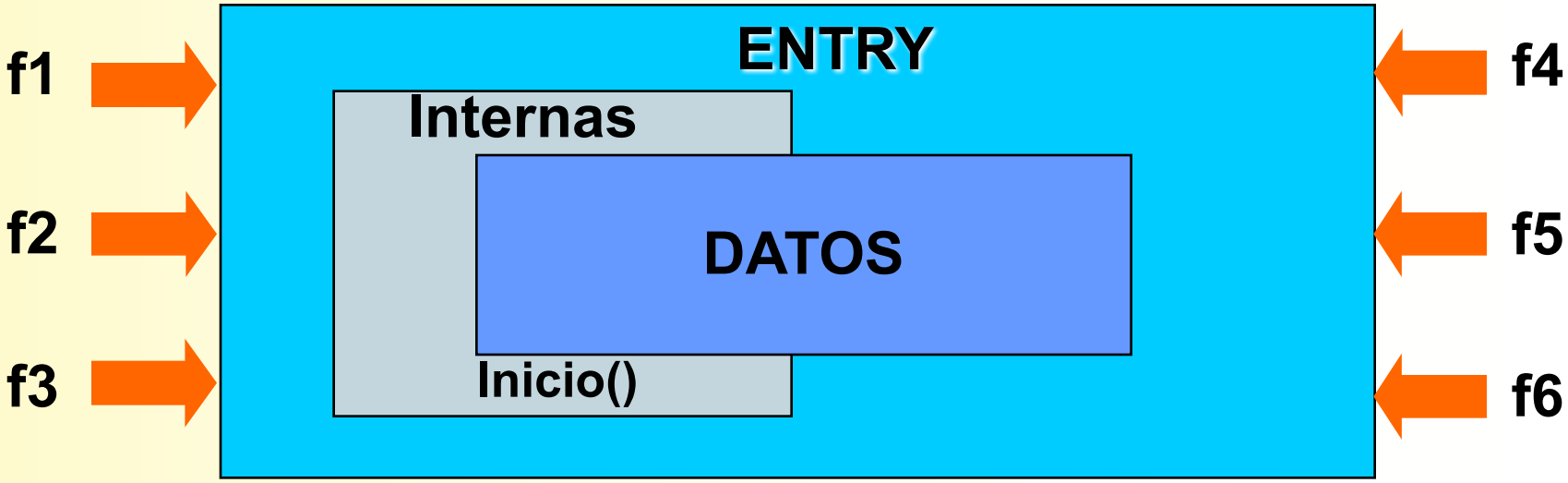
- Objetivo: aislar el código de sincronización del código de los procesos.



Monitor: tipo abstracto de datos que garantiza el acceso en exclusión mutua a los *datos*.

Monitores(II). Introducción

- Sólo un proceso puede ejecutar simultánemente una de las *funciones de acceso* (*ENTRY)* definidas para el Monitor.
- El Monitor puede incluir *funciones internas* con el objetivo de estructurar la codificación de las *funciones de acceso*.
- Los procesos no pueden acceder a las *funciones internas* ni a los *datos* del Monitor.



Monitores (III). Condicionales

- Se incluyen como parte de los *datos* del Monitor *Variables de Condición:*
- Una *variable de condición* `v` consta de:
  - Una cola de procesos.
  - Tres funciones de acceso:
    - `wait(v)`: suspende al proceso en la cola asociada a la *variable de condición*.
    - `signal(v)`: si existe algún proceso suspendido en la cola asociada a la *variable de condición* se despierta el más prioritario.
    - `empty(v)`: función `boolena` que retorna `verdadero` si no existe ningún proceso suspendido en la cola asociada a la *variable de condición* (`falso` en caso contrario)

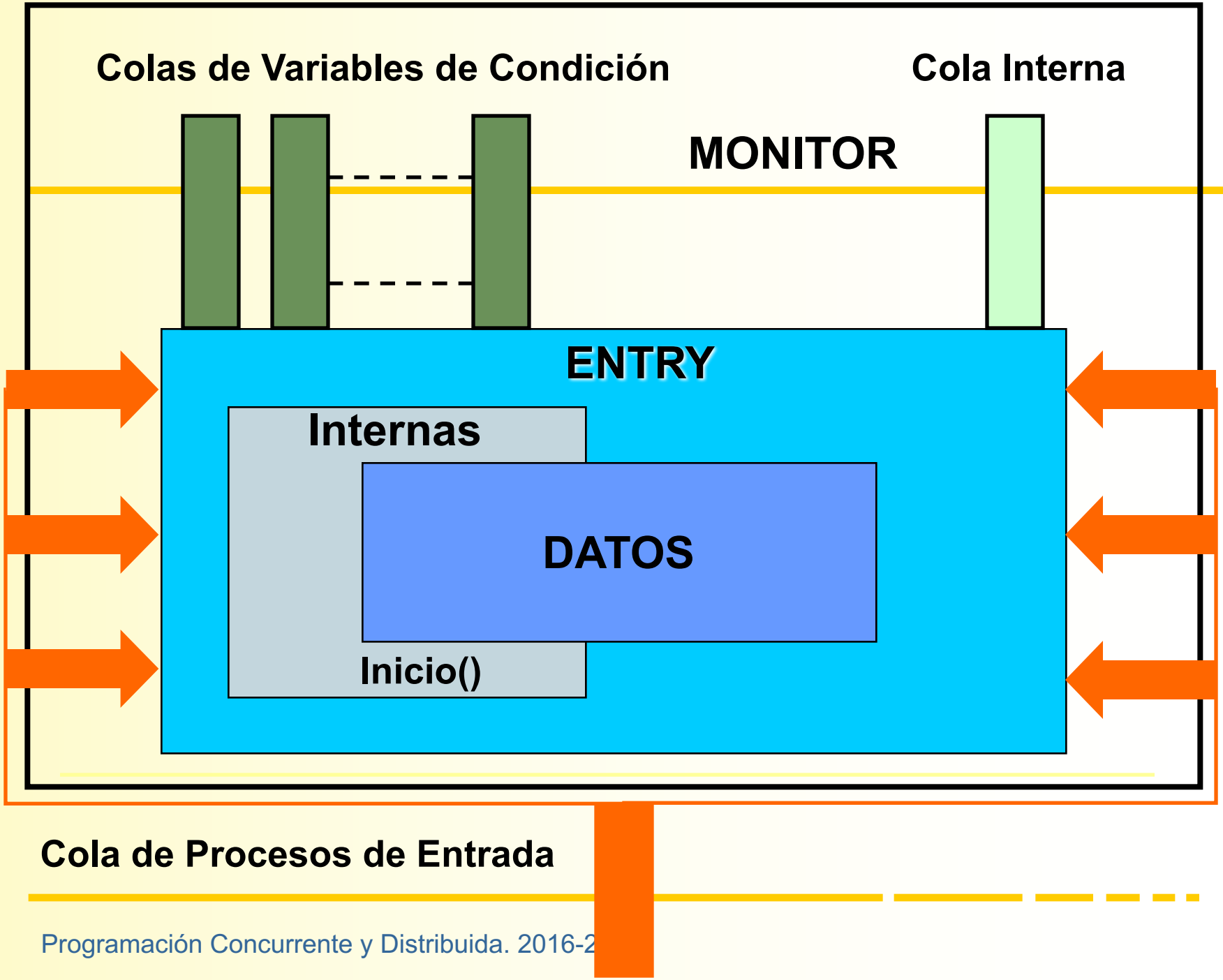
Monitores(IV). Condicionales

```
Monitor Semaforo
 int s;
 condition v;
 ENTRY wait() {

  if (s == 0) wait(v);
  else s--;
 }
 Process P {
  M.wait();
  S;
  M.signal();
 }
```

```
inicio() {
 s = N;
}
ENTRY signal() {

 if (empty(v)) s++;
 else signal(v);
}
```

Monitores (V). Condicionales

- Problema: cuando un proceso *P* despierta a otro proceso *Q* suspendido sobre una *variable de condición,* ¡existirán dos procesos preparados dentro del Monitor!.
- Solución: como el Monitor garantiza la exclusión mutua en el acceso a los datos, uno de los dos procesos se suspenderá hasta que el otro **abandone el Monitor** (retorna o se suspende sobre una *variable de condición*).
- Dos posibilidades:
  1. Continúa el proceso señalado (*Q)* y se demora el que ha ejecutado el `signal` (*P).*
  2. Continúa el proceso que ha ejecutado el `signal` (*P)* y se demora el proceso señalado (*Q).*
- En general, las soluciones son dependientes del tipo de Monitor.
- Soluciones independientes del tipo de Monitor: después de un `signal`, el proceso abandona el Monitor.


Colas de Variables de Condición
Cola Interna
MONITOR
ENTRY
Internas
DATOS
Inicio()
Cola de Procesos de Entrada

Monitores(VI). Implementación

```
semaforo mutex(1), next(0);
int next_count = 0;
```

[condition v]:

```
   semaforo v_sem(0);
   int v_count = 0;
```

[Monitor.f()]:

```
   wait(mutex);
   f();
   siguiente(); →
```

```
if (next_count > 0)
     signal(next);
else signal(mutex);
```

Monitores(VII). Implementación

[signal(v)]: Tipo 1 (cont. señalado)

```
if (v_count > 0) {
      next_count++;
      signal(v_sem);
      wait(next);
      next_count--;
}
```

[wait(v)]: Tipo 1 (cont. Señalado)

```
v_count++;
siguiente(); →  | if (next_count > 0)
wait(v_sem);    |    signal(next);
v_count--;      | else signal(mutex);
```