Universidad de Huelva


**Universidad de Huelva**

Grado en Ingeniería Informática

# Tema 2. Primeras aproximaciones a la solución de los problemas de la Programación Concurrente

*Resumen*

Autor: Alberto Fernández Merchán
Asignatura: Programación Concurrente y Distribuida

# 1. Introducción

Los recursos que no son compartibles por los procesos se gestionan mediante la exclusión mutua. Por otro lado, los procesos concurrentes que cooperan compartiendo información se relacionan mediante condiciones de sincronización.

# 2. Tipos de sincronización y su solución

- **Exclusión Mutua**: Acción de sincronización neceasria para que dos o más procesos puedan usar un recurso no compartible. Para garantizarla se diseña un protocolo de entrada y otro de salida. Estos protocolos deben satisfacer ñas siguientes condiciones:
  - Exclusión Mutua: Dos procesos no pueden estar a la vez en la sección crítica.
  - Limitación de la espera: Ningún proceso espera de forma indefinida.
  - Progreso en la ejecución: Un proceso que quiera acceder a la sección crítica lo hará si esta está libre.
- **Condición de sincronización**: Un proceso no realiza un evento hasta que otro proceso haya realizado una acción determinada.

Para resolver estos problemas existen diferentes mecanismos:

- Inhibición de Interrupciones: En un sistema con un procesador, la concurrencia se consigue mediante interrupciones de E/S.
- Espera ocupada
- Semáforos
- Regiones Críticas Condicionales
- Monitores
- Paso de mensajes
- Invocaciones remotas

# 3. Soluciones de Espera Ocupada

Implementan la sincronización basándose en que un proceso espera comprobando de forma continua el valor de una variable (manteniendo ocupada a la CPU). Se pueden distinguir:

- **Soluciones Software**: Las únicas instrucciones atómicas son las de leer/escribir direcciones de memoria.
  La exclusión mutua está asegurada mediante el protocolo de entrada (comprueba la condición para enrar a la S.C.) y el protocolo de salida (indica que ha terminado de ejecutar la S.C.).

  Un proceso **no** puede pararse durante la ejecución de los protocolos o de la sección crítica.

  Si varios procesos están intentando entrar en sus secciones críticas, uno de ellos siempre conseguirá entrar en ella.

  Cada proceso tendrá exito al acceder a su sección crítica.

- Soluciones Hardware: Se usan instrucciones especiales para llevar a cabo una serie de acciones.

## 3.1. Algoritmo de Dekker

- c0 y c1 se inicializan a fueraSC. El valor de turno no influye.

```
process P0
repeat
  c0 := quiereentrar;
  while c1 = quiereentrar do
   if turno = 1 them
    begin
     c0 := fueraSC;
     while turno = 1 do;
     c0 := quiereentrar;
    end
  Sección Crítica0
  turno := 1
  c0 := fueraSC;
  Resto0
forever
```

```
process P1
repeat
  c1 := quiereentrar;
  while c0 = quiereentrar do
   if turno = 0 them
    begin
     c1 := fueraSC;
     while turno = 0 do;
     c1 := quiereentrar;
    end
  Sección Crítica1
  turno := 0
  c1 := fueraSC;
  Resto1
forever
```

## 3.2. Algoritmo de Peterson

- c0 y c1 se inicializan a fueraSC. El valor de turno no influye.

```
process P0
repeat
  c0 := quiereentrar;
  turno := 1;
  while (c1 = quiereentrar)
          and (turno = 1) do;
  Sección Crítica0
  c0 := fueraSC;
  Resto0
forever
```

```
process P1
repeat
  c1 := quiereentrar;
  turno := 0;
  while (c0 = quiereentrar)
          and (turno = 0) do;
  Sección Crítica1
  c1 := fueraSC;
  Resto1
forever
```

## 3.3. Algoritmo de Lamport

- Es posible usarlo en entornos distribuidos

```
process Pi
repeat //inicialmente numero[i]=0 -> el proceso no tiene numero
 c[i] := cognum;
 numero[i]:= 1+max(numero[0],...,numero[n-1]);
 c[i] := nocognum;
 for j=0 to n-1 do
  begin
   while (c[j] = cognum) do;
   while ((numero[j]<>0) and ((numero[i],i)>(numero[j],j))) do;
  end
 Sección Crítica1
 numero[i]=0;
 Resto1
forever
```

24

# 4. Soluciones Hardware

Hay procesadores que proporcionan instrucciones para llevar a cabo de forma indivisible varias acciones. Estas instrucciones pueden suarse para resolver problemas de exclusión mutua con espera ocupada.

## 4.1. Exchange(r,m)

Intercambia el contenido de las posiciones de memoria r y m de forma atómica.

- m inicialmente vale 1 y los $r_i$ valen 0

```
process P0
repeat
   repeat
      exchange(r0,m)
   until r0 = 1;
   Sección Crítica0
   exchange(r0,m)
   Resto0
forever
```

```
process P1
repeat
   repeat
      exchange(r1,m)
   until r1 = 1;
   Sección Crítica1
   exchange(r1,m)
   Resto1
forever
```

## 4.2. Subc(r,m)

Decrementa en 1 el contenido de m y copia el resultado en r de forma atómica.

- m inicialmente vale 1

```
process P0
repeat
   repeat
      subc(r0,m)
   until r0 = 0;
   Sección Crítica0
   m := 1;
   Resto0
forever
```

```
process P1
repeat
   repeat
      subc(r1,m)
   until r1 = 0;
   Sección Crítica1
   m := 1;
   Resto1
forever
```

## 4.3. Addc(r,m)

Incrementa en 1 el contenido de m y copia el resultado en r de forma atómica.

- m inicialmente vale -1

```
process P0
repeat
   repeat
      addc(r0,m)
   until r0 = 0;
   Sección Crítica0
   m := -1;
   Resto0
forever
```

```
process P1
repeat
   repeat
      addc(r1,m)
   until r1 = 0;
   Sección Crítica1
   m := -1;
   Resto1
forever
```

## 4.4. Testset(m)

Comprueba el valor de la variable m:

- Si $m == 0$ lo cambia a 1 y devuelve true
- Si $m == 1$ no modifica el valor y devuelve false.

- La implementación del protocolo sería: (`m` inicialmente vale 0)

```
process P0
repeat
   repeat until testset(m);
   Sección Crítica0
   m := 0;
   Resto0
forever
```

```
process P1
repeat
   repeat until testset(m);
   Sección Crítica1
   m := 0;
   Resto1
forever
```