Tema 2. El Problema de la Sección Crítica

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

El Problema

- Escenario genérico con N procesos concurrentes.
- Sección crítica: parte del código de un proceso en el que se tiene que limitar la concurrencia.
- Exclusión mutua: caso particular en el que se limita la concurrencia a un único proceso en la sección crítica.
- Solución:
  - Incluir mecanismos de sincronización añadiendo código antes y después de las secciones críticas.
  - Restricciones de la solución:
    - Ausencia de *interbloqueo (deadlock)*.
    - Ausencia de *inanición (starvation).*

Exclusión Mutua (2 procesos) (I)

- Primer intento: turno.

```
                    int turno = 0;

Process P                          Process Q
  while (1) {                        while (1) {
p1: ……;                            q1: ……;
p2: while(turno == 1);             q2: while(turno == 0);
p3: SECCIÓN CRÍTICA;               q3: SECCIÓN CRÍTICA;
p4: turno = 1;                     q4: turno = 0;
  }                                  }
```

¿Espera Activa?

¿Es válida? Es decir, ¿garantiza la exclusión mutua en la sección crítica de ambos procesos sin interbloqueo ni inanición?

Exclusión Mutua (2 procesos) (II)

- Segundo intento: (post)solicitud.

```
int quiere_p = 0, quiere_q = 0;

Process P                          Process Q
  while (1) {                        while (1) {
p1: ……;                            q1: ……;
p2: while(quiere_q);               q2: while(quiere_p);
p3: quiere_p = 1;                  q3: quiere_q = 1;
p4: SECCIÓN CRÍTICA;               q4: SECCIÓN CRÍTICA;
p5: quiere_p = 0;                  q5: quiere_q = 0;
  }                                  }
```

¿Es válida? Es decir, ¿garantiza la exclusión mutua en la sección crítica de ambos procesos sin interbloqueo ni inanición?

Exclusión Mutua (2 procesos) (III)

- Tercer intento: (pre)solicitud.

```
int quiere_p = 0, quiere_q = 0;

Process P                    Process Q
  while (1) {                  while (1) {
p1: ……;                      q1: ……;
p2: quiere_p = 1;            q2: quiere_q = 1;
p3: while(quiere_q);         q3: while(quiere_p);
p4: SECCIÓN CRÍTICA;         q4: SECCIÓN CRÍTICA;
p5: quiere_p = 0;            q5: quiere_q = 0;
  }                            }
```

¿Es válida? Es decir, ¿garantiza la exclusión mutua en la sección crítica de ambos procesos sin interbloqueo ni inanición?

Exclusión Mutua (2 procesos) (IV)

- Cuarto intento: (pre)solicitud + contienda.

```
int quiere_p = 0, quiere_q = 0;
```

```
Process P                          Process Q
  while (1) {                        while (1) {
p1: ……;                            q1: ……;
p2: quiere_p = 1;                  q2: quiere_q = 1;
p3: while(quiere_q) {              q3: while(quiere_p) {
p4:   quiere_p = 0;                q4:   quiere_q = 0;
p5:   quiere_p = 1;                q5:   quiere_q = 1;
    }                                  }
p6: SECCIÓN CRÍTICA;               q6: SECCIÓN CRÍTICA;
p7: quiere_p = 0;                  q7: quiere_q = 0;
  }                                  }
```

¿Espera Activa?

¿Es válida? Es decir, ¿garantiza la exclusión mutua en la sección crítica de ambos procesos sin interbloqueo ni inanición?

Exclusión Mutua (2 procesos) (V)

- Solución 1: (pre)solicitud + turno si hay contienda.
  - Algoritmo de *Dekker:*

```
   int quiere_p = 0, quiere_q = 0, turno = 0;
```

```
Process P                          Process Q
  while (1) {                        while (1) {
p1:  ……;                          q1:  ……;
p2:  quiere_p = 1                  q2:  quiere_q = 1;
p3:  while(quiere_q) {             q3:  while(quiere_p) {
p4:    if (turno == 1) {           q4:    if (turno == 0) {
p5:      quiere_p = 0              q5:      quiere_q = 0;
p6:      while (turno == 1);       q6:      while (turno == 0);
p7:      quiere_p = 1;             q7:      quiere_q = 1;
       }                                  }
     }                                  }
p8:  SECCIÓN CRÍTICA;              q8:  SECCIÓN CRÍTICA;
p9:  turno = 1;                    q9:  turno = 0;
p10: quiere_p = 0;                 q10: quiere_q = 0;
  }                                  }
```

Exclusión Mutua (2 procesos) (VI)

- Solución 2: (pre)solicitud + primero si hay contienda.
  - Algoritmo de *Peterson:*

```
int quiere_p = 0, quiere_q = 0, ultimo = 0;
```

```
Process P
  while (1) {
p1: ……;
p2: quiere_p = 1;
p3: ultimo = 0;
p4: while(quiere_q AND ultimo == 0);
p5: SECCIÓN CRÍTICA;
p6: quiere_p = 0;
  }
```

```
Process Q
  while (1) {
q1: ……;
q2: quiere_q = 1;
q3: ultimo = 1;
q4: while(quiere_p AND ultimo == 1);
q5: SECCIÓN CRÍTICA;
q6: quiere_q = 0;
}
```