Sincronización con Espera Activa

PROGRAMACIÓN CONCURRENTE – TEMA 2

Universidad
Rey Juan Carlos

Sincronización con Espera Activa

- Introducción
- Sincronización Condicional
- **Exclusión Mutua**
  - Propiedades de Corrección
  - 1ª Aproximación: Alternancia Obligatoria
  - 2ª Aproximación: No Exclusión Mutua
  - 3ª Aproximación: Interbloqueo
  - 4ª Aproximación: Espera indefinida
  - Solución: Algoritmo de Dekker
  - Uso de la Exclusión Mutua
- Espera Activa vs Herramientas de sincronización
- Conclusiones

Exclusión Mutua

Sincronización con Espera Activa

- **¿Qué es la Exclusión Mutua?**
  - Desde un punto de vista general la exclusión mutua se tiene cuando varios **procesos compiten** por un **recurso común de acceso exclusivo**
  - Sólo **uno de los procesos** puede estar accediendo al recurso a la vez y los demás tienen que **esperar**
  - Cuando un proceso **libera** el recurso de acceso exclusivo y otro proceso estaba esperando, el proceso que espera **accederá** al recurso
  - De forma mas **concreta**, para comprender su funcionamiento se estudia **El Problema de la Exclusión Mutua**

Exclusión Mutua

Sincronización con Espera Activa

- **El problema de la Exclusión Mutua**
  - Se tienen dos o más procesos concurrentes, que ejecutan indefinidamente una secuencia de instrucciones dividida en dos secciones
    - Sección crítica
    - Sección no crítica

```
public static void p1() {

    while(true) {
        // Sección Crítica
        printlnI("P1_SC1 ");
        printlnI("P1_SC2 ");

        // Sección No Crítica
        printlnI("P1_SNC1 ");
        printlnI("P1_SNC2 ");
    }
}
```

El método ***printlnI*** imprime en una columna diferente la información de cada proceso. Esto permite diferenciar de forma muy sencilla qué instrucciones ejecuta cada proceso.

Exclusión Mutua

Sincronización con Espera Activa

- **Sección Crítica**
  - Secuencia de instrucciones que acceden a un recurso compartido de **acceso exclusivo**
  - Puesto que el recurso es de acceso exclusivo y solo un proceso puede acceder a él al mismo tiempo, cada proceso debe ejecutar las **instrucciones de la sección crítica** sin que haya **intercalación** de **instrucciones de la sección crítica** de otros procesos
  - Se pueden intercalar instrucciones que no hagan uso del recurso, es decir, instrucciones fuera de la sección crítica de otros procesos
- **Sección No Crítica**
  - Secuencia de instrucciones que pueden ser ejecutadas concurrentemente por todos los procesos
  - Se pueden intercalar con las instrucciones de la sección crítica de otros procesos

Exclusión Mutua

Sincronización con Espera Activa

```
while(true) {

   //Preprotocolo

   // Sección Crítica
   println("P1_SC1");
   println("P1_SC2");

   //Postprotocolo

   // Sección No Crítica
   println("P1_SNC1");
   println("P1_SNC2");
}
```

- El **preprotocolo** y **postprotocolo** son las secuencias de instrucciones que deben ejecutar los procesos para garantizar que las instrucciones de la **sección crítica** se ejecutan cumpliendo los requisitos
- Se asume que los procesos finalizan su ejecución fuera de la sección crítica, de forma que siempre hayan ejecutado el **postprotocolo**

Sincronización con Espera Activa

- Introducción
- Sincronización Condicional
- Exclusión Mutua
  - **Propiedades de Corrección**
  - 1ª Aproximación: Alternancia Obligatoria
  - 2ª Aproximación: No Exclusión Mutua
  - 3ª Aproximación: Interbloqueo
  - 4ª Aproximación: Espera indefinida
  - Solución: Algoritmo de Dekker
  - Uso de la Exclusión Mutua
- Espera Activa vs Herramientas de sincronización
- Conclusiones

Propiedades de Corrección

Sincronización con Espera Activa

- Pese a que pueda parecer sencillo, implementar correctamente un programa para **El Problema de la Exclusión Mutua** es complejo y hay que tener en cuenta que se deben cumplir ciertas **propiedades de corrección**
- Estas **propiedades de corrección** se estudian con detenimiento porque las tendremos que cumplir **siempre** que desarrollemos cualquier programa concurrente, no sólo en este problema concreto
- Una **propiedad de un programa concurrente** es una característica que se debe cumplir en cualquier posible **intercalación de las instrucciones atómicas**

Propiedades de Corrección

Sincronización con Espera Activa

- **Exclusión Mutua**
  - En cada instante sólo puede estar un proceso ejecutando su sección crítica porque sólo un proceso puede acceder al recurso compartido a la vez
- **Ausencia de Retrasos Innecesarios**
  - Si un proceso intenta entrar en su sección crítica y no hay otro proceso que también quiera entrar, entonces deberá entrar sin retrasos
- **Ausencia de Inanición (*starvation*)**
  - Todo proceso que quiera entrar en la sección crítica, en algún momento deberá conseguirlo
  - Se produce inanición cuando un proceso quiere entrar en la sección critica pero sistemáticamente es otro proceso el que entra en ella

Propiedades de Corrección

Sincronización con Espera Activa

- **Ausencia de Interbloqueos**
  - Si varios procesos intentan entrar simultáneamente a su sección crítica, al menos uno de ellos debe lograrlo. Un interbloqueo se produciría si ambos procesos quedaran esperando
  - **Interbloqueo activo (*livelock*)**
    - Cuando los procesos ejecutan instrucciones que no producen un avance real del programa, son inútiles
  - **Interbloqueo pasivo (*deadlock*)**
    - Cuando todos los procesos están esperando
    - Como todos están esperando, ninguno de ellos podrá cambiar las condiciones para que dejen de esperar

Propiedades de Corrección

- Las propiedades se dividen en dos tipos
  - **De seguridad (*safety*)**
    - Si alguna de estas propiedades se incumple en alguna ocasión, el programa se comportará de forma errónea
    - Aseguran que nada malo sucederá durante la ejecución y deben cumplirse **siempre**
  - **De Vida (*liveness*)**
    - Si alguna de estas propiedades se incumple en "alguna" ocasión, el programa se comportara de forma correcta pero será lento y desaprovechará los recursos
    - Aseguran que algo bueno ocurrirá **alguna** vez durante la ejecución del programa

Propiedades de Corrección

- **Propiedades de seguridad (*safety*)**
  - Exclusión Mutua
  - Ausencia de Interbloqueo pasivo
- **Propiedades de Vida (*liveness*)**
  - Ausencia de Retrasos innecesarios
  - Ausencia de inanición (*starvation*)
  - Ausencia de interbloqueo activo (*livelock*)

Propiedades de Corrección

Sincronización con Espera Activa

- El acceso a la sección crítica se puede especificar de una forma más **justa** (***fairness***)
  - **Espera lineal:** Si un proceso quiere entrar en la sección crítica, entrará antes de que otro proceso entre más de una vez
  - **Espera FIFO:** Si un proceso quiere entrar en la sección crítica, entrará antes de que entre otro proceso que lo solicite después que él
  - **Prioridad:** Los procesos entran en la sección crítica en función de su prioridad. A igual prioridad, el comportamiento es FIFO
  - **Aleatorio:** Los procesos esperan un tiempo aleatorio (de tmin a tmax) antes de entrar en la sección crítica

Sincronización con Espera Activa

- Introducción
- Sincronización Condicional
- Exclusión Mutua
  - Propiedades de Corrección
  - **1ª Aproximación: Alternancia Obligatoria**
  - 2ª Aproximación: No Exclusión Mutua
  - 3ª Aproximación: Interbloqueo
  - 4ª Aproximación: Espera indefinida
  - Solución: Algoritmo de Dekker
  - Uso de la Exclusión Mutua
- Espera Activa vs Herramientas de sincronización
- Conclusiones

1ª Aproximación: Alternancia Obligatoria

- Para implementar esta solución se utiliza una variable **turno** que indica el proceso que puede entrar en la **sección crítica**
- Cuando un proceso va a entrar en la sección crítica, en el **preprotocolo** se comprueba si es su turno
- Al salir de la sección crítica, en el **postprotocolo** se indica que el turno es del otro proceso

1ª Aproximación: Alternancia Obligatoria

```
static volatile int turno;
```

```
public static void p1() {
   for (int i = 0; i < 5; i++) {
      // Preprotocolo
      while (turno != 1);

      // Sección Crítica
      printlnI("P1_SC1");
      printlnI("P1_SC2");

      // Postprotocolo
      turno = 2;

      // Sección No Crítica
      printlnI("P1_SNC1");
      printlnI("P1_SNC2");
   }
}
```

```
public static void p2() {
   for (int i = 0; i < 5; i++) {
      // Preprotocolo
      while (turno != 2);

      // Sección Crítica
      printlnI("P2_SC1");
      printlnI("P2_SC2");

      // Postprotocolo
      turno = 1;

      // Sección No Crítica
      printlnI("P2_SNC1");
      printlnI("P2_SNC2");
   }
}
```

1ª Aproximación: Alternancia Obligatoria

- A continuación se muestra una posible intercalación de las instrucciones
- Se supone que el turno comienza en 1
- Se puede observar cómo las instrucciones de la sección crítica del proceso p1 no se intercalan con las instrucciones de la sección crítica de p2
- Se puede observar cómo las instrucciones que no están en la sección crítica se pueden intercalar con instrucciones de la sección crítica

	pA	pB	turno
1	`while (turno != 1);`		1
2	`println("P1_SC1");`		1
3		`while (turno != 2);`	1
4	`println("P1_SC2");`		1
5	`turno = 2;`		2
6		`while (turno != 2);`	2
7	`println("P1_SNC1");`		2
8		`println("P2_SC1");`	2
9	`println("P1_SNC2");`		2
10		`println("P2_SC1");`	2
11	`while (turno != 1);`		2
12	`while (turno != 1);`		2
13		`turno = 1;`	1
14	`while (turno != 1);`		1
15	`println("P1_SC1");`		1
16		`println("P2_SNC1");`	1
17	`println("P1_SC2");`		1
18	…		

Iª Aproximación: Alternancia Obligatoria

- **Requisitos que cumple**
  - Exclusión Mutua
  - Ausencia de Interbloqueos
  - Ausencia de Inanición
- **Requisitos que no cumple**
  - Ausencia de Retrasos Innecesarios (debido a que la alternancia es obligatoria y no debería serlo)

Sincronización con Espera Activa

- Introducción
- Sincronización Condicional
- Exclusión Mutua
  - Propiedades de Corrección
  - 1ª Aproximación: Alternancia Obligatoria
  - **2ª Aproximación: No Exclusión Mutua**
  - 3ª Aproximación: Interbloqueo
  - 4ª Aproximación: Espera indefinida
  - Solución: Algoritmo de Dekker
  - Uso de la Exclusión Mutua
- Espera Activa vs Herramientas de sincronización
- Conclusiones

2ª Aproximación: No exclusión mutua

- Para evitar la alternancia obligatoria, podemos usar una **variable booleana** por cada proceso que indique si dicho proceso **está en la sección crítica**
- Un proceso al entrar en la sección crítica comprueba si ya hay otro proceso y si no hay nadie, indica que entra él
- Al salir de la sección crítica el proceso indica que ya no está en ella

2ª Aproximación: No exclusión mutua

```
static volatile boolean p1sc;
static volatile boolean p2sc;
```

```
for (int i = 0; i < 5; i++) {
  // Preprotocolo
  while (p2sc);
  p1sc = true;

  // Sección Crítica
  println("P1_SC1");
  println("P1_SC2");

  // Postprotocolo
  p1sc = false;

  // Sección No Crítica
  println("P1_SNC1");
  println("P1_SNC2");
}
```

```
for (int i = 0; i < 5; i++) {
  // Preprotocolo
  while (p1sc);
  p2sc = true;

  // Sección Crítica
  println("P2_SC1");
  println("P2_SC2");

  // Postprotocolo
  p2sc = false;

  // Sección No Crítica
  println("P2_SNC1");
  println("P2_SNC2");
}
```

	p1	p2	p1sc	p2sc
1	`while (p2sc);`		false	false
2	`p1sc = true;`		false	false
3		`while (p1sc);`	true	false
4	`println("P1_SC1");`		true	false
5		`while (p1sc);`	true	false
6	`println("P1_SC2");`		true	false
7		`while (p1sc);`	true	false
8	`p1sc = false;`		false	false
9		`while (p1sc);`	false	true
10	`println("P1_SNC1");`		false	true
11		`p2sc = true;`	false	true
12		`println("P2_SC1");`	false	true
13	`println("P1_SNC1");`		false	true
14	`while (p2sc);`		false	true
15		`println("P2_SC2");`	false	true
16	`while (p2sc);`		false	true
17		`p2sc = false;`	false	false
18	`while (p2sc);`		false	false
19	...			

2ª Aproximación: No exclusión mutua

- Aunque existan intercalaciones que cumplen con los requisitos, existen intercalaciones en las que **no se tiene la exclusión mutua**
- Los dos procesos pueden ejecutar las instrucciones de la sección crítica de forma **intercalada**

2ª Aproximación: No exclusión mutua

- Requisitos que no cumple
  - Exclusión Mutua

	p1	p2	p1sc	p2sc
1	`while (p2sc)`		false	false
2		`while (p1sc)`	false	false
3	`p1sc = true;`		true	false
4		`p2sc = true;`	true	true
5	`println("P1_SC1");`		true	true
6		`println("P2_SC1");`	true	true
7	`println("P1_SC2");`		true	true
8		`println("P2_SC2");`	true	true
9	…			

Sincronización con Espera Activa

- Introducción
- Sincronización Condicional
- Exclusión Mutua
  - Propiedades de Corrección
  - 1ª Aproximación: Alternancia Obligatoria
  - 2ª Aproximación: No Exclusión Mutua
  - **3ª Aproximación: Interbloqueo**
  - 4ª Aproximación: Espera indefinida
  - Solución: Algoritmo de Dekker
  - Uso de la Exclusión Mutua
- Espera Activa vs Herramientas de sincronización
- Conclusiones

3ª Aproximación: Interbloqueo

- El problema de la aproximación anterior es que los dos procesos miran y después entran, pudiendo entrar los dos a la vez
- En la 3ª aproximación, antes de comprobar si hay alguien dentro, vamos a hacer una petición
- Si alguien lo ha pedido ya, nos esperamos

3ª Aproximación: Interbloqueo

```
static volatile boolean p1p;
static volatile boolean p2p;
```

```
for (int i = 0; i < 5; i++) {
  // Preprotocolo
  p1p = true;
  while (p2p);

  // Sección Crítica
  println("P1_SC1 ");
  println("P1_SC2 ");

  // Postprotocolo
  p1p = false;

  // Sección No Crítica
  println("P1_SNC1 ");
  println("P1_SNC2 ");
}
```

```
for (int i = 0; i < 5; i++) {
  // Preprotocolo
  p2p = true;
  while (p1p);

  // Sección Crítica
  println("P2_SC1 ");
  println("P2_SC2 ");

  // Postprotocolo
  p2p = false;

  // Sección No Crítica
  println("P2_SNC1 ");
  println("P2_SNC2 ");
}
```

	p1	p2	p1p	p2p
1	`p1p = true;`		true	false
2	`while (p2p)`		true	false
3		`p2p = true;`	true	true
4		`while (p1p)`	true	true
5	`printlnI("P1_SC1 ");`		true	true
6		`while (p1p)`	true	true
7	`printlnI("P1_SC2 ");`		true	true
8	`p1p = false;`		false	true
9		`while (p1p)`	false	true
10	`printlnI("P1_SNC1 ");`		false	true
11	`printlnI("P1_SNC2 ");`		false	true
12		`printlnI("P2_SC1 ");`	false	true
13	`p1p = true;`		true	true
14	`while (p2p)`		true	true
15		`printlnI("P2_SC2 ");`	true	true
16		`p2p = false;`	true	false
17	`while (p2p)`		true	false
18	`printlnI("P1_SC1 ");`		true	false
19	…			

3ª Aproximación: Interbloqueo

- Requisitos que cumple
  - Exclusión Mutua
- Requisitos que no cumple
  - Ausencia de interbloqueos

	p1	p2	c.p1p	c.p2p
1	`p1p = true;`		`true`	`false`
2		`p2p = true;`	`true`	`true`
3	`while (p2p)`		`true`	`true`
4		`while (p1p)`	`true`	`true`
5		`while (p1p)`	`true`	`true`
6		`while (p1p)`	`true`	`true`
7	`while (p2p)`		`true`	`true`

Sincronización con Espera Activa

- Introducción
- Sincronización Condicional
- Exclusión Mutua
  - Propiedades de Corrección
  - 1ª Aproximación: Alternancia Obligatoria
  - 2ª Aproximación: No Exclusión Mutua
  - 3ª Aproximación: Interbloqueo
  - **4ª Aproximación: Espera indefinida**
  - Solución: Algoritmo de Dekker
  - Uso de la Exclusión Mutua
- Espera Activa vs Herramientas de sincronización
- Conclusiones

4ª Aproximación: Espera indefinida

- La 3ª aproximación falla porque los dos procesos, una vez que anuncian su intención de entrar en su sección crítica, insisten en su derecho de entrar en ella
- La 4ª aproximación cede su derecho a entrar en la sección crítica si descubre que hay competencia con otro proceso

4ª Aproximación: Espera indefinida

```
for (int i = 0; i < 5; i++) {
   // Preprotocolo
   p1p = true;
   while (p2p){
      p1p = false;
      p2p = true;
   }

   // Sección Crítica
   println("P1_SC1");
   println("P1_SC2");

   // Postprotocolo
   p1p = false;

   // Sección No Crítica
   println("P1_SNC1");
   println("P1_SNC2");
}
```

```
static volatile boolean p1p;
static volatile boolean p2p;
```

4ª Aproximación: Espera indefinida

- Requisitos que cumple
  - Exclusión Mutua
- Requisitos que no cumple
  - Interbloqueo activo
    - Hay intercalaciones en las que ningún proceso entra en la sección crítica

Espera Indefinida

	p1	p2	p1p	p2p
1	`p1p = true;`		`true`	`false`
2		`p2p = true;`	`true`	`true`
3	`while (p2p)`		`true`	`true`
4		`while (p1p)`	`true`	`true`
5	`p1p = false;`		`false`	`true`
6		`p2p = false;`	`false`	`false`
7	`p1p = true;`		`true`	`false`
8		`p2p = true;`	`true`	`true`
9	`while (p2p)`		`true`	`true`
10		`while (p1p)`	`true`	`true`
11	`p1p = false;`		`false`	`true`
12		`p2p = false;`	`false`	`false`

4ª Aproximación: Espera indefinida

- Requisitos que no cumple
  - Inanición
    - Hay intercalaciones en las que un proceso no entra nunca en la sección crítica mientras que otro proceso entra repetidas veces

	p1	p2	p1p	p2p
1	`p1p = true;`		`true`	`false`
2		`p2p = true;`	`true`	`true`
3		`while (p1p)`	`true`	`true`
4		`p2p = false;`	`true`	`false`
5	`while (p2p)`		`true`	`false`
6		`p1p = true;`	`true`	`true`
7	`println("P1_SC1");`		`true`	`true`
8		`while (p1p)`	`true`	`true`
9	`println("P1_SC2");`		`true`	`true`
10		`p2p = false;`	`true`	`false`
11	`p1p = false;`		`false`	`false`
12	`println("P1_SNC1");`		`false`	`false`
13	`println("P1_SNC2");`		`false`	`false`
14		`p1p = true;`	`false`	`true`
15	`p1p = true;`		`true`	`true`
16		`while (p1p)`	`true`	`true`
17		`p2p = false;`	`true`	`false`
18	`while (p2p)`		`true`	`false`
19	`println("P1_SC1");`		`true`	`false`

Sincronización con Espera Activa

- Introducción
- Sincronización Condicional
- Exclusión Mutua
  - Propiedades de Corrección
  - 1ª Aproximación: Alternancia Obligatoria
  - 2ª Aproximación: No Exclusión Mutua
  - 3ª Aproximación: Interbloqueo
  - 4ª Aproximación: Espera indefinida
  - **Solución: Algoritmo de Dekker**
  - Uso de la Exclusión Mutua
- Espera Activa vs Herramientas de sincronización
- Conclusiones

Solución: Algoritmo de Dekker

- Es una combinación de la 1ª y 4ª aproximación
- Las variables booleanas aseguran la exclusión mutua (como en la 4ª aproximación)
- Al detectar competencia, una variable turno se encarga del "desempate"

Solución: Algoritmo de Dekker

```
for (int i = 0; i < 5; i++) {

   // Preprotocolo
   p1p = true;
   while (p2p){
      if(turno != 1){
         p1p = false;
         while(turno != 1);
         p1p = true;
      }
   }

   // Sección Crítica
   println("P1_SC1 ");
   println("P1_SC2 ");

   // Postprotocolo
   p1p = false;
   turno = 2;

   // Sección No Crítica
   println("P1_SNC1 ");
   println("P1_SNC2 ");
}
```

```
for (int i = 0; i < 5; i++) {

   // Preprotocolo
   p2p = true;
   while (p1p){
      if(turno != 2){
         p2p = false;
         while(turno != 2);
         p2p = true;
      }
   }

   // Sección Crítica
   println("P2_SC1 ");
   println("P2_SC2 ");

   // Postprotocolo
   p2p = false;
   turno = 1;

   // Sección No Crítica
   println("P2_SNC1 ");
   println("P2_SNC2 ");
}
```

Solución: Algoritmo de Dekker

- Requisitos que cumple
  - Exclusión Mutua
  - Ausencia de Interbloqueos
  - Ausencia de Retrasos Innecesarios
  - Ausencia de Inanición

Una demostración más rigurosa se puede encontrar en el libro Ben-Ari, M. *Principles of Concurrent and Distributed Programming*. Ed. Prentice Hall, 1.990.

Sincronización con Espera Activa

- Introducción
- Sincronización Condicional
- Exclusión Mutua
  - Propiedades de Corrección
  - 1ª Aproximación: Alternancia Obligatoria
  - 2ª Aproximación: No Exclusión Mutua
  - 3ª Aproximación: Interbloqueo
  - 4ª Aproximación: Espera indefinida
  - Solución: Algoritmo de Dekker
  - **Uso de la Exclusión Mutua**
- Espera Activa vs Herramientas de sincronización
- Conclusiones

Uso de la Exclusión Mutua

- Una instrucción atómica es aquella que se ejecuta como una unidad indivisible
- El lenguaje de programación y el hardware definen las instrucciones atómicas en las que se divide cada sentencia

Sentencia

```
x = x+1
```

Instrucciones Atómicas

```
LOAD R,x
ADD R,#1
STR R,x
```

Uso de la Exclusión Mutua

- Supongamos que dos procesos quieren usar una **variable común** para contar las acciones realizadas
- Según hemos visto, si dos procesos quieren **incrementar la misma variable** existen intercalaciones de las instrucciones atómicas que producen errores en la cuenta
- Para el desarrollador sería muy interesante que el **incremento** de una variable fuese una **instrucción atómica**

Uso de la Exclusión Mutua

- Tipos de instrucciones atómicas
  - De grano fino
    - Ofrecidas por el lenguaje de programación y el hardware al desarrollador
    - Habitualmente se corresponden con las instrucciones máquina del procesador
  - De grano grueso
    - Conjunto de sentencias que ejecuta un proceso sin interferencias de otros procesos
    - Los lenguajes de programación y el sistema hardware disponen de mecanismos para hacer que un grupo de sentencias se ejecuten como una instrucción atómica de grano grueso

Uso de la Exclusión Mutua

- La **sección crítica** de la exclusión mutua es una **instrucción atómica de grano grueso**
  - Es **indivisible** en el sentido de que ningún otro proceso puede interferir en el uso del recurso compartido de acceso exclusivo
    - Si dentro de la sección crítica se incrementa una variable, ningún otro proceso podrá interferir en esa sentencia
  - Pero es **divisible** en el sentido de que se pueden intercalar instrucciones atómicas de la sección no crítica de otros procesos

Uso de la Exclusión Mutua

- Se ha visto un algoritmo de Exclusión Mutua para dos procesos.
- **El algoritmo de Lamport** es un algoritmo que permite tener una exclusión mutua para más de dos procesos
- Este algoritmo es complejo y no vamos a ver su implementación en detalle
- Hasta que estudiemos los **semáforos**, usaremos la exclusión mutua implementada con espera activa

Uso de la Exclusión Mutua

- **Condición de carrera (*race condition*)**
  - Se dice que se ha producido una condición de carrera cuando un programa concurrente se **comporta** de una forma **anómala** debido a que los procesos no están bien sincronizados entre sí
  - Habitualmente las condiciones de carrera ocurren sólo **ocasionalmente** y aparecen en ciertas intercalaciones de instrucciones

Sincronización con Espera Activa

- Introducción
- Sincronización Condicional
- Exclusión Mutua
- Espera Activa vs Herramientas de sincronización
  - **Problemas de la Espera Activa**
  - Herramientas de Espera Pasiva
- Conclusiones

Problemas de la Espera Activa

- Se ha trabajado en el modelo de Memoria Compartida usando variables compartidas para la sincronización
- Se denomina **Espera Activa** porque los procesos están ejecutando instrucciones (están activos) incluso cuando tienen que esperar para poder continuar su ejecución

```
while (!continuar);
```

- También se la conoce como ***Busy waiting*** o ***spinning*** *o* ***polling*** *(aunque este último término es más usado en entrada/salida)*

Problemas de la Espera Activa

- La Espera Activa tiene bastantes **problemas**
  - Multiprogramación
    - Los procesos que están esperando están malgastando el procesador que podría usarse por otros procesos que realmente estén realizando un trabajo útil
  - Multiproceso
    - Un procesador ejecutando instrucciones consume energía y por tanto disipa calor
    - Si las instrucciones no son útiles, el procesador podría estar en reposo

La Espera Activa es muy ineficiente y en general debería evitarse

Problemas de la Espera Activa

- Para solucionar estos problemas, se desarrollaron **herramientas de sincronización de procesos** en los procesadores, librerías y lenguajes
- Con estas herramientas, cuando un proceso no puede continuar ejecutando las sentencias se **bloquea** y deja de ejecutar sentencias hasta que otro proceso lo desbloquea cuando se cumplen las condiciones para que siga ejecutando
- Esto permite aprovechar de forma mucho mas adecuada los recursos (capacidad de cómputo, energía, …)

Sincronización con Espera Activa

- Introducción
- Sincronización Condicional
- Exclusión Mutua
- Espera Activa vs Herramientas de sincronización
  - Problemas de la Espera Activa
  - **Herramientas de sincronización**
- Conclusiones

Herramientas de sincronización

Sincronización con Espera Activa

- En la programación funcional y en la programación orientada a objetos, la gran mayoría de los lenguajes de programación implementan los mismos conceptos
  - Funcional: Funciones, listas, patrones…
  - Orientación a Objetos: Clases, objetos, métodos, atributos…
- En la programación concurrente no ocurre lo mismo y cada **lenguaje de programación , cada librería y cada sistema operativo** implementan sus **propias** herramientas de sincronización
- Algunas herramientas básicas suelen estar disponibles en todos las tecnologías

Herramientas de sincronización

Sincronización con Espera Activa

- Herramientas de sincronización de procesos
  - Modelo de Memoria Compartida
    - **Semáforos**
    - Regiones Críticas
    - Regiones Críticas Condicionales
    - Monitores
    - Sucesos
    - Buzones
    - Recursos
  - Modelo de Paso de Mensajes
    - Envío asíncrono
    - Envío síncrono o cita simple
    - Invocación Remota o cita extendida

Sincronización con Espera Activa

- Introducción
- Sincronización Condicional
- Exclusión Mutua
- Espera Activa vs Herramientas de sincronización
- **Conclusiones**

Conclusiones

- Propiedades de corrección que debe cumplir un programa concurrente
  - **Propiedades de seguridad (*safety*)**
    - Exclusión Mutua
    - Ausencia de Interbloqueo pasivo
  - **Propiedades de Vida (*liveness*)**
    - Ausencia de Retrasos innecesarios
    - Ausencia de inanición (*starvation*)
    - Ausencia de interbloqueo activo (*livelock*)

Conclusiones

Sincronización con Espera Activa

- La Espera Activa es una técnica muy **ineficiente**
- **No debe usarse nunca** para desarrollar programas concurrentes
- Se estudia en detalle por diversos motivos
  - Para introducir al alumno los **conceptos** de la programación concurrente y sepa estudiar el comportamiento de un programa de este tipo
  - Para que conozca las **propiedades de corrección** que todo programa concurrente debe cumplir
  - Por **motivos históricos**, ya que al inicio de la programación concurrente era la única técnica disponible