SINCRONIZACIÓN Y COMUNICACIÓN

Karim Guevara Puente de la Vega

2017

Mecanismos de sincronización

- Exclusión mutua con espera activa
- SLEEP y WAKEUP
- Semáforos
- Monitores
- Transferencia de mensajes

Exclusión mutua con espera activa

- Inhabilitación de interrupciones
- Variables de bloqueo
- Alternancia estricta
- Peterson
- Hardware de sincronización: TSL

Inhabilitación de interrupciones

- Cada proceso inhabilite las interrupciones justo después de ingresar en su región crítica, y vuelva a habilitarlas justo antes de salir de ella.
  - **Entrar_RC** → DISABLE (inhabilitar las interrupciones)
  - **Salir_RC** → ENABLE (habilitar las interrupciones).
- Con las interrupciones inhabilitadas, no hay interrupciones de reloj.
  - CPU sólo se conmuta como resultado de interrupciones de reloj o de otro tipo.
- Proceso puede examinar y actualizar la memoria compartida sin temor: **operaciones atómicas**.

Inhabilitación de interrupciones

shared double balance,cantidad;

Programa para p1

```
. . .
deshabilitar_Interrupciones();
balance=balance+cantidad;
habilitar_Interrupciones();
. . .
```

shared double balance, cantidad;

Programa para p2

```
. . .
deshabilitar_Interrupciones();
balance=balance-cantidad;
habilitar_Interrupciones();
. . .
```

Inhabilitación de interrupciones

- Región crítica debe ser corta
  - Perder interrupciones generadas por los dispositivos de E/S.
- Inhabilitando las interrupciones, también se impide la ejecución de otros procesos
  - Procesos cooperantes que no intentan entrar en la región crítica y aquellos que no lo son.
- Peligroso darle al usuario la posibilidad de inhabilitar las interrupciones
  - Pérdida de las subsiguientes interrupciones y el acaparamiento en exclusiva de la CPU.

Inhabilitación de interrupciones

- Por lo tanto…
  - Técnica útil o práctica para el núcleo
  - Método apropiado para sistemas monoprocesador.
  - Pero no es apropiada como mecanismo general de exclusión mutua para los procesos de usuario

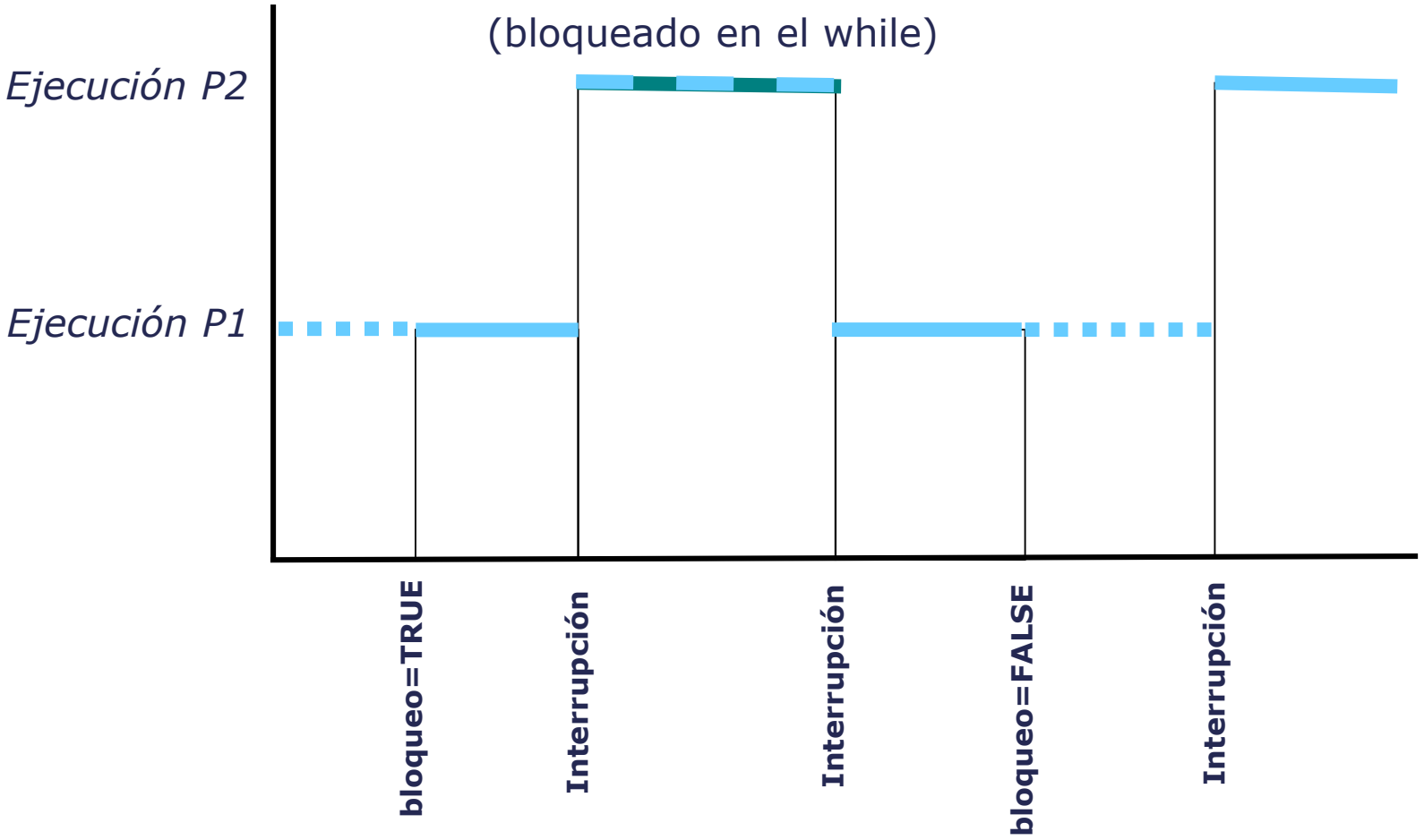
Variables de bloqueo

- Variable compartida booleana, indica si la SC está ocupada o no:
  - 0 (false) : libre
  - 1 (true) : ocupada

- Cuando un proceso llegue a la entrada de la SC debe consultar el valor de la variable bloqueo(candado)
  - Si es 0 → la pone en 1 e ingresa a la SC
  - Si es 1 → el proceso debe de esperar hasta que sea 0

- … Espera activa

Espera activa



Variables de bloqueo

```
shared boolean bloqueo = FALSE;
shared double cantidad, balance;

Programa para P1
. . .
while (bloqueo) { NULL;}
bloqueo=TRUE;
balance=balance+cantidad;
bloqueo=FALSE;
. . .
```

```
Programa para P2
. . .
while (bloqueo) { NULL;}
bloqueo=TRUE;
balance=balance-cantidad;
bloqueo=FALSE;
. . .
```

- La consulta del candado y su posterior puesta *true*, no es atómica
  - No se asegura la exclusión mutua

Variables de bloqueo

- … una nueva sección crítica?

```
shared boolean bloqueo = FALSE;

Programa para P1
. . .
while (bloqueo) { NULL;}
bloqueo=TRUE;

<Sección crítica>;
bloqueo=FALSE;
. . .
```

```
Programa para P2
. . .
while (bloqueo) { NULL;}
bloqueo=TRUE;
<Sección crítica>;
bloqueo=FALSE;
. . .
```

Variables de bloqueo

```
entrar(bloqueo) {
    deshabilitar_Interrupciones();
    while(bloqueo){
            habilitar_Interrupciones();
            deshabilitar_Interrupciones();
    }
    bloqueo=TRUE;
    habilitar_Interrupciones();
}
```

```
salir(bloqueo) {
    deshabilitar_Interrupciones();
    bloqueo=FALSE;
    habilitar_Interrupciones();
}
```

shared double balance, cantidad;
shared boolean bloqueo=FALSE;

Programa para p1

entrar(bloqueo);
balance=balance+cantidad;
salir(bloqueo);

Programa para p2

entrar(bloqueo);
balance=balance-cantidad;
salir(bloqueo);

Alternancia estricta

- Utiliza una variable compartida que indica a qué proceso le corresponde ingresar a la SC.
- Inicialmente es FALSE:
  - El primer proceso examina la variable y ve que es FALSE ingresa a la SC.
  - El segundo proceso también ve que el FALSE, pero…
    - No ingresa a la SC, da vueltas en ciclos (espera activa)
    - Problema: bloqueo giratorio
  - Al salir el primero pone a TRUE la variable
    - El segundo puede ingresar.

Alternancia estricta

```
shared boolean bloqeo = FALSE;
shared boolean cantidad, balance;

Programa para P1
. . .
while (bloqueo) { NULL;}
balance = balance + cantidad;
bloqueo = TRUE;
. . .
```

```
Programa para P2
. . .
while (!bloqueo) { NULL;}
balance = balance - cantidad;
bloqueo = FALSE;
. . .
```

- Esta solución viola la condición de espera limitada (hambruna)
- Requiere que los procesos se alternen para ingresar a sus SC.

Peterson

- Combina la idea de tomar turnos con la de tener variables de candado y variables de advertencia
  - Solución para el problema de la exclusión mutua sin alternancia estricta .
- 1981, Peterson descubre una forma más sencilla de lograr la exclusión mutua
  - Manejar más información, una variable más.
    - Será verdadera si un proceso quiere entrar a la sección crítica.

Peterson

```
#define FALSE 0
#define TRUE 1
#define N  2

int turno;
int interesado [N];

void enter_region (int process)
{
      int otro;
      otro = 1-process;
      interesado [process] = TRUE;
      turno = process;
      while (turno == process && interesado [otro] == TRUE);
}

void leave_region (int process)
{
      interesado [process] = FALSE;
}
```

- Dos procesos 0 y 1
- Cada proceso invoca ***enter_region*** para ingresar a SC.
- Después de SC el proceso invoca ***leave_region()***

Hardware de sincronización-TSL

- Instrucción especial para resolver el problema de la sección crítica.
- Instrucción que combinan dos o tres operaciones en una sola operación atómica.

```
TSL RX,bloqueo
```

Test and Set Lock – probar y escribir

  - Lee el contenido de la palabra de memoria (*bloqueo*),
  - Lo coloca en el registro *RX*
  - Le pone un valor diferente a *0* o *FALSE*

Hardware de sincronización-TSL

- Se garantiza que: la lectura de la palabra y la escritura en ella son indivisibles.
  - Ningún otro proceso o procesador puede tener acceso a la palabra de memoria antes de acabarse de ejecutar TSL.
- En sistemas multiprocesador, la CPU que ejecuta TSL bloquea el bus de memoria.
- Se requiere una variable compartida **bloqueo** para coordinar el acceso a la memoria compartida

Hardware de sincronización-TSL

- ¿cómo se puede evitar que dos procesos entren a sus SC a la vez?

```
enter_region:
        tsl register, lock
        cmp register, 0
        jne enter_region
        ret

leave_region:
        move lock,0
        ret
```

```
Pi:
        ….
        ….
        enter_region;
        seccion_critica;
        leave_region;
        ….
        ….
```

Mecanismos de sincronización

- Exclusión mutua con espera activa
- SLEEP y WAKEUP
- Semáforos
- Monitores
- Transferencia de mensajes

SLEEP & WAKEUP

- Problema:
  - Espera activa
  - Inversión de prioridades, hambruna
- Solución…. SLEEP y WAKEUP
  - Se bloquean, no desperdician tiempo de CPU
- **SLEEP**: llamada al sistema, hace que el invocador se bloquee.
- **WAKEUP**: llamada que activa al proceso que lleva como parámetro.
- Se relacionan a partir de una dirección de memoria

Productor -Consumidor

- Buffer limitado
  - Dos procesos comparten un buffer de tamaño fijo
  - **Proceso productor**: coloca información
  - **Proceso consumidor**: saca información
- Problemas:
  - Buffer lleno y el productor quiere colocar información
  - Buffer vacío y el consumidor quiere sacar información
- Solución: el productor y consumidor deben **bloquearse** en el momento adecuado

Productor -Consumidor

```
#define N 100
int count = 0;

void producer(void){
  int ítem;
  while (TRUE) {
    item = produceItem();
    if (count == N)
       sleep();
    insertItem(item);
    count = count + 1;
    if (count == 1)
       wakeup(consumer);
  }
}
```

```
void consumer(void){
  int item;

  while (TRUE) {
    if (count == 0)
       sleep();
    item = removeItem();
    count = count - 1;
    if (count == N - 1)
       wakeup(producer);
    consumeItem(item);
  }
}
```

- ¿Hay condiciones de competencia?
  - Bit de espera para activar: alcancía de señales de activar (despertar)
- ¿Si hay n procesos?.... ¿cuánta memoria es necesaria?