Sistemas Operativos

Comunicación y Sincronización - I

Sistemas Operativos

- ☑ Versión: Abril 2017
- ☑ Palabras Claves: Proceso, Comunicación, Mensajes, mailbox, port, send, receive, IPC, Productor, Consumidor

Algunas diapositivas han sido extraídas de las ofrecidas para docentes desde el libro de Stallings (Sistemas Operativos) , el de Silberschatz (Operating Systems Concepts)

Linux Kernel Development - 3er Edicion – Robert Love (Caps. 9 y 10)

Comunicación entre procesos

- ¿Como hacer para pasar información de un proceso a otro?
- ¿Cómo hacer para que no se “superpongan” entre sí?
- ¿Cómo obtener una secuencia apropiada de dependencias? (p.e., cuando uno produce y otro consume)

Comunicación entre Threads

- Facilidad por compartir el espacio de direcciones
  - Hilos de un mismo proceso
- Los problemas de superposición y dependencias siguen ocurriendo.

Para qué sirven los procesos cooperativos?

- ☑ Para compartir información (por ejemplo, un archivo)
- ☑ Para acelerar el cómputo (separar una tarea en subtareas que cooperan ejecutándose paralelamente)
- ☑ Para planificar tareas de manera tal que se puedan ejecutar en paralelo.

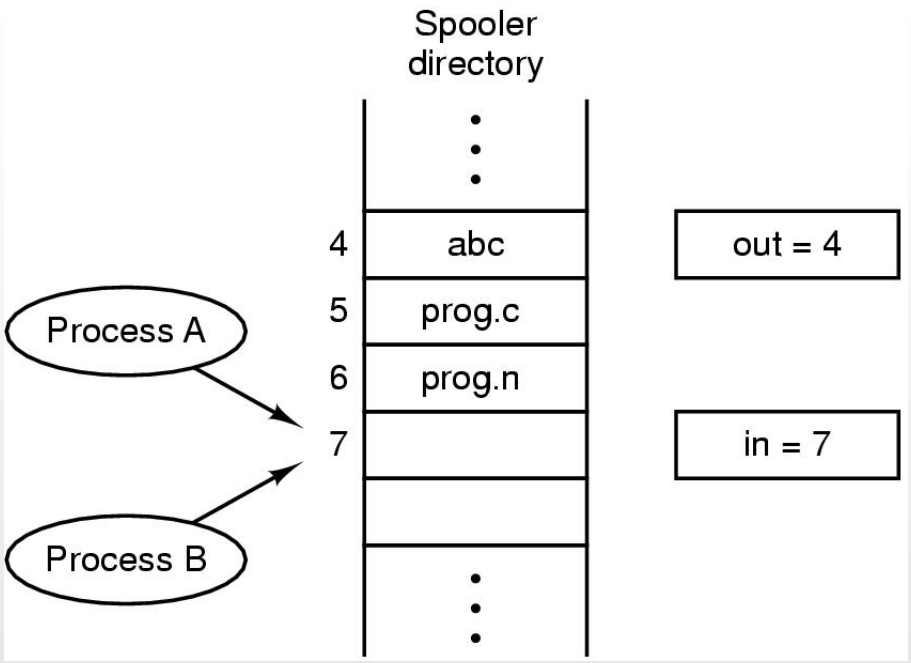
Dificultades de la Concurrencia

- **Compartir recursos globales**: Si dos procesos hacen uso de una variable compartida, el orden de acceso al recurso es crítico
- **Gestión de la asignación óptima de recursos**: Puede dar lugar a bloqueos mutuos (*interbloqueo*)
- **Dificultad de localizar errores de programación**: Los resultados no son ni deterministas ni reproducibles

Condición de carrera

- El resultado final depende del orden en que se ejecuten los procesos.
- Ejemplo:
  - P3 y P4 comparten la variable b y c.
  - Están inicializadas b=1, c=2
  - P3 ejecuta b=b+c
  - P4 ejecuta c=b+c
  - El valor final depende del orden de ejecución

Condición de carrera



- in: apunta a la siguiente ranura libre
- out: apunta al siguiente archivo a imprimir
- Ranuras 0 a 3, vacías (ya Se imprimieron)
- Ranuras 4 a 6, con archivos a imprimir

A lee in(7) y deja el procesador
B se ejecuta y lee in(7)
B almacena en el lugar 7 el nombre del archivo a imprimir.
Cuando le toca a A, sobreescribe la ranura 7

Ejemplo en monoprocesador

```
void echo()
    {
      cent=getchar();
      csal=cent;
      putchar(csal)
    }
```

- P1 invoca echo y se interrumpe cuando getchar devuelve el valor (p.e., x). Cent se modificó (vale x)
- P2 invoca echo. Este se ejecuta hasta concluir y muestra el carácter y.
- P1 se reactiva. Cent ya no tiene el valor x (contiene y). En el putchar se muestra y.

Ejemplo en multiprocesador

Proceso 1	Proceso 2
cent= getchar()	..
…	cent= getchar()
csal=cent	csal=cent
putchar(csal)	…
…	putchar(csal)

Implementación de soluciones

- Prohibir que más de un proceso lea y escriba datos compartidos al mismo tiempo
  - Exclusión mutua
- Delegar responsabilidad en los procesos (modo usuario)
  - Herramientas del SO
  - Herramientas de los Lenguajes
- Diseñar el Kernel para garantizar EM (modo kernel)

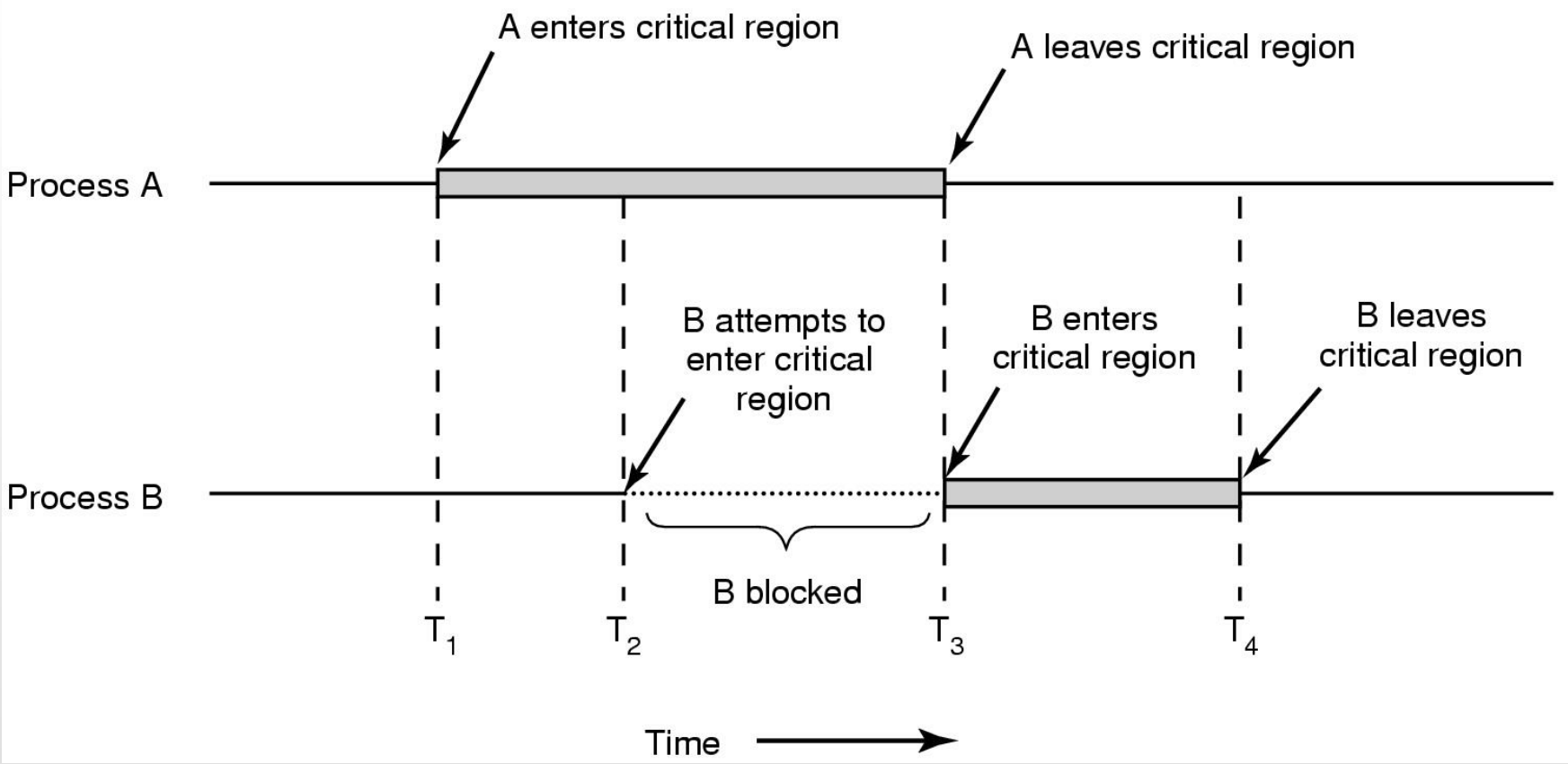
Concepto de sección crítica

- Sección de código en un proceso que accede a recursos compartidos y que no puede ser ejecutada mientras otro proceso esté en esa sección de código
  - Se protegen datos, no código
  - El SO también presenta secciones criticas.

Condiciones para la Exclusión Mutua

- ☑ Dos procesos no pueden estar simultáneamente dentro de sus regiones críticas
- ☑ No se pueden hacer suposiciones en cuanto a velocidades o cantidad de CPUs
- ☑ Ningún proceso que se ejecute fuera de su SC puede bloquear otros procesos
- ☑ Ningún proceso tiene que esperar “para siempre” para entrar en su SC.

Sección Critica



La solución a SC debe satisfacer

- ☑ Exclusión mutua (mutual exclusion)
- ☑ Continuidad (Progress)
- ☑ Espera limitada (Bounded Waiting)

Estructura General de un Proceso

Repeat

Entry section

Critical section

Exit section

Remainder section

Until false;

Posibles Soluciones

- Soluciones por Software:
  - Variables lock
  - Solución de Peterson
- Soluciones por Hardware:
  - Deshabilitar interrupciones
  - Instrucción TSL/Test and set lock
  - Instrucción xchg/swap

Espera activa, cíclica o ocupada

- Busy waiting o spin waiting
- El proceso no hace nada hasta obtener permiso para entrar en la SC.
- Continúa ejecutando la/las instrucciones de la entrada a la SC.
- Se usa cuando hay expectativa razonable que la espera será corta

Variables candado (lock)

- Solución por software
- Variable compartida, con valor inicial 0
- Valores posibles: 0, ningún proceso está en la SC; 1, algún proceso está en la SC.
- El proceso lo fija en 1 al entrar a la SC.
- Posible condición de carrera (antes de ponerlo en 1, se le da la CPU al otro proceso...)

Ejemplo: Alternancia Estricta

Proceso 0

```
Repeat
   while turno not equal 0
         do no-op;
      Critical section
         turno = 0;
   Remainder section
Until  false;
```

Proceso 1

```
Repeat
   while turno not equal 1
         do no-op;
      Critical section
         turno = 1;
   Remainder section
Until  false;
```

Supongamos que P0 sale de su SC y turno=1. Si quiere volver a entrar no puede, y P1 puede estar en sección no crítica.
Problema: la diferente velocidad de los procesos.
El proceso 0 puede ser bloqueado por un proceso que NO esta en su SC

Solución de Peterson

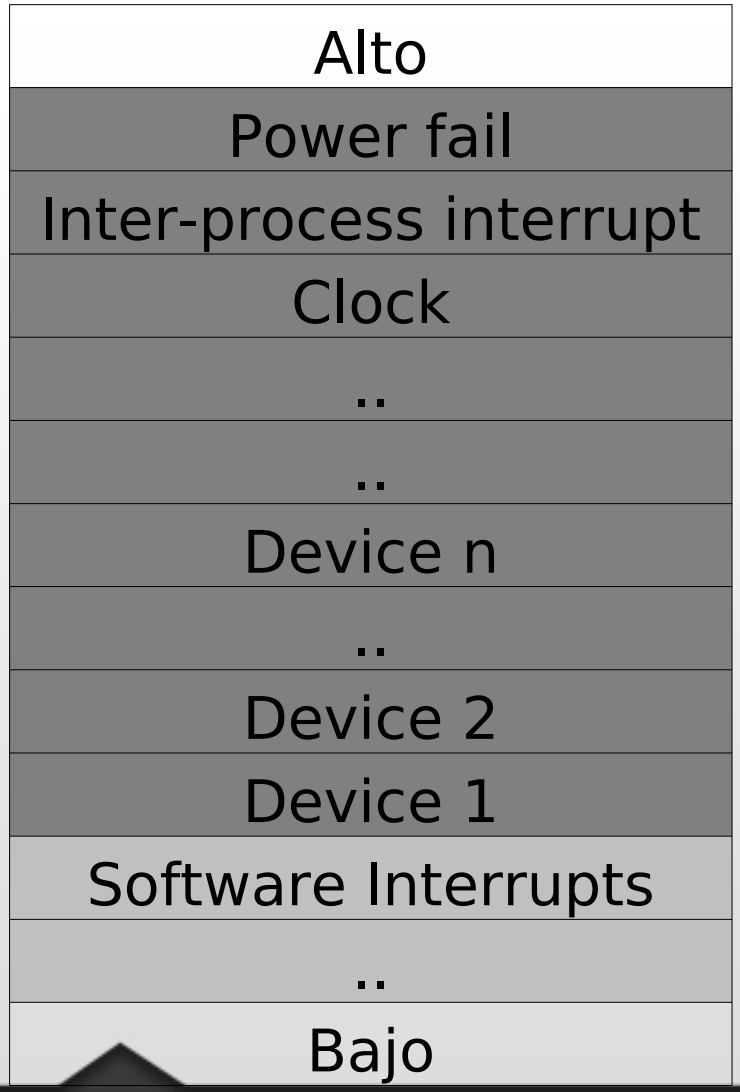
El arreglo flag se inicializa en false
Estructura del proceso i:

```
Var flag:array [0..1] of boolean;
Proc (i: int)
    J = el otro prceso
    Repeat
         flag[i]:= true;
         turn := j;
         While (flag[j] and turn=j) do no-op;
              Critical section
          flag [i] := false;
         Remainder section
    Until  false;
```

Deshabilitar interrupciones

- Conocida también como “elevar el nivel de procesador”
- Un proceso se ejecuta hasta que se invoca una System Call o es interrumpido
  - No pueden ocurrir interrupciones de reloj
- Multiprocesadores
  - Deshabilitar las interrupciones en un procesador no garantiza Exclusión Mutua
- Peligroso su uso por parte de procesos de usuario

Deshabilitar interrupciones




```
While (true)
   {
     /* deshabilitar interrupciones */;
     /* sección crítica */;
     /* habilitar interrupciones */;
     /* resto */;

     }
```

Deshabilitar interrupciones

- Ejemplo en el Kernel:
  - Deshabilitar las interrupciones mientras se está trabajando con la lista de procesos

Solución por Hardware – T&S

```
function test-and-set (var target:boolean):
   boolean;
      Begin
         test-and-set:= target:
         target:= true;
      end:
```

Exclusión mutua usando test-and-set

```
Repeat
        while test-and-set(lock) do no-op;
            Critical section
         lock := false;
           Remainder section
Until  false;
```

Solución por Hardware - Swap

```
Procedure swap (var a,b; boolean);
    var temp:boolean;
    begin
        temp := a;
        a := b;
        b := temp;
    end;
```

Exclusión mutua con swap

- ☑ Se declara una variable booleana *lock*, inicializada a *false*,
- ☑ Cada proceso cuenta con una variable booleana *key*, que es local

```
Repeat
    key := true;
    repeat
        swap (lock, key);
    until key= false;
            Critical section
   lock := false;
            Remainder section
Until  false;
```

Sección Crítica en el kernel

- Kernel Apropiativo
  - Un proceso en Modo Kernel puede ser expulsado de la CPU
- Kernel No Apropiativo
  - Un proceso en Modo Kernel no puede ser expulsado de la CPU (salvo que se bloquee o deje la CPU voluntariamente)

Sección Crítica en el kernel

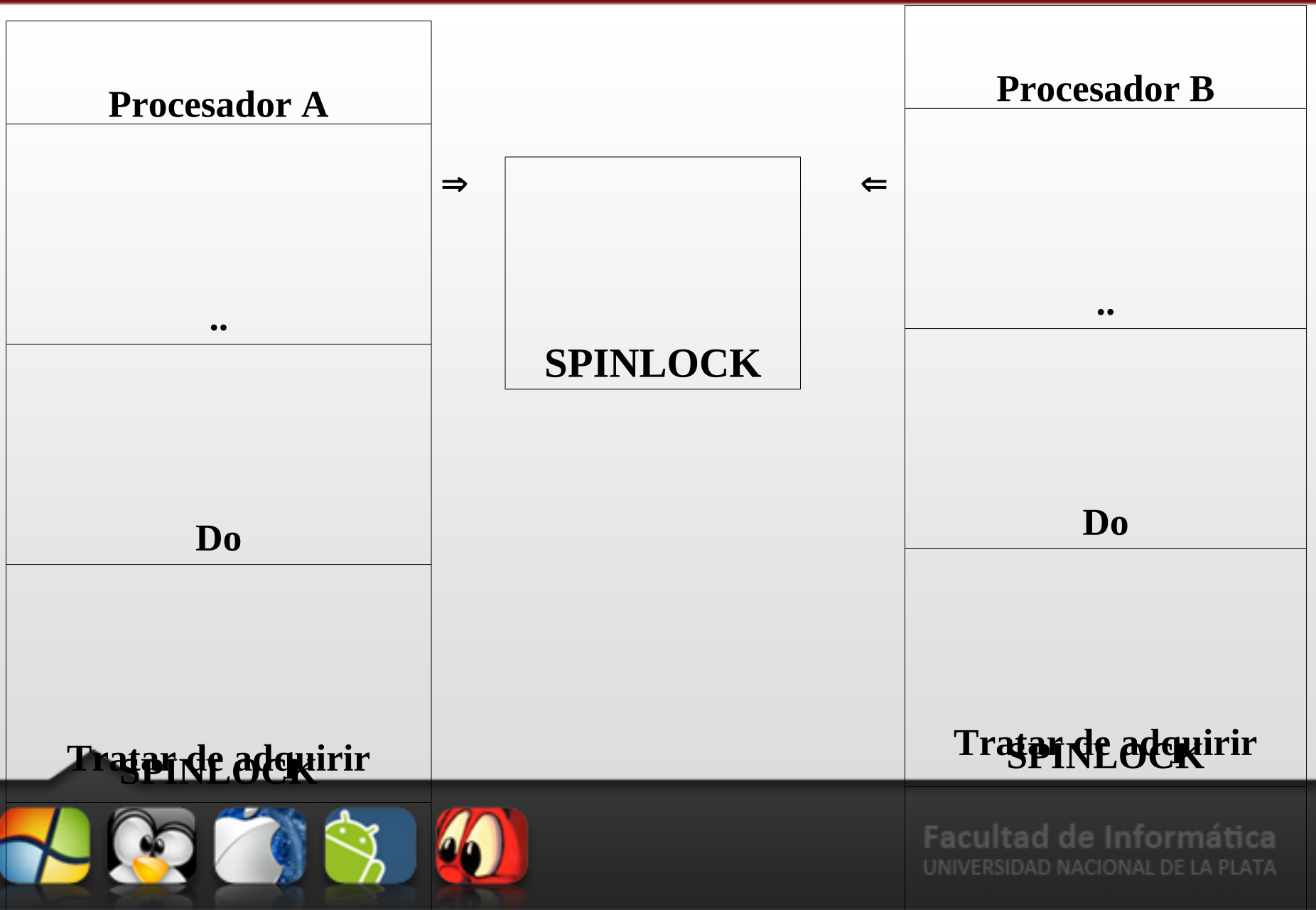
- Tener en cuenta:
  - Sistemas monoprocesador

    vs
  - Sistemas multiprocesador, simétricos, tightly coupled (comparten memoria y clock)
- Sistemas Reentrantes
  - Mas de un proceso puede estar en Kernel Mode
- Código del SO puede estar ejecutandose simultáneamente en los distintos procesadores
  - Datos globales que se comparten
- Sección crítica en el kernel: secciones de código que acceden a datos globales

Uso de instrucciones tipo lock

- Se “bloquea” el uso del bus multiprocesador para proteger la ubicación de memoria que se está accediendo.
- SPINLOCK: mecanismo que asocia una primitiva de locking a la estructura de datos que se quiere proteger
- El spinlock se asocia a la estructura a proteger
- Los procesos deben “adquirir” el spinlock

Spinlock


Procesador A
..
Do
Tratar de adquirir SPINLOCK
⇒
SPINLOCK
⇐
Procesador B
..
Do
Tratar de adquirir SPINLOCK

Spinlock (cont.)

- Se puede implementar por test-and-set
- Cuando está tratando de adquirir el spinlock se eleva nivel de procesador.
- Problemas:
  - si el código de spinlock provoca interrupciones de menor nivel (por ejemplo, si se invoca un handler de page fault).
  - Procesos Ready de Mayor Prioridad

Sincronización en Kernel Linux

Técnica	*Descripción*	*Alcance*
Operaciones Atómicas	Leer-modificar-escribir contadores atómicamente	Global
Spinlock	Lock con busy wait	Global
Semáforos	Lock con blocking wait	Global
Deshabilitar interrupciones	Sin interrupciones en un CPU	Local
Barrera de Memoria	Evitar re-ordenamiento de instrucciones	Local

Sincronización en Kernel Linux

- Operaciones atómicas
  - Tipo atomic_t (contador de 24 bits)
  - Algunas funciones
    - atomic_read(v) / atomic_set(v,i)
    - atomic_add(v,i) / atomic_sub(v,i)
    - atomic_sub_and_test(v,i)
    - atomic_inc(v) / atomic_dec(v)
  - En Multiprocesadores las funciones tienen el prefijo “lock”

Sincronización en Kernel Linux

- Spinlocks
  - Tipo spinlock_t,
  - Algunas functiones
    - spin_lock_init()
    - spin_lock() / spin_unlock()
    - spin_is_locked() / spin_trylock()
    - spin_lock_irq() / spin_unlock_irq()
  - Spinlocks de Lectura/Escritura
    - Tipo rwlock_t (32 bits, 2 datos: cantidad leyendo, flag de escritura)
    - read_lock() / read_unlock()
    - write_lock() / write_unlock()

Sincronización en Kernel Linux

- Semaforos
  - Tipo semaphore
    - count: > 0 libre, = 0 ocupado
    - wait: cola de espera de procesos
  - Algunas funciones
    - up() / down() TASK_UNINTERRUPTIBLE
    - down_trylock()
    - down_interrumpible() TASK_INTERRUPTIBLE

Sincronización en Kernel Linux

- Deshabilitar Interrupciones
  - Funciones
    - __cli() / local_irq_disable()
    - __sti() / local_irq_enable()
  - Pone a 0 en flag IF
  - Como se pueden ejecutar de manera anidada por lo que no se conoce el valor previo a __cli()
    - __save_flags(valor) / local_irq_save()
    - __restore_flags(valor) / local_irq_restore()

Sincronización en Kernel Linux

- Barreras de Memoria
  - Problemas con optimización de código
    - Compilador y/o procesador
  - Asegura que todas las instrucciones previas a la barrera se ejecuten antes de comenzar las opciones subsiguientes.
  - Algunas Funciones
    - mb()
    - rmb() / wmb()