8 – Control de recursos

1. Introducción

La sincronización entre procesos es necesaria para tareas cooperativas, ya que deben cumplir una serie de restricciones lógicas en su orden de ejecución. La resolución de la sincronización entre procesos puede resolverse con:

- **Mecanismos de bajo nivel**, como la espera activa o los semáforos.
- **Mecanismos de alto nivel**, como los monitores.

Incluso en el caso de procesos independientes desde un punto de vista lógico es necesario también utilizar mecanismos de sincronización. Esto es debido a que se comparte, en mayor o menor medida, tanto el entorno de ejecución como los recursos (que son limitados). Aparecen así tareas competitivas, pues tienen que acceder a recursos compartidos. En los sistemas en tiempo real no hay recursos ilimitados y hay que garantizar el correcto acceso mediante estos mecanismos de sincronización.

2. Mecanismos de sincronización

Los mecanismos de sincronización cumplen dos funciones básicas, por un lado, asegurar la exclusividad de acceso a un recurso y por otro, encargarse de la planificación del uso de un recurso en función de su prioridad. Estas restricciones de sincronización de ambos tipos pueden expresarse en los esquemas:

- Si se cumple una determinada condición, se atiende al proceso A.
- Si se cumple la condición, entonces la ejecución del proceso A es prioritaria sobre la de otro proceso B.

Las condiciones que se deben evaluar para realizar la sincronización tendrán en cuenta información sobre las restricciones, el recurso o la solicitud de acceso. entre otros. Esta información se puede clasificar en diferentes categorías:

- La operación de acceso solicitada.
- Temporización.
- Parámetros de la petición.
- Estado de sincronización del recurso.
- Estado local del recurso.
- Historial de uso.
- Prioridad del proceso.

Las primitivas de sincronización que se utilizan para tratar el acceso sobre los recursos son:

- Monitores o métodos sincronizados.
- Servidores por el paso de mensajes.
- Recursos protegidos, que serán objetos protegidos.

Un aspecto de la implementación a tener en cuenta es el uso de la **sincronización de condición** frente a la sincronización de evitación. En la primera se aceptan todas las peticiones que puedan llegar, pero los procesos que no puedan ser atendidos se suspenden en una cola de espera asociados a una variable de condición hasta que el recurso esté disponible. En la **sincronización de evitación**, las peticiones sólo se aceptan en el caso de que puedan ser atendidas y las condiciones que determinan cuándo un recurso puede ser accedido con seguridad se expresan como una condición de guardia en la aceptación de la acción.

Además, hay que tener en cuenta la atomicidad en la interacción con el gestor de recursos, pues muchos problemas de sincronización de recursos en sistemas multitarea y multiprocesador recaen en el hecho de que las operaciones no se pueden realizar de forma atómica.

3. Propiedades de los mecanismos de sincronización

Es deseable disponer de un mecanismo de sincronización de alto nivel que resuelva el control de recursos, aunque no es fácil expresar claramente sus requisitos. Para realizar un análisis sistemático y objetivo, los requisitos que debe cumplir un mecanismo de sincronización se definen en base a tres categorías:

- **Modularidad:** una implementación es modular si está estructurada de tal manera que es sencillo modificar una parte sin afectar al resto del sistema. Esta propiedad es importante para que el sistema sea más fácil de entender y de mantener.
- **Expresividad:** un mecanismo es expresivo si permite una definición sencilla de los requisitos de sincronización. En contraparte, un mecanismo es poco expresivo cuando es sea posible usar la información de la que se dispone sobre el problema de sincronización.
- **Facilidad de uso:** un sistema debe permitir la descomposición en restricciones de sincronización individuales, que puedan ser implementadas de forma aislada. Si no, a medida que aumenten las restricciones aumentaría la complejidad de la implementación.

```
/* Clase que encapsula un tipo de recurso */
public class Recurso {
    ...
}

/* Interfaz genérica para un controlador */
public interface Control {
    public Recurso reservar();
    public void liberar();
}
/* Implementación de un controlador de recursos */
public class Controlador implements Control {
    public synchronized void Recurso reservar() {

    }
    Public synchronized void Recurso liberar() {
    }
}
```

La implementación en Java del control de recursos se basa en el uso de monitores y métodos sincronizados, y cumple con los requisitos de modularidad. En función a la expresividad, un mecanismo de sincronización deberá permitir expresar adecuadamente las restricciones de sincronización del sistema.

Categorías de información relacionadas a los mecanismos de sincronización

- **Tipo de operación**: algunos recursos admiten distintos tipos de acceso (acceso compartido o acceso exclusivo). Esta información puede utilizarse para dar prioridad a un tipo de acceso frente a otro.
- **Temporización**: el orden de llegada de las solicitudes de acceso se utiliza habitualmente para gestionar el reparto de los recursos. Considerando únicamente este factor, un proceso que solicite un recurso antes que otro, conseguirá el acceso en primer lugar.
- **Parámetros**; puede ser conveniente utilizar la información contenida en los parámetros asociados a la petición. Con frecuencia, esta información se refiere al tamaño o número de elementos solicitados (si son recursos cuantificables), o bien a la identidad del recurso. De esta forma, se puede hacer un uso más fino del recurso.
- **Estado del recurso**: hay que tener en cuenta la información del estado del recurso además de la información relacionada con el solicitante del propio recurso (orden de llegada, tipo de petición, etc.). Esta información puede ser de distintos tipos:
    - **Estado de la sincronización**: incluye toda la información del estado del recurso necesaria únicamente para propósitos de sincronización (no sería necesaria si el recurso no se accede concurrentemente). Incluye información sobre los procesos que tienen acceso al recurso y las operaciones que están ejecutando. Un ejemplo es el número de procesos que acceden a un recurso.
    - **Estado local**: contiene información sobre el recurso que es necesaria, independientemente de si el recurso se accede de forma concurrente o secuencial. Por ejemplo, un buffer de memoria.
    - **Historial de uso o acceso**: contiene información sobre los eventos ocurridos previamente en relación con un recurso, por lo que afectan explícitamente a la sincronización del recurso (son pasado). Por ejemplo, qué operaciones se han ejecutado.
- **Prioridad del proceso**: puede utilizarse para determinar el orden de acceso a los recursos. Un proceso de mayor prioridad puede ser atendido antes que otro de menor prioridad, a pesar de que este último haya realizado su solicitud con anterioridad al primero.

4. Evaluación de los mecanismos de sincronización

En base a los diferentes criterios anteriores, se pueden evaluar algunos mecanismos de sincronización típicos. En el caso de los **monitores**:

- Tipo de operación: permiten expresar diferentes tipos de solicitud implementando diferentes métodos. Sin embargo, éstos se planifican normalmente por orden FIFO, por lo que no es posible hacerlo por tipo de operación.
- Restricciones temporales: su implementación FIFO garantiza el orden en el que se atienden las peticiones, pero no es posible controlar el orden por el tipo de operación
- Estado del recurso: representan adecuadamente restricciones con variables de condición.
- Parámetros: pueden usarse de forma natural.
- Prioridad: es difícil de implementar debido a colas FIFO, pero pueden definirse distintos monitores basados en prioridades.

Para los servidores y objetos protegidos:

- Permiten una mejor gestión del tipo de operación.
- Las Restricciones de estado se pueden expresar con condiciones de guarda.
- Parámetros: sólo pueden ser tenidos en cuenta después de atender la llamada, por lo que debe construirse una interacción en dos pasos.
- Puesto que la espera de guardas se hace forma arbitraria o mediante FIFO, no es sencillo hacer gestión de prioridades.

5. Funcionalidad de reencolado

Supone una mejora en la usabilidad de los mecanismos de evitación que proporcionan algunos lenguajes de programación concurrente. Consiste en mover explícitamente una tarea que se encuentra en una barrera o guarda a otra cola de espera, donde deberá conseguir de nuevo el acceso. Se trata de una mejora en la usabilidad en los mecanismos de evitación que proporcionan algunos lenguajes de programación concurrente.

Un reencolado puede hacerse a la misma barrera, a otra barrera en la misma unidad, o en otra unidad diferente. En cuanto al flujo de control, a diferencia de lo que ocurre en una llamada a otro procedimiento, el invocador no retoma el control.

Cuando un punto de entrada se reencola en otro, el flujo de control del primero termina. Cuando el segundo finaliza, el control se devuelve al objeto que realizó la llamada inicial. Como consecuencia, cuando se ejecuta un reencolado de un objeto protegido a otro, la exclusión mutua se libera.

6. Uso de los recursos

El modo normal de operación cuando las tareas competitivas o cooperativas necesitan de unos determinados recursos pasa por solicitar el recurso, esperar si es necesario, usar el recurso y por último liberarlo. Un recurso puede ser requerido en modo:

- **Recurso compartido**: el recurso puede usarse por varias tareas al mismo tiempo. Por ejemplo, un fichero de lectura.
- **Recurso exclusivo**: sólo una tarea puede usarlo al mismo tiempo en un instante determinado. Por ejemplo, un recurso físico como una impresora.

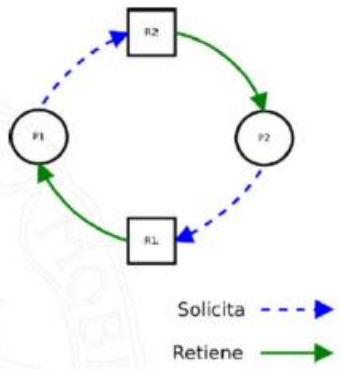
Algunos recursos pueden ser usados de las dos maneras. Así, si un proceso solicita acceso a un recurso en modo compartido mientras está siendo usado en modo exclusivo, el proceso debe esperar y si se está usando en modo compartido, puede continuar. Si la tarea solicita acceso en modo exclusivo, debe esperar a que los procesos que están accediendo en modo compartido terminen.

Por tanto, sólo deben solicitarse recursos cuando se necesiten para evitar el riesgo de quedar bloqueadas antes de tiempo. De la misma manera, los recursos no deben liberarse hasta que se complete la acción, pues el rendimiento del sistema se vería afectado porque las tareas estarían esperando continuamente. También puede ocurrir que, si las tareas liberan los recursos demasiado pronto y luego fallan, pueden haber pasado información errónea al recurso por lo que el estado de los mismos debe modificarse de nuevo.

Cualquier procedimiento de recuperación debe liberar los recursos si la acción es completada con éxito o deshacer los efectos en el recurso para que el sistema pueda ser restaurado a un estado seguro. Con la recuperación de errores hacia delante, estas operaciones pueden ser realizadas por los manejadores de excepción. Con la recuperación hacia atrás, es necesario soporte adicional por parte del lenguaje de programación.

7. Interbloqueo

Uno de los problemas más importantes en el control de recursos es el de las **condiciones de carrera**. Cuando dos o más tareas acceden, o intentan acceder, a más de un recurso compartido, se da la posibilidad de que se genere la situación en que los procesos involucrados queden incapacitados para continuar su ejecución normal, bloqueándose mutuamente. Esta situación es la que se conoce como **interbloqueo**.




```
/* P1 */
// Código previo a la solicitud del recurso
espera(&s1); /* espera en el recurso R1 */
...
/* Empieza el uso del recurso R1 */
espera(&s2); // En este punto, el proceso ha adquirido el recurso R1 y
espera a obtener R2.
// Aquí existe la posibilidad de llegar a un punto muerto
...
/ * Usa el recurso R2 * /
señala(&s2); // Libera los recursos R1 y R2
señala(&s1);
...
```

```
/* P2 */
...
espera(&s2); // Espera en el recurso R2
...
/ * Usa el recurso R2 */
espera(&s1); // Espera en el recurso R1
// Aquí se llegaría a un punto muerto.
...
/* Usa el recurso R1 */
señal(&s1); // Libera R1
señal(&s2);
```

Dependiendo de las características del sistema, pueden aparecer distintas condiciones de bloqueo:

- **Punto muerto (deadlock):** los procesos quedan suspendidos a la espera de que se libere el recurso que se necesitan.
- **Bloqueo Activo (livelock):** los procesos continúan su ejecución, pero no pueden evolucionar (bucle de espera activa).
- **Inanición (starvation):** algunas tareas no obtienen recursos ya que siempre hay otra de mayor prioridad que los están usando.

Para que se produzca un interbloqueo, deben darse, de forma simultánea, las siguientes cuatro condiciones:

- **Exclusión mutua:** garantiza que dos procesos no tengan acceso a un recurso al mismo tiempo.
- **Retención y espera:** los procesos pueden retener recursos mientras esperan a otros.
- **No apropiación:** un proceso no puede ser expropiado de un recurso por parte del sistema operativo, sino que tiene que cederlo voluntariamente.
- **Esperar circular:** existe una lista circular de procesos tal que, cada uno de ellos, retiene a un recurso solicitado por el siguiente.

Aunque no se den estas cuatro condiciones simultáneamente, puede haber situaciones en las que un proceso no consiga acceso a un recurso si la política de reserva no es justa y existen procesos que intentan acceder continuamente al mismo recurso. De esta forma, las peticiones de acceso a un recurso por parte de un proceso no serán atendidas, retrasándose su

ejecución indefinidamente en favor de otros procesos, dando lugar a la inanición. La inanición, a diferencia del interbloqueo, no impide la evolución del sistema, pero provoca un reparto injusto de los recursos, ya que algunos procesos podrán hacer uso de ellos y otros no. En particular, para un sistema basado en el uso prioridades, puede ocurrir que un proceso de alta prioridad queda a la espera de un resultado de otro proceso con menor prioridad que puede llegar a no ejecutarse nunca, dando lugar a la inversión de prioridades.

Estrategias para tratar los Interbloqueos

- **Prevención**: considera la eliminación de alguna de las cuatro condiciones que provocan un interbloqueo.
- **Evitación**: se concede acceso a los recursos únicamente si el sistema se mantiene en un estado seguro (un estado inseguro es aquel que puede llevar a un interbloqueo).
- **Detección y recuperación**: consiste en actuar después de que aparezca un estado de interbloqueo. Para ello, es necesario detectar la ocurrencia del problema e identificar los procesos y recursos implicados, y, a continuación, tomar las medidas necesarias para restablecer un estado consistente. El algoritmo de detección, que se ejecuta periódicamente, se basa generalmente en técnicas centradas en la búsqueda de esperas circulares. Una vez se ha detectado el interbloqueo, para eliminarlo se lleva a cabo:
    - Terminar los procesos implicados.
    - Retrocede a un estado anterior.
    - Expropiar los recursos para llegar a un estado consistente.

En cualquier caso, se debe evitar la inanición de los procesos.