

**Universidad Nacional Autónoma de México**

**Facultad de Ingeniería**

**Sistemas Operativos**





Primitivas de Sincronización en JAVA

Profesor: Ing. Wolf

Trabajo Elaborado por:

Ortega González Luis Abraham

Páez Lampón Bidkar Herson

**Candados o Mutexes**

¿Qué son?

Mecanismo que asegura que cierta región del código será ejecutada como si fuera atómica; mecanismo de prevención, que mantiene en espera a cualquier hilo o proceso que quiera entrar a la sección crítica protegida por el mutex, reteniéndolo antes de entrar a esta hasta que el proceso que la está ejecutando salga de ella.

¿Cómo implementar?

Debemos marcar las variables como privadas y sincronizar el código que modifica a dichas variables.

Ejemplo:

Supongamos un método en el que deseamos aplicar un mutex, la sintaxis sería la siguiente

privated synchronized void nameMethod(int var)

Cada objeto en JAVA posee un seguro que previene su acceso, dicho seguro se activa únicamente cuando el objeto se encuentra dentro de un método sincronizado.

**Semáforos**

¿Qué son?

Variable tipo entero que tiene definida la siguiente interfaz:

Inicialización. Se puede inicializar el semáforo a cualquier valor entero, pero después de esto, su valor no puede ya ser leído. Un semáforo es una estructura abstracta, y su valor es tomado como opaco (invisible) al programador

Semaphore semaphore = new Semaphore(n);

Decrementar. Cuando un hilo decrementa al semáforo, si el valor es negativo, el hilo se bloquea y no puede continuar hasta que otro hilo incremente el semáforo. Según la implementación, esta operación puede denominarse wait, down, acquiere o incluso p

Semaphore.acquire();

Incrementar. Cuando un hilo incrementa el semáforo, si hay hilos esperando, uno de ellos es despertado. Los nombres que recibe esta operación son signal, up, reléase, post o V

Semaphore.release();

¿Cómo implementar?

* Se necesita de la clase java.util.concurrent.Semaphore
* El constructor de la clase Semaphore recibe un argumento entero
* El semáforo tiene dos métodos principales
  + acquire()
  + release()
* La clase Semaphore también soporta Fairness y métodos de no bloqueo como tryacquire() y tryacquire(int permits)

Variables de Condición:

Extensión sobre el comportamiento de los mutexes, buscan darles la “inteligencia” de responder ante determinados eventos. Siempre operan en conjunto con un mutex.

¿Cómo implementar?

Utilizando los métodos heredados de la clase java.lang.Object y un mutex

Dichos métodos son

wait() : Posiciona al hilo en la lista de espera del objeto

notify(): Libera el seguro para un hilo

notifyAll(): Libera el seguro para todos los hilos que estén en espera

Monitores:

Módulo opacto que encapsula servicios mediante métodos de acceso, así como sus variables locales y globales. Busca evitar errores al distinguir entre el uso que debe tener el semáforo. Sólo permite un hilo y un método de servicio a la vez. Posee las operaciones wait() and signal() cuyo funcionamiento es semejante al de un semáforo con su acquire() y release()

¿Cómo implementar?

La manera más fácil de abordar esta pregunta es asumiendo que los monitores son una combinación de mutexes y semáforos, aunque realmente no es esto, sólo se asemeja a ambos.

Ejemplo:

Public synchronized void nameMethod(type\_var name\_var)

{

If(condición1) wait(var\_condition);

If(condición2)notify(var\_condition);

}

Mesografía (Consultada 18/09/2016)

<http://sistop.org/pdf/sistemas_operativos.pdf>

<http://tutorials.jenkov.com/java-util-concurrent/semaphore.html>

<https://examples.javacodegeeks.com/core-java/util/concurrent/semaphore/java-util-concurrent-semaphore-example/>

<https://docs.oracle.com/javase/7/docs/api/java/util/concurrent/Semaphore.html>

<http://www.cs.buap.mx/~mtovar/doc/Semaforos/monitores.pdf>