

Instituto Politécnico Nacional

Escuela Superior de Computo

3CV16

Aplicaciones para comunicaciones en red

Badillo de la Rosa Jorge Antonio.

Gutierrez Rodriguez Omar

Reyes Pérez Edward Daniel

Tarea 2

Índice

* Introducción…………………………………………. 1
* Sección critica…………………………………………. 1
* Operaciones atómicas…………………………………….2
* Death lock…………………………………………………3
* Tipos de espera……………………………………………4
* Semáforos…………………………5
* Referencias……………………………………….7

**Introducción**

Se conoce por programación concurrente a la rama de la informática que trata de las

técnicas de programación que se usan para expresar el paralelismo entre tareas y para

resolver los problemas de comunicación y sincronización entre procesos.

El principal problema de la programación concurrente corresponde a no saber en que

orden se ejecutan los programas (en especial los programas que se comunican). Se debe

tener especial cuidado en que este orden no afecte el resultado de los programas.

La palabra reservada synchronized se usa para indicar que ciertas partes del código, (habitualmente, una función miembro) están sincronizadas, es decir, que solamente un subproceso puede acceder a dicho método a la vez. Cada método sincronizado posee una especie de llave que puede cerrar o abrir la puerta de acceso.

**¿Qué es una sección critica? y ¿Qué es una exclusión mutua?**

El método más sencillo de comunicación entre los procesos de un programa concurrente

es el uso común de unas variables de datos. Esta forma tan sencilla de comunicación

puede llevar, no obstante, a errores en el programa ya que el acceso concurrente puede

hacer que la acción de un proceso interfiera en las acciones de otro de una forma no

adecuada.

Para evitar este tipo de errores se pueden identificar aquellas regiones de los procesos

que acceden a variables compartidas y dotarlas de la posibilidad de ejecución como si

fueran una única instrucción. Se denomina Sección Crítica a aquellas partes de los

procesos concurrentes que no pueden ejecutarse de forma concurrente o, también, que

desde otro proceso se ven como si fueran una única instrucción. Esto quiere decir que si

un proceso entra a ejecutar una sección crítica en la que se accede a unas variables

compartidas, entonces otro proceso no puede entrar a ejecutar una región crítica en la

que acceda a variables compartidas con el anterior.

Las secciones críticas se pueden mutuo excluir. Para conseguir dicha exclusión se

deben implementar protocolos software que impidan o el acceso a una sección crítica

mientras está siendo utilizada por un proceso.

**¿Qué sin las operaciones atómicas?**

Una operación atómica es una operación en la que un procesador puede simultáneamente leer una ubicación y escribirla en la misma operación del bus. Esto previene que cualquier otro procesador o dispositivo de E/S escriba o lea la memoria hasta que la operación se haya completado.

El término atómico implica la indivisibilidad e irreductibilidad del proceso, ya que este debe realizarse en su totalidad o en caso de ser interrumpido poder deshacer sus acciones de modo que fuese como si no se hubiese realizado acción alguna.

**¿Qué es un death lock?**

A cada proceso se le asignan varios recursos para su ejecución. En sistemas de multiprogramación, uno de los principales objetivos del sistema operativo es el compartimiento de los recursos. Cuando se comparten los recursos entre una población de usuarios, cada uno de los cuales mantiene un control exclusivo sobre ciertos recursos asignados a él, es posible que otros usuarios no terminen sus procesos (bloqueo entre usuarios).

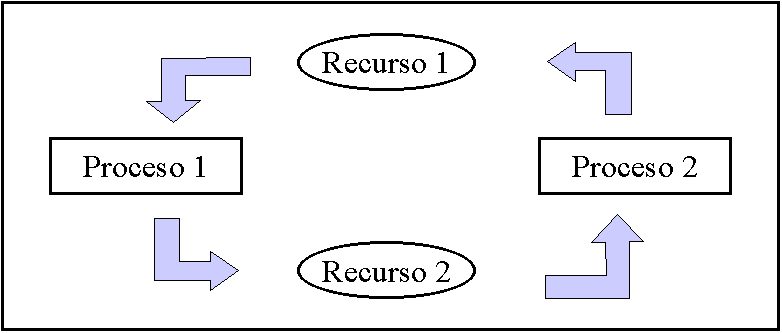
El bloqueo mutuo puede aparecer de muchas formas:

- Si a un proceso se le asigna la tarea de esperar a que ocurra un evento y el sistema no incluye providencias para señalar la ocurrencia de dicho evento, habrá un bloqueo mutuo con un solo proceso. Es muy difícil detectar bloqueos mutuos de esta naturaleza.

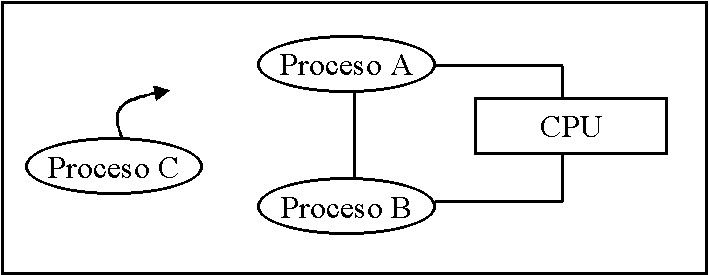
- La mayor parte de los bloqueos mutuos en sistemas reales implican una competencia entre varios procesos por varios recursos.

Bloqueo mutuo = Deadlock = Abrazo mortal

El bloqueo mutuo o abrazo mortal (deadlock) es el problema más serio que se puede presentar en un ambiente de multiprogramación. Es la actividad en que dos o más procesos esperan un recurso (CPU, memoria, entrada/salida) que nunca les va a ser otorgado.



Como se ve, aquí el sistema está bloqueado: Proceso 1 tiene a Recurso 1 y necesita a Recurso 2 para continuar. Y Proceso 2 tiene asignado a Recurso 2, y necesita a Recurso 1 para continuar. Cada proceso espera que el otro libere un recurso que no liberará hasta que el otro libere su recurso, lo cual no sucederá si el primero no libera su recurso. Aquí se presenta una espera circular.



Otro ejemplo, dos procesos adueñados de CPU. Proceso A se ejecuta y llama a Proceso B, Proceso B se ejecuta y llama a Proceso A,..., si llega otro proceso, no lo van a dejar entrar a CPU a ejecutarse.

**¿Tipos de espera?**

**Activa**

“La espera activa es una técnica en la cual un proceso comprueba continuamente si una condición se cumple o si se produce un evento”

**Texto

Descripción generada automáticamente**

**Pasiva**

“La espera activa es una técnica en la cual un proceso comprueba continuamente si una condición se cumple o si se produce un evento”

**¿Cuáles mecanismos de sincronización existen?**

**Optimistas**

Este mecanismo debe emplearse si el programador considera que la frecuencia de acceso a un recurso compartido es baja, es decir, que supone que la probabilidad de coincidencia de dos o más procesos al recurso compartido es baja.

Los inconvenientes de este tipo de mecanismo son dos:

* Se consume más memoria, pues hay que realizar una copia del recurso compartido para efectuar la actualización.
* En situaciones de coincidencia, se tiene que volver a realizar la operación, por tanto, se desperdician recursos de procesamiento.

Un mecanismo de sincronización optimista funciona de la siguiente manera:

* Paso 1. Realiza una copia del estado en el que está el recurso compartido.
* Paso 2. Opera con esa copia.
* Paso 3. Compruebo si mi copia alterada coincide con el original aplicándole también el tratamiento.

**Pesimistas**

Este mecanismo debe emplearse si se considera que la frecuencia de acceso al recurso compartido es alta.

En este mecanismo disponemos de tres partes:

* El protocolo de entrada, en el que se emplea un mecanismo que no permite continuar con la ejecución si otro u otros procesos están accediendo al recurso compartido.
* La sección crítica, en el que se realizan las operaciones pertinentes con el recurso compartido.
* El protocolo de salida, en el que se vuelve a permitir el acceso al recurso compartido.

Los protocolos de entrada y salida son, generalmente, operaciones costosas en términos de recursos de procesamiento, pues requieren el uso de instrucciones atómicas cuyo tiempo de ejecución es alto.

Por último, se podría emplear un control de concurrencia pesimista para resolver un problema que se resuelve con un control de concurrencia optimista, pero no al revés.

**Funciones para semáforos**

A veces es necesario que dos o más procesos o hilos (threads) accedan a un recurso común (escribir en un mismo fichero, leer la misma zona de memoria, escribir en la misma pantalla, etc). El problema es que si lo hacen simultáneamente y de forma incontrolada, pueden "machacar" el uno la operación del otro (y dejar el fichero o la memoria con un contenido inservible o la pantalla ilegible).

Para evitar este problema, están los semáforos. Un semáforo da acceso al recurso a uno de los procesos y se lo niega a los demás mientras el primero no termine. Los semáforos, junto con la memoria compartida y las colas de mensajes, son los recursos compartidos que suministra UNIX para comunicación entre procesos.

El funcionamiento del semáforo es como el de una variable contador. Imaginemos que el semáforo controla un fichero y que inicialmente tiene el valor 1 (está "verde"). Cuando un proceso quiere acceder al fichero, primero debe decrementar el semáforo. El contador queda a 0 y como no es negativo, deja que el proceso siga su ejecución y, por tanto, acceda al fichero.

Ahora un segundo proceso lo intenta y para ello también decrementa el contador. Esta vez el contador se pone a -1 y como es negativo, el semáforo se encarga de que el proceso quede "bloqueado" y "dormido" en una cola de espera. Este segundo proceso no continuará por tanto su ejecución y no accederá al fichero.

Supongamos ahora que el primer proceso termina de escribir el fichero. Al acabar con el fichero debe incrementar el contador del semáforo. Al hacerlo, este contador se pone a 0. Como no es negativo, el semáforo se encarga de mirar el la cola de procesos pendientes y "desbloquear" al primer proceso de dicha cola. Con ello, el segundo proceso que quería acceder al fichero continua su ejecución y accede al fichero.

Cuando este proceso también termine con el fichero, incrementa el contador y el semáforo vuelve a ponerse a 1, a estar "verde".

Es posible hacer que el valor inicial del semáforo sea, por ejemplo, 3, con lo que pasarán los tres primeros procesos que lo intenten. Pueden a su vez quedar muchos procesos encolados simultáneamente, con lo que el contador quedará con un valor negativo grande. Cada vez que un proceso incremente el contador (libere el recurso común), el primer proceso encolado despertará. Los demás seguirán dormidos.

Como vemos, el proceso de los semáforos requiere colaboración de los procesos. Un proceso debe decrementar el contador antes de acceder al fichero e incrementarlo cuando termine. Si los procesos no siguen este "protocolo" (y pueden no hacerlo), el semáforo no sirve de nada.

Documentación de POSIX en : https://www.delftstack.com/es/howto/c/semaphore-example-in-c/

**Referencia**

Deadlock; <http://cidecame.uaeh.edu.mx/lcc/mapa/PROYECTO/libro26/interbloqueo_deadlock.html>

Programación concurrente:

<https://www.fing.edu.uy/tecnoinf/mvd/cursos/so/material/teo/so07-concurrencia.pdf>

Tipos de esperas:

<https://babel.upm.es/teaching/concurrencia/material/slides/groman/CC_EsperaActiva.pdf>

Semaforos:

https://www.delftstack.com/es/howto/c/semaphore-example-in-c/