

Sintesis de Deadlock

El grupete

19 de octubre de 2008

1. Deadlock

1.1. Introducción

En un entorno de multiprogramación varios procesos pueden competir por un numero finito de recursos. Un procesos solicita recursos y si estos no estan disponibles, el proceso queda en estado de espera. Puede pasar que los procesos en espera nunca cambien de estado porque los recursos que necesitan los tienen otros procesos que tambien están en espera, a esta situación se la llama Dreadlock.

1.2. Modelo del sistema

Los recursos se dividen en varios tipos, cada uno de los cuales consiste en ejemplares idénticos. En el modo de operacion normal, un proceso solo puede utilizar un recurso en la secuencia siguiente:

- Solicitud: Si la solicitud no se puede atender de inmediato entonces el proceso solicitante debe esperar a que pueda adquirir el recurso.
- Utilizacion
- Liberacion

La solicitud y liberacion de recursos son llamadas a sistema, estas pueden lograrse a travez de las operacion wait y signal sobre semaforos. Una tabla del sistema registra si cada uno de los recursos esta libre o asignado, y si esta asignado a que proceso. Si un proceso solicita un recurso que ya esta asigando entoces lo puede encolar en la cola de espera de ese proceso. Un conjunto de procesos se encuentra en Deadlock si cada uno espera un suceso que solo puede originar otro proceso del mismo conjunto.

1.3. Caracterización de los bloqueos mutuos

1.3.1. Condiciones necesarias

Una situacion de deadlock puede surgir si y solo si en un sistema se presentan las siguientes cuatro condiciones:

- Exclusion mutua: Por lo menos un recurso debe reternerse en modo no compartido; es decir, solo un proceso puede usar el recurso a la vez. Si otro proceso solicita el recurso debera esperar a que sea liberado.

- Retencion y espera: debe haber por lo menos un proceso que retenga un recurso y espere adquirir otros recursos retenidos por otros procesos.
- No apropiacion: Los recursos no se pueden quitar. Es decir, un recurso solo puede ser liberado por el proceso que lo tiene despues de haber cumplido su tarea.
- Espera circular: Debe haber un conjunto P_0, \dots, P_n de procesos en espera, tales que P_0 espera un recurso retenido por P_1 , P_1 espera un recurso retenido por P_2, \dots , P_{n-1} espera un recurso retenido por P_n y P_n espera un recurso retenido por P_0 .

La condicion de espera circular implica la de retención y espera, sin embargo es util verlas por separado.

1.3.2. Grafo de asignación de recursos

Es un grafo dirigido donde los nodos son procesos o recursos. Una arista de un recurso a un proceso indica que ese recurso esta siendo utilizado por ese proceso (arista de asignación). Una arista de un proceso a un recurso indica que ese proceso solicito ese recurso (arista de solicitud). A los procesos los dibujamos con redondeles y a los recursos con cajitas. Como pueden haber muchas instancias de un mismo tipo de recurso en las cajitas dibujamos puntitos que representan cuantas instancias de ese recurso hay. Ejemplo: si tenemos dos impresoras dibujaremos una cajita con dos puntos dentro, en este caso podran salir dos flechas de este nodo (una por impresora). IMPORTANTE: Si el grafo no contiene ciclos entonces ningun proceso esta en deadlock, por otra parte, si hay ciclo entonces PUEDE haber un deadlock. Si cada tipo de recurso tiene un solo ejemplar, entonces un ciclo representa que ha ocurrido un bloqueo mutuo y todos los procesos en el ciclo estan en deadlock. Si el ciclo comprende solo un conjunto de tipos de recursos, cada uno con un solo ejemplar, entonces ha ocurrido un bloqueo mutuo entre todos los procesos en el ciclo.

1.3.3. Metodos para manejar Deadlocks

Existen dos metodos principales para tratar el tema del deadlock. Podemos usar un protocolo para que el sistema nunca entre en deadlock o podemos permitir que entre y se recupere. Primero consideraremos metodos para que el sistema nunca entre en deadlock. Para ello contamos con dos metodos comunes: la prevencion y la evitacion.

1.4. Prevencion de bloqueos mutuos

Para que haya deadlock deben presentarse cada una de las cuatro condiciones necesarias. Veremos distintas maneras de evitar cada una de estas condiciones, previniendo el deadlock.

1.4.1. Exclusion Mutua

Los recursos compartibles no pueden participar en un deadlock. Por lo general no es posible evitar el deadlock negando la condicion de exclusion mutua. Por su propia condicion algunos recursos no son compartibles.

1.4.2. Retencion y espera

Para asegurar que la condicion de retencion y espera nunca se de, debemos garantizar que cuando un proceso solicita un recurso, no retenga otros.

- Un protocolo que puede usarse es que cada proceso solicite todos los recursos antes de empezar su ejecución.
- Otro similar es que un proceso solo pueda pedir recursos si no tiene ninguno asignado. Si necesita mas recursos que los que tiene en un momento dado, debe liberar todo para volver a pedir lo que necesita.

La desventaja de estas politicas es que puede pasar que se pidan recursos que nunca seran usados por el proceso que los solicito. Ademas es posible un bloqueo indefinido, si un proceso pide varios recursos populares, es posible que nunca le sean adjudicados porque estan en uso por otros procesos (es una especie de inanicion, pero el libro no lo nombra asi).

1.4.3. No apropiacion

La tercer condicion es que no haya apropiación de recursos: para evitar esto podemos permitir la apropiación ;P

- Si un proceso que tiene un recurso solicita otro que no se le puede dar de inmediato, entonces se le sacan todos los recursos que tiene y volvera a ejecutar cuando se le puedan asignar todos los recursos que pidio mas los que tenia.
- Si un proceso solicita recursos que no estan disponibles comprobamos si estan asignados otro proceso que espera mas recursos, en cuyo caso expropiamos los recursos deseados del proceso en espera y los asignamos al proceso solicitante. Si los recursos no estan disponibles ni asignados a un proceso en espera, el proceso debera esperar.

Este protocolo se usa cuando se puede guardar con facilidad el estado de los recursos, como CPU y memoria principal. Es un garron para impresora o cintas.

1.4.4. Espera circular

Una forma de asegurar que no se presente la condicion de espera circular es imponer una ordenacion total de todos los tipos de recursos y requerir que cada proceso solicite los recursos en orden ascendente.

1.5. Evitación de bloqueos mutuos

Otro metodo para evitar los bloqueos mutuos consiste en requerir informacion adicional sobre como se solicitaran los recursos. Cada solicitud requiere que el sistema considere los recursos disponibles en ese momento, los actualmente asignados a cada proceso, y las futuras solicitudes y liberaciones de cada proceso, para decidir si puede satisfacer la solicitud presente o debe esperar para evitar un posible deadlock futuro. Los diversos algoritmos difieren en la cantidad y tipo de informacion que requieren. El modelo mas sencillo y util requiere que el proceso declare la cantidad maxima de recursos que usará para cada tipo

de recurso. Un algoritmo de evitacion de deadlock examina dinamicamente el estado de asignacion de recursos para asegurar que no pueda presentar una condicion de espera circular. El estado de asignacion de recursos viene definido por el numero de recursos disponibles y asignados, y por la demanda maxima de los procesos. Un estado es seguro si el sistema puede asignar recursos a cada proceso (hasta el maximo) siguiendo algun orden y aun asi avitar el deadlock. Mas formalmente un sistema se encuentra en estado seguro sólo si existe una secuencia segura. Una secuencia segura de procesos P_1, \dots, P_n es segura para el estado actual de asignaciones si, para cada P_i , los recursos que aun puede solicitar P_i pueden satisfacerse con los recursos actualmente disponibles mas los retenidos por todos los P_j , donde $j < i$. En esta situacion si los recursos que necesita P_i no estan inmediatamente disponibles, entonces P_i puede esperar a que terminen todos los procesos P_j . Una vez terminados, P_i puede obtener todos los recursos necesarios, completar la tarea, devolver los recursos que se le han asignado y terminar. Cuando P_i termina, P_{i+1} puede obtener todos los recursos necesarios, etc. Si no existe esta secuencia, se dice que el estado es *inseguro*. Un estado seguro NO ES un estado de bloqueo mutuo, y un estado de bloqueo mutuo ES un estado inseguro, pero no todos los estados inseguros son de deadlock. Un estado inseguro puede llevar a un estado de deadlock. Ya establecido el concepto de estado seguro podemos ahora definir algoritmos de evitacion que aseguren que el sistema nunca caera en un estado de deadlock. La idea es asegurar que el sistema nunca caera en un estado inseguro. Al principio el sistema esta en un estado seguro, cuando un proceso solicita un recurso que en ese momento esta disponible, el sistema debe decidir si en ese momento le puede asignar ese recurso inmediatamente o si el proceso tiene que esperar. La solicitud se atiende solo si deja al sistema en un estado seguro. Observe que en este esquema si un proceso solicita un recurso que esta disponible puede llegar a tener que esperar, por esto la utilizacion de recursos puede ser menor que sin la utilizacion de este algoritmo.

1.5.1. Varios ejemplares de un mismo tipo de recurso

Describiremos el algoritmo del banquero. Cuando un proceso entra en el sistema debe declarar la cantidad maxima de recursos que puede llegar a necesitar, esta cantidad no puede exceder la cantidad que hay en el sistema. Cuando un proceso solicita un conjunto de recursos, el sistema debe determinar si esta asignacion deja al sistema en un estado seguro. Si es asi, los recursos se asignan, sino el proceso debe esperar hasta que otro libere los recursos que el necesita. Para mayor detalle mirar el libro, no se puede resumir y no tiene sentido que lo copie.

1.5.2. Un ejemplar unico de cada tipo de recurso

Este algoritmo utiliza una variante del grafo de asignacion de recursos, ademas de las aristas de solicitud y asignacion, aadimos un nuevo tipo de arista, llamada *arista de reserva*. Una arista de reserva $P_i \rightarrow R_j$ indica que en el futuro el proceso P_i puede pedir el recurso R_j . Esta linea semaja a una linea de reserva en cuanto a direccion, pero la linea es punteada. Cuando el proceso P_i solicita el recurso R_j , la arista de reserva $P_i \rightarrow R_j$ se convierte en una arista de solicitud. De manera parecida cuando P_i libera un recurso R_j , la arista de

asignacion $R_j \rightarrow P_i$ vuelve a ser una arista de reserva $P_i \rightarrow R_j$. Observemos que en el sistema todos los recursos deben reservarse a priori; es decir antes de que el proceso P_i comience su ejecución, todas sus aristas de reserva ya deben aparecer en el grafo de asignacion de recursos. Podemos hacer mas flexible permitiendo que una arista de reserva $P_i \rightarrow R_j$ se agregue al grafo si todas las aristas relacionada con P_i son de reserva. Suponga que el proceso P_i solicita el recurso R_j . Esta solicitud puede atenderse solo si al convertir la arista de solicitud $P_i \rightarrow R_j$ en una de asignacion $R_j \rightarrow P_i$ no se forma un ciclo en el grafo de asignacion de recursos. Si no hay ningun ciclo, entonces la asignacion del recurso dejara al sistema en un estado seguro. Si se detecta un ciclo, entonces la asignacion colocara al sistema en un estado inseguro, por lo que el proceso P_i tendra que esperar a que se satisfagan sus solicitudes.

1.6. Deteccion de Deadlocks

Si un sistema no emplea un algoritmo de prevencion o evitacion de deadlocks, entonces puede ocurrir una situacion de deadlock. En este entorno el sistema debe ofrecer:

- Un algoritmo que examine el estado del sistema para determinar si ha ocurrido un deadlock.
- Un algoritmo para recuperarse del deadlock.

1.6.1. Varios ejemplares de un tipo de recurso

El algoritmo de detección emplea estructuras de datos que varian en el tiempo, similares a las utilizadas en el banquero.

- Disponible: Un vector de longitud m que indica el numero de recursos disponibles de cada tipo.
- Asignacion: Una matriz de $m \times n$ que define el numero de recursos de cada tipo actualmente asignados a cada proceso.
- Solicitud: Una matriz de $n \times m$ que indica la solicitud actual de cada proceso.

El algoritmo de deteccion descrito aqui se limita a investigar cada una de las posibles secuencias de asignacion para los procesos que quedan por terminar. (ES PARECIDO AL DEL BANQUERO, OTRA VEZ MIREN ESTE ALGORITMO EN EL LIBRO PORQUE NO TIENE SENTIDO QUE LO COPIE Y ADEMAS LOS ALGORITMOS, SI SON BUENOS, SON NO COMPRESIBLES ;P).

1.6.2. Un unico ejemplar de cada tipo de recurso

Una vez mas usaremos una variante del grafo de asignacion, llamada *grafo de espera*. Podemos obtener este grafo sacando los nodos correspondientes a recursos del grafo de asignacion de recursos y uniendo las aristas respectivas. Como antes, hay un deadlock si y solo si, hay un ciclo. Para detectar deadlock el sistema tiene que mantener actualizado este grafo y cada una cierta cantidad de tiempo correr un algoritmo de deteccion de ciclos ($O(n^2)$, n procesos).

1.6.3. Utilizacion de los algoritmos de deteccion

Cuando debemos invocar el algoritmo de deteccion? La respuesta depende de dos factores:

- Con que frecuencia es probable que ocurra un deadlock?
- Cuantos procesos se veran afectados al ocurrir el deadlock?

Tener en cuenta: Frecuencia de llamada al algoritmo de deteccion (Intervalo de tiempo); costo de overhead de este procesamiento. Si existe un deadlock, hasta que se solucione se puede .agrandar.^{el} deadlock porque otros procesos pueden pedir recursos que estan trabados por estos giles. Los bloqueos mutuos solo pueden aparecer cuando una solicitud no se puede atender de inmediato, es posible que esta solicitud sea la que cierra un ciclo en el grafo de asignacion. Por un lado podemos invocar el algoritmo de deteccion de deadlock cada vez que no pueda atenderse inmediatamente un pedido de asignacion. En este caso podemos identificar los procesos que estan en el ciclo, pero mejor aun, podemos saber cual de ellos es el que cerro el ciclo. Si existen muchos tipos de recursos, una solicitud puede ocasionar varios ciclos en el grafo de recursos, cada uno de ellos completado por la solicitud mas reciente y "provocado" por un proceso identificable (si lo bleteamos nos sacamos flor de quilombo de encima). No podemos llamar al algoritmo de deteccion cada vez que se efectua una solicitud porque es demasiado costoso, en vez de esto lo que se puede hacer es llamarlo cada tanto (una hora por ejemplo), o cuando el uso de CPU es menor a algun porcentaje (%40 ponele)