Deadlocks 5

Sistemas Operativos y Distribuidos

Mg. Javier Echaiz
D.C.I.C. – U.N.S.
http://cs.uns.edu.ar/~jechaiz
je@cs.uns.edu.ar

Deadlocks

A deadlock is a situation wherein two or more competing actions are waiting for the other to finish, and thus neither ever does. It is often seen in a paradox like the "chicken or the egg ". (En.Wikipedia)

"When two trains approach each other at a crossing, both shall come to a full stop and neither shall start up again until the other has gone."

— Illogical statute passed by the Kansas Legislature

El Problema del Interbloqueo

- Un conjunto de procesos bloqueados, cada uno teniendo recursos y esperando adquirir otro tenido por procesos en el conjunto.
- Ejemplo 1
  - Un sistema tiene 2 cintas.
  - P_1 y P_2 cada uno tiene una cinta y necesita la otra.
- Ejemplo 2
  - semáforos *A* y *B*, inicializados a 1.

P_0	P_1
wait (A);	*wait(B)*
wait (B);	*wait(A)*

Modelo de Sistema

- Tipos de Recursos R_1, R_2, . . ., R_m
  ciclos CPU, espacio de memoria, dispositivos E/S
- Cada recurso tipo R_i tiene W_i instancias.
- Cada proceso utiliza un recurso como sigue:
  - requerimiento
  - uso
  - liberalización

Caracterización de Interbloqueos

El Interbloqueo puede alcanzarse si se cumplen las cuatro condiciones simultáneamente.

- **Exclusión Mutua :** sólo un proceso a la vez puede usar un recurso.
- **Retener y Esperar:** un proceso mantiene al menos un recurso y está esperando adquirir recursos adicionales tenidos por otros procesos.
- **No Apropiación:** Un recurso puede ser liberado sólo voluntariamente por el proceso que lo tiene, después que el proceso ha completado su tarea.
- **Espera Circular:** existe un conjunto $\{P_0, P_1, \ldots, P_0\}$ de procesos esperando tal que P_0 está esperando por un recurso que es retenido por P_1, P_1 está esperando por un recurso que es retenido por P_2, …, P_{n-1} está esperando por un recurso que es retenido por P_n, y P_0 está esperando por un recurso que es retenido por P_0.

Modelado de Interbloqueos

Grafo dirigido: es un par (N,E) donde N es un conjunto no vacío de nodos y E es un conjunto de lados dirigidos. Un lado dirigido es un par (*a,b*) donde $a,b \in N$.

Camino: Un camino es una secuencia de nodos (*a,b,c,…,i,j*) de un grafo dirigido tales que (*a,b*), (*b,c*), …, (*i,j*) son lados dirigidos.

Ciclo: Un ciclo es un camino donde el primer y último nodo son los mismos.

Modelado de Interbloqueos (cont.)

Conjunto alcanzable: el conjunto alcanzable de un nodo *a* es el conjunto de todos los nodos *b* tales que existe un camino de *a* hasta *b*.

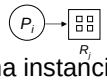
Nudo: Un nudo es un conjunto no vacío K de nodos tales que el conjunto alcanzable de cada nodo en K es exactamente el conjunto K.

Grafo de Asignación de Recursos

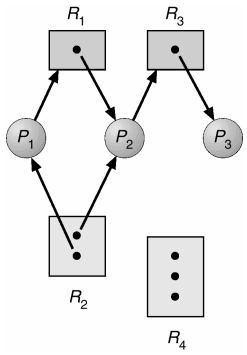
Un conjunto de vértices *V* y un conjunto de lados *E*.

- V está particionado en dos tipos:
  - $P = \{P_1, P_2, \ldots, P_n\}$, el conjunto consistente de todos los procesos en el sistema.
  - $R = \{R_1, R_2, \ldots, R_m\}$, el conjunto consistente de todos los recursos tipo en el sistema.
- lado de requerimiento – lado dirigido $P_1 \rightarrow R_j$
- lado de asignación – lado dirigido $R_j \rightarrow P_i$

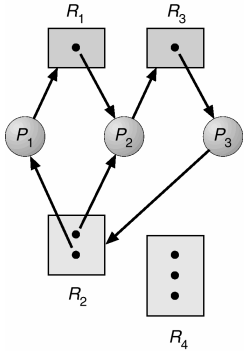
Grafo de Asignación de Recursos (Cont.)

- Proceso 
- Recurso tipo con 4 instancias 
- P_i requiere una instancia de R_j 
- P_i mantiene una instancia R_j 

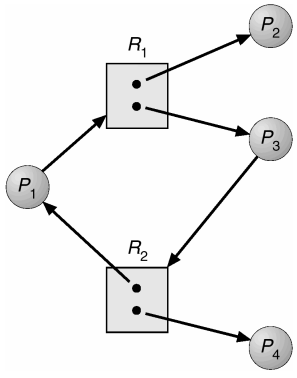
Ejemplo: Grafo de Asignación de Recursos



Grafo de Asignación de Recursos con Interbloqueo



Grafo de Asignación de Recursos con un Ciclo pero sin Interbloqueo



Cuestiones Básicas

- Si un grafo no contiene ciclos ⇒ no hay interbloqueo.
- Si un grafo contiene un ciclo ⇒
  - Si hay una sola instancia por tipo de recurso, entonces hay interbloqueo.
  - Si hay varias instancias por tipo de recurso, hay posibilidad de interbloqueo.

Métodos para Manejo de Interbloqueos

- Asegure que el sistema no entrará *nunca* en estado de interbloqueo.
- Permitir al sistema entrar en un estado de interbloqueo y luego recuperarse.
- Ignore el problema y pretenda que el interbloqueo nunca ocurrió en el sistema; usado en la mayoría de los sistemas operativos incluido UNIX.

Estrategias para Manejo de Interbloqueos

Las estrategias utilizadas para atacar el problema de interbloqueos se establecen en tres niveles de acción: antes que suceda, cuando los procesos están corriendo o con hechos consumados.

Estas estrategias son:

- Prevención
- Evasión
- Detección

Prevención de Interbloqueos

Restringir los modos en que se pueden hacer los requerimientos

- Exclusión Mutua – no requerido para recursos compartidos; debe mantenerse para recursos no compartidos.
- Mantener y Esperar – debe garantizar que siempre que un proceso requiera un recursos no mantiene otros.
  - Un proceso debe requerir y alocar todos sus recursos antes que comience su ejecución, o permitir a los procesos requerir recursos solo cuando no tenga ninguno.
  - Baja utilización de recursos; es posible inanición.

Prevención de Interbloqueos (Cont.)

- No Apropiación –
  - Si un proceso que mantiene algunos recursos requiere otro recurso, no le puede ser inmediatamente asignado, entonces todos los recursos mantenidos son liberados.
  - Los recursos apropiados son agregados a la lista de recursos por los cuales el proceso está esperando.
  - El proceso se reiniciará solo cuando pueda reobtener sus viejos recursos, tanto como los nuevos que está requiriendo.
- Espera Circular – impone un orden total de todos los tipos de recursos, y requiere que cada proceso requiera recursos en un orden creciente de enumeración.

Evasión de Interbloqueos

Requiere que el sistema tenga disponible alguna información adicional *a priori*.

- El modelo más simple y útil requiere que cada proceso declare el *máximo número* de recursos de cada tipo que puede necesitar.
- El algoritmo de evasión de interbloqueos dinámicamente examina el estado de asignación de recursos para asegurar que no puede haber una condición de espera circular.
- El *estado* de asignación de recursos está definido por el número de recursos disponibles y alocados, y las máximas demandas de los procesos.

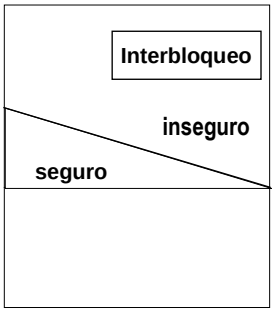
Estado Seguro

- Cuando un proceso requiere un recurso disponible, el algoritmo debe decidir si su asignación inmediata deja al sistema en un estado seguro.
- El sistema está en un estado seguro si existe una secuencia segura de ejecución de todos los procesos.
- La secuencia <P_1, P_2, …, P_n> es segura si por cada P_i, los recursos que P_i puede aún requerir pueden ser satisfechos por recursos disponibles corrientes mas los recursos mantenidos por todos los P_j, con $j<i$.
  - Si los recursos que P_i necesita no están inmediatamente disponibles, entonces P_i puede esperar hasta que todos los P_j hayan finalizado.
  - Cuando P_j finaliza, P_i obtiene los recursos necesitados, ejecuta, retorna los recursos alocados, y termina.
  - Cuando P_i termina, P_{i+1} puede obtener los recursos que necesita, y así sucesivamente.

Cuestiones Básicas

- Si un sistema está en un estado seguro ⇒ no hay interbloqueo.
- Si un sistema está en un estado inseguro ⇒ posibilidad de interbloqueo.
- Evasión ⇒ asegura que el sistema *nunca* va a entrar en un estado inseguro.

Espacio de estados Seguro, Inseguro e Interbloqueo



Algoritmo del Banquero

- Múltiples instancias.
- Cada proceso debe reclamar a priori el máximo uso.
- Cuando un proceso requiere un recurso puede tener que esperar.
- Cuando un proceso obtiene todos sus recursos debe retornarlos en una cantidad finita de tiempo.

Estructura de Datos del Algoritmo del Banquero

Sea n = número de procesos, y m = número de tipos de recursos.

- *Disponible:* Vector de longitud m. Si disponible [j] = k, hay k instancias del tipo de recurso R_j disponible.
- *Max:* matriz $n \times m$. Si *Max* [i,j] = k, entonces el proceso P_i puede requerir a lo sumo k instancias del recurso de tipo R_j.
- *Asignación:* matriz $n \times m$. Si Asignación[i,j] = k entonces P_i tiene alocadas k instancias de R_j.
- *Necesidad:* matriz $n \times m$. Si *Necesidad*[i,j] = k, entonces P_i puede necesitar k instancias mas de R_j para completar su tarea.

Necesidad [i,j] = *Max*[i,j] – *Asignación* [i,j].

Algoritmo

1. Sean *Trab y Final* vectores de longitud m y n, respectivamente. Se inicializan:
   Trab := *Disponible*
   Final [i] = *falso para* i - 1,3, …, n.
2. Encuentre un i tal que cumplan:
   (a) *Final* [i] = *falso*
   (b) $Necesidad_i \leq Trab$
   Si tal i *no* existe, vaya al paso 4.
3. *Trab* := *Trab* + $Asignación_i$
   Final[i] := *verdadero*
   vaya al paso 2.
4. Si *Final* [i] = verdadero para todo i, entonces el sistema está en un estado seguro.

Algoritmo de Requerimiento de Recursos para el Proceso P_i

$Request_i$ = vector de requerimento para el proceso P_i.
Si $Request_i[j] = k$ entonces el proceso P_i quiere k instancias del tipo de recurso R_j.

1. Si $Request_i \leq Necesidad_i$ vaya al paso 2. Sino condición de error, dado que el proceso ha excedido su máximo.
2. Si $Request_i \leq Disponible$, vaya al paso 3. Sino P_i debe esperar, dado que los recursos no están disponibles.
3. Se simula alocar los recursos requeridos P_i modificando el estado como sigue:

   $Disponible := Disponible - Request_i$;
   $Asignación_i := Asignación_i + Request_i$;
   $Necesidad_i := Necesidad_i – Request_i$;;

   - *Si seguro ⇒ los recursos son alocados a P_i.*
   - *Si inseguro ⇒ P_i debe esperar, y es restaurado el viejo estado de asignación de recursos*

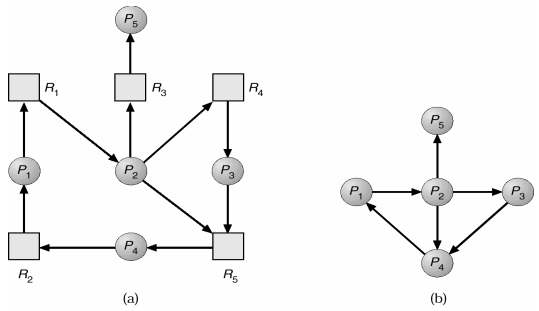
Detección de Interbloqueos

- Permite al sistema entrar en estado de interbloqueo
- Algoritmo de Detección
- Esquema de Recuperación

Instancia Simple de Cada Tipo de Recurso

- Se mantiene un grafo *wait-for*
  - Los nodos son procesos.
  - $P_i \rightarrow P_j$ si P_i está esperando por P_j.
- Periódicamente se invoca un algoritmo que busca por ciclos en el grafo.
- Un algoritmo para detectar un ciclo en un grafo requiere un orden de n^2 operaciones, donde n es el número de vértices en el grafo.

Grafo de Asignación de Recursos y Grafo Wait-for



Grafo de asignación de recursos

Grafo wait-for correspondiente

Varias Instancias de un Tipo de Recurso

- *Disponible:* Vector de longitud m indica el número de recursos disponibles de cada tipo.
- *Asignación:* Una matriz de $n \times m$ que define el número recursos de cada tipo corrientemente alocados por cada proceso.
- *Request:* Una matriz de $n \times m$ matrix que indica el requerimiento corriente de cada proceso. Si $Request[ij] = k$, entonces el proceso P_i está requiriendo k instancias mas del recurso de tipo R_j.

Algoritmo de Detección

1. Sean *Trab* y *Final* vectores de longitud m y n, respectivamente inicializados:
   (a) *Trab = Disponible*
   (b) Para $i = 1,2, \ldots, n$, si $Asignación_i \neq 0$, entonces *Final*[i] := falso; sino *Final*[i] := verdadero.
2. Encuentre un índice i tal que:
   (a) *Final*[i] = *falso*
   (b) $Request_i \leq Trab$
   Si no existe tal i , vaya al paso 4.
3. *Trab* := *Trab* + $Asignación_i$
   Final[i] := *verdadero*
   vaya al paso 2.
4. Si *Final*[i] = falso, para algún i, $1 \leq i \leq n$, entonces el sistema está en un estado de interbloqueo. Es mas, si *Final*[i] = *falso*, entonces P_i está interbloqueado.

Uso del Algoritmo de Detección

- Cuándo y qué tan frecuentemente debe invocarse depende de:
  - ¿Qué tan *frecuentemente* es probable que ocurra un interbloqueo?
  - ¿*Cuántos* procesos serán afectados cuando ocurra un interbloqueo?
- Si el algoritmo de detección es invocado arbitrariamente, puede haber muchos ciclos en el grafo de recursos y no se puede decir cual de los muchos procesos interbloqueados **causó** el interbloqueo.

Recuperación de Interbloqueos: Terminación de Procesos

- Aborto de todos los procesos interbloqueados.
- Aborto de un proceso a la vez hasta que el ciclo de interbloqueo haya sido eliminado.
- ¿En qué orden se elige abortar?
  - Prioridad del proceso.
  - Cuánto tiempo ha computado el proceso, y cuánto falta para completar.
  - Recursos que el proceso ha usado.
  - Recursos que el proceso necesita para completar.
  - Cuántos procesos necesitarán ser terminados.
  - ¿Es el proceso interactivo o batch?

Recuperación de Interbloqueos: Apropiación de Recursos

- Selección de una víctima – minimiza costos.
- Rollback – retorna a algún estado seguro, reinicia el proceso desde ese estado.
- Inanición – algunos procesos pueden ser elegidos siempre como víctimas, incluir un número de rollbacks en el factor de costo.

Extensión a Distribuidos

(ver desde slide 105 del módulo anterior)