Bloqueos Mutuos

UNIVERSIDAD DEL BÍO-BÍO

Agenda

1. Objetivos
2. Introducción
3. Grafos de Asignación de Recursos
4. Condiciones de Deadlock
5. Métodos de manejo de deadlocks

UNIVERSIDAD DEL BÍO-BÍO

Objetivos

- Describir los bloqueos mutuos, que impiden que un conjunto de procesos concurrentes completen sus tareas.
- Presentar una serie de métodos para prevenir o evitar los bloqueos mutuos en un sistema informático.

Concepto

- En un entorno de multiprogramación (varios procesos en MP) varios procesos pueden competir por un número finito de recursos.
- Un proceso solicita recursos, que si no hay disponibles debe esperar.
- **Los recursos que espera los poseen otros procesos que también podrían estar en espera**.
- Ocurre un bloqueo mutuo (Deadlock).
- Un proceso que se encuentra en este estado y que tiene recursos asignados que son demandados por otros procesos afecta al desempeño y funcionamiento del sistema.

UNIVERSIDAD DEL BÍO-BÍO

Modelo del Sistema

- El sistema consiste en un número finito de **recursos** que se distribuyen entre un grupo de **procesos competitivos**.
- Los recursos se clasifican en tipos, pudiendo existir varias instancias idénticas de un mismo tipo.
  - Ciclos de CPU, espacio de memoria, dispositivo de E/S, etc.
- Los procesos para usar un recurso deben respetar el siguiente protocolo:
  - **Solicitar** el recurso
  - **Usar** el recurso
  - **Liberar** el recurso

UNIVERSIDAD DEL BÍO-BÍO

Definición

- "Un Conjunto de procesos está en (estado de) bloqueo mutuo cuando cada uno de los procesos está esperando por un evento que puede ser causado sólo por otro proceso del conjunto".

Condiciones necesarias para Deadlock

- Una situación de deadlock puede surgir si se dan **simultáneamente** las siguientes cuatro condiciones en un sistema:
  - **Exclusión Mutua**. Al menos un recurso debe estar en modo no compartido; es decir, sólo un proceso puede usarlo cada vez. Si otro proceso solicita el recurso, el proceso solicitante tendrá que esperar hasta que el recurso sea liberado.
  - **Retención y Espera**. Un proceso deber estar reteniendo al menos un recurso y esperando para adquirir otros recursos adicionales que actualmente estén retenidos por otros procesos.

Condiciones necesarias para Deadlock

- Una situación de deadlock puede surgir si se dan **simultáneamente** las siguientes cuatro condiciones en un sistema:
  - **Sin Desalojo**. Los recursos no pueden ser desalojados; es decir,un recurso sólo puede ser liberado voluntariamente por el proceso que lo retiene, después de que dicho proceso haya completado su tarea.
  - **Espera Circular**. Debe existir un conjunto $\{P_0, P_1, ..., P_n\}$ de procesos en espera, tal que P_0 esté esperando a un recurso retenido por P_1, P_1 esté esperando a un recurso detenido por P_2, ..., P_{n-1} esté esperando un recurso retenido por P_n y P_n esperando por P_0.

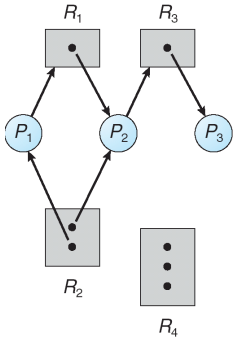
Grafos de Asignación de Recursos

- Los Deadlocks pueden describirse con mayor precisión en función de un grafo dirigido*(V,E)* llamado **Grafo de asignación de recursos**. El cual consiste en:
  - Vértices V, de dos tipos:
    - P = $\{P_1, P_2, ..., P_n\}$, Conjunto de procesos.
    - R = $\{R_1, R_2, ..., R_m\}$, Conjunto de tipos de recursos.
  - Aristas E, de dos tipos:
    - $P_i -> R_j$, Arista de solicitud
    - $R_j -> P_i$, Arista de asignación.

UNIVERSIDAD DEL BÍO-BÍO

Grafos de Asignación de Recursos - Ejemplo

- P = $\{P_1, P_2, P_3\}$
- R = $\{R_1, R_2, R_3, R_4\}$
- $\{P_1 -> R_1, P_2 -> R_3, R_1 -> P_2, R_2 -> P_1, R_2 -> P_2, R_3 -> P_3\}$
- Instancias: #R1=1, #R2=2, #R3=1, #R4=3



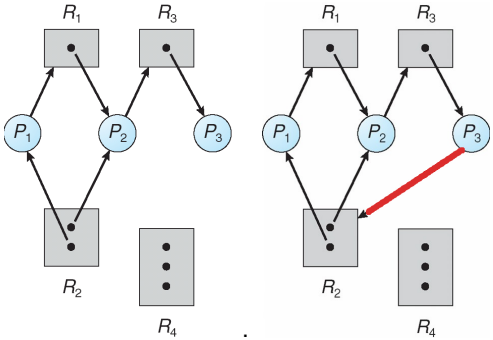
Condiciones de Deadlock

- Si el grafo no tiene un ciclo, entonces no existe deadlock.
- Si existe un ciclo, entonces es posible que exista deadlock (**condición necesaria**).
- Si existe un ciclo, y todos los recursos involucrados tienen sólo una instancia, entonces existe un deadlock (**condición necesaria y suficiente**).

UNIVERSIDAD DEL BÍO-BÍO

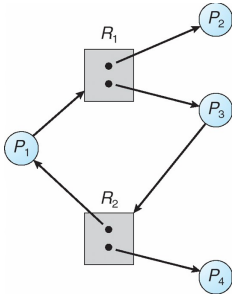
Ejemplo Deadlock

- Suponga que P3 solicita un recurso R2, se crearán dos ciclos:
  - P1 –>R1 –>P2 –>R3 –>P3 –>R2 –>P1
  - P2 –>R3 –>P3 –>R2 –>P2



Ejemplo sin Deadlock

- Hay un ciclo
  - $\{P_1 -> R_1 -> P_3 -> R_2 -> P_1\}$
- No hay bloqueo. P_2 liberará recurso de R_1 y P_4 liberará recurso de R_2.



Métodos de manejo de deadlocks

- Existen tres métodos principales:
  - Ignorar el problema y actuar como si nunca se produjeran deadlocks en el sistema.
  - Protocolos que aseguran la **NO** ocurrencia de deadlock:
    - **Prevención de deadlocks**. Conjunto de métodos que aseguran que al menos una de las condiciones necesarias **NO** ocurra (negación de las condiciones necesarias para que ocurra un bloqueo mutuo).
    - **Evitación de deadlocks**. Conociendo a priori las necesidades máximas de recursos para cada proceso, se analiza para cada solicitud si conduce a un potencial deadlock del sistema (Uso de algoritmos).
  - Protocolos que aceptan la ocurrencia de deadlocks.
    - **Detección y recuperación de deadlocks**. Se usan algoritmos para analizar si el sistema se encuentra en estado deadlock, y si es así, para eliminarlo.

UNIVERSIDAD DEL BÍO-BÍO

Prevención de deadlocks

Se logra por negación de alguna de las cuatro condiciones necesarias.

- **Negación de la exclusión mutua**. Esta es normalmente una propiedad del recurso, por lo tanto no es posible obligar a compartir un recurso que no es posible hacerlo de forma simultánea, por ejemplo, una impresora.
- **Evaluación**
  - La exclusión mutua no es una condición que se pueda negar, por lo tanto hay que apuntar a negar las otras condiciones.

Prevención de deadlocks

- **Negación de retención y espera**.
  - Cuando un proceso solicita recurso, no retenga otros.
  - El proceso debe solicitar todos sus recursos de una sola vez y prosigue cuando se le asignen.
- **Evaluación**
  - Mal uso de los recursos, se pueden asignar recursos pero no se usen por un periodo de tiempo prolongado.
  - Puede producir postergación indefinida en la asignación de recursos.

UNIVERSIDAD DEL BÍO-BÍO

Prevención de deadlocks

- **Negación de NO desalojo**.
  - Si un proceso que retiene recursos solicita nuevos recursos que no pueden ser asignados, el proceso debe liberar (o se le expropian) todos los recursos retenidos.
  - El proceso se recomienza cuando estén disponibles los recursos que tenía asignado y los que solicitaba.
- **Evaluación**
  - Pérdida de trabajo.
  - Puede producir postergación indefinida.
  - Algunos recursos no son posibles de expropiar (impresoras, cintas)

Prevención de deadlocks

- **Negación de Espera Circular**.
  - Se asigna un número entero único a cada tipo de recurso, que permitirá comparar dos recursos y determinar si uno precede al otro dentro de cierto orden (p.e. creciente de numeración).
  - Tipos de recursos $R = \{R_1, R_2, ..., R_m\}$
  - Función $F:R -> N$, donde N pertenece al conjunto de los número naturales.
    - $F(\textit{unidad de cinta}) = 1$
    - $F(\textit{unidad de disco}) = 5$
    - $F(\textit{impresora}) = 12$
  - Si proceso tiene asignado unidades de disco y solicita impresora, entonces se le asigna.
  - Si proceso tiene asignado unidades de disco y solicita unidad de cinta, entonces debe liberar discos antes.
- **Evaluación**
  - Difícil definir un orden a los recursos en un ambiente dinámico.
  - Es bastante restrictivo y poco flexible (depende de la lógica de los procesos).

Evitar deadlocks

- Necesidades de recursos de cada proceso deben ser conocidas de antemano.
- Con esta información, haciendo una asignación cuidadosa de los recursos se puede evitar un deadlock.
- Se logra una mejor utilización de los recursos que la estrategia de prevención.
- Los métodos más simples requieren que los procesos declaren a priori sus necesidades máximas de recursos.

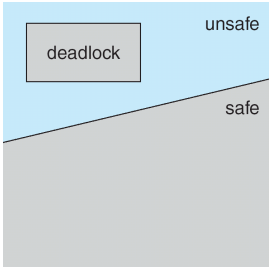
Evitar deadlocks

- **Estado Seguro**
  - Una secuencia de procesos $<P_1, P_2, ..., P_n>$, es una secuencia segura para la asignación actual, si para cada proceso P_i los recursos que aun pueden solicitar pueden ser satisfechos con los recursos actualmente disponibles y con los recursos retenidos por P_j, con $j < i$.
    - Si recursos que necesita P_i no están inmediatamente disponibles, entonces debe esperar que P_j termine.
    - Cuando termine P_j, P_i puede obtener los recursos que necesite de P_j, completar sus tareas y devolver los recursos.
    - Cuando P_i termina, P_{i+1} puede obtener los recursos que necesita, etc.
  - Un sistema está en **estado seguro**, si existe una **secuencia segura** donde se puedan realizar las asignaciones solicitadas de recursos.
  - Si no existe tal secuencia segura, entonces el sistema está en un **estado no seguro**.

UNIVERSIDAD DEL BÍO-BÍO

Evitar deadlocks

- Observaciones:
  - Un estado seguro (safe) **evita** un estado de deadlock.
  - Un estado NO seguro (unsafe) no necesariamente significa un estado de deadlock, pero puede conducir a uno.
  - Por lo tanto, si el SO evita estados NO seguros (unsafe), también evita posibles deadlocks.



Evitar deadlocks

- Dado tres procesos con un sistema de 12 unidades de algún dispositivo. En el tiempo cero, se encuentra en **Estado Seguro**.

Proceso	Nec. máximas	Nececidades actuales
P_0	10	tiene 5 (necesita 5)
P_1	4	tiene 2 (necesita 2)
P_2	9	tiene 2 (necesita 7)
-	-	9 asignados (**3 disponibles**)

- $<P_1, P_0, P_2>$es una secuencia segura.

Algoritmo del Banquero (Dijkstra 1965)

- Tiene este nombre porque podría utilizarse en los sistemas bancarios para garantizar que el banco nunca asigne sus fondos disponibles de tal forma que no pueda satisfacer las necesidades de todos sus clientes.
- Cuando una nueva hebra entra al sistema, debe declarar sus necesidades máximas para cada tipo de recurso.
- Cuando un usuario solicita un recurso, el sistema determina si tal asignación deja al sistema en estado seguro.
  - Si es el caso, lo asigna.
  - En otro caso, el proceso debe esperar hasta que otro proceso libere suficientes recursos y se garantice que el sistema permanezca en estado seguro.

UNIVERSIDAD DEL BÍO-BÍO

Algoritmo del Banquero (Dijkstra 1965)

- **n** = número de procesos, y **m** = número de tipos de recursos.
- **disponible[m]**. vector de longitud m. si disponible[j] = k, entonces hay k instancias de recurso tipo R_j disponible.
- **maximo[n][m]**. matriz n x m. si maximo[i][j] = k, entonces proceso P_i indica necesidad máxima de k instancias del recurso tipo R_j.
- **asignacion[n][m]**. matriz n x m. si asignación[i][j] = k, entonces proceso P_i tiene asignado actualmente k instancias del recurso R_j.
- **necesidad[n][m]**. matriz n x m. si necesidad[i][j] = k, entonces P_i puede necesitar k instancias mas de R_j para completar su tarea.
  - necesidad[i][j] = maximo[i][j] - asignacion[i][j]

Algoritmo del Banquero (Dijkstra 1965)

- n = 5 (5 procesos)
- m = 2 (cinta, CPU)
- disponible[2] = 2 (2 CPUs)
- maximo[3][2] = 1 (P_3 necesidad máxima de 1 CPU)
- asignacion[2][2] = 1 (P_2 tiene asignado 1 CPU)
- necesidad[4][2] = 2 (P_4 necesita 2 CPUs más)

UNIVERSIDAD DEL BÍO-BÍO

Ejemplo - Algoritmo del Banquero

Proceso	Asignación	Máximo	Disponible
-	A B C	A B C	A=3 B=3 C=2
P_0	0 1 0	7 5 3	-
P_1	2 0 0	3 2 2	-
P_2	3 0 2	9 0 2	-
P_3	2 1 1	2 2 2	-
P_4	0 0 2	4 3 3	-

Ejemplo - Algoritmo del Banquero

- P_1 termina y libera recursos.

Proceso	Asignación	Máximo	Disponible
-	A B C	A B C	A B C
P_0	0 1 0	7 5 3	-
$*P_1$	- - -	- - -	5 3 2
P_2	3 0 2	9 0 2	-
P_3	2 1 1	2 2 2	-
P_4	0 0 2	4 3 3	-

Ejemplo - Algoritmo del Banquero

- P_3 termina y libera recursos.

Proceso	Asignación	Máximo	Disponible
-	A B C	A B C	A B C
P_0	0 1 0	7 5 3	-
P_1	- - -	- - -	-
P_2	3 0 2	9 0 2	-
$*P_3$	- - -	- - -	7 4 3
P_4	0 0 2	4 3 3	-

Ejemplo - Algoritmo del Banquero

- P_4 termina y libera recursos.

Proceso	Asignación	Máximo	Disponible
-	A B C	A B C	A B C
P_0	0 1 0	7 5 3	-
P_1	- - -	- - -	-
P_2	3 0 2	9 0 2	-
P_3	- - -	- - -	-
$*P_4$	- - -	- - -	7 4 5

Ejemplo - Algoritmo del Banquero

- P_0 termina y libera recursos.

Proceso	Asignación	Máximo	Disponible
-	A B C	A B C	A B C
$*P_0$	- - -	- - -	7 5 5
P_1	- - -	- - -	-
P_2	3 0 2	9 0 2	-
P_3	- - -	- - -	-
P_4	- - -	- - -	-

Ejemplo - Algoritmo del Banquero

- P_2 termina y libera recursos.

Proceso	Asignación	Máximo	Disponible
-	A B C	A B C	A B C
P_0	- - -	- - -	-
P_1	- - -	- - -	-
$*P_2$	- - -	- - -	10 5 7
P_3	- - -	- - -	-
P_4	- - -	- - -	-

- Este estado es seguro dado que existe la secuencia segura: $<P_1, P_3, P_4, P_0, P_2>$.

Detección y Recuperación de Deadlocks

- Mediante un algoritmo se verifica la existencia o no de un estado de deadlock.
- Se permite la existencia de las tres primeras condiciones necesarias y se detecta la existencia de una espera circular (análisis del grafo de asignación).
- **Idea**:
  - El SO puede chequear periódicamente la existencia de una deadlock.
  - En caso de detectar deadlock, iniciar un procedimiento de recuperación.

Detección y Recuperación de Deadlocks

- Cuando un algoritmo detecta que hay un bloqueo puede realizar:
  - **Recuperación manual**: Avisar de la existencia de deadlock y dejar que se resuelva manualmente.
  - **Recuperación automática**: Realizada por el sistema:
    - Abortar un (o más) proceso (s).
    - Expropiar un (o varios) recurso (s).

Detección y Recuperación de Deadlocks

- **Término de procesos**
  - Técnicas:
    - Abortar todos los procesos bloqueados.
    - Abortar un proceso por vez y chequear si persiste el deadlock.
  - Criterios para elegir un proceso:
    - Prioridad del proceso.
    - Tiempo de ejecución realizado del proceso
    - Cantidad y tipos de recursos retenidos.
    - etc.

Detección y Recuperación de Deadlocks

- **Expropiación de Recursos**
  - Quitar sucesivamente recursos de los procesos y se asignan a otros procesos hasta romper el ciclo del bloqueo.
  - Considerar:
    - Seleccionar una víctima. Tratar de elegir aquella que minimiza los costos.
    - Rollback (vuelta atrás). Volver a un punto de seguridad y recomenzar el proceso.
    - Inanición. Asegurar que proceso seleccionado para expropiar no sea siempre el mismo.