Estrategias para evitar interbloqueos
Algoritmo del banquero

Adaptación (ver referencias al final)

Evitar la ocurrencia de un interbloqueo

- En las estrategias de prevención
  - Diseñar protocolos para que no ocurran al menos una de las cuatro condiciones que llevan a un interbloqueo.
- En las estrategias para evitar entrar en un interbloqueo
  - Se necesita información previa sobre las peticiones futuras de recursos
  - Se analiza el sistema con la información de peticiones futuras
  - Si la petición futura lleva a un interbloqueo, no hay asignación de recursos

R_1	R_2	R_3
10	5	7

Inventario de recursos

Asignación actual

	R_1	R_2	R_3
T_0	0	1	0
T_1	2	0	0
T_2	3	0	2
T_3	2	1	1
T_4	0	0	2

Máximo

R_1	R_2	R_3
7	5	3
3	2	2
9	0	2
2	2	2
4	3	3

Necesario

R_1	R_2	R_3
7	4	3
1	2	2
6	0	0
0	1	1
4	3	1

Disponible

R_1	R_2	R_3
3	3	2

Estado del sistema en un momento dado

- **Asignación actual:** recursos asignados a cada T_i en este momento.
- **Máximo:** Máximo de recursos que serán solicitados por cada T_i
- **Necesario:** Recursos que necesita cada T_i para terminar. Máximo – Asignación actual.

Inventario de recursos

R_1	R_2	R_3
10	5	7

Asignación actual

	R_1	R_2	R_3
T_0	0	1	0
T_1	2	0	0
T_2	3	0	2
T_3	2	1	1
T_4	0	0	2

Máximo

R_1	R_2	R_3
7	5	3
3	2	2
9	0	2
2	2	2
4	3	3

Necesario

R_1	R_2	R_3
7	4	3
1	2	2
6	0	0
0	1	1
4	3	1

Disponible

R_1	R_2	R_3
3	3	2

Llega una nueva solicitud de T_1

- Solicitud de $T_1 = [1, 0, 2]$
- ¿Solicitud de $T_1 \leq$ Disponible?

Inventario de recursos

R_1	R_2	R_3
10	5	7

Asignación actual

	R_1	R_2	R_3
T_0	0	1	0
T_1	3	0	2
T_2	3	0	2
T_3	2	1	1
T_4	0	0	2

Máximo

R_1	R_2	R_3
7	5	3
3	2	2
9	0	2
2	2	2
4	3	3

Necesario

R_1	R_2	R_3
7	4	3
0	2	0
6	0	0
0	1	1
4	3	1

Disponible

R_1	R_2	R_3
2	3	0

¿Se puede asignar la solicitud sin entrar en un estado **no seguro**?

- Supongamos que se realiza la asignación de recursos a $T_1 = [1, 0, 2]$. $[2, 0, 0] + [1, 0, 2] = [3, 0, 2]$
- La asignación se hace de lo que hay en Disponible: $[3, 3, 2] - [1, 0, 2] = [2, 3, 0]$

- Para responder a la pregunta es necesario verificar si con los recursos disponibles puedo hacer otra asignación de tal manera que algún T_i termine y devuelva los recursos asignados.
  - La idea es no quedarse sin recursos para permitir que otros T_i terminen y devuelvan lo asignado

- Después de la asignación, con lo que queda disponible se puede nuevamente asignar a T_1 todo lo que necesite para que termine y devuelva lo asignado.
  - Hacemos esta asignación…

Inventario de recursos

R_1	R_2	R_3
10	5	7

Asignación actual

	R_1	R_2	R_3
T_0	0	1	0
T_1	3	2	2
T_2	3	0	2
T_3	2	1	1
T_4	0	0	2

Máximo

R_1	R_2	R_3
7	5	3
3	2	2
9	0	2
2	1	2
4	3	3

Necesario

R_1	R_2	R_3
7	4	3
0	0	0
6	0	0
0	1	1
4	3	1

Disponible

R_1	R_2	R_3
2	1	0

- Se hace la asignación a $T_1 = [0, 2, 0]$ para que termine: $[3, 0, 2] + [0, 2, 0] = [3, 2, 2]$
- Se resta Máximo – Asignación actual = Necesario: $[3, 2, 2] - [3, 2, 2] = [0, 0, 0]$

Inventario de recursos

R_1	R_2	R_3
10	5	7

Asignación actual

	R_1	R_2	R_3
T_0	0	1	0
T_1	0	0	0
T_2	3	0	2
T_3	2	1	1
T_4	0	0	2

Máximo

R_1	R_2	R_3
7	5	3
3	2	2
9	0	2
2	2	2
4	3	3

Necesario

R_1	R_2	R_3
7	4	3
0	0	0
6	0	0
0	1	1
4	3	1

Disponible

R_1	R_2	R_3
5	3	2

- T_1 ya no requiere más recursos y puede devolver lo asignado: $[3, 2, 2]$ que se suma a lo disponible $[2, 1, 0]$: $[5, 3, 2]$
- Con lo disponible podemos ver que se puede satisfacer Necesario de T_3 para que termine: hacemos esta asignación.

Inventario de recursos

R_1	R_2	R_3
10	5	7

Asignación actual

	R_1	R_2	R_3
T_0	0	1	0
T_1	0	0	0
T_2	3	0	2
T_3	2	2	2
T_4	0	0	2

Máximo

R_1	R_2	R_3
7	5	3
3	2	2
9	0	2
2	2	2
4	3	3

Necesario

R_1	R_2	R_3
7	4	3
0	0	0
6	0	0
0	0	0
4	3	1

Disponible

R_1	R_2	R_3
5	2	1

T_3 ya no requiere más recursos y puede devolver lo asignado: $[2, 2, 2]$ que se suma a lo disponible $[5, 2, 1] = [7, 4, 3]$

R_1	R_2	R_3
10	5	7

Inventario de recursos

Asignación actual

	R_1	R_2	R_3
T_0	0	1	0
T_1	0	0	0
T_2	3	0	2
T_3	0	0	0
T_4	0	0	2

Máximo

R_1	R_2	R_3
7	5	3
3	2	2
9	0	2
2	2	2
4	3	3

Necesario

R_1	R_2	R_3
7	4	3
0	0	0
6	0	0
0	0	0
4	3	1

Disponible

R_1	R_2	R_3
7	4	3

Con lo disponible se puede satisfacer a T_4 a T_0 y por último a T_2

- La solicitud $\boldsymbol{T_1 = [1, 0, 1]}$ se puede satisfacer en el momento solicitado ya que la secuencia $< \boldsymbol{T_1, T_3, T_4, T_0, T_2} >$ garantiza un estado seguro del sistema.

  - El sistema no se queda sin recursos para asignar y permitir que todos los T_i terminen y devuelvan lo asignado.

Inventario de recursos

R_1	R_2	R_3
10	5	7

Asignación actual

Máximo

Necesario

	R_1	R_2	R_3	R_1	R_2	R_3	R_1	R_2	R_3
T_0	0	1	0	7	5	3	7	4	3
T_1	3	0	2	3	2	2	0	2	0
T_2	3	0	2	9	0	2	6	0	0
T_3	2	1	1	2	2	2	0	1	1
T_4	0	0	2	4	3	3	4	3	1

Disponible

R_1	R_2	R_3
2	3	0

Nos devolvemos al estado en el que se asignó $T_1 = [1, 0, 1]$
En donde el sistema tiene el estado indicado

- Se hace la solicitud $T_4 = [3, 3, 0]$
- ¿Solicitud $T_4 \leq$ Disponible ? No se puede satisfacer.

R_1	R_2	R_3
10	5	7

Inventario de recursos

Asignación actual

	R_1	R_2	R_3
T_0	0	1	0
T_1	3	0	2
T_2	3	0	2
T_3	2	1	1
T_4	0	0	2

Máximo

R_1	R_2	R_3
7	5	3
3	2	2
9	0	2
2	2	2
4	3	3

Necesario

R_1	R_2	R_3
7	4	3
0	2	0
6	0	0
0	1	1
4	3	1

Disponible

R_1	R_2	R_3
2	3	0

Nos devolvemos al estado en el que se asignó $T_1 = [1, 0, 1]$
En donde el sistema tiene el estado indicado

- Se hace la solicitud $T_0 = [0, 2, 0]$
- ¿Solicitud $T_0 \leq$ Disponible? Si, pero analicemos qué pasa si se hace la asignación solicitada

Inventario de recursos

R_1	R_2	R_3
10	5	7

Asignación actual

	R_1	R_2	R_3
T_0	0	3	0
T_1	3	0	2
T_2	3	0	2
T_3	2	1	1
T_4	0	0	2

Máximo

R_1	R_2	R_3
7	5	3
3	2	2
9	0	2
2	2	2
4	3	3

Necesario

R_1	R_2	R_3
7	1	3
0	2	0
6	0	0
0	1	1
4	3	1

Disponible

R_1	R_2	R_3
2	1	0

- Se hace asignación $T_0 = [0, 2, 0]$ desde Disponible
- Se resta a Disponible la asignación $[0, 2, 0] = [2, 1, 0]$
- Se suma a Asignación actual de T_0 lo asignado $[0, 2, 0]$.
- Se resta a Necesario de T_0 lo Actual de T_0.

- Es este punto vemos que la solicitud $\boldsymbol{T_0 = [0, 2, 0]}$, de ser asignada, lleva al sistema a un estado en el queda sin recursos disponibles.
  - Todos los T_i se quedarán esperando en interbloqueo

- La solicitud $\boldsymbol{T_0 = [0, 2, 0]}$, aunque puede pensarse que es viable ser asignada, llevaría al sistema a un estado de interbloqueo.

- $\boldsymbol{T_0}$ tendría que esperar a que aparezcan otras solicitudes o devoluciones de recursos que lo satisfagan y que no pongan al sistema en un estado no seguro.

Algoritmo del banquero

- Se requieren varias estructuras de datos para su implementación.
- Sea $\boldsymbol{n}$ el número de hilos/procesos en el sistema $(T_1, T_2, \ldots, T_n)$
- Sea $\boldsymbol{m}$ el número de tipos de recursos $(R_1, R_2, \ldots R_n)$
- Disponible
  - Vector de longitud $\boldsymbol{m}$. Indica el número de recursos disponibles de cada tipo.
  - $Disponible[j] = k$, indica que hay $\boldsymbol{k}$ instancias disponibles del recurso $\boldsymbol{R_j}$.
- Máximo
  - Matriz de $\boldsymbol{n \times m}$ que indica la demanda máxima de recursos por cada hilo/proceso.
  - $Máximo[i][j] = \boldsymbol{k}$, indica que el hilo $\boldsymbol{T_i}$ requiere máximo $\boldsymbol{k}$ instancias del recurso $\boldsymbol{R_j}$.

Algoritmo del banquero

- Asignación (actual)
  - Matriz de $\boldsymbol{n \times m}$ que indica el número de recursos actualmente asignados a cada $\boldsymbol{T_i}$.
  - $Asignación[i][j] = k$, indica que $\boldsymbol{T_i}$ tiene asignado $\boldsymbol{k}$ instancias del recurso $\boldsymbol{R_j}$.
- Necesario
  - Matriz de $\boldsymbol{n \times m}$ que indica el número de cada recurso que le hacen falta a cada hilo/proceso para terminar.
  - $Necesario[i][j] = k$, indica que $\boldsymbol{T_i}$ necesita $\boldsymbol{k}$ instancias adicionales de $\boldsymbol{R_j}$ para terminar.
  - $\boldsymbol{Necesario[i][j] = Máximo[i][j] - Asignación[i][j]}$

Algoritmo del banquero: estado seguro

- Paso 1: Inicializar las siguientes estructuras
  - Sea $\boldsymbol{Trabajo}[]$ un vector de tamaño $\boldsymbol{m}$
  - Sea $\boldsymbol{Finaliza}[]$ un vector de tamaño $\boldsymbol{n}$
  - Inicializar $\boldsymbol{Trabajo} = \boldsymbol{Disponible}$
  - Inicializar $\boldsymbol{Finaliza}[\boldsymbol{i}] = \boldsymbol{Falso}$ para $i = 0, 1, 2, \dots, n-1$
- Paso 2: Encontrar un índice i tal que, se cumplan la siguiente condición
  - $\boldsymbol{Finaliza}[\boldsymbol{i}] = \boldsymbol{Falso}$ y $\boldsymbol{Necesario_i} \leq \boldsymbol{Trabajo}$
  - Si $\boldsymbol{i}$ no existe, ir al paso 4.
  - $\boldsymbol{Necesario_i}$ es el vector fila de la matriz $\boldsymbol{Necesario}$

Algoritmo del banquero: estado seguro

- Paso 3
  - $\boldsymbol{Trabajo = Trabajo + Asignación_i}$
  - $\boldsymbol{Finaliza[i] = Verdadero}$
  - Regresa al paso 2
- Paso 4
  - Si $\boldsymbol{Finaliza[i] == Verdadero}$ para todo $\boldsymbol{i}$, entonces el sistema está en estado seguro.

Algoritmo del banquero: solicitud de recurso

- Sea $Solicitud_i$ un vector que indica la solicitud de T_i.
- Si $Solicitud_i[j] == k$, entonces T_i solicita k instancias del recurso R_j.
- Cuando llega una nueva solicitud al sistema de T_i, se ejecutan las siguientes acciones.
  - Paso 1: Si $Solicitud_i \leq Necesita_i$, ir al paso 2. Sino, error ya que T_i excede su demanda máxima de recursos
  - Paso 2: $Solicitud_i \leq Disponible$, ir al paso 2. Sino, T_i debe esperar por recursos disponibles.
  - Paso 3: Se modifican los estados de la siguiente manera (siguiente *slide*)

Algoritmo del banquero: solicitud de recurso

- Paso 3: Se modifican los estados de la siguiente manera
  - $\boldsymbol{Disponible = Disponible - Solicitud_i}$
  - $\boldsymbol{Asignación_i = Asignación_i + Solicitud_i}$
  - $\boldsymbol{Necesita_i = Necesita_i - Solicitud_i}$
- Si la asignación lleva a un estado seguro
  - Se asignan los recursos a $\boldsymbol{T_i}$
- Si la asignación lleva a un estado inseguro
  - $\boldsymbol{T_i}$ debe esperar por $\boldsymbol{Solicitud_i}$ y se restaura el estado anterior del sistema (antes de la asignación de recursos).

Referencias

- Silberschatz, A., Baer Galvin, P., & Gagne, G. (2018). Deadlock avoidance. In *Operating Systems Concepts* (10th ed., pp. 330–337). John Wiley & Sons, Inc.