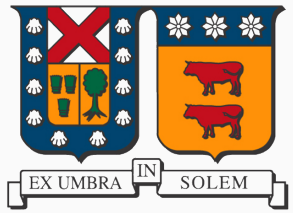


Sistemas Operativos

Deadlocks

Viktor Andrés Tapia Vásquez

Segundo Semestre 2021

Departamento de Informática, Campus SSJJ.

Índice de Contenidos

Definición

Definición

- Es un ciclo de espera entre un conjunto de tareas por ejecutarse.
- Cada tarea espera que otra realice alguna acción.
- Dicha acción no puede ocurrir.
- Es un desafío para el desarrollo de aplicaciones multi hebras.

Definición

Ejemplo:

```
Thread A

lock1.acquire();
...
lock2.acquire();
while (necesito_esperar) {
      cv.wait(&lock2);
}
lock2.release();
.......
lock1.release();
```

```
Thread B

lock1.acquire();
.........
lock2.acquire();
.........
cv.signal()
lock2.release();
....
lock1.release();
```

En este caso, Thread A llama lock2 reteniendo lock1 y espera sobre cv la liberación de lock2. Ocurre deadlock si Thread B necesita lock1

Definición

- Deadlock e Inanición se relacionan con la vida de un proceso o hebra.
- En la inanición se queda detenido por un tiempo indefinido.
- Deadlock es inanición, pero tiene una condición más fuerte.
- No se puede continuar.
- El problema es que ambos no ocurren siempre.
- Herramientas de testing no descubren directamente esto.
- Será necesario un esfuerzo adicional en el desarrollo.

Definición - Condiciones Generales

- Un sistema con un número finito de recursos.
- Los recursos se distribuyen entre distintas tareas en competición.
- Distintos tipos y cada uno puede tener distintas instancias.
- Ciclos de CPU, archivos, E/S, memoria, etc.

Definición - Condiciones Generales

- Si una tarea solicita un recurso, la asignación de cualquier instancia de éste satisface la solicitud realizada.
- Un recurso no puede ser utilizado si no ha sido asignado.
- Se pueden solicitar tantos como se necesiten.
- El total de solicitados no puede exceder al total disponible.

Definición - Condiciones Generales

- La utilización de los recursos sigue la secuencia:
  - ✓ **Solicitud:** La tarea espera mientras la solicitud sea respondida.
  - ✓ **Uso:** Se utiliza el recurso.
  - ✓ **Liberación:** Se libera el recurso.
- Solicitud y Liberación requieren uso de llamadas al sistema.

Definición - Condiciones para el Deadlock

- **Exclusión Mutua:** Una tarea utilizando un recurso a la vez.
- **Espera y Retención:** Tarea en espera retiene el recurso.
- **No se puede quitar:** Recursos no pueden ser quitados.
- **Espera circular:** Existe un conjunto de tareas que esperan, y cada tarea espera por un recurso que tiene otra tarea.

Felices las 4

Las condiciones deben ocurrir de forma simultánea.

Definición - Grafo de asignación

- Podemos definir el deadlock mediante un grafo.
- Modela la asignación de recursos del sistema.
- Está formado por un conjunto de vértices (V) y arcos (E).
- V está particionado en dos conjuntos:
  - ✓ $T = \{T_1, T_2, ..., T_n\}$. Tareas del sistema.
  - ✓ $R = \{R_1, R_2, ..., R_n\}$. Recursos del sistema.
- Arco de Solicitud: $T_i \rightarrow R_j$.
- Arco Asignación: $R_j \rightarrow T_i$.

Definición - Grafo de asignación

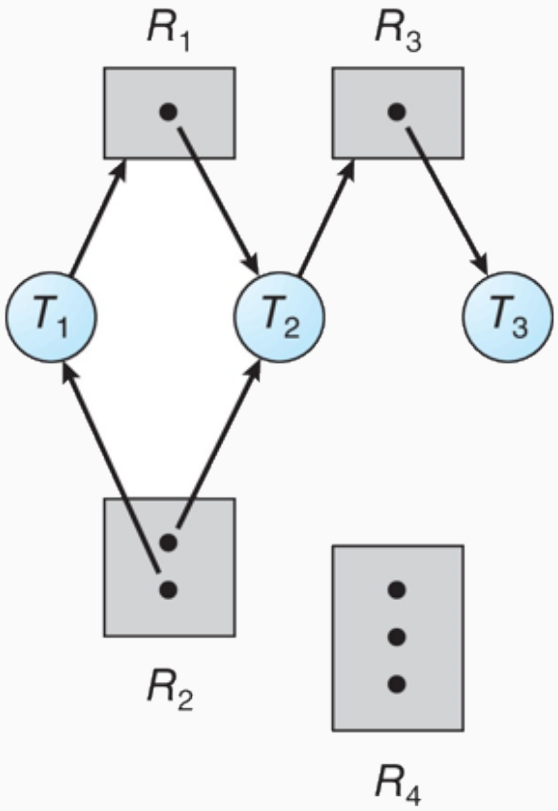


Figure 1: Ejemplo: Grafo de Asignación

Definición - Grafo de asignación

- Si un grafo no tiene ciclos, no hay deadlock.
- Si un grafo tiene ciclos:
  - ✓ Si hay una instancia por tipo de recurso, existe deadlock.
  - ✓ Si hay varias instancias, existe la posibilidad de deadlock.

Definición - Grafo de asignación

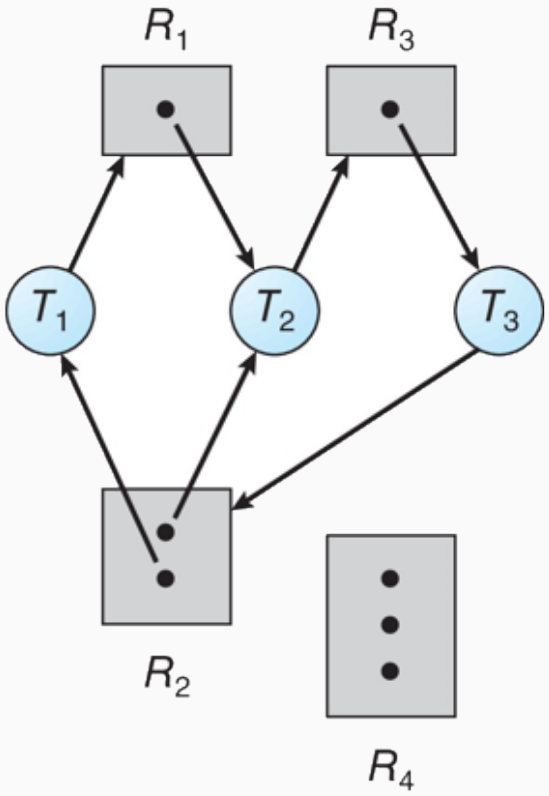


Figure 2: Grafo de Asignación con Deadlock

Definición - Grafo de asignación

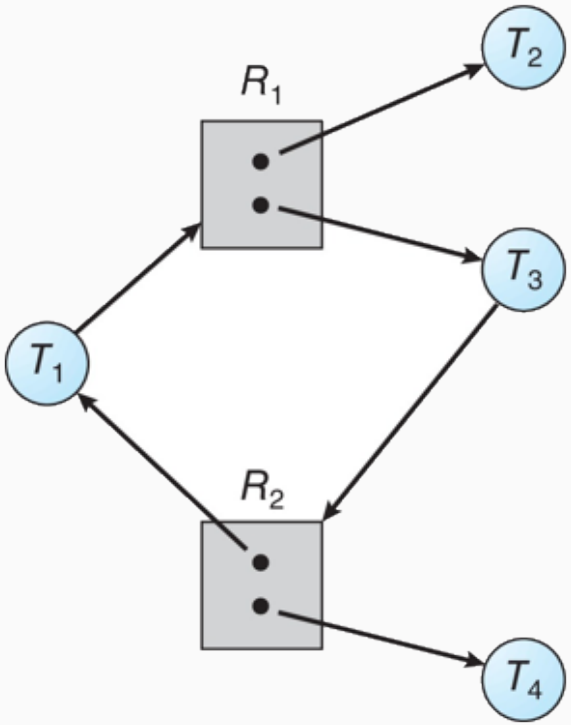


Figure 3: Grafo de Asignación sin Deadlock

¿Qué Hacemos?

¿Qué Hacemos?

- Asegurarnos que el sistema **nunca** entrará en deadlock.
  - ✓ Prevención.
  - ✓ Evasión.
- Permitir el deadlock y recuperar el sistema.
- Ignorar el problema y pretender que no hay problema.

¿Qué Hacemos? - Prevención

Limitar las 4 condiciones. Algunas de las acciones a realizar:

- Exclusión mutua no requerida para recursos compartidos.
- Obligar que las tareas no estén reteniendo para solicitar recursos.
- Si una tarea está reteniendo y solicita un recurso que no se puede asignar inmediatamente, se libera el recurso retenido.
- Imponer un orden en la solicitud de recursos por parte de las tareas.

¿Qué Hacemos? - Evasión

El sistema debe manejar información a priori:

- Cada tarea declara el máximo número de recursos que necesita.
- Dinámicamente se examina el estado de asignaciones.
- El estado de asignación queda definido por los recursos disponibles, asignados y demandados.

¿Qué Hacemos? - Evasión

- El análisis se realiza cuando una tarea solicita un recurso disponible.
- Si su asignación deja el sistema en **estado seguro**, se realiza.
- Sistema en estado seguro no tiene deadlock.
- Sistema en estado inseguro podría tener deadlock.

Estado Seguro

Debe existir al menos una secuencia de asignación de recursos para todas las tareas del sistema tal que para cada una de ellas se puedan asignar los recursos que aún necesitan.

¿Qué Hacemos? - Evasión

- La idea es evitar entrar en un estado inseguro.
- Cantidad de instancias por tipo de procesos:
  - ✓ Una $\rightarrow$ Grafo de asignación de recursos.
  - ✓ Múltiples $\rightarrow$ Algoritmo del banquero.

¿Qué Hacemos? - Evasión - Grafo

- **Arco demanda:** $T_i \rightarrow R_j$ (línea punteada).
- Indica que la tarea podría requerir un recurso.
- Se convierte en requerimiento cuando se requiere un recurso.
- Se convierte en asignación cuando el recurso es asignado.
- Cuando se libera el recurso, se convierte en demanda.
- Los recursos deben ser demandados a priori.

¿Qué Hacemos? - Evasión - Grafo

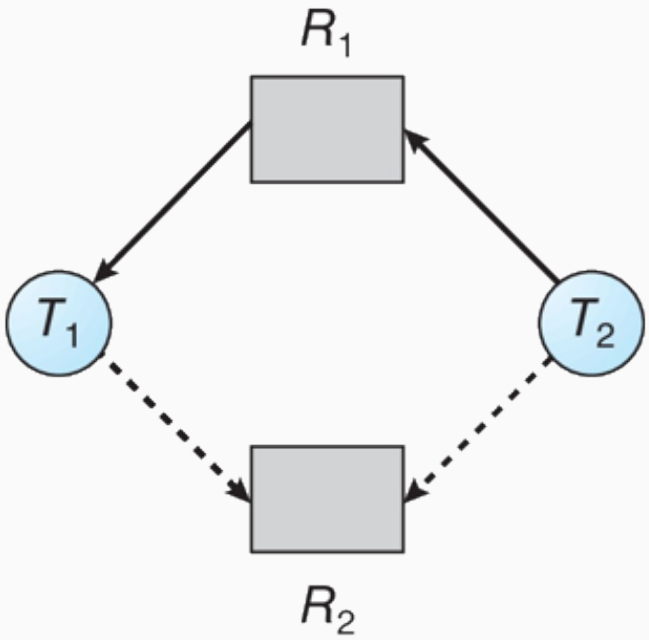


Figure 4: Grafo de Asignación Recursos

¿Qué Hacemos? - Evasión - Grafo

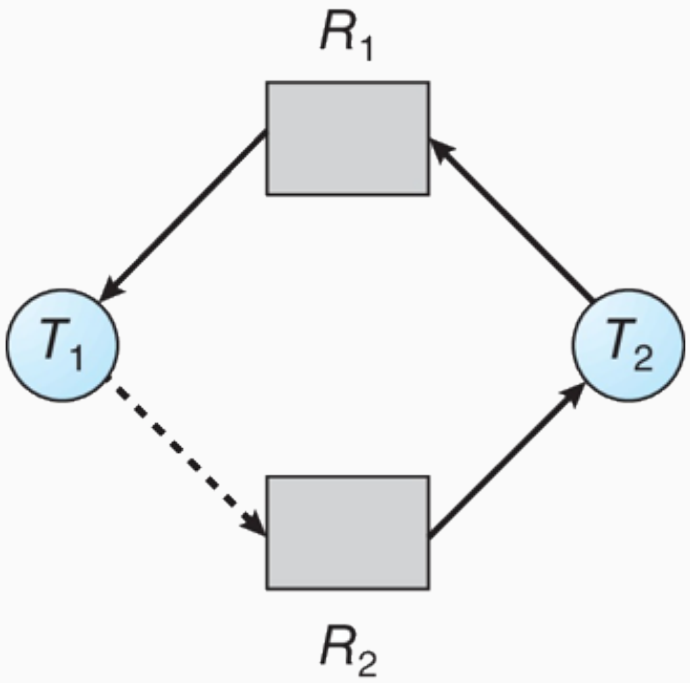


Figure 5: Estado Inseguro

¿Qué Hacemos? - Evasión - Grafo

- Cuando una tarea hace una petición de un recurso:
- Puede ser resuelta solo si al convertir un arco de requerimiento en un arco de asignación no se genera un ciclo en el grafo.

¿Qué Hacemos? - Evasión - Banquero

- Utilizado para múltiples instancias de tipos de recursos.
- Cada tarea debe demandar a priori el máximo uso de recursos.
- Una tarea puede tener que esperar.
- Cuando obtiene todos sus recursos los utiliza y los libera.

¿Qué Hacemos? - Evasión - Banquero

Sean $n =$ número de tareas y $m =$ número de tipos de recursos.

- **Disponible:**
  - ✓ Vector de largo m.
  - ✓ Si Disponible $[j] = k$, hay k instancias disponibles de R_j.
- **Max:**
  - ✓ Matriz $n \times m$.
  - ✓ Si Max $[i, j] = k$, la tarea T_i puede requerir a lo más k instancias del recurso tipo R_j.

¿Qué Hacemos? - Evasión - Banquero

Sean $n =$ número de tareas y $m =$ número de tipos de recursos.

- **Asignado:**
  - ✓ Matriz $n \times m$.
  - ✓ Si Asignado $[i, j] = k$, T_i tiene asignado k instancias de R_j.
- **Necesita:**
  - ✓ Matriz $n \times m$.
  - ✓ Si Necesita $[i, j] = k$, la tarea T_i puede necesitar k instancias adicionales del recurso tipo R_j.
  - ✓ Necesita $[i, j] =$ Max $[i, j]$ - Asignado $[i, j]$.

¿Qué Hacemos? - Evasión - Banquero

(1) Defina los vectores *Trabajo* y *Fin* de largo m y n respectivamente:

Trabajo $=$ Disponible
Fin$[i] =$ false, for $i = 0, 1, ..., n - 1$

(2) Busque un i tal qué:

Fin$[i] =$ false
Necesita$_i \leq$ Trabajo.
Si no existe, saltar al paso 4.

¿Qué Hacemos? - Evasión - Banquero

(3) Ejecute:

Trabajo = Trabajo + Asignado[i]
Fin[i] = true
Volver al paso 2.

(4) Si Fin[i] = true para todos los i, el sistema está seguro.

¿Qué Hacemos? - Evasión - Banquero

En caso que se requiera realizar una nueva solicitud debemos:

(1) Definir el vector $Solicitud_i$ para la tarea que está solicitando.
Si $\text{Solicitud}_i[j] = k$, la tarea T_i quiere k instancias de R_j.

(2) Si $\text{Solicitud}_i \leq \text{Necesita}_i$ avanza. Caso contrario, error.

(3) Si $\text{Solicitud}_i \leq \text{Disponible}_i$ avanza. Caso contrario, espera.

¿Qué Hacemos? - Evasión - Banquero

(4) Ejecute:

Disponible = Disponible - Solicitud$_i$

Asignado$_i$ = Asignado$_i$ + Solicitud$_i$

Necesita$_i$ = Necesita$_i$ - Solicitud$_i$

(5) Volvemos a correr el banquero.

Si es seguro, se realiza la asignación.

Caso contrario, se debe esperar y restaurar el estado anterior.

¿Qué Hacemos? - Evasión - Banquero

- Ejemplo: 5 procesos, 3 recursos (A, B y C).
- Instancias: A = 10, B = 5 y C = 7.

	Allocation	*Max*	*Available*
	A B C	*A B C*	*A B C*
P_0	0 1 0	7 5 3	3 3 2
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	

¿Qué Hacemos? - Evasión - Banquero

- Calculamos el contenido de la matriz Necesita:

	Need
	A B C
P_0	7 4 3
P_1	1 2 2
P_2	6 0 0
P_3	0 1 1
P_4	4 3 1

- El sistema está en estado seguro: P1, P3, P4, P2 y P0.
- Consideremos ahora que P1 solicita (1,0,2).

¿Qué Hacemos? - Evasión - Banquero

- Solicita $\leq$ Disponible es verdadero.

	Allocation	*Need*	*Available*
	A B C	*A B C*	*A B C*
P_0	0 1 0	7 4 3	2 3 0
P_1	3 0 2	0 2 0	
P_2	3 0 2	6 0 0	
P_3	2 1 1	0 1 1	
P_4	0 0 2	4 3 1	

- El sistema está en estado seguro: P1, P3, P4, P0 y P2.

¿Qué Hacemos? - Detección

Otra opción es permitir que el sistema entre en Deadlock. Necesitamos:

- Algoritmo de detección.
- Mecanismo de recuperación.

¿Qué Hacemos? - Detección

- Recursos con única instancia utilizaremos el grafo *wait-for*.
- Los nodos son tareas.
- $T_i \rightarrow T_j$. Si T_i espera por T_j.
- Se invoca periodicamente en busca de un ciclo.
- Si hay un ciclo, hay deadlock.
- Esto suele ser bastante costoso.

¿Qué Hacemos? - Detección

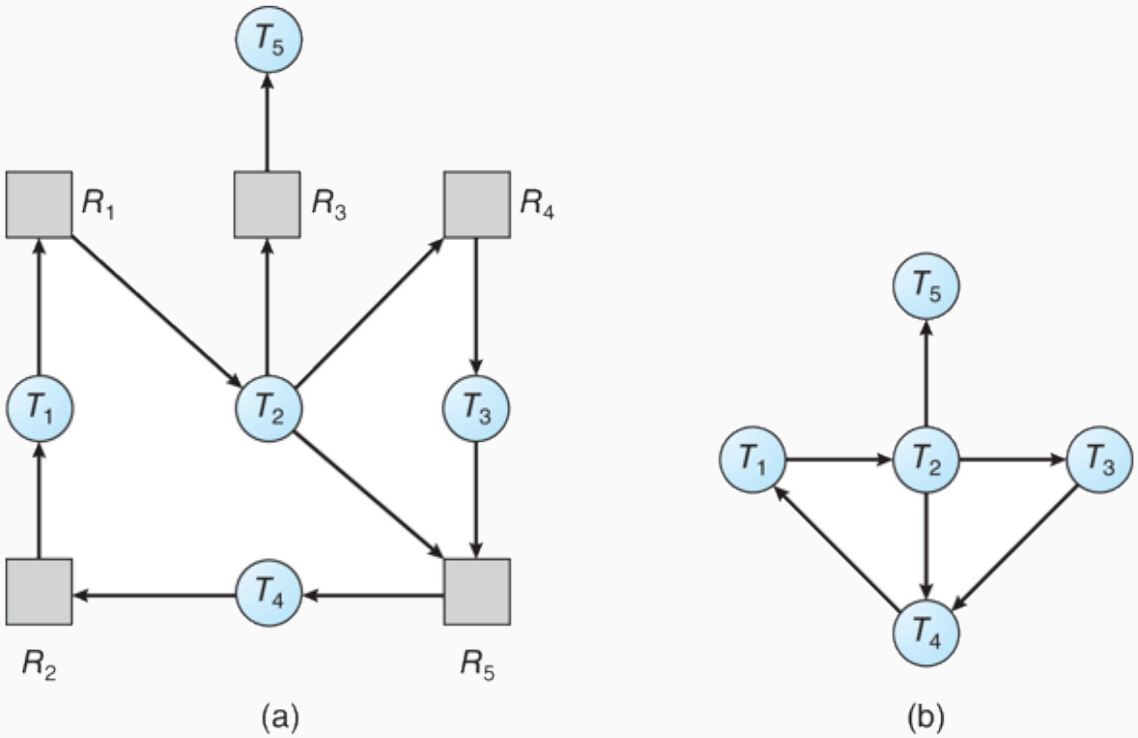


Figure 6: Grafo Asignación v/s Grafo wait-for

¿Qué Hacemos? - Detección

Para recursos con múltiples instancias usamos un Banquero modificado:

- **Asignado:**
  - ✓ Matriz $n \times m$.
  - ✓ Define número de tipos de recursos asignados.
- **Disponible:**
  - ✓ Vector de largo m. Indica las instancias disponibles.
- **Solicitud:**
  - ✓ Matriz $n \times m$.
  - ✓ Indica las siguientes solicitudes de las tareas.

¿Qué Hacemos? - Detección

(1) Defina los vectores *Trabajo* y *Fin* de largo m y n respectivamente:

Trabajo $=$ Disponible
For $i = 1, 2, ..., n$. If Asignado$_i \neq 0$ then Fin$[i] =$ false
Caso contrario Fin$[i] =$ true

(2) Busque un i tal qué:

Fin$[i] =$ false
Solicitud$_i \leq$ Trabajo.
Si no existe, saltar al paso 4.

¿Qué Hacemos? - Detección

(3) Ejecute:

Trabajo = Trabajo + Asignado$_i$
Fin[i] = true
Vuelva paso 2.

(4) Si Fin[i] == false para algún valor de i, el sistema está en deadlock.

If Fin[i] = false, entonces T_i está en deadlock.

Requiere realizar operaciones con un orden de $O(m \times n^2)$.

¿Qué Hacemos? - Detección

- Ejemplo: 5 procesos, 3 recursos (A, B y C).
- Instancias: A = 7, B = 2 y C = 6.

	Allocation	*Request*	*Available*
	A B C	*A B C*	*A B C*
P_0	0 1 0	0 0 0	0 0 0
P_1	2 0 0	2 0 2	
P_2	3 0 3	0 0 0	
P_3	2 1 1	1 0 0	
P_4	0 0 2	0 0 2	

- La secuencia P0, P2, P3, P1 y P4 resulta Fin$[i]$ = true para todo i.

¿Qué Hacemos? - Detección

- Consideremos que P2 solicita una instancia adicional de C.
- ¿Estado del sistema?
- Puede reclamar los recursos de P0 pero es insuficiente.
- Hay deadlock. Involucrados: P1, P2, P3 y P4.

¿Qué Hacemos? - Detección

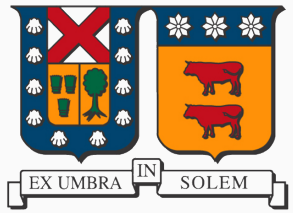
- Cuándo y qué tan seguido utilizamos los algoritmos?
  - ✓ Qué tan seguido puede ocurrir un deadlock?
  - ✓ Cuántos procesos hay que restaurar?
- Si se invoca de forma arbitraria:
  - ✓ Podría encontar una situación con muchos ciclos.
  - ✓ No podríamos determinar quién causo el deadlock.

¿Qué Hacemos? - Recuperación

- Dos opciones:
  - ✓ Finalizar tareas.
  - ✓ Quitar recursos.
- Finalización:
  - ✓ Terminar todos los procesos en deadlock.
  - ✓ Terminar un proceso por vez hasta eliminar el ciclo.
- ¿En qué orden deberíamos elegir?
  - ✓ Prioridad y/o tipo de tarea.
  - ✓ Uso de CPU utilizado y/o restante.
  - ✓ Uso de recursos.

¿Qué Hacemos? - Recuperación

- Quitar recursos:
  - ✓ Seleccione una vítctima.
  - ✓ Quitar los recursos y restaurarlo.
  - ✓ Volver a un estado seguro.
  - ✓ Reiniciar procesos.
  - ✓ Puede ocurrir inanición.
  - ✓ Guardar el número de restauraciones como id.



Sistemas Operativos

Deadlocks

Viktor Andrés Tapia Vásquez

Segundo Semestre 2021

Departamento de Informática, Campus SSJJ.