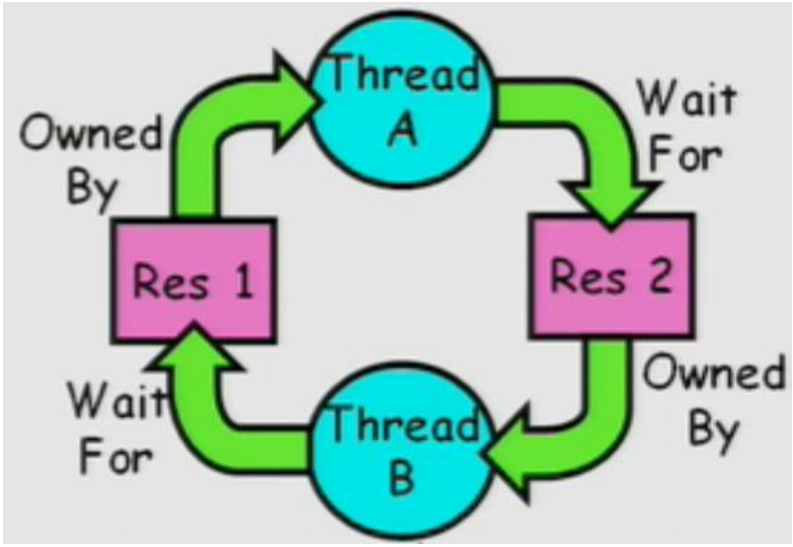
7: Deadlocks

Sistemas Operativos 1
Ing. Alejandro León Liu

- Describir deadlocks
- Métodos para prevenir y evitar deadlocks

- Sistema
  - Recursos
    - CPU
    - Impresoras
    - Memoria
    - Semaphores
    - Múltiples instancias de ciertos recursos
    - Exclusivos o compartidos
  - Proceso
    - Solicita recurso
      - request(), open(), allocate(), wait()
    - Utiliza recurso
    - Libera recurso
      - release(), close(), free(), notify(), signal()

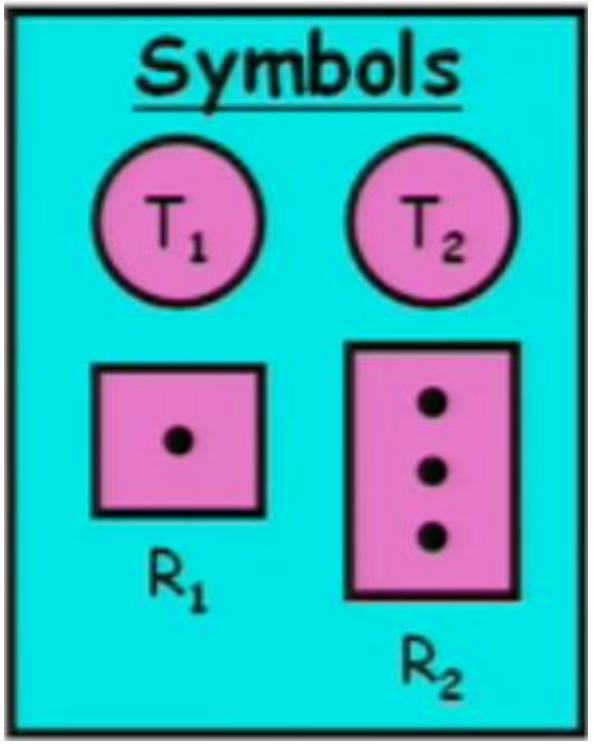
- Starvation vs. Deadlocks
  - Starvation
    - Thread espera indefinidamente
    - Ocurre en SJF y en algoritmo por prioridad
      - Ejemplo: Proceso con prioridad baja nunca se ejecuta ya que siempre hay procesos de mayor prioridad.
    - No puede ocurrir en FCFS.
    - Puede terminar
  - Deadlock
    - Espera circular por recursos. Procesos no pueden ejecutarse.
    - No puede terminar sin intervención externa


Thread A
Wait For
Res 2
Owned By
Thread B
Wait For
Res 1
Owned By

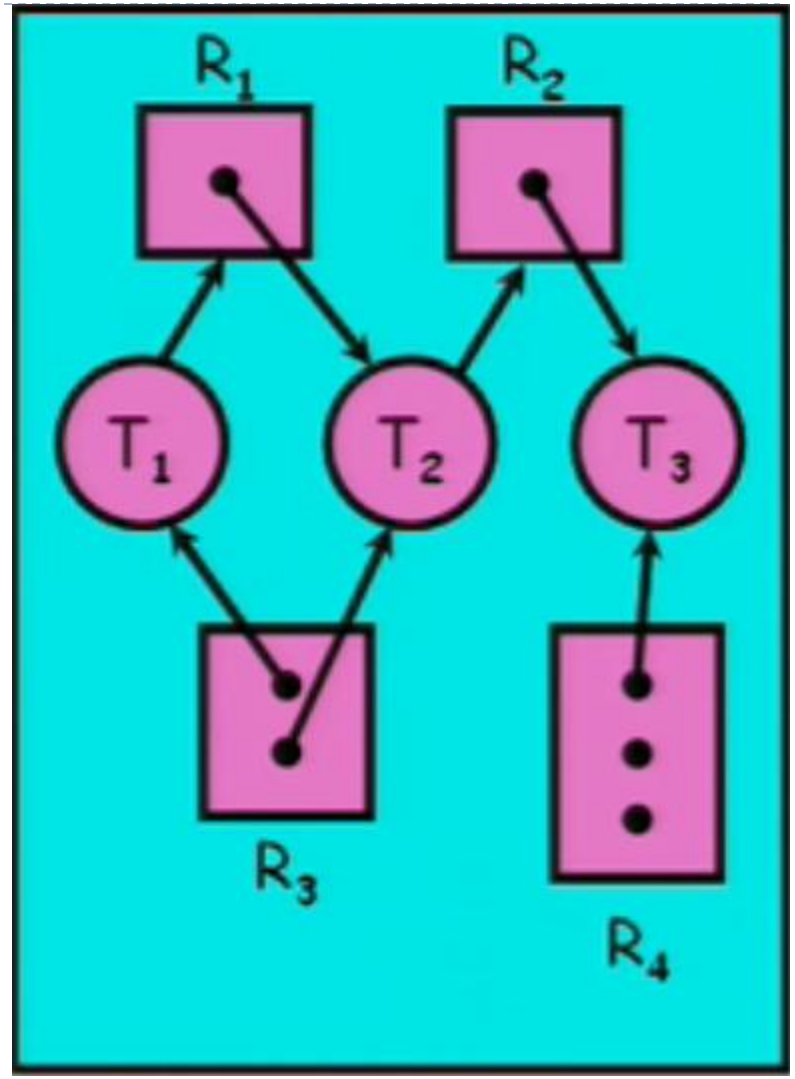
- Condiciones
  - Mutual exclusion
    - Por lo menos un recurso no pude ser compartido (Utilizado por una thread)
  - Hold and wait
    - Thread debe tener un recurso y esperando que otros recursos sean liberados
  - No preemption
    - Thread libera recurso por su propia cuenta.
  - Circular wait
    - Existe conjunto {T1, T2 … Tn} tal que T1 espera por un recurso de T2, Ti espera por un recurso de Ti+1,Tn espera por un recurso de T1

RESOURCE ALLOCATION GRAPH

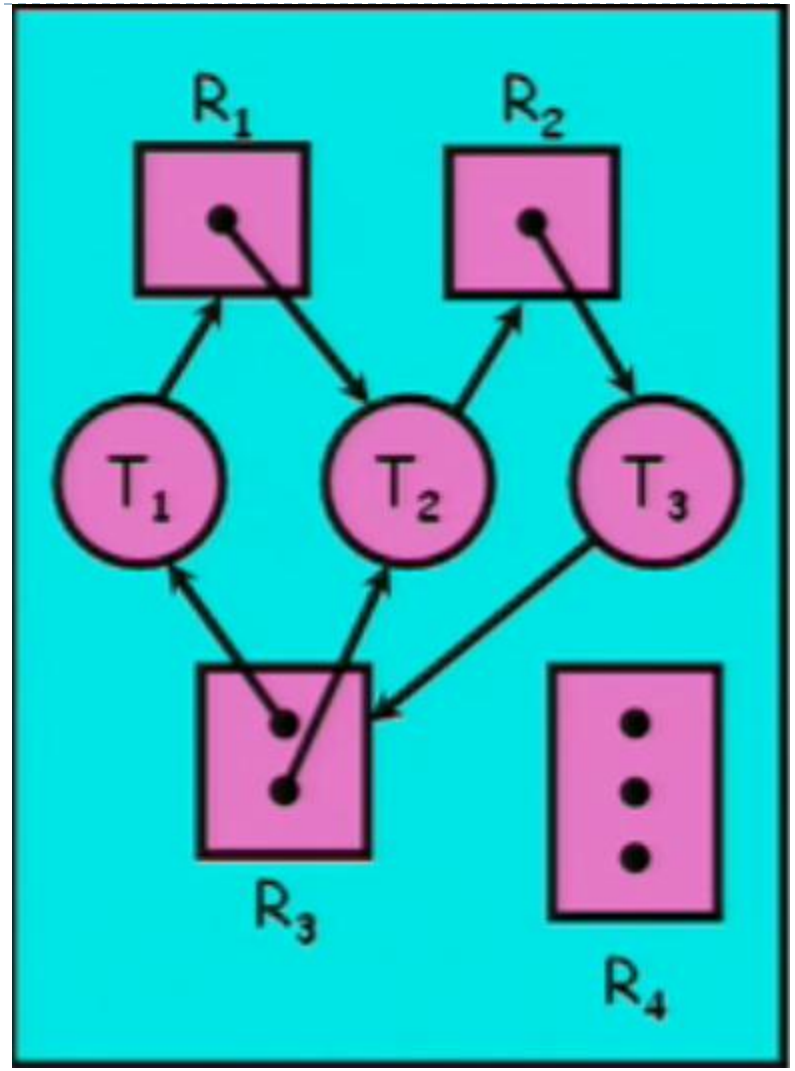
- Threads {T1, T2 … Tn}
- Recursos {R1, R2 … Rn}
- Grafo ‘espera por’
  - Solicitud de recurso T → R
  - Recurso en uso R → T



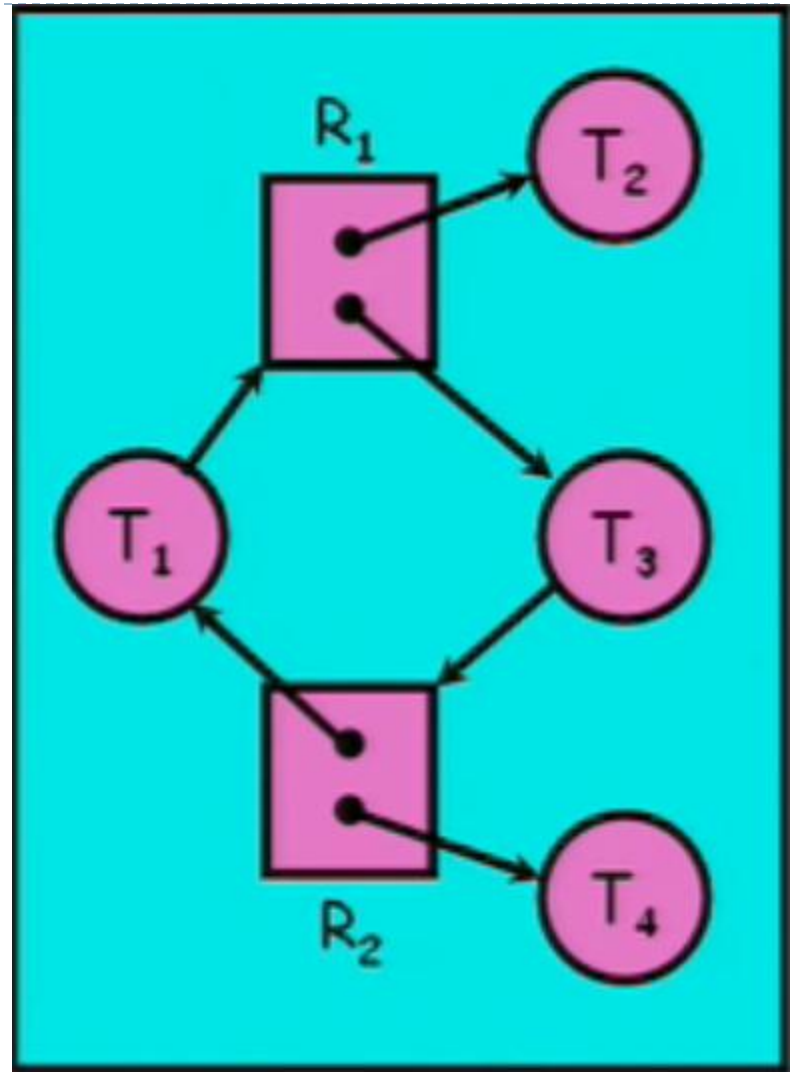
- Deadlock?
  - No
  - No hay ciclo (Circular wait)



- Deadlock?
  - Si
  - Ciclo implica deadlock?



- Deadlock?
  - No
  - T2 o T4 puede terminar
  - Si grafo no tiene ciclo
    - No hay deadlock
  - Si grafo tiene ciclo
    - Una instancia por recurso
      - Si hay deadlock
    - Varias instancias por recurso
      - Posiblemente hay deadlock



PREVENCIÓN DE DEADLOCKS

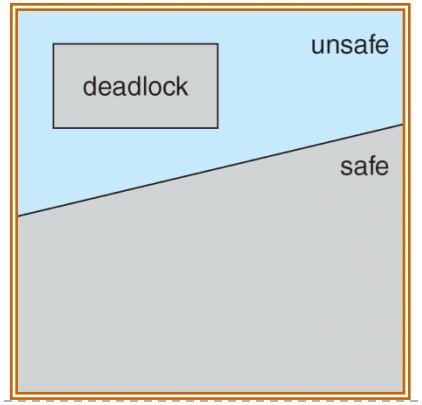
- Evitar que una de las cuatro condiciones no se cumplan
  - Mutual exclusion
    - Recursos compartidos como archivos en modo lectura
    - No es posible evitar (Depende del tipo de recurso)
  - Hold and wait
    - Solicitar recursos antes de ejecución
      - Mala utilización de recursos (Tiempo alocado sin uso)
    - Thread puede solicitar recursos si no tiene ninguno asignado
      - Restrictivo
      - Starvation

- No preemption
  - Generalmente recursos son no preemptive
  - Recursos preemptive
    - CPU. Fácil guardar su estado y asignar a otro thread (Context switch)
    - Memoria
- Circular wait
  - Ordenar recursos
    - F(Disco) = 5. F(Impresora) = 12
  - Cada thread debe asignar recursos en orden incremental
  - Prueba por contradicción
    - {T1, T2 … Tn}. Ti espera por Ri (en uso por Ti+1)
    - F(R0) < F(R1) <…..< F(Rn) < … < F(R0) lo cual es imposible F(R0)
  - Responsabilidad de los desarrolladores

EVITAR DEADLOCKS

- Prevención deadlocks
  - Baja utilización de recursos
  - Throughput reducido
- Evitar deadlocks
  - Dinámicamente examinar el estado de asignación de recursos para evitar que se de una espera circular
  - Cada proceso declara el máximo de cada recurso que utilizará

- Estado seguro
  - Existe secuencia {T1, T2 … Tn} tal que para cada Ti, las solicitudes de recursos pueden ser exitosas ya sea por
    - Recursos disponibles
    - Recursos que estén siendo utilizados por Tj, tal que j < i
  - De lo contrario el sistema está en un estado inseguro
  - Un estado inseguro puede ser un deadlock
  - Evitar entrar en un estado inseguro



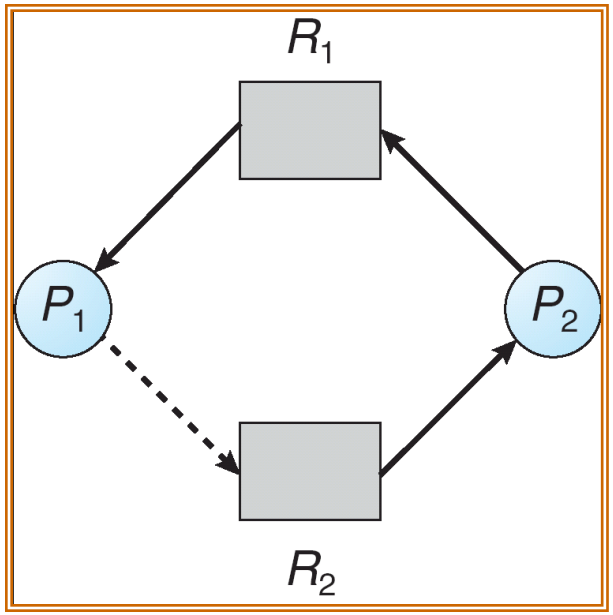
- En t0, el sistema está en un estado seguro.
  - Secuencia T1, T0, T2

Recurso R (12 instancias)	Máximo	Actual
T0	10	5
T1	4	2
T2	9	2

- En t1, T2 solicita un R más
  - Sistema se encuentra en estado inseguro
  - 2 instancias libres, que se pueden asignar a:
    - T0: No tendrá el máximo y no podrá terminar
    - T2: No tendrá el máximo y no podrá terminar
    - T1: Puede terminar. Libera 4 instancias
      - Si se asignan a T0 o a T2, ninguna podrá terminar.
      - Deadlock

Recurso R (12 instancias)	Máximo	Actual
T0	10	5
T1	4	2
T2	9	3

- Utilizar resource allocation graph para recursos con una sola instancia
- Evitar ciclos
  - Al asignar R2 a P1, se forma un ciclo (deadlock)



- Banker's Algorithm
  - Múltiples instancias de recursos
  - Cada proceso debe declarar el número máximo de cada recurso
  - Estructuras de datos
    - Available[]. Available[j] indica el # instancias de Rj disponibles
    - Max[][]. Max[i][j]. Numero maximo de instancias de Rj de Ti
    - Allocation[][]. Allocation[i][j]. Numero de instancias de Rj asignadas a Ti
    - Need[][]. Need[i][j]. Numero de instancias de Rj que necesita Ti

Safety Algorithm

1. Work = Available. Finish[i] = false. Para cada i
2. Encontrar i tal que
    1. Finish[i] == false
    2. Need[i] <= work
    3. De lo contrario ir a 4
3. Work = work + allocation[i]

   Finish[i] = true
4. If finish[i] == true para todos los i, Estado seguro

   De lo contrario, Ti está en deadlock

- Resource Request algorithm
  - T1 solicita los recursos en Request[i]

1. If request[i] <= need[i]. Ir a 2. De lo contrario error “Thread Solicita más recursos de los que declaró”
2. If request[i] <= available. Ir a 3. De lo contrario Ti debe esperar
3. Alocar recursos y revisar si se llegó a un estado seguro

available = available – request[i]
allocation[i] = allocation[i] + request[i]
need[i] = need[i] – request[i]

Probar Safety Algorithm

- Ejemplo
  - Threads {T0… T4}
  - Recursos
    - A (10), B (5), C(7)
  - En T0:

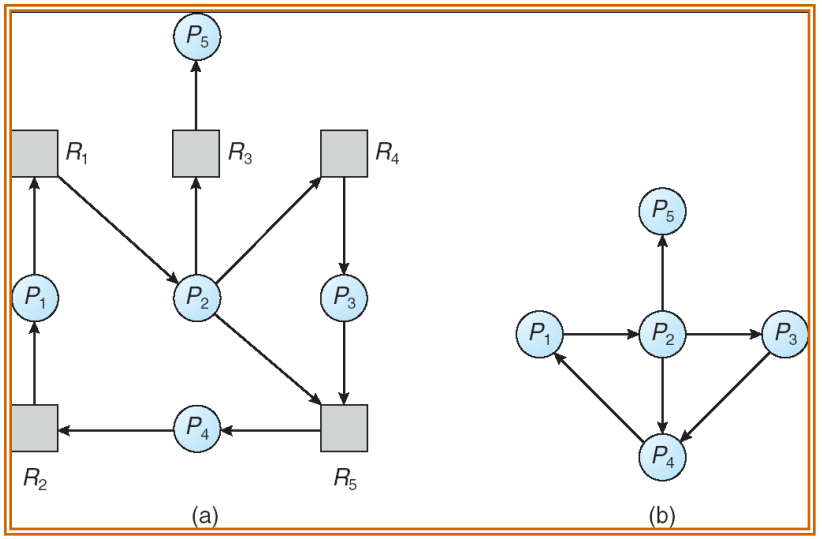
	Allocation	*Max*	*Need*	*Available*
	A B C	*A B C*	*A B C*	*A B C*
T_0	0 1 0	7 5 3	7 4 3	3 3 2
T_1	2 0 0	3 2 2	1 2 2	
T_2	3 0 2	9 0 2	6 0 0	
T_3	2 1 1	2 2 2	0 1 1	
T_4	0 0 2	4 3 3	4 3 1	

  - Estado Seguro
    - T_1, T_3, T_4, T_2, T_0

- T_1 Request (1,0,2)
- T_4 Request (3,3,0)
- T_0 Request (0,2,0)

DETECCIÓN DE DEADLOCKS

- Sistema debe proveer:
  - Algoritmo para detectar deadlock
  - Algoritmo para recuperarse del deadlock
- Una instancia por recurso
  - Resource allocation graph
  - Convertir a wait-for-graph
  - Ciclo implica deadlock
- Múltiples instancias
  - Banker's algorithm
  - Safety algorithm



- Overhead para detectar deadlock
  - ¿Cuándo ejecutar algoritmo?
  - Deadlocks se producen al solicitar recursos y la thread debe esperar
    - Esta solicitud ocacionó el deadlock
  - Ejecutar algoritmo en cada solicitud puede ser costoso
    - Ejecutar algoritmo cada período de tiempo

RECUPERACIÓN

- Solución manual (Notificar al usuario)
- Solución automática
  - Terminar procesos
    - Terminar todas las threads en deadlock. Garantizamos romper deadlock
    - Terminar una thread a la vez
      - Evaluar algoritmo de detección de deadlock
      - Criterios para seleccionar threads a terminar
        - Prioridad
        - Tiempo de ejecución
        - Tipo de recursos utilizados
        - Tipo de thread (interactivo o batch)
        - Terminar procesos que puedan ejecutar rollback

- Desasignar recursos
  - Seleccionar thread
  - Rollback si es posible
  - Starvation
    - Garantizar que no siempre se desasignen recursos del mismo thread

IGNORAR DEADLOCKS

- Poca probabilidad de ocurrencia
- Evitar overhead para prevenir/evitar/detectar deadlocks
- Utilizado por la mayoría de S.O.

RESUMEN

- Deadlock
  - Mutual exclusion
  - Hold and wait
  - No preemption
  - Circular wait
- Starvation vs. Deadlocks
- Métodos para manejar deadlocks
  - Prevención
    - Evitar que se cumpla alguna condición
  - Evitar deadlocks
  - Detectar deadlocks
    - Recuperarse
  - Ignorar deadlocks