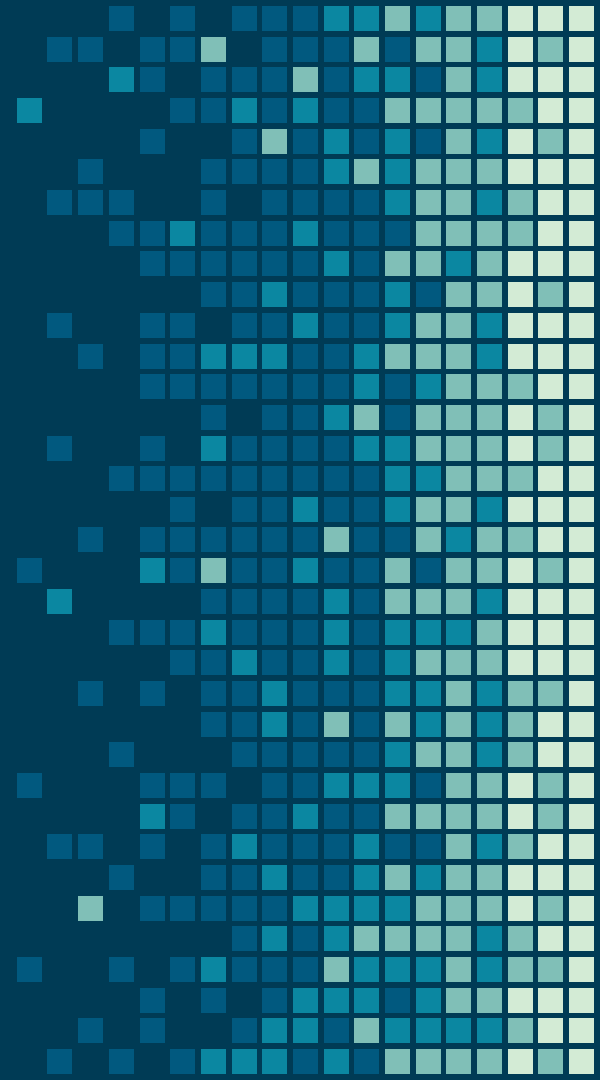

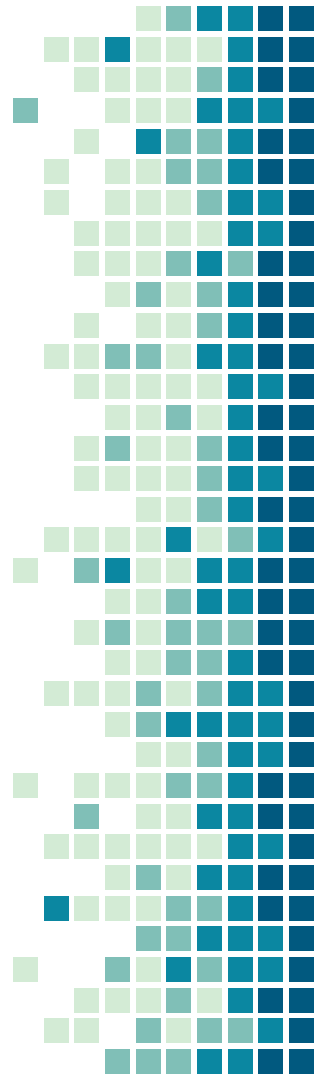


Deadlock

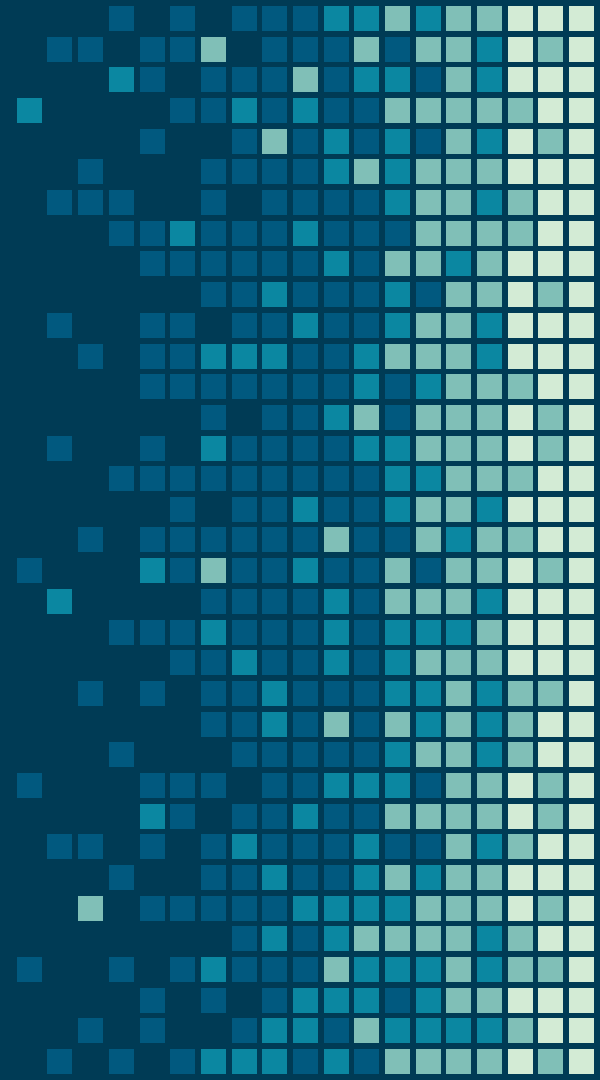


Agenda

- 
- **Introducción**
 - **Interbloqueo**
 - **Detección de un interbloqueo**
 - **Evasión de un interbloqueo**
 - **Prevención de interbloqueo**



Introducción



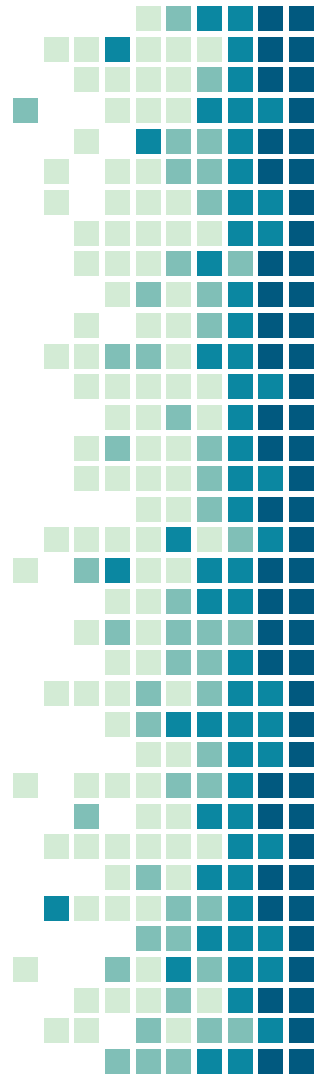


¿Cómo es que la concurrencia de procesos es posible?

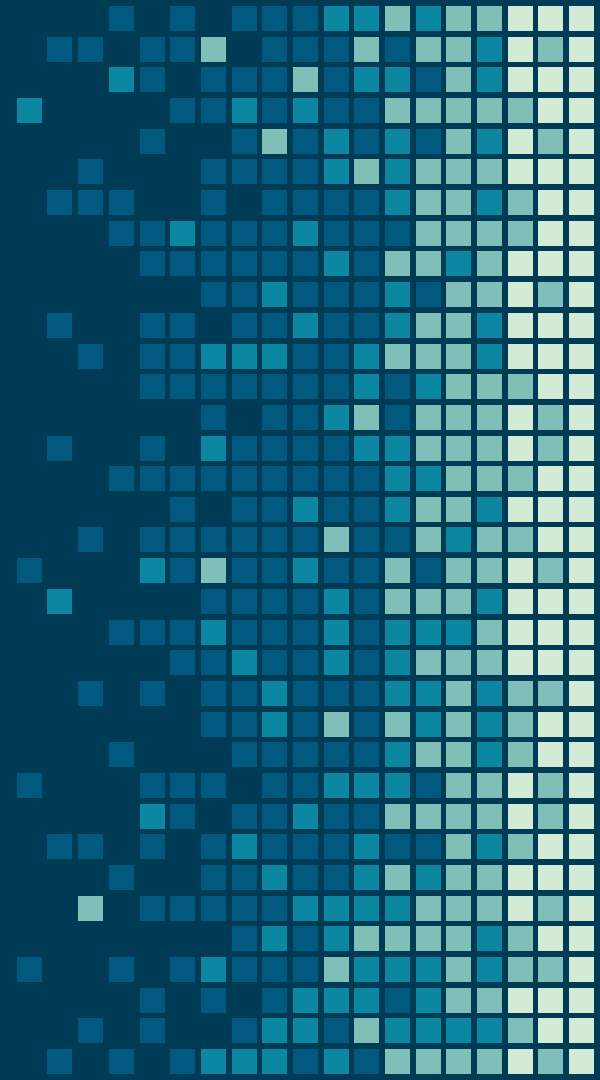
¿Qué se puede asegurar, hasta ahora, sobre el paralelismo de procesos?

¿Cómo ayuda los semáforos a evitar conflicto entre procesos?


¿Si se tiene dos procesos independientes y dos procesadores duplicados se tendría el mismo problema de concurrencia ?

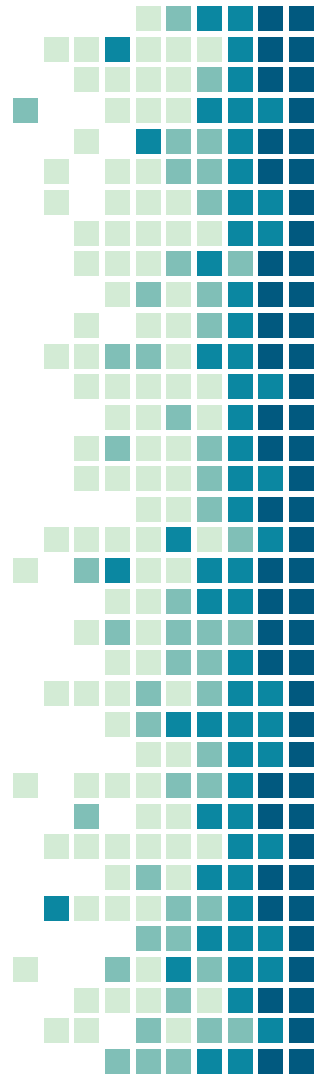


Interbloqueo

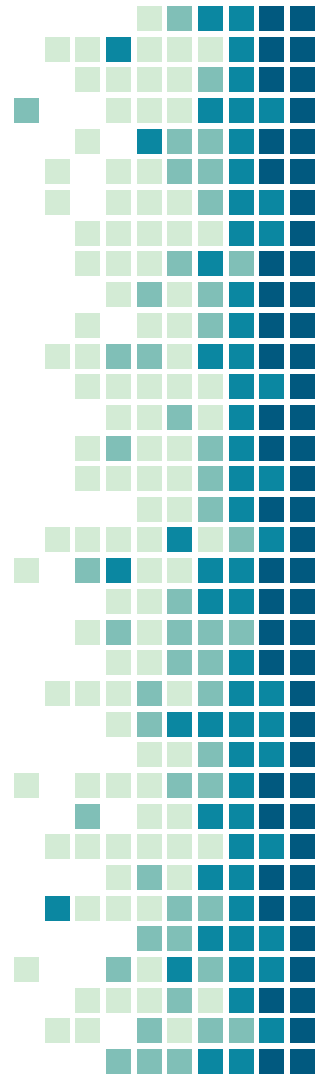


Escenario

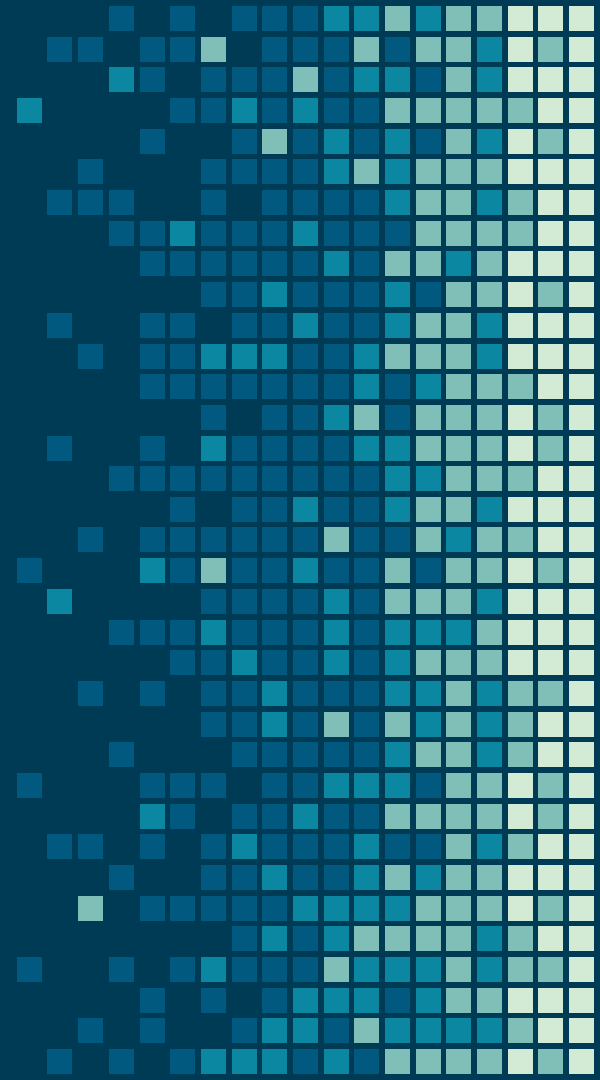
- 
- ❏ Suponga que hay dos procesos que requieren imprimir y grabar en disco información.
 - ❏ El proceso A pide la impresora y se le concede, mientras que el proceso B pide el grabador y también se le concede.
 - ❏ El proceso A o B termina, necesita el recurso que el otro posee, lo que hace que se produzca una espera hasta que se libere.
 - ❏ A esto se le llama interbloqueo.



Ingenieros CE hambrientos.



Recursos

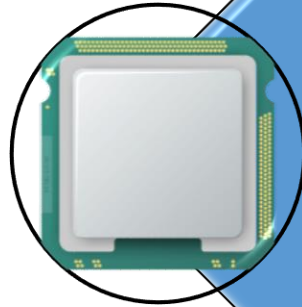


Recursos

- Los interbloqueos pueden ocurrir cuando a los procesos se les otorga acceso exclusivo a los dispositivos, registros, archivos entre otros. Cabe destacar que un recurso no es sólo Hardware.



Recursos apropiativo



Se refiere a un recurso al que se le puede quitar el proceso que lo posee sin efectos dañinos para el sistema.

Un ejemplo sería el procesador, el cual se puede otorgar a otro proceso sin que el actual haya terminado.

Recursos no apropiativos



- Un recurso no apropiativo es uno que no se puede quitar a su propietario actual sin hacer que el cómputo falle.
- Si un proceso ha iniciado a imprimir un documento y se trata de quitar el recurso para darlo a otro proceso, sucede un error ya que la impresión no se completa de la manera correcta.

Secuencia de eventos para utilizar un recurso



Solicitar el
recurso.



Utilizar el
recurso.



Liberar el
recurso.



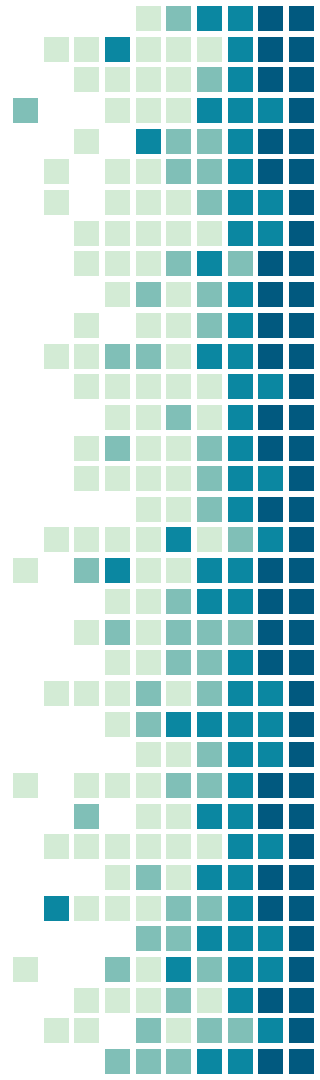
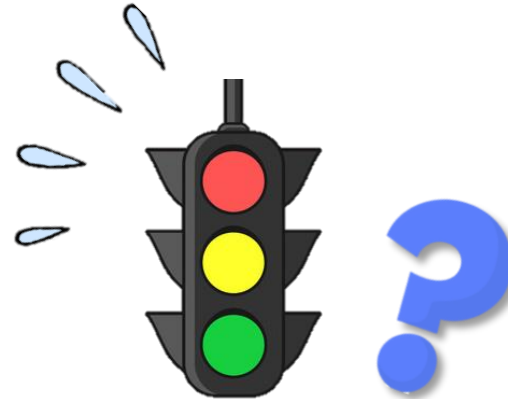
Si el recurso no
está disponible
cuando se solicita,
el proceso
solicitante se ve
obligado a
esperar.

¿Qué ocurre cuando a un
proceso se le niega el uso del
recurso?

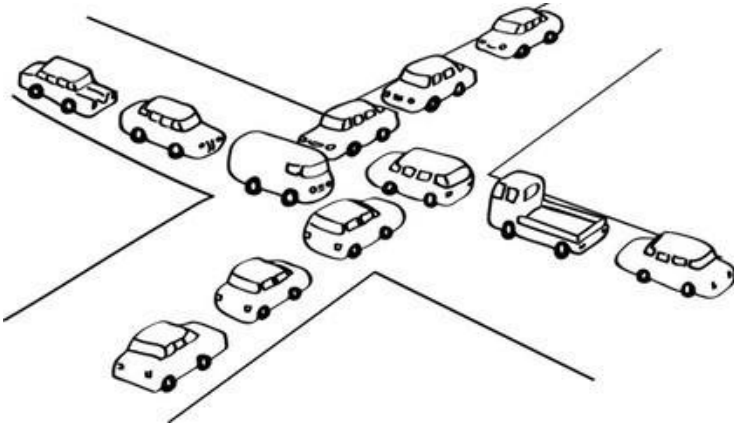


Adquisición de un recurso

- Para ciertos recursos, es responsabilidad de los usuarios administrar la adquisición de los mismos.
- Una posible manera es con famosos semáforos.
- ¿Nos aseguramos de eliminar el problema con los semáforos?



¿Qué es un interbloqueo formalmente?



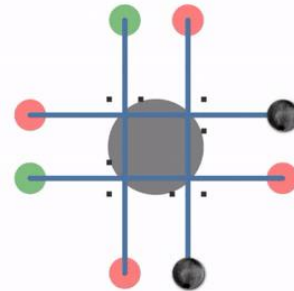
Un conjunto de procesos se encuentran en un interbloqueo si cada proceso en el conjunto está esperando un evento que sólo puede ser ocasionado por otro proceso en el mismo conjunto.

¿Cuál es el problema de un interbloqueo?



Debido a que todos los procesos están en espera, ninguno producirá algún evento para despertar a otro proceso, lo que implica una espera para siempre.

El evento que se necesita es la liberación de un recurso, generalmente.



¿Cómo se podría hacer la adquisición de un recurso?



- Ceder el control a usuario para que se encargue del control del mismo.
- Dejar esta tarea al sistema operativo sería ineficiente. ¿Por qué?
- Entonces, ¿ Como puede controlar el acceso un programador a los recursos y con ello adquirirlos?

Semáforos

—
Semaforo recurso_1

```
Void proceso_A(void) {  
    down(&recurso_1);  
    usar_recurso_1();  
    up(&recurso_1)  
}
```

—
Semaforo recurso_1
Semaforo recurso_2

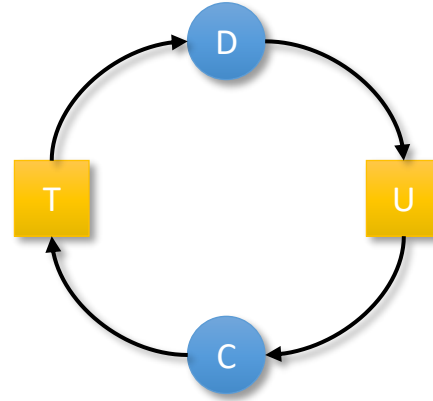
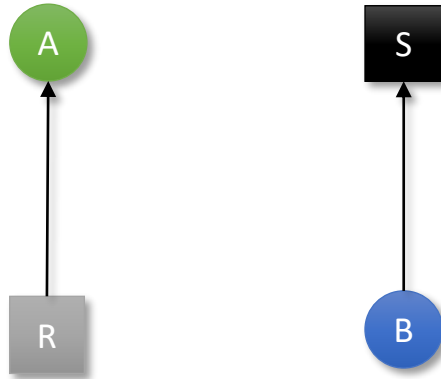
```
Void proceso_A(void) {  
    down(&recurso_1);  
    down(&recurso_2),  
    usar_ambos_recurso();  
    up(&recurso_2);  
    up(&recurso_1);  
}
```

Condiciones de Coffman para un interbloqueo

- 💡 Condición de exclusión mutua (que no se presente exclusión mutua).
- 💡 Condición de contención y espera.
- 💡 Condición no apropiativa
- 💡 Condición de espera circular.







Notación

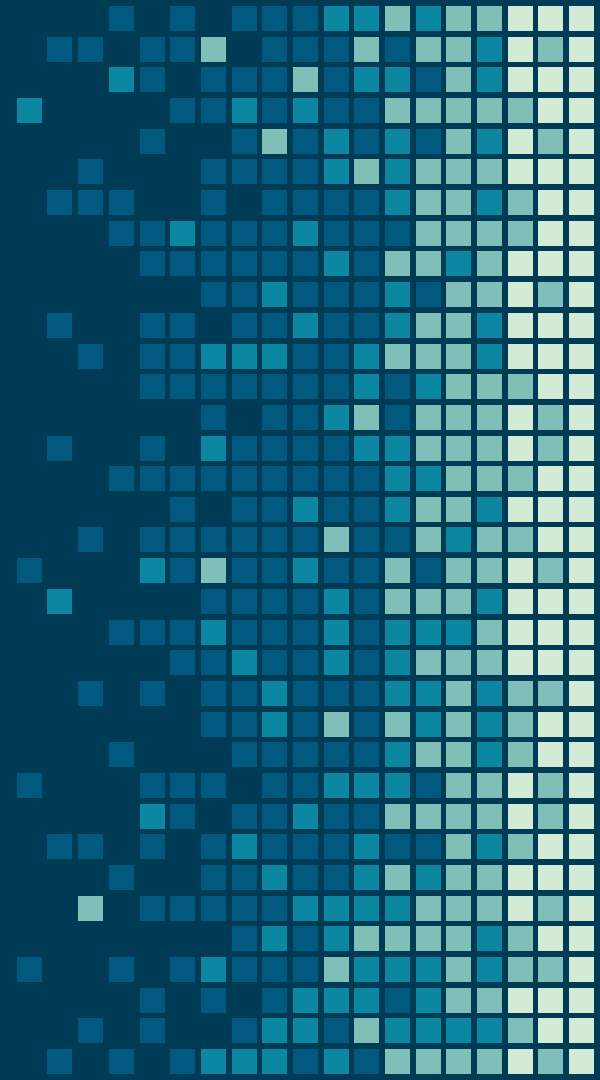


- El recurso R es asignado al proceso A, el proceso B espera por el recurso S .

¿Cómo lidiar con los interbloqueos?

-  Sólo ignorar el problema.
-  Detección y recuperación.
-  Evitarlos en forma dinámica mediante la asignación cuidadosa de los recursos.
-  Prevención, evite alguna de las cuatro condiciones de Coffman.

Ignorar el
problema



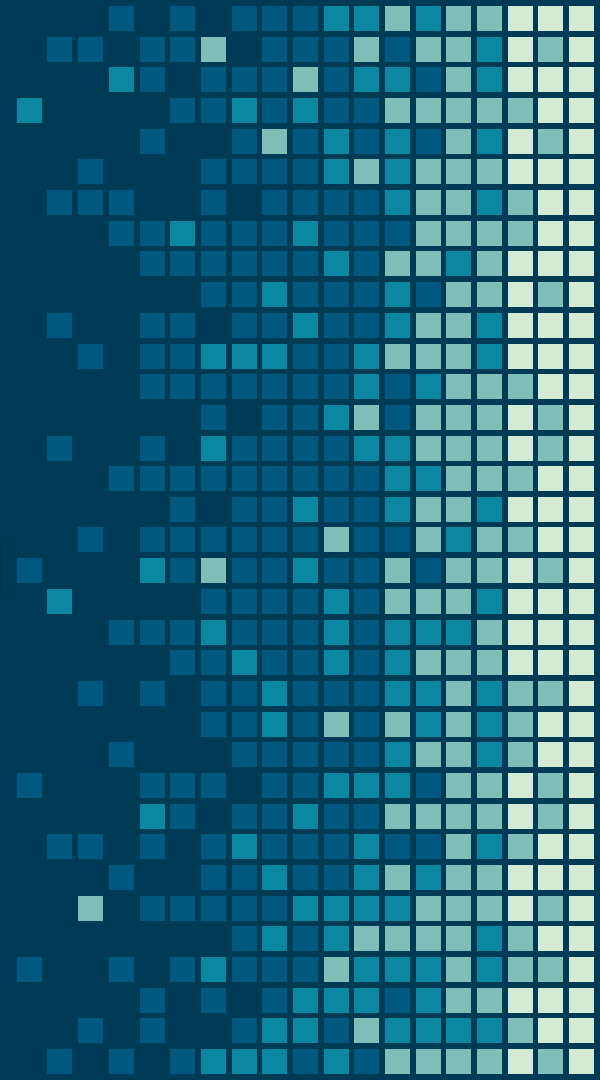
El algoritmo de la avestruz.

- Es el método más simple.
- Meta la cabeza en la tierra y haga como si no hubiera problema.
- Matemáticos VS Ingenieros.
- Consiste en ser optimista

¿Y qué es ser optimista?



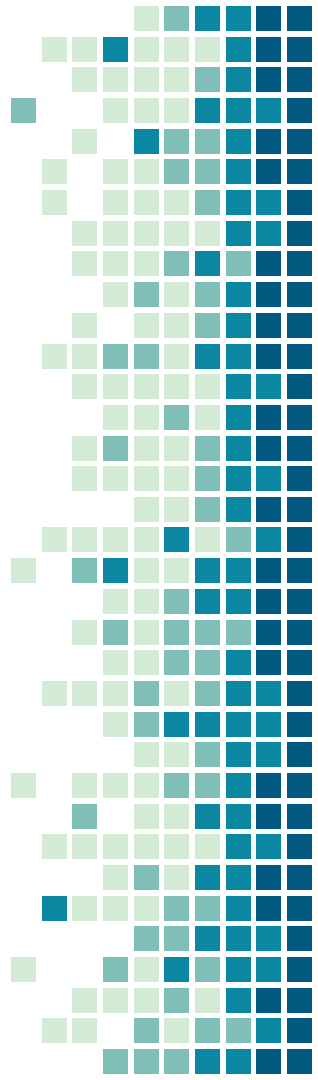
Detección y recuperación de un interbloqueo



Detección y recuperación



- No se trata de evitar los interbloqueos.
- La idea fundamental es identificarlos cuando ocurran y luego realizar alguna acción para recuperarse.
- Existen algunas maneras de recuperarse de un interbloqueo.

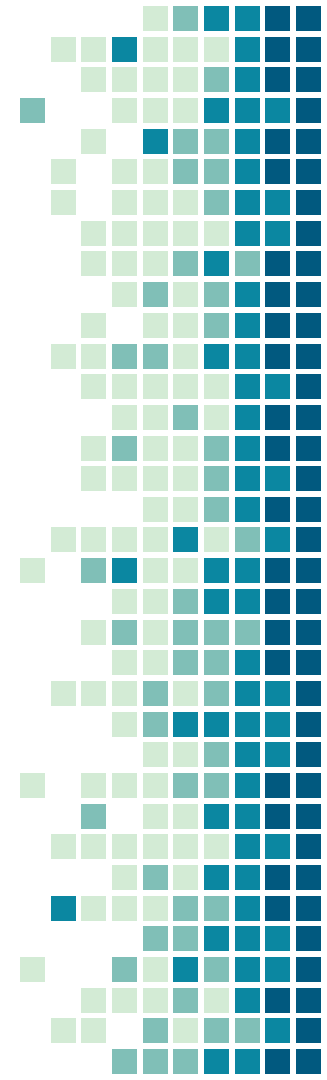


Detección y recuperación

- Se construye un grafo de procesos y recursos.

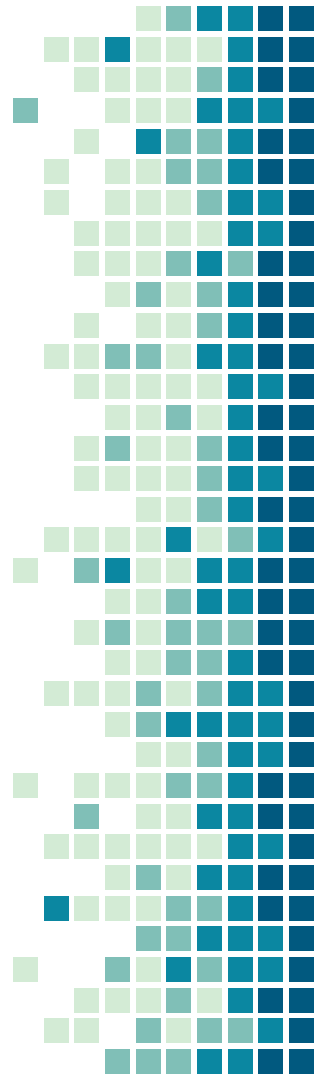
- Si el grafo contiene uno o más ciclos, significa que existe un interbloqueo.

- Cualquier proceso que forme parte del ciclo se considera que está en interbloqueo.

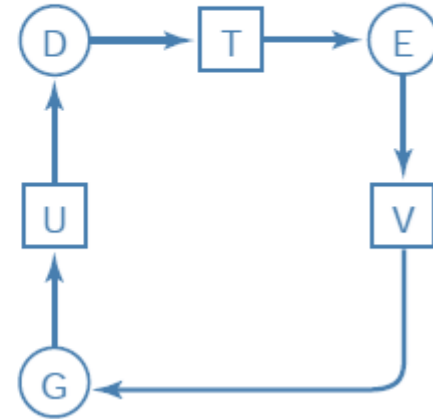
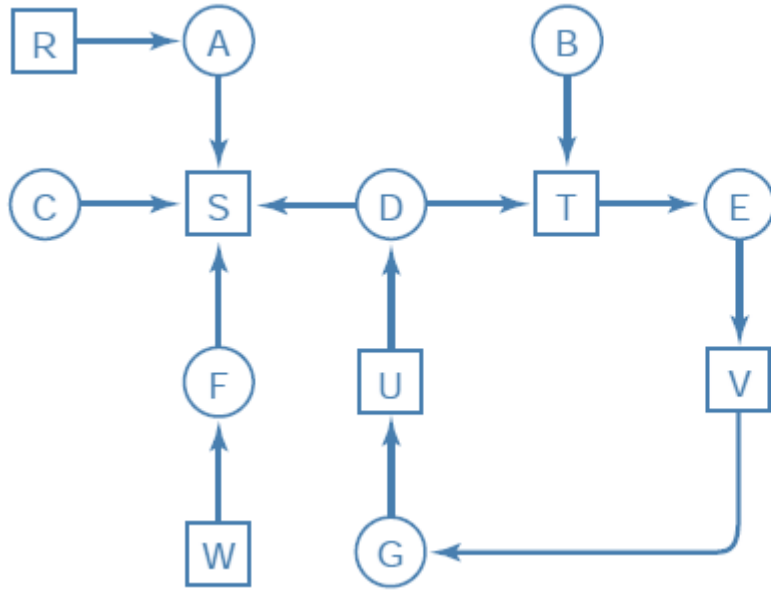


Ejemplo

- ✓ El proceso A contiene a R y quiere a S.
- ✓ El proceso B no contiene ningún recurso pero quiere a T.
- ✓ El proceso C no contiene ningún recurso, pero quiere a S.
- ✓ El proceso D contiene a U y quiere a S y a T.
- ✓ El proceso E contiene a T y quiere a V.
- ✓ El proceso F contiene a W y quiere a S.
- ✓ El proceso G contiene a V y quiere a U.



Solución



¿Cómo se puede implementar en la práctica?

Detección del interbloqueo con varios recursos de cada tipo.

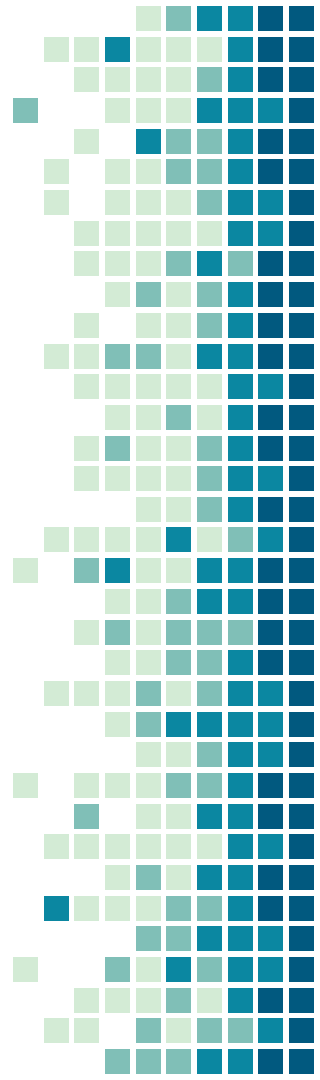
Algoritmo basados en matrices.

A es el vector de recursos disponibles.

E es el vector de recursos existentes.

C es la matriz de asignaciones actuales.

R es la matriz de peticiones.




Detección del interbloqueo con varios recursos de cada tipo.

Recursos disponibles
($A_1, A_2, A_3, \dots, A_m$)

Recursos en existencia
($E_1, E_2, E_3, \dots, E_m$)


Matriz de solicitudes

Matriz de asignaciones actuales



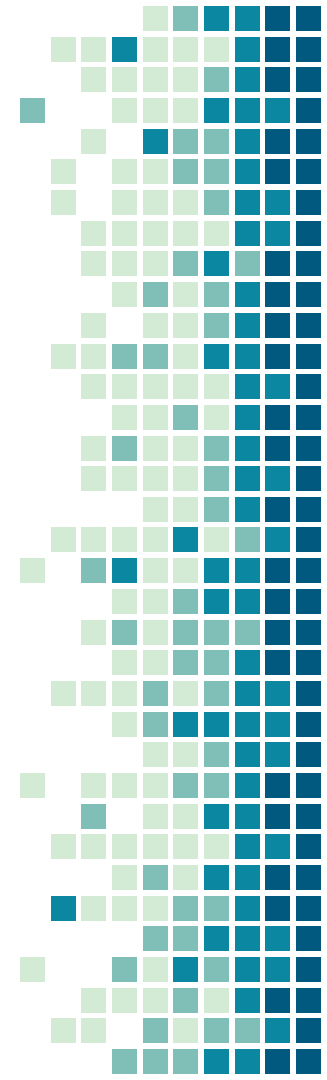
| | | | | |
|----------|----------|----------|---------|----------|
| R_{11} | R_{12} | R_{13} | \dots | R_{1m} |
| R_{21} | R_{22} | R_{23} | \dots | R_{2m} |
| \vdots | \vdots | \vdots | | \vdots |
| R_{n1} | R_{n2} | R_{n3} | \dots | R_{nm} |

La fila 2 es lo que necesita
el proceso 2



| | | | | |
|----------|----------|----------|---------|----------|
| C_{11} | C_{12} | C_{13} | \dots | C_{1m} |
| C_{21} | C_{22} | C_{23} | \dots | C_{2m} |
| \vdots | \vdots | \vdots | | \vdots |
| C_{n1} | C_{n2} | C_{n3} | \dots | C_{nm} |

La fila n es la asignación
actual al proceso n



Ejemplo. ¿Hay interbloqueo?

Unidades de cinta
Trazadores
Escáneres
Unidades de CD-ROM

$$E = (4 \quad 2 \quad 3 \quad 1)$$

Unidades de cinta
Trazadores
Escáneres
Unidades de CD-ROM

$$A = (2 \quad 1 \quad 0 \quad 0)$$

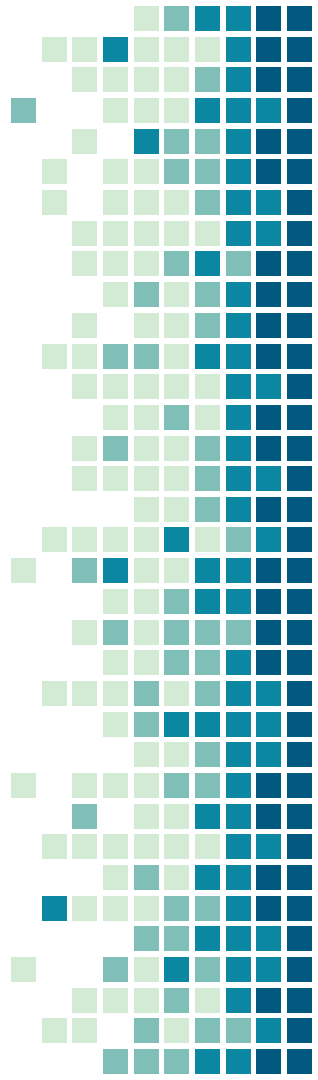
Matriz de asignaciones actuales

$$C = \begin{bmatrix} 0 & 0 & 1 & 0 \\ 2 & 0 & 0 & 1 \\ 0 & 1 & 2 & 0 \end{bmatrix}$$

Matriz de peticiones

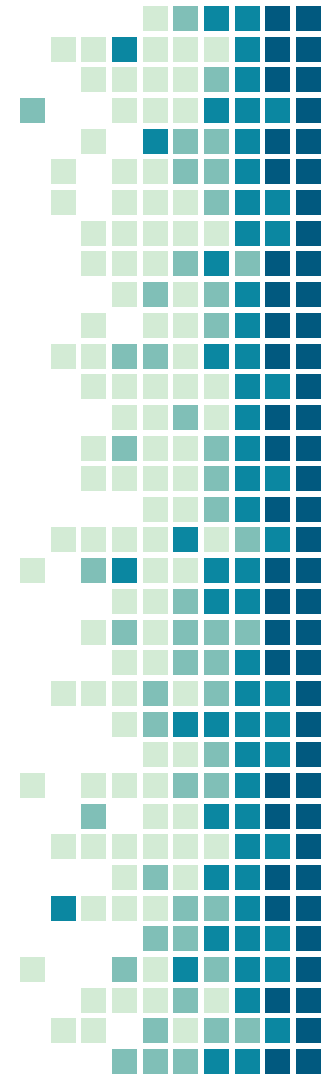
$$R = \begin{bmatrix} 2 & 0 & 0 & 1 \\ 1 & 0 & 1 & 0 \\ 2 & 1 & 0 & 0 \end{bmatrix}$$

No existe interbloqueo



Recuperación de interbloqueo.

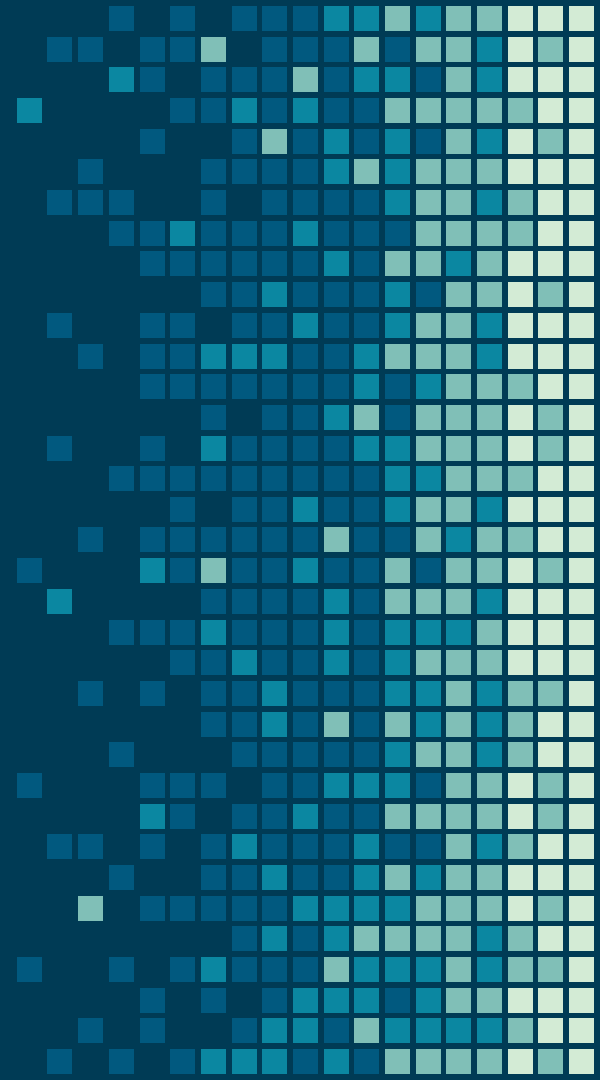
Una vez que se ha detectado los interbloqueos, se debe de buscar alguna forma de recuperar la situación normal.



Recuperación de interbloqueo.

- ✓ Las técnicas de recuperación no son muy finas y por lo tanto actualmente casi no se utilizan, sin embargo, solucionan el problema.
- ✓ Una está relacionada con la intervención manual en el procesamiento con lotes.
- ✓ Puntos de recuperación de manera periódica.
- ✓ Eliminación de proceso para que los otros sigan ejecutando.

Evasión de
interbloqueo.



¿Cómo evitar los interbloqueos?

Cuando se hace la solicitud de un recurso el SO debe ser capaz de decidir si lo asigna o no.

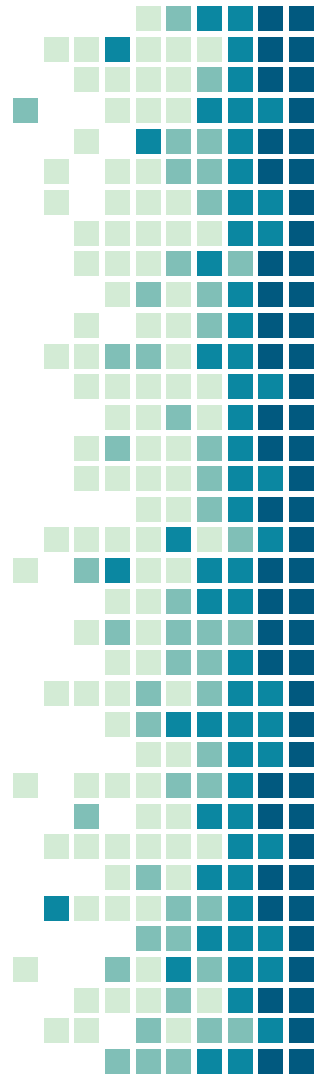


- ☒ La asignación dependerá de si es seguro asignar el recurso a un proceso en específico.
- ☒ ¿Habrá manera de siempre evitar un interbloqueo?

Implicaciones

La mayoría de los algoritmos se basan en el concepto de estados seguros.

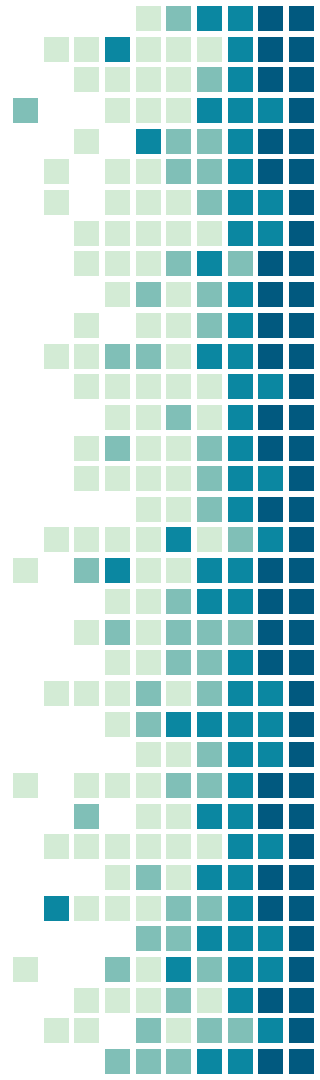
El cual es un método que evalúa el estado actual, tomando en cuenta condiciones muy específicas para el futuro (asignación de recursos).



Estados Seguros



- ➔ Se dice que un estado es seguro si existe cierto orden de programación en el que se puede ejecutar cada proceso hasta completarse.
- ➔ Esto implica que se debe tomar en cuenta si en algún momento los procesos solicitaran de manera repentina el número máximo de recursos.



Ejemplo de estado seguro.

| | | | | | | | | | | | | | | |
|------------|---|---|------------|---|---|------------|---|---|------------|---|---|------------|---|---|
| Tiene Máx. | | | Tiene Máx. | | | Tiene Máx. | | | Tiene Máx. | | | Tiene Máx. | | |
| A | 3 | 9 | A | 3 | 9 | A | 3 | 9 | A | 3 | 9 | A | 3 | 9 |
| B | 2 | 4 | B | 4 | 4 | B | 0 | – | B | 0 | – | B | 0 | – |
| C | 2 | 7 | C | 2 | 7 | C | 2 | 7 | C | 7 | 7 | C | 0 | – |
| Libres: 3 | | | Libres: 1 | | | Libres: 5 | | | Libres: 0 | | | Libres: 7 | | |

Suponga que hay 10 instancias de recursos

Ejemplo de estado no seguro.

A solicita un recurso y es otorgado.

Tiene Máx.

| | | |
|---|---|---|
| A | 3 | 9 |
| B | 2 | 4 |
| C | 2 | 7 |

Libres: 3

Tiene Máx.

| | | |
|---|---|---|
| A | 4 | 9 |
| B | 2 | 4 |
| C | 2 | 7 |

Libres: 2

Tiene Máx.

| | | |
|---|---|---|
| A | 4 | 9 |
| B | 4 | 4 |
| C | 2 | 7 |

Libres: 0

Tiene Máx.

| | | |
|---|---|---|
| A | 4 | 9 |
| B | – | – |
| C | 2 | 7 |

Libres: 4

En conclusión la petición de A debe ser denegada. Porque no produce un estado seguro.

El algoritmo del banquero para un recurso en general.

Creado por Dijkstra.

Tiene Máx.

| | | |
|---|---|---|
| A | 0 | 6 |
| B | 0 | 5 |
| C | 0 | 4 |
| D | 0 | 7 |

Libres: 10

Tiene Máx.

| | | |
|---|---|---|
| A | 1 | 6 |
| B | 1 | 5 |
| C | 2 | 4 |
| D | 4 | 7 |

Libres: 2

Tiene Máx.

| | | |
|---|---|---|
| A | 1 | 6 |
| B | 2 | 5 |
| C | 2 | 4 |
| D | 4 | 7 |

Libres: 1

Lo que hace el algoritmo es verificar si al otorgar la petición de un recurso se produce un estado inseguro.

Algoritmo del banquero para varios recursos.

Es una generalización del anterior, pero ahora se debe de mantener dos matrices con el fin de controlar el uso de recursos

| | Proceso | Unidades de cinta | Trazadores | Impresoras | Unidades de CD-ROMs |
|---|---------|-------------------|------------|------------|---------------------|
| A | 3 | 0 | 1 | 1 | |
| B | 0 | 1 | 0 | 0 | |
| C | 1 | 1 | 1 | 0 | |
| D | 1 | 1 | 0 | 1 | |
| E | 0 | 0 | 0 | 0 | |

Recursos asignados

| | Proceso | Unidades de cinta | Trazadores | Impresoras | Unidades de CD-ROMs |
|---|---------|-------------------|------------|------------|---------------------|
| A | 1 | 1 | 0 | 0 | |
| B | 0 | 1 | 1 | 2 | |
| C | 3 | 1 | 0 | 0 | |
| D | 0 | 0 | 1 | 0 | |
| E | 2 | 1 | 1 | 0 | |

Recursos que aún se necesitan

E = (6342)
P = (5322)
A = (1020)

Algoritmo de Banquero

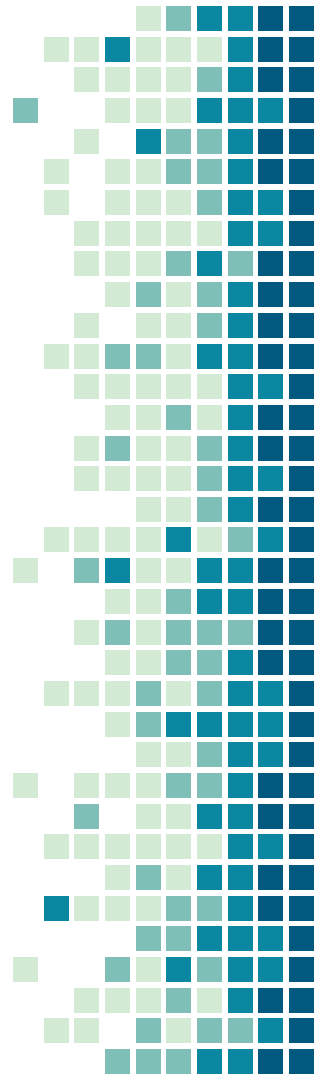
- Se publicó por primera vez en 1965.
- Se realizaron muchas publicaciones sobre esta genialidad de algoritmo.

Pero, hay un pequeño problema. ¿Cuál es?

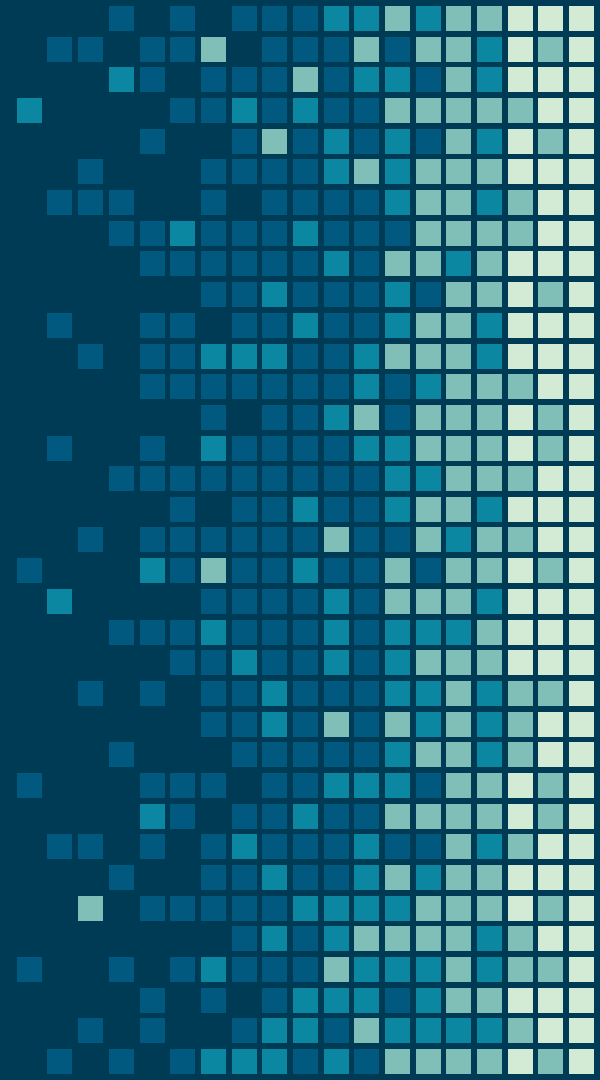
...



Es poco útil en los sistemas amplios porque, muchos SO no saben de antemano los recursos que se usan.

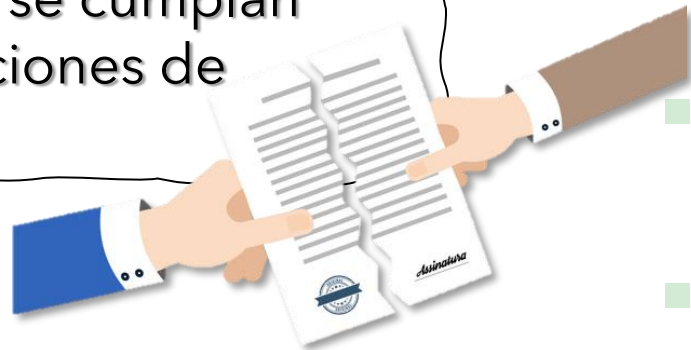


Prevención de interbloqueos.



¿Cómo prevenir interbloqueos?

- Como ya se mencionó evitar el interbloqueo en SO es casi imposible.
- Entonces cómo hacen los SO para lidiar con los interbloqueos entre los procesos.
- La solución es impedir que se cumplan alguna de las cuatro condiciones de Coffman.



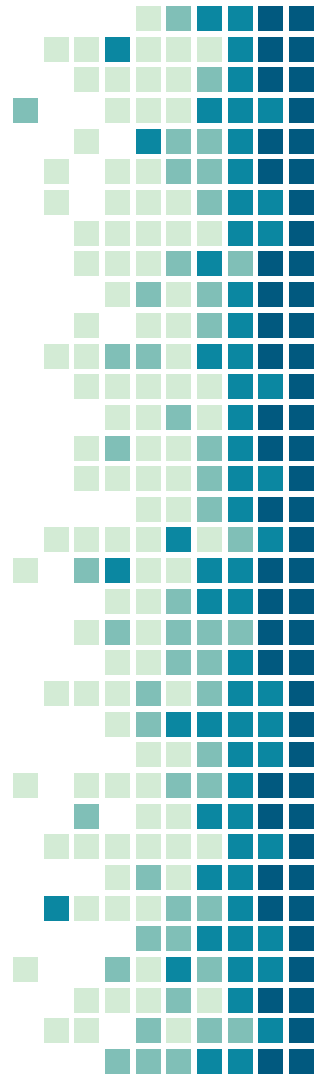
¿Cómo atacar la condición exclusión mutua?

¿Qué ocurre cuando se imprimen 200 documentos de manera simultánea?

La idea fundamental es utilizar una estructura intermedia, que sirva de "controlador", con el fin de que otro proceso pueda tomar los datos desde dicha estructura.

- Por ejemplo el funcionamiento de la impresora.

Principio: Evitar asignar un recurso cuando no sea necesario y tratar de que sea la menor cantidad de procesos.



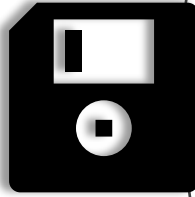
¿Cómo atacar la condición de contención y espera?

- Lo que hay que lograr es que el proceso no espere por más recursos.
- Una solución es que los procesos soliciten todos sus recursos antes de la ejecución.



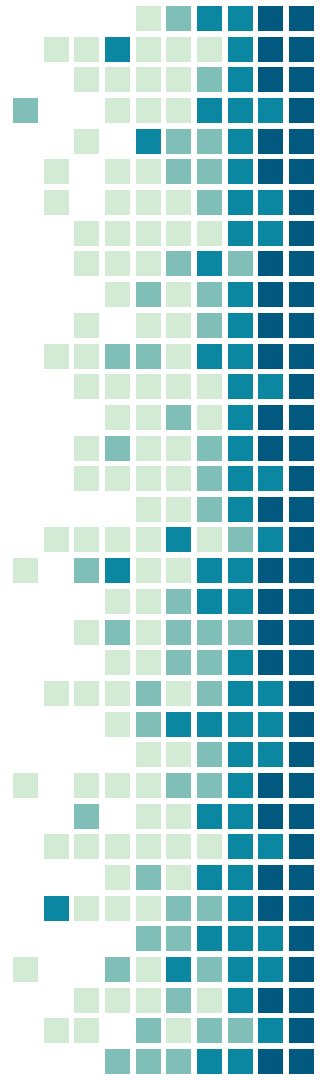
¿Cuáles son los dos problemas?

¿Cómo atacar la condición de no apropiativa?



Virtualizar (se crea una imagen de los datos en disco) recursos y con ello evitar esta situación. Esto con el fin de recuperar la información posteriormente.

Sin embargo, no todos los recursos se pueden virtualizar, como por ejemplo las tablas de los sistemas operativos.

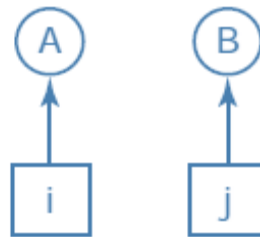


¿Cómo atacar la condición de lista circular?

💡 Tener una regla que diga que un proceso tiene derecho sólo a un recurso en cualquier momento. Si necesita un segundo recurso, debe liberar el primero. ¿Qué ocurre cuando se copian datos? Es inaceptable esta regla.

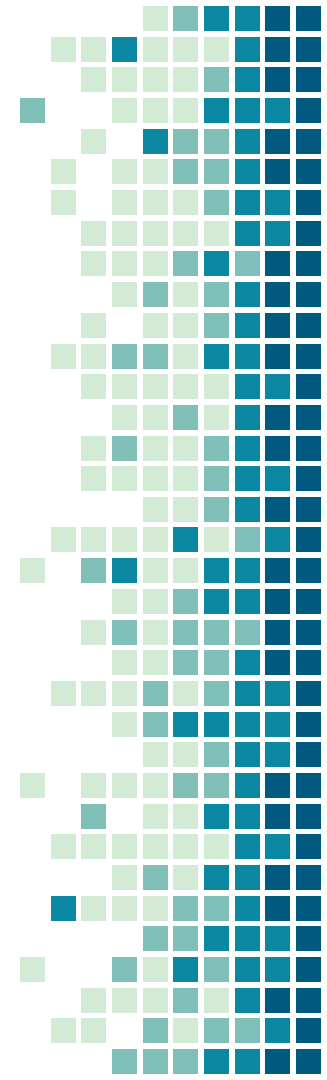
💡 Crear un orden global y ahora la regla es que un proceso no puede solicitar uno menor.

1. Fotocomponedora
2. Escáner
3. Trazador
4. Unidad de cinta
5. Unidad de CD-ROM



Resumen

| Condición | Método |
|---------------------|---|
| Exclusión mutua | Poner todo en la cola de impresión |
| Contención y espera | Solicitar todos los recursos al principio |
| No apropiativa | Quitar los recursos |
| Espera circular | Ordenar los recursos en forma numérica |



Referencias

-▶ Tanenbaum, A. S. (2015). *Sistemas operativos modernos*. Pearson Educación.
-▶ Stallings, W. (1997). *Sistemas operativos*. Martin Iturbide.
-▶ King, S. T., Dunlap, G. W., & Chen, P. M. (2003, June). Operating System Support for Virtual Machines. In *USENIX Annual Technical Conference, General Track* (pp. 71-84).



¿Preguntas?

Realizado por: Jason Leitón Jiménez.

Tecnológico de Costa Rica

Ingeniería en Computadores

2023

TEC