INTERBLOQUEOS

INTERBLOQUEOS

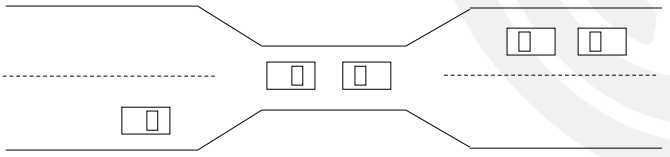
- Modelo de Sistema
- Caracterización de Interbloqueos
- Métodos para el Manejo de Interbloqueos
- Prevención de Interbloqueos
- Evasión de Interbloqueos
- Detección de Interbloqueos
- Recuperación de Interbloqueos

EL PROBLEMA DE INTERBLOQUEO

- Un conjunto de procesos bloqueados, cada uno contiene recursos y espera para adquirir otro recurso mantenido por otro proceso en el conjunto.
- Ejemplo
  - Un sistema tiene 2 discos.
  - P_1 y P_2 cada uno tiene un disco y necesita otro.
- Ejemplo
  - semáforos *A* y *B*, inicializados a 1

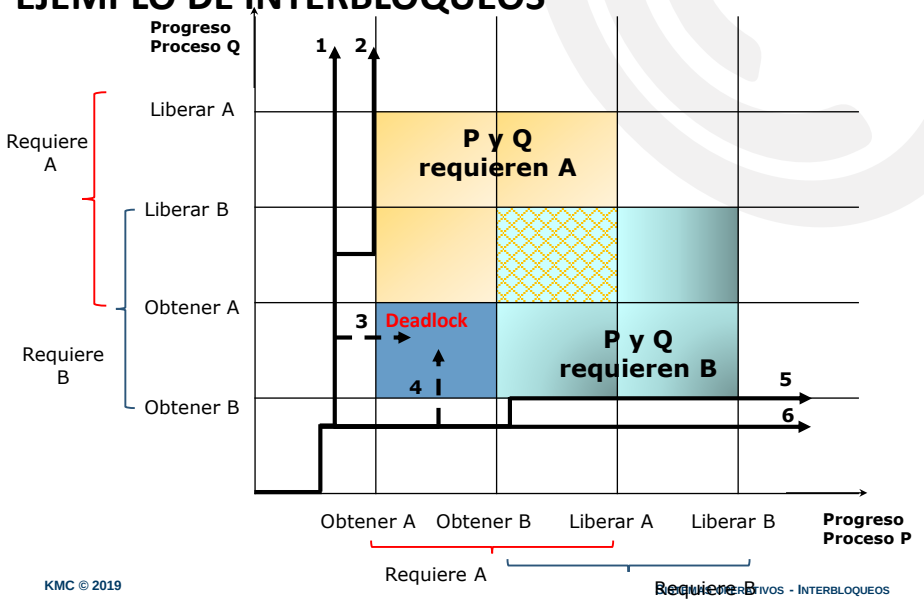
P_0	P_1
wait (A);	*wait(B)*
wait (B);	*wait(A)*

EJEMPLO DEL CRUCE EN EL PUENTE



- Tránsito en una sola dirección.
- Cada sección de un puente puede ser vista como un recurso.
- Si un interbloqueo ocurre, puede ser resuelto si cada auto retrocede (se apropia de recursos y *rollback*).
- Varios autos pueden retroceder si un interbloqueo occurre.
- Es posible la inanición.

EJEMPLO DE INTERBLOQUEOS



MODELO DE SISTEMA

- Tipos de Recursos R_1, R_2, . . ., R_m
  ciclos CPU, espacio de memoria, dispositivos E/S
- Cada recurso tipo R_i tiene W_i instancias.
- Cada proceso utiliza un recurso como sigue:
  - **requerimiento**
  - **uso**
  - **liberalización**

Uso correcto

CARACTERIZACIÓN DE INTERBLOQUEOS

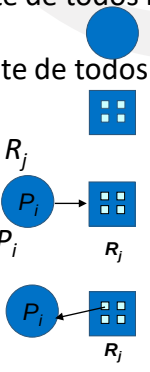
El Interbloqueo puede alcanzarse si se cumplen las cuatro condiciones simultáneamente.

- **Exclusión Mutua :** solo un proceso a la vez puede usar un recurso.
- **Retener y Esperar:** un proceso mantiene al menos un recurso y está esperando adquirir recursos adicionales tenidos por otros procesos.
- **No Apropiación:** Un recurso puede ser liberado solo voluntariamente por el proceso que lo tiene, después que el proceso ha completado su tarea.
- **Espera Circular:** existe un conjunto $\{P_0, P_1, \ldots, P_n\}$ de procesos esperando tal que P_0 está esperando por un recurso que es retenido por P_1, P_1 está esperando por un recurso que es retenido por P_2, …, P_{n-1} está esperando por un recurso que es retenido por P_n, y P_n está esperando por un recurso que es retenido por P_0.

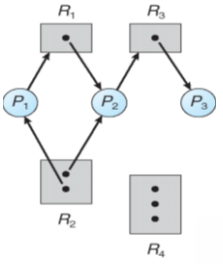
GRAFO DE ALOCACIÓN DE RECURSOS

Un conjunto de vértices *V* y un conjunto de lados *E*.

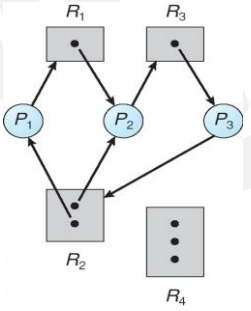
- V está particionado en dos tipos:
  - $P = \{P_1, P_2, \ldots, P_n\}$, el conjunto consistente de todos los procesos en el sistema . Proceso
  - $R = \{R_1, R_2, \ldots, R_m\}$, el conjunto consistente de todos los recursos tipo en el sistema. Recurso
- lado de requerimiento – lado dirigido $P_i \rightarrow R_j$
- lado de asignamiento – lado dirigido $R_j \rightarrow P_i$



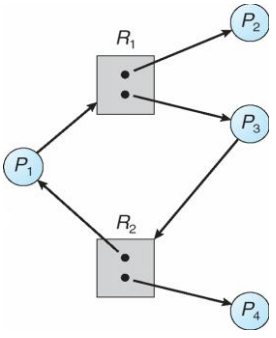
EJEMPLOS: GRAFO DE ALOCACIÓN DE RECURSOS



a) Grafo sin ciclo



b) Grafo con interbloqueo



c) Grafo con ciclo sin interbloqueo

CUESTIONES BÁSICAS

- Si un grafo no contiene ciclos ⇒ no hay interbloqueo.
- Si un grafo contiene un ciclo ⇒
  - Si hay una sola instancia por tipo de recurso, entonces hay interbloqueo.
  - Si hay varias instancias por tipo de recurso, hay posibilidad de interbloqueo.

MÉTODOS PARA MANEJO DE INTERBLOQUEOS

- Asegure que el sistema no entrará *nunca* estado de interbloqueo.
- Permitir al sistema entrar en un estado de interbloqueo y luego recuperarse.
- Ignore el problema y pretenda que el interbloqueo nunca ocurrió en el sistema; usado en la mayoría de los sistemas operativos incluído UNIX, Linux, Windows.

ESTRATEGIAS PARA MANEJO DE INTERBLOQUEOS

Las estrategias utilizadas para atacar el problema de interbloqueos se establecen en tres niveles de acción: antes que suceda, cuando los procesos están corriendo o con hechos consumados.

Estas estrategias son:

- Prevención
- Evasión
- Detección

PREVENCIÓN DE INTERBLOQUEOS

Restringir los modos en que se pueden hacer los requerimientos

- **Exclusión Mutua** – no requerido para recursos compartidos; debe mantenerse para recursos no compartidos.
- **Mantener y Esperar** – debe garantizar que siempre que un proceso requiera un recurso no mantiene otros.
- **No Apropiación** - Si un proceso que mantiene algunos recursos requiere otro recurso, no le puede ser inmediatamente alocado, entonces todos los recursos mantenidos son liberados.
- **Espera Circular** – impone un orden total de todos los tipos de recursos, y requiere que cada proceso requiera recursos en un orden creciente de enumeración.

EVASIÓN DE INTERBLOQUEOS

Requiere que el sistema tenga disponible alguna información adicional *a priori* .

- El modelo más simple y útil requiere que cada proceso declare el *máximo número* de recursos de cada tipo que puede necesitar.
- El algoritmo de evasión de interbloqueos dinámicamente examina el estado de alocación de recursos para asegurar que no puede haber una condición de espera circular.
- El *estado* de alocación de recursos está definido por el número de recursos disponibles y alocados, y las máximas demandas de los procesos.

ESTADO SEGURO

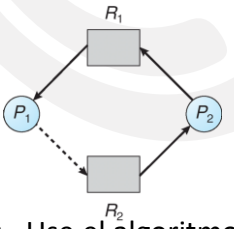
- Cuando un proceso requiere un recurso disponible, el algoritmo debe decidir si su alocación inmediata deja al sistema en un estado seguro.
- El sistema está en un **estado seguro** si existe una secuencia segura de ejecución de todos los procesos.
- La secuencia $<P_1, P_2, \ldots, P_n>$ es segura si por cada P_i, los recursos que P_i puede aún requerir pueden ser satisfechos por recursos disponibles corrientes mas los recursos mantenidos por todos los P_j, con $j<i$.
  - Si los recursos que P_i necesita no están inmediatamente disponibles, entonces P_i puede esperar hasta que todos los P_j hayan finalizado.
  - Cuando P_j finaliza, P_i obtiene los recursos necesitados, ejecuta, retorna los recursos alocados, y termina.
  - Cuando P_i termina, P_{i+1} puede obtener los recursos que necesita, y así sucesivamente.

CUESTIONES BÁSICAS

- Si un sistema está en un estado seguro ⇒ no hay interbloqueo.
- Si un sistema está en un estado inseguro ⇒ posibilidad de interbloqueo.
- Evasión ⇒ asegura que el sistema *nunca* va a entrar en un estado inseguro.

ALGORITMOS DE EVASIÓN

- Si sólo hay una instancia de un tipo de recurso. Use un grafo de alocación de recursos.



- Para múltiple instancias de un tipo de recursos. Use el algoritmo del banquero.

ALGORITMOS DE EVASIÓN

- Ejemplo simple de evasión:

Considerar 12 recursos iguales y 3 procesos con las siguientes necesidades máximas y alocación corriente.

	Max	Aloc	
P_0	10	5	
P_1	4	2	
P_2	9	2	probar P_2 con alocación=3

Sec. segura <P_1,P_0,P_2>

Sec. insegura <P_1,.....>

ALGORITMO DEL BANQUERO

- Múltiples instancias.
- Cada proceso debe reclamar a priori el máximo uso.
- Cuando un proceso requiere un recurso puede tener que esperar.
- Cuando un proceso obtiene todos sus recursos debe retornarlos en una cantidad finita de tiempo.

ESTRUCTURA DE DATOS DEL ALGORITMO DEL BANQUERO

Sea *n* = número de procesos, y *m* = número de tipos de recursos.

- *Disponible:* Vector de longitud *m*. Si disponible[*j*] = *k*, hay *k* instancias del tipo de recurso R_j disponible.
- *Max:* matriz *n x m*. Si *Max*[*i,j*] = *k*, entonces el proceso P_i puede requerir a lo sumo *k* instancias del recurso de tipo R_j.
- *Alocación:* matriz *n x m*. Si Alocación[*i,j*] = *k* entonces P_i tiene alocadas *k* instancias de R_j.
- *Necesidad:* matriz *n x m*. Si *Necesidad*[*i,j*] = *k*, entonces P_i puede necesitar *k* instancias más de R_j para completar su tarea.

Necesidad[*i,j*] = *Max*[*i,j*] – *Alocación*[*i,j*].

ALGORITMO DE SEGURIDAD

1. Sean *Trab y Final* vectores de longitud *m* y *n*, respectivamente. Se inicializan:

   Trab := *Disponible*

   Final[*i*] = *falso para i* - 1,3, …, *n.*

2. Encuentre un *i* tal que cumplan:

   (a) *Final*[*i*] = *falso*

   (b) $Necesidad_i \leq Trab$

   Si tal *i no* existe, vaya al paso 4.

3. *Trab* := *Trab* + $Alocación_i$
   Final[*i*] := *verdadero*
   vaya al paso 2.

4. Si *Final*[*i*] = verdadero para todo *i*, entonces el sistema está en un estado seguro.

ALGORITMO DE REQUERIMIENTO DE RECURSOS PARA EL PROCESO P_I

$Request_i$ = vector de requerimento para el proceso P_i.
Si $Request_i[j] = k$ entonces el proceso P_i quiere *k* instancias del tipo de recurso R_j.

1. Si $Request_i \leq Necesidad_i$ vaya al paso 2. Sino **condición de error**, dado que el proceso ha excedido su máximo.
2. Si $Request_i \leq Disponible$, vaya al paso 3. Sino P_i debe esperar, dado que los recursos no están disponibles.
3. Se simula alocar los recursos requeridos P_i modificando el estado como sigue:

   $Disponible := Disponible - Request_i;$

   $Alocación_i := Alocación_i + Request_i;$

   $Necesidad_i := Necesidad_i - Request_i;$

   - *Si* ***seguro*** $\Rightarrow$ *los recursos son alocados a* P_i.
   - *Si* ***inseguro*** $\Rightarrow$ P_i *debe esperar, y es restaurado el viejo estado de alocación de recursos*

EJEMPLO DEL ALGORITMO DEL BANQUERO

- 5 procesos P_0 a P_4; 3 recursos tipo *A* (10 instancias), *B* (5 instancias), y *C* (7 instancias).
- Instantánea en el instante T_0:

	Alocación	*Max*	*Disponible*	***Necesidad***
	A B C	*A B C*	*A B C*	***A B C***
P_0	0 1 0	7 5 3	3 3 2	**7 4 3**
P_1	2 0 0	3 2 2		**1 2 2**
P_2	3 0 2	9 0 2		**6 0 0**
P_3	2 1 1	2 2 2		**0 1 1**
P_4	0 0 2	4 3 3		**4 3 1**

El sistema está en un estado seguro dado que la secuencia $< P_1, P_3, P_4, P_2, P_0>$ satisface los criterios de seguridad.

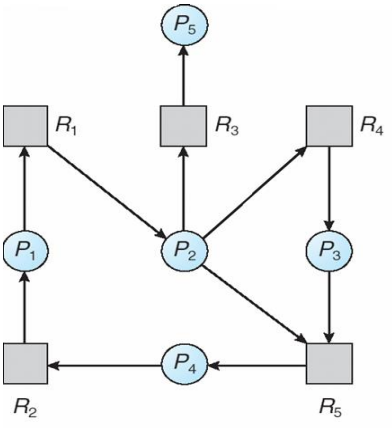
DETECCIÓN DE INTERBLOQUEOS

- Permite al sistema entrar en estado de interbloqueo
- Algoritmo de Detección
- Esquema de Recuperación

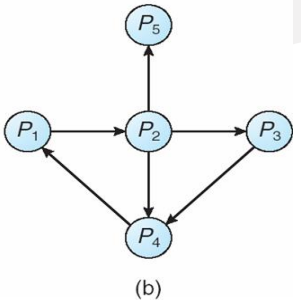
INSTANCIA SIMPLE DE CADA TIPO DE RECURSO

- Se mantiene un grafo *wait-for*
  - Los nodos son procesos.
  - $P_i \rightarrow P_j$ si P_i está esperando por P_j.
- Periódicamente se invoca un algoritmo que busca por ciclos en el grafo.
- Un algoritmo para detectar un ciclo en un grafo requiere un orden de n^2 operaciones, donde n es el número de vértices en el grafo.

GRAFO DE ALOCACIÓN DE RECURSOS Y GRAFO WAIT-FOR



Grafo de alocación de recursos



Grafo wait-for correspondiente

VARIAS INSTANCIAS DE UN TIPO DE RECURSO

- *Disponible:* Vector de longitud *m* indica el número de recursos disponibles de cada tipo.
- *Alocación:* Una matriz de *n x m* que define el número de recursos de cada tipo corrientemente alocados por cada proceso.
- *Request:* Una matriz de *n x m* matrix que indica el requerimiento corriente de cada proceso. Si *Request*[*i,j*] = *k*, entonces el proceso P_i está requiriendo *k* instancias más del recurso de tipo R_j.

ALGORITMO DE DETECCIÓN

1. Sean *Trab* y *Final* vectores de longitud *m* y *n*, respectivamente inicializados:
   (a) *Trab* = *Disponible*
   (b) Para *i* = 1,2, …, *n*, si $Alocación_i \neq 0$, entonces *Final*[i] := falso; sino *Final*[i] := verdadero.
2. Encuentre un índice *i* tal que:
   (a) *Final*[*i*] = *falso*
   (b) $Request_i \leq Trab$
   Si no existe tal *i*, vaya al paso 4.
3. *Trab* := *Trab* + $Alocación_i$
   Final[*i*] := *verdadero*
   vaya al paso 2.
4. Si *Final*[*i*] = falso, para algún *i*, $1 \leq i \leq n$, entonces el sistema está en un estado de interbloqueo. Es mas, si *Final*[*i*] = *falso*, entonces P_i está interbloqueado.

El algoritmo requiere un orden de O(m x n^2) operaciones para detectar si el sistema está en un estado de interbloqueo.

EJEMPLO DE ALGORITMO DE DETECCIÓN

- Cinco procesos P_0 a P_4; tres tipos de recursos A (7 instancias), *B* (2 instancias), y *C* (6 instancias).
- En el instante T_0:

	Alocación	*Request*	*Disponible*
	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 <P_0, P_2, P_3, P_1, P_4> resultará en

  Final[*i*] = verdadero para todo *i*.

USO DEL ALGORITMO DE DETECCIÓN

- Cuándo y qué frecuente debe invocarse depende de:
  - ¿Con qué ***frecuencia*** es probable que ocurra un interbloqueo?
  - ¿***Cuántos*** procesos serán afectados cuando ocurra un interbloqueo?
- Si el algoritmo de detección es invocado arbitraria-mente, puede haber muchos ciclos en el grafo de recursos y no se puede decir cual de los muchos procesos interbloqueados **causó** el interbloqueo.

RECUPERACIÓN DE INTERBLOQUEOS: TERMINACIÓN DE PROCESOS

- Aborto de todos los procesos interbloqueados.
- Aborto de un proceso a la vez hasta que el ciclo de interbloqueo haya sido eliminado.
- ¿En qué orden se elige abortar?
  - Prioridad del proceso.
  - Cuánto tiempo ha computado el proceso, y cuánto falta para completar.
  - Recursos que el proceso ha usado.
  - Recursos que el proceso necesita para completar.
  - Cuántos procesos necesitarán ser terminados.
  - ¿Es el proceso interactivo o batch?

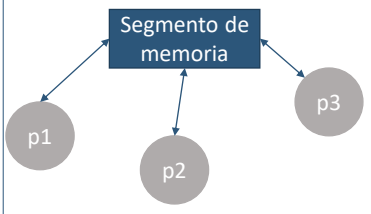
RECUPERACIÓN DE INTERBLOQUEOS: APROPIACIÓN DE RECURSOS

- Selección de una víctima – minimiza costos.
- Rollback – retorna a algún estado seguro, reinicia el proceso desde ese estado.
- Inanición – algunos procesos pueden ser elegidos siempre como víctimas, incluir un número de rollbacks en el factor de costo.

COMUNICACIÓN ENTRE PROCESOS

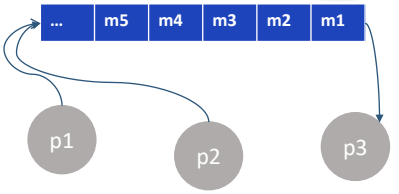
Memoria Compartida

- Segmento de memoria compartida



Mensajes

- Cola de mensajes



COMUNICACIÓN ENTRE PROCESOS: SEGMENTO DE MEMORIA

OPERACIONES

1.- Creación de un nuevo segmento de memoria compartida o acceder a uno existente.

Llamada al sistema: **shmget**

int shmget(key_t *key*, size_t *size*, int *shmflg);*

2.- Mapeado del segmento de memoria compartida al espacio de direcciones del proceso.

Llamada al sistema: **shmat**

char *shmat(int *shmid*, void **shmaddr*, int *shmflg*);

COMUNICACIÓN ENTRE PROCESOS: SEGMENTO DE MEMORIA

OPERACIONES

3.- Lectura o escritura. En esta zona se lee y se escribe como en cualquier dirección de memoria del proceso. Por ser una zona de memoria compartida, para realizar el acceso a esta zona hay que utilizar semáforos para obtener acceso exclusivo.

4.- Desenlace del segmento de memoria compartida por parte del proceso.

Llamada al sistema: **shmdt**

int shmdt(void *_shmaddr_)

Librerías:
#include <sys/ipc.h>
#include <sys/shm.h>

COMUNICACIÓN ENTRE PROCESOS: SEGMENTO DE MEMORIA: EJEMPLO

Proceso 1

```
struct info *ctrl;
// obtener un identificador
id = shmget(KEY, SEGSIZE, 0);
if (id <0){
   printf(" FALLO EL shmget \n");    ex
  }
//asociar la memoria a la estructura
ctrl = (struct info*) shmat(id,0,0);
if (ctrl <= (struct info *) (0)){
    printf(" Error en el shmat \n");   exit(2);
  }
// operar sobre memoria compartida
 if ( argc == 1 )
    strcpy( ctrl->mensaje, "\0" );
 else
    strcpy( ctrl->mensaje, argv[1]);
```

Proceso 2

```
struct info{
  char mensaje[255];
};

#define KEY   ((key_t) (1243))
#define SEGSIZE  sizeof (struct info)
//tamaño de la estructura
```

```
    printf(" Error fallo shmat \n");   exit(2);
  }
//valor inicial al segmento de memoria
compartida
strcpy(ctrl->mensaje, "Sistemas
Operativos");
```

COMUNICACIÓN ENTRE PROCESOS: COLAS DE MENSAJES

OPERACIONES

1.- Creación de la cola de mensajes.

Llamada al sistema: **msgget**

int msgget(key_t *key*, int *shmflg);

Librerías:
#include <sys/ipc.h>
#include <sys/msg.h>

2.- Envío de un mensaje.

Llamada al sistema: **msgsnd**

int msgsnd(int *msqid*, const void **msgp*, size_t *msgsz*, int *msgflg*);

3.- Recepción de un mensaje.

Llamada al sistema: **msgrcv**

int msgrcv(int *msqid*, const void **msgp*, size_t *msgsz*, long msgtyp, int *msgflg*);

COMUNICACIÓN ENTRE PROCESOS: COLAS DE MENSAJES

FORMATO DEL MENSAJE

```
struct mensaje{
        long tipo;
        int dato_1;
        int dato_2;
};
```

Primer campo del mensaje. Define el tipo de mensaje

COMUNICACIÓN ENTRE PROCESOS: COLAS DE MENSAJES: EJEMPLO

Emisor

```
.... inicialización de la cola....
idmsg = msgget(key, IPC_CREAT|0666);
mimensaje.tipo = 1;
mimensaje.dato_1 = 24;
mimensaje.dato_2 = 4;

longitud = sizeof(struct mensaje) -
sizeof(long);

if (msgsnd(idmsg, &mimensaje, longitud,
0) == -1)
{
printf("Error en la escritura mensaje\n");
exit(1);    }
```

Receptor

```
.... vinculación a la cola....
idmsg = msgget(key, 0666);

longitud = sizeof(struct mensaje) -
sizeof(long);

if (msgrcv(idmsg, &mimensaje, longitud,
0, 0) == -1)
{
printf("Error en la lectura mensaje\n");
exit(1);    }
```


Bibliografía:

- Silberschatz, A., Gagne G., y Galvin, P.B.; "*Operating System Concepts*", 7ma Edición 2009, 9na Edición 2012, 10ma Edición 2018.
- Stallings, W. *"Operating Systems: Internals and Design Principles"*, Prentice Hall, 6ta Edición 2009, 7ma Edición 2011, 9na Edición 2018.