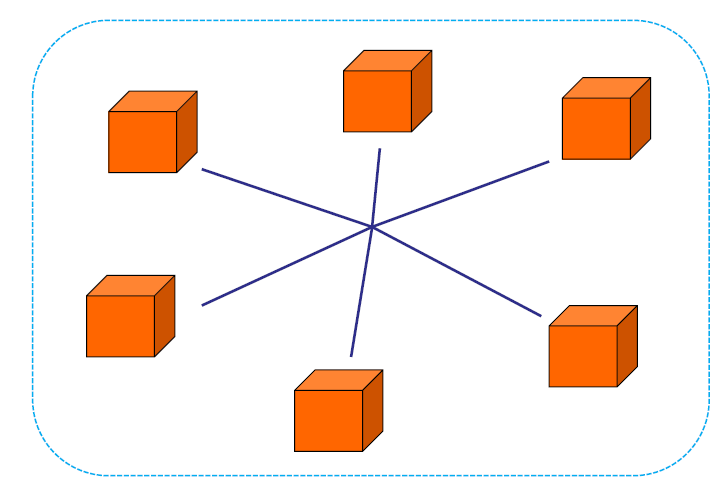
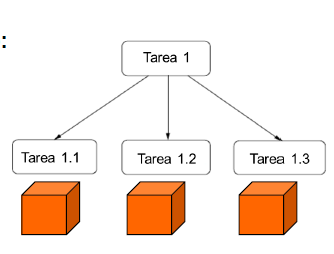
**Sistemas operativos distribuidos:**



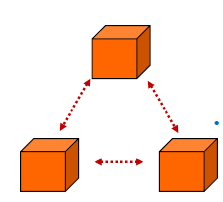
Se utiliza cuando tengo un conjunto de computadoras que se integran y hacen parecer a las aplicaciones del usuario y al usuario, que se tratan de un sistema único. Estos distintos nodos (sistema distribuido) hacen desaparecer la dualidad local/remoto.

Objetivos del sistema distribuido:

* Distribuir el trabajo (aprovecha el procesador y la memoria que tiene cada nodo del sistema)



* Compartir los recursos (aprovecha los recursos locales de cada nodo, si se ejecuta un proceso en otro nodo, también se pueda acceder)



* Logrando así:

1. Alto rendimiento
2. Alta escalabilidad
3. Alta disponibilidad

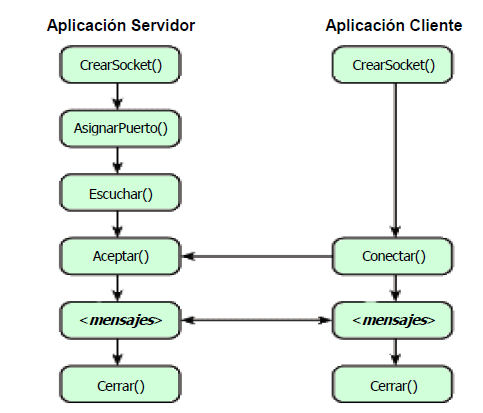
Cuestiones para implementar un sistema distribuido:

* ¿Cómo lograr la sincronización de procesos?
* ¿Cómo manejar el deadlock?
* ¿Cómo lograr un estado consistente?

¿Cómo compartir info entre procesos?:

El método será el intercambio de mensajes, con el protocolo tcp/ip por ej.

* Primitivas de comunicación:
* socktets=>



* mensajes=>

función enviar (sockets, datos)

función recibir (socket): datos (no tenga ningún dato para recibir, por ende, devuelve un vacío y también puede ser bloqueante)

todo esto mediante:

1. intercambio de mensajes confiables (tcp) vs no confiables (uvp)
2. intercambio de mensajes sincrónicos (hasta que no se envié todos los datos o no se hallan cargando todos los datos desde el canal de comunicaciones, la función de enviar recibir no funcionara) vs asincrónicos (puedo seguir ejecutando, a pesar a que no se ejecutaron los datos en el canal de comunicaciones)
3. configuración bloqueante (si envió un enviar bloqueante, hasta que el proceso que hace recibir no ejecuta, la función enviar queda bloqueada hasta que el otro proceso no haya recibido la info) vs no bloqueante (el enviar manda los datos y no se preocupa en confirmar si fueron recibidos o no) ahora en una función recibir no bloqueante, el proceso que lo ejecuta puede recibir datos o datos vacíos. (si el recibir fuera bloqueante, habrá datos recibidos o un error, pero jamás datos vacíos)

como ordenar los eventos de los procesos?:

* sincronización de la hora:

•Protocolo de Tiempo de Red (NTP)

-permite sincronizar las computadoras con hora utc de un reloj atómico.

-aplicable a redes locales o amplias por utilizar niveles jerárquicos.

-mecanismos:

a) Con cierta frecuencia, un servidor de tiempo manda la hora a todas las computadoras para que se sincronizan (redes LAN).

b) Cuando se quiere sincronizar, el servidor de tiempo le manda a otra computadora la hora a utilizar que considera el tiempo de latencia del canal de comunicación.

* Relojes Virtuales:
  + - Permiten ordenar eventos en un sistema distribuido donde no hay una hora global confiable.
    - Sólo permiten conocer que evento se efectuó antes o después de otro.
    - Mecanismo: Registrar la cantidad de eventos que sucedieron e incluirlo como un *timestamp* en cada mensaje enviado.

•Relojes Virtuales de Lamport

•Relojes Vectoriales

Relojes virtuales de Lamport:

Por cada evento que produce un proceso o so de pc, va incrementando un contador.

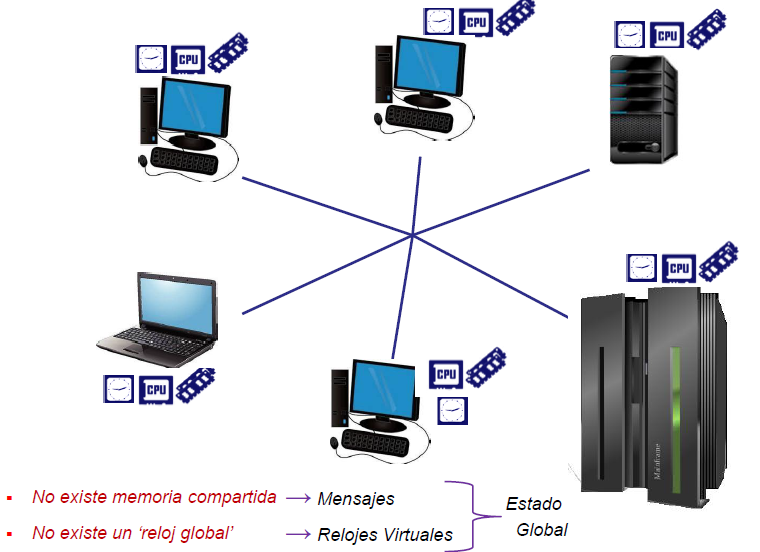
Esto tiene un problema:

El hecho de que un valor de un contador de un evento sea menor a otro, no implica que el evento anterior de verdad se produjo antes que el siguiente.

Relojes vectoriales:

En lugar de tener un contador único por cada proceso, yo tengo un vector de contadores, cada proceso, va actualizar su posición que corresponde a su reloj de Lamport interno, los otros procesos van a actualizar otra posición. Podemos saber que evento si se va a producir antes que otro, cual se produce después y que eventos son independientes uno de otros.

Estado consistente:



no existe memoria compartida, a la cual puedan acceder a todos los procesos y los sistemas operativos en un sistema distribuido.

La solución a la falta de memoria compartida, es el intercambio de mensajes.

La solución a no tener un reloj global son la utilización de relojes virtuales (vectoriales).

Pero… con estos 2 mecanismos tenemos que lograr un estado global.

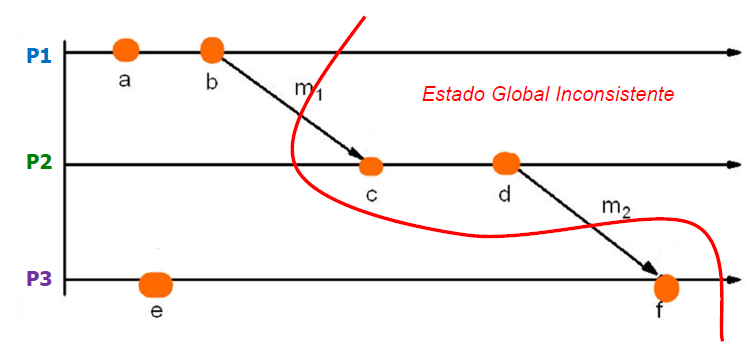
¿Cómo obtener un Estado Global Consistente?:

* Estado Global:
  + - Está definido en un Sistema Distribuido por:
      * el estado interno de cada computadora (memoria*)*
      * el estado de los canales de comunicación (mensajes *encolados)*

\*Se utiliza para:

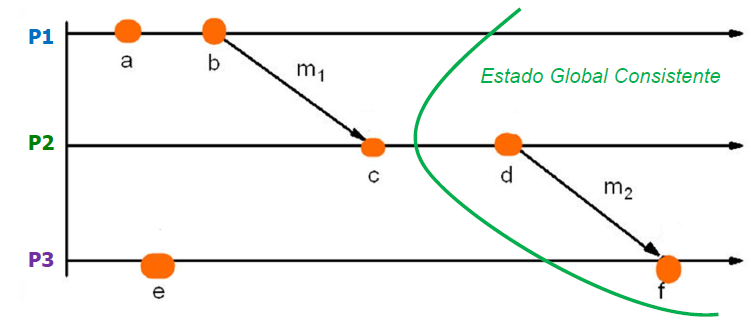
* *detección del deadlock.*
* *establecimiento de puntos de recuperación.*
* *detección de objetos que no se encuentren referenciados o utilizados por los procesos.*
* *detección de procesos finalizados (correctamente o por error).*

Un Estado Global se considera *consistente* si por cada evento recolectado también se tienen recolectados todos que sucedieron anteriormente.

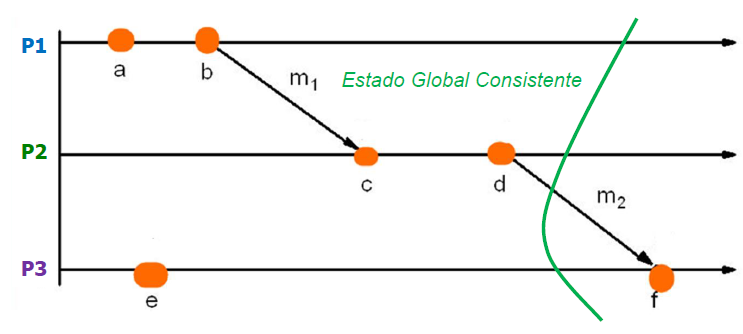
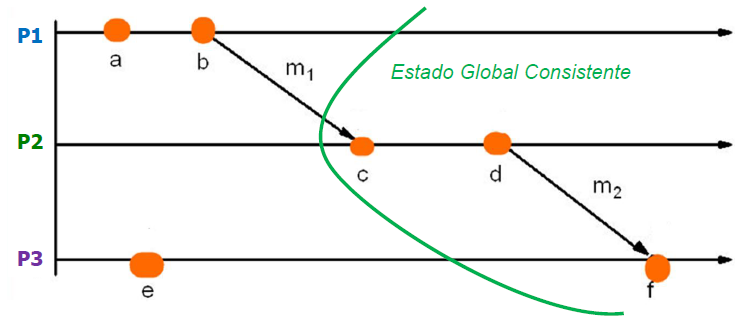


Estaría manteniendo dentro de mi estado los eventos a b e y f, pero me faltaría el d que se debe recolocar antes que f.

En cambio, si yo recolecto un estado global incluyendo abc y e porque tengo todos los eventos y todos los que sucedieron antes.



Otras formas



* Algoritmo de la *Instantánea* o *Snapshot* [Chandy & Lamport]:

1)El proceso que inicializa el algoritmo guarda su estado local y envía el mensaje *snapshot* al resto de los procesos.

2)Cuando un proceso recibe el mensaje *snapshot* desde el proceso P:

‒Si es la primera vez que lo recibe, guarda su estado y reenvía el mensaje a todos los otros procesos. Además, comienza a registrar todos los mensajes recibidos de otros procesos.

‒Si no es la primera vez que lo recibe, se almacenan los mensajes anteriores recibidos del proceso P como el estado de ese canal. Además, deja de registrar los nuevos mensajes que lleguen de P.

Cuando el proceso haya recibido un mensaje *snapshot* de todos los otros procesos, el algoritmo finaliza enviándose el estado local y de los canales al proceso que lo solicitó.

De esta forma logramos responder la pregunta de cómo lograr un estado consistente.

Sincronización de procesos:

* Busca que los Procesos no se interfieran entre sí al ejecutarse en forma concurrente.
* Controla *el acceso de los Recursos Compartidos (****Región Crítica)***de los Procesos. buscando garantizar la ***Exclusión Mutua***.

\*Implementación de Exclusión Mutua mediante **Semáforos**:

- Se basa en la utilización de un mecanismo especial provistos por el Sistema Operativo para controlar el acceso a la región crítica: los *Semáforos*

*-* Un *Semáforo* es una clase formada por:

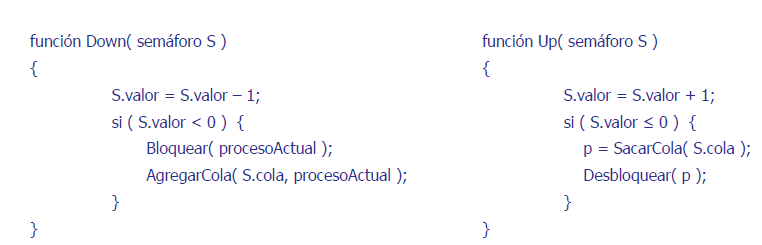
+*Contador Entero*

+*Cola de Espera de Procesos (FIFO)*

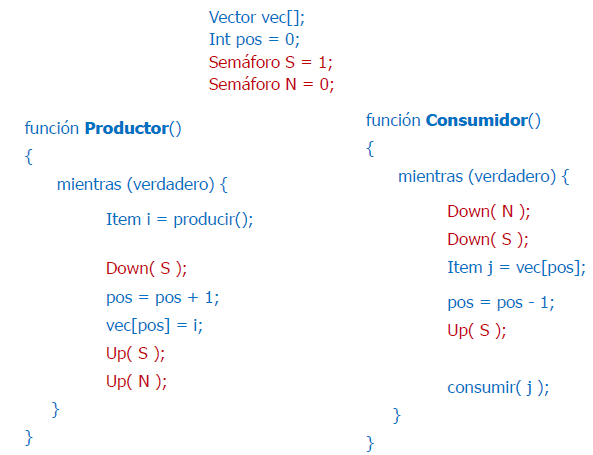
+*Funciones Atómicas: Up & Down*

*-* *Requiere:*

* *memoria compartida*
* *colas de espera globales*
* *Eventos globales*

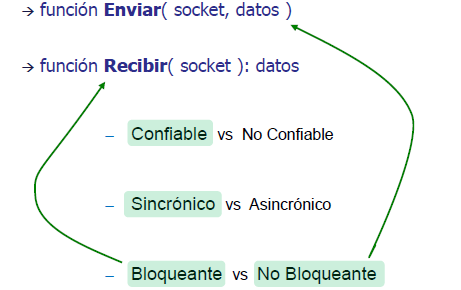


\*ejemplo: productor / consumidor con semáforos



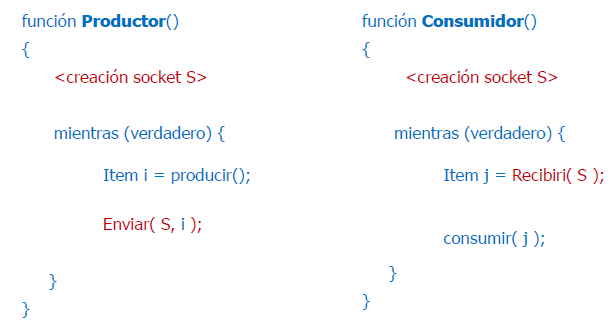
\*\*implementación de exclusión mutua mediante mensajes:

* Se basa en la utilización de las primitivas de comunicación (sockets + mensajes) para controlar el acceso a la región critica.
* Funciones:



Solo cambia el método de bloqueante y no bloqueante.

\*\*ejemplo: productor / consumidor con mensajes

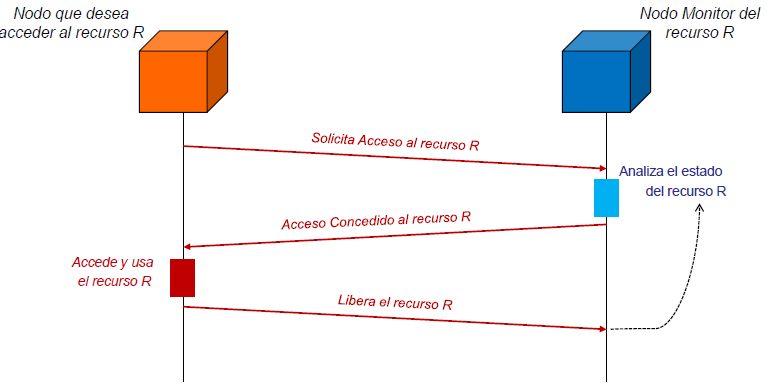


¿Cómo manejar la sincronización con mensajes cuando existen múltiples nodos y procesos?:

* Estrategias

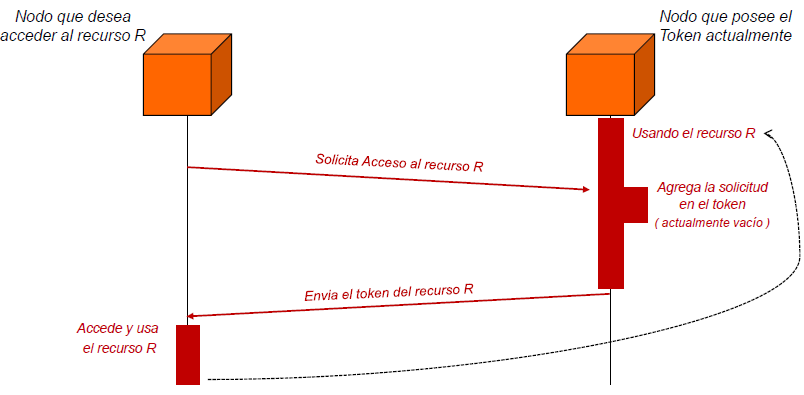
-centralizada:

Existe un nodo de control (monitor) asociado a cada recurso que determina que proceso puede utilizarlo en cada momento.



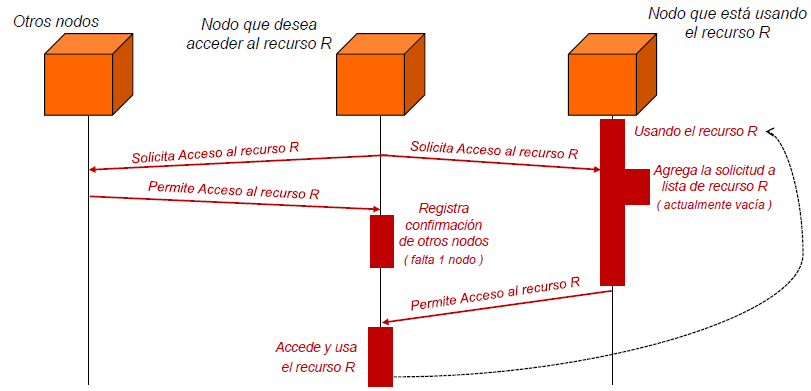
-descentralizada con token:

No existe un nodo de control, pero si un mensaje especial (token) que le permite al que lo posee acceder al recurso.



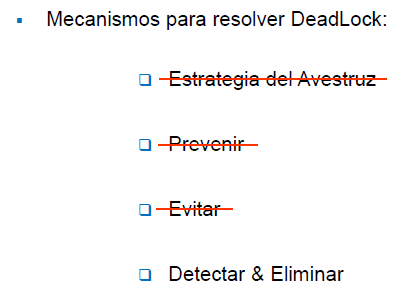
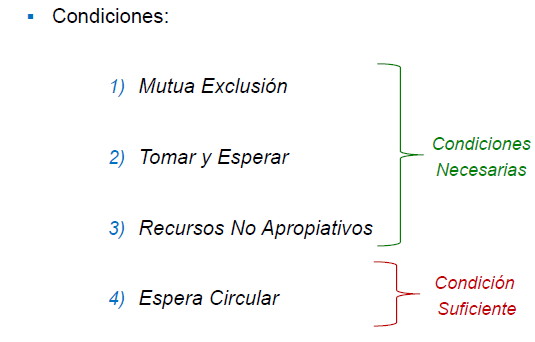
-descentralizada sin token:

No existe un nodo de control ni un mensaje especial. Se utiliza el orden de las solicitudes para permitir el acceso sincronizado al recurso.



De esta forma logramos responder la pregunta de cómo lograr la sincronización de procesos.

Deadlock:



* Detectar y eliminar el deadlock:

En forma *recurrente* se analiza si hay *Espera/s Circular/es* entre los procesos (se simula la ejecución de los procesos para determinar si terminan o no).

si no hay, no hace nada

si hay, se elimina

‒liberar recurso/s tomado/s

‒rollback de proceso/s

‒matar proceso/s

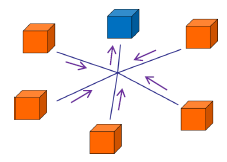
Problema:

*los procesos de la espera circular pueden estar ejecutando en nodos diferentes.*

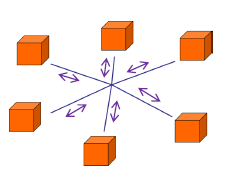
¿Cómo detectar el deadlock entre múltiples procesos ejecutando en distintos nodos?:

* Estrategias:

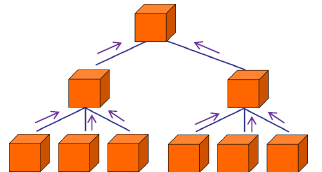
-centralizada: Toda la info se va a concentrar en un único nodo (algoritmo snapshot) para que ese nodo registre toda la info de todos los nodos del sist distribuido y de esta manera detectar si hay o no una espera circular.



-descentralizada: toda la info sobre el estado del sistema, se les manda a todos los nodos y estos tratan de detectar la espera circular (están todos realizando la misma tarea al mismo tiempo, se genera conflictos con un deadlock, por ende, se están matando procesos de mas). Se vuelve un cuello de botella si el sistema distribuido es muy grande.



-jerárquica: se arman grupos de nodos (parecido a una grilla donde tengo subredes que se ocupan de distintas acciones) y un nodo de ese grupo será encargando de detectar si hay un deadlock en los procesos de ese grupo de nodos. Puede pasar que un nodo de un grupo tenga un proceso que está en espera circular con otro proceso que está en otro grupo, por ende, el nodo que está detectando el deadlock no lo hará, va haber un nodo que está en un nivel superior, que solo va analizar situaciones donde un proceso de un grupo comparte algún recurso o esperar un recurso que tiene tomado un nodo de otro recurso. Entonces al detectar el deadlock, sea de forma más ordenada, pero que tampoco un nodo se vuelva un cuello de botella de la detección. Tiene la ventaja de las 2 estrategias anteriores y ninguna desventaja.



De esta forma logramos responder la pregunta de cómo manejar el deadlock.