Sincronización y concurrencia en sistemas distribuidos

UNIVERSIDAD NEBRIJA

Índice

- Relojes y eventos
- Sincronización
- Tiempo lógico
- Exclusión mutua
- Bloqueo

Relojes y eventos

- **PROBLEMÁTICA**: **No existe un tiempo absoluto** y global al que podamos invocar
  - Supongamos que un sistema distribuido tenga **N procesos**, $\mathbf{p_i}$ para **i=1:N**
- Cada proceso $\mathbf{p_i}$ se ejecuta en un procesador diferente, y cada proceso tiene su estado $\mathbf{s_i}$ asociado ($\mathbf{s_i}$ se transforma dinámicamente)
  - $\mathbf{s_i}$ contiene el valor de **todas la variables** y **objetos** que afectan a $\mathbf{p_i}$
- Cada proceso únicamente puede comunicarse con otros procesos diferentes mediante el **paso de mensajes**
- Definimos el **evento** como **una única acción de comunicación o de transformación** de $\mathbf{s_i}$ por parte de $\mathbf{p_i}$
- Definimos **historia** del proceso $\mathbf{p_i}$ como la **serie de eventos que tienen lugar en él**:

$$\text{Historia}(p_i) = \mathbf{h_i} = \langle e_i0, e_i1, e_i2, \ldots, e_iM \rangle$$

  - Mediante la historia somos capaces de **ordenar eventos**

Relojes y eventos

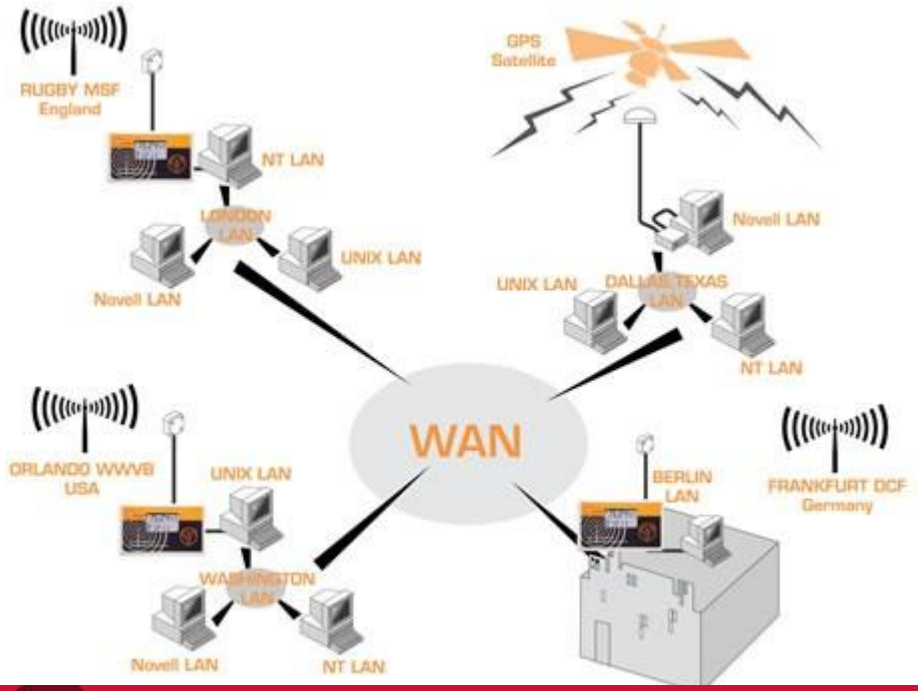
- **Eventos <u>sucesivos</u>** corresponderán a **<u>marcas de tiempo diferentes</u>**, si la **<u>resolución</u>** del reloj es más pequeña que el tiempo entre eventos
- El tiempo entre eventos depende de factores como por ejemplo el **<u>tipo de procesador</u>**
- Los principales parámetros de los relojes son:
  - **Sesgo**: **<u>diferencia</u>** instantánea entre las lecturas de **<u>dos relojes diferentes</u>**
  - **Deriva**: **<u>variación del ritmo</u>** al que cuenta el reloj (variaciones físicas internas:
    oscilador, variaciones físicas externas: temperatura) 1s cada 10^8 días
- Para mejorar las características de los relojes distribuidos se recurre a la **<u>sincronización externa</u>**. Los relojes atómicos utilizados para sincronizar tienen una deriva de 1s cada 10^{13} días

Relojes y eventos

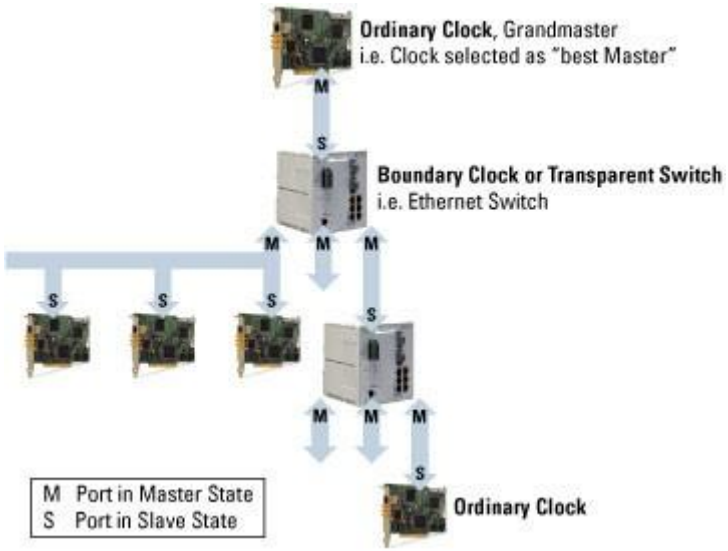
- El **tiempo universal coordinado** (UTC) es el estándar utilizado para sincronizar sistemas. Esta medida temporal se **difunde mediante radio**, **servidores** y **satélites**, consiguiendo con **cada sistema una precisión diferente**.
  - El **UTC** se basa en **reloj atómico**
- **Sincronización externa**: consiste en utilizar una fuente externa autorizada para sincronizar los diferentes relojes del sistema distribuido.
- **Sincronización interna**: consiste en sincronizar los relojes de un sistema distribuido utilizando para ello medidas de **magnitudes internas al propio sistema**, **sin** intervención de **fuentes externas**.
- Los **fallos** de reloj son de **ruptura** cuando el **reloj no pulsa**, y **fallos arbitrarios** el **resto** de ocasiones. Por tanto, el **reloj** no tiene porque ser correcto, sólo **capaz de sincronizarse**

Relojes y eventos

Sincronización externa



Sincronización interna



Sincronización

•Sincronización de **procesos síncronos**:

- •Suponemos que se conoce la **deriva**, el **retardo de transmisión de mensajes** y el **tiempo de procesado**
- •Un proceso **envía el tiempo *t*** en un mensaje ***m***.
- •Se definen **tiempos máximos** (max) y **mínimos** (min) de transmisión.
- •Se calcula el **sesgo** máximo como **u=max-min**
- •Se **sincroniza a la mitad del sesgo para reducir el error**: **t+u/2**

•La mayoría de sistemas distribuidos son asíncronos, lo cual hace que se **desconozca el límite superior** (max) de los retardos. **Ejemplo**: Internet

Sincronización

- Sincronización **Método Cristian** (‘89):
  - El valor de $\mathbf{T_{round}}$ se calcula de forma **estadística**, es decir, se toman múltiples medidas a lo largo del tiempo para conseguir un valor lo más preciso posible
  - Al ser un sistema de sincronización externo, el algoritmo Cristian **dejará de funcionar si falla el servidor UTC**. Por ello, Cristian propone solicitar varios tiempo y quedarse con el primero que responda. Pero, ¿y si el que responde se equivoca?

Sincronización

El Algoritmo de Cristian propone realizar la sincronización de los relojes de múltiples máquinas utilizando una máquina externa que cuente con un reloj más preciso, posiblemente sincronizado otros aún más precisos. Partimos de un sistema distribuido con varios nodos i, cada uno de ellos cuenta con un reloj local Ci. En cualquier instante t se cumple para todos los nodos Ci(t) = t, es decir, todos los relojes locales tienen la misma hora y coinciden con la hora "verdadera". El algoritmo de Cristian sirve para sincronizar el reloj local de un ordenador cliente (CC) con el reloj local de un ordenador servidor (CS).[2]

1. Un proceso *p* hace una petición de tiempo al servidor en un mensaje m_r.
2. El servidor responde con un mensaje m_t en el que incluye su tiempo T_{UTC}.
3. El proceso que recibe el mensaje m_t actualiza su reloj con el tiempo T_{UTC}, pero hay que considerar el error cometido pues se ha requerido un tiempo para la transmisión del mensaje desde el servidor.

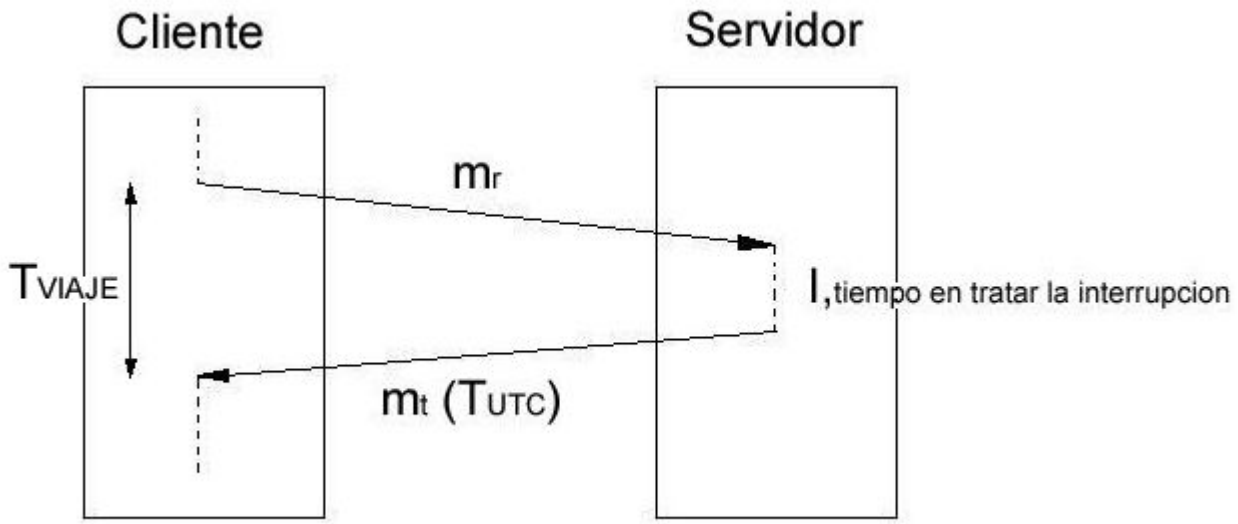
Se mide el tiempo que se tarda en recibir la respuesta desde que de envía el mensaje de petición, T_{VIAJE}. El tiempo estimado de propagación, en ausencia de otra información, será $T_{VIAJE}/2$ por lo que el cliente sincroniza su reloj a $T_{UTC} + T_{VIAJE}/2$. Esta estimación puede mejorarse si se conoce el tiempo mínimo de propagación del mensaje:

- El tiempo mínimo en el que el servidor escribió el mensaje es *min*.
- El tiempo máximo en el que el servidor escribió el mensaje es $T_{VIAJE} - min$.

Por lo tanto el tiempo en el que el mensaje del servidor es recibido se sitúa en el rango $[C_{UTC} + min, C_{UTC} + T_{VIAJE} - min]$ cuya anchura es $T_{VIAJE} - 2min$ así que la precisión es $\pm T_{VIAJE}/2 - min$.

Sincronización

El Algoritmo de Cristian



Sincronización

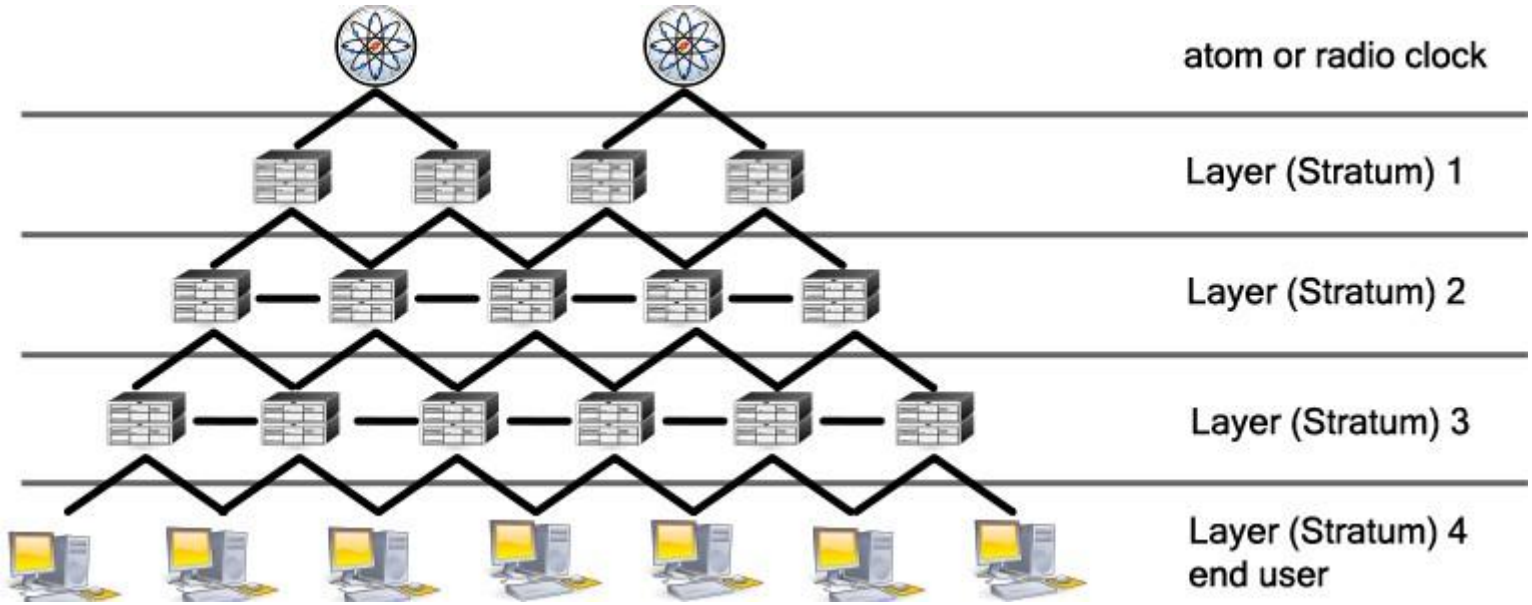
- Sincronización **Método Berkeley**:
  - Se **elige** a un computador como **coordinador o maestro**
  - El **maestro consulta** periódicamente a los **esclavos**
  - El maestro calcula los **tiempos** de **ida** y **vuelta** a cada uno de los **esclavos** y la **promedia**, incluyendo los tiempos que han proporcionado los esclavos con la consulta
  - Envía el **factor de corrección** necesario para cada uno de los **esclavos**, no el valor corregido.
    - El **factor de corrección** puede ser **positivo o negativo**
    - El algoritmo **descarta** directamente los **relojes defectuosos** para realizar el cálculo
    - Se considera un reloj **defectuoso** el que se desvía más de un determinado **umbral**
  - A diferencia del método Cristian, **si falla el coordinador**, se define un **nuevo coordinador** entre los esclavos. Esto se puede realizar porque es un método de sincronismo interno

Sincronización

- Sincronización **Protocolo del tiempo de Red** (NTP):
  - **Cristian** y **Berkeley** son algoritmos para **intranets**
  - **NTP** pretende proporcionar un servicio que permita a los clientes a lo largo de Internet estar sincronizados a UTC
  - NTP emplea técnicas estadísticas de **filtrado** a los valores temporales
  - NTP permite sobrevivir a pérdidas largas de la conexión mediante **redundancia de servidores y redes**
  - NTP presenta una **elevada frecuencia de sincronización** para evitar derivas en los clientes
  - NTP **protege contra la interferencia** (accidental o maliciosa)
  - Existen **varios estratos en la red de NTP.** El estrato inferior siempre se sincroniza con el estrato superior. El primer estrato está directamente conectado a las fuentes UTC
  - NTP aplica el **algoritmo Cristian en cada nivel para calcular los sesgos**

Sincronización

- Sincronización **<u>Protocolo del tiempo de Red</u>** (NTP):

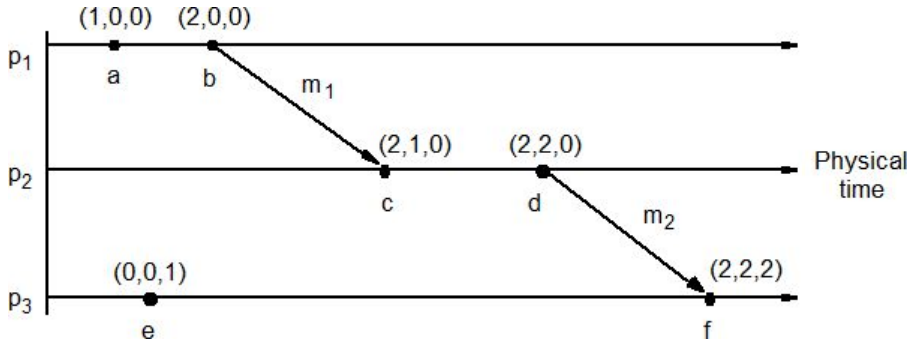


Tiempo lógico

- **Lamport** (‘78): **No podemos sincronizar perfectamente los relojes** en un sistemas distribuido, por ello **no podemos usar el tiempo físico**
  - Recurrimos a la **causalidad** para ordenar sucesos que ocurren en diferentes procesos
- La ordenación causal o causal potencial consiste en establecer la relación “***sucedió antes***” en los eventos de un proceso
  - El reloj lógico consiste en capturar numéricamente la relación “sucedió antes”
- **CADA PROCESO *pi*** tiene su propio reloj lógico, L_i, que sirve para aplicar marcas de tiempo
    - Con cada evento en p_i, L_i se incrementa: $L_i=L_i+1$
    - Cuando se envía un mensaje el tiempo t **es igual a** L_i
    - Cuando se recibe un mensaje el proceso calcula $L_i=\max(L_i,t_{recibido})$

Tiempo lógico

- **Relojes vectoriales**:
  - Si existe un sistema con **N procesos** el **vector** de tiempo en de **N enteros**
  - Cada proceso mide su propio reloj vectorial para **ordenar los sucesos locales**
  - El resto de valores del vector son las **marcas de tiempo de los mensajes que envía a otros procesos**
  - **Aumentan** el número de recursos de **almacenamiento** necesarios y **sobrecargan** la **red** con mensajes más largos

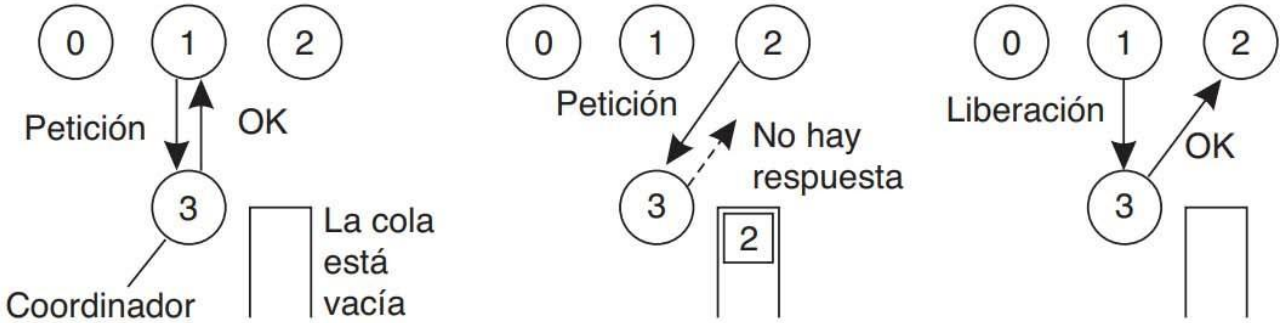


Exclusión mutua

- **Exclusión mutua**: prevención de interferencias para asegurar la consistencia cuando varios procesos quieren **acceder a un mismo recurso compartido**
  - En ocasiones se puede controlar mediante el uso de servidores, pero en general se **requiere un mecanismo separado**
- En sistemas **peer to peer** es necesario implementar un **sistema de exclusión independiente**
  - Para proteger los recursos compartidos surge el concepto de **sección crítica**
- Una **sección crítica** es un mecanismo que encierra un determinado **recurso** en una región a la que únicamente **puede acceder uno de los componentes a la vez**
  - Las acciones que se pueden realizar en la sección crítica son:
    - **Entrar**: que **bloquea el recursos** hasta recibir el mensaje de salir
    - **Salir**
    - **Acceder a recurso**

Exclusión mutua

- **Requisitos** de la exclusión mutua para sistemas distribuidos:
  - **Seguridad**: Un **único proceso** puede estar ejecutándose en la sección crítica
  - **Pervivencia**: Las **peticiones** para entrar y salir de la sección crítica **se conceden** (tarde o temprano se acepta la petición, pero **no se deniega ninguna**)
  - **Ordenación**: Si una petición para entrar en la sección crítica ocurre antes que otra **se garantiza ese orden**

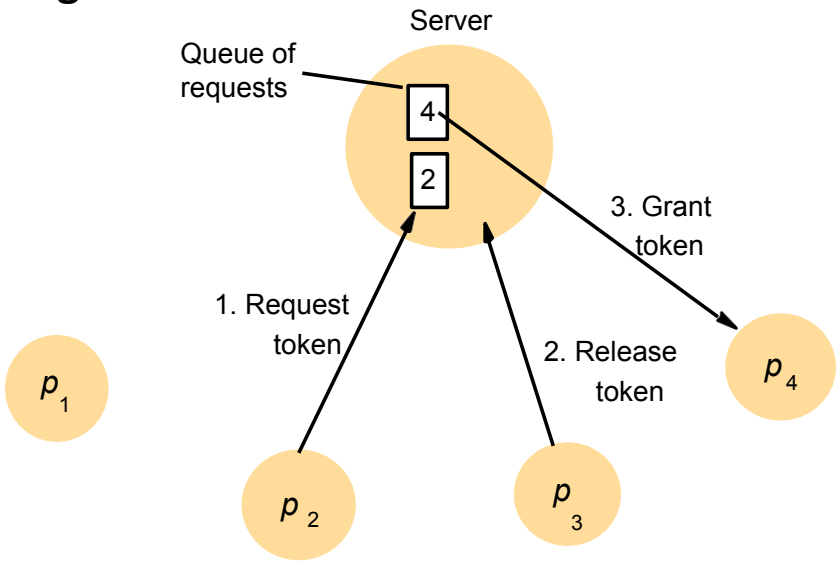


Exclusión mutua

- **Evaluación** de los sistemas distribuidos:
  - **Ancho de banda consumido**: proporcional al **número de mensajes Entrar/Salir**
  - **Retraso del cliente**: cada vez que un proceso entra en la **operación Entrar/Salir**
  - **Efecto sobre la capacidad de procesamiento del sistema**: a mayor retardo para entrar en sección crítica, **menor eficiencia en el procesado**
- Principales algoritmos:
  - **Servidor central**
  - **Anillo**
  - **Ordenación de eventos** (no utilizado en la práctica)
  - **Algoritmo de Maekawa**

Exclusión mutua

- **<u>Servidor central</u>**: un servidor da **permisos** para entrar en la sección crítica mediante un **testigo**. Si el proceso **no tiene el testigo** no puede entrar en la sección crítica, pero el servidor **pone en la cola de espera su petición**. Al **salir** de la sección crítica se **devuelve el testigo**

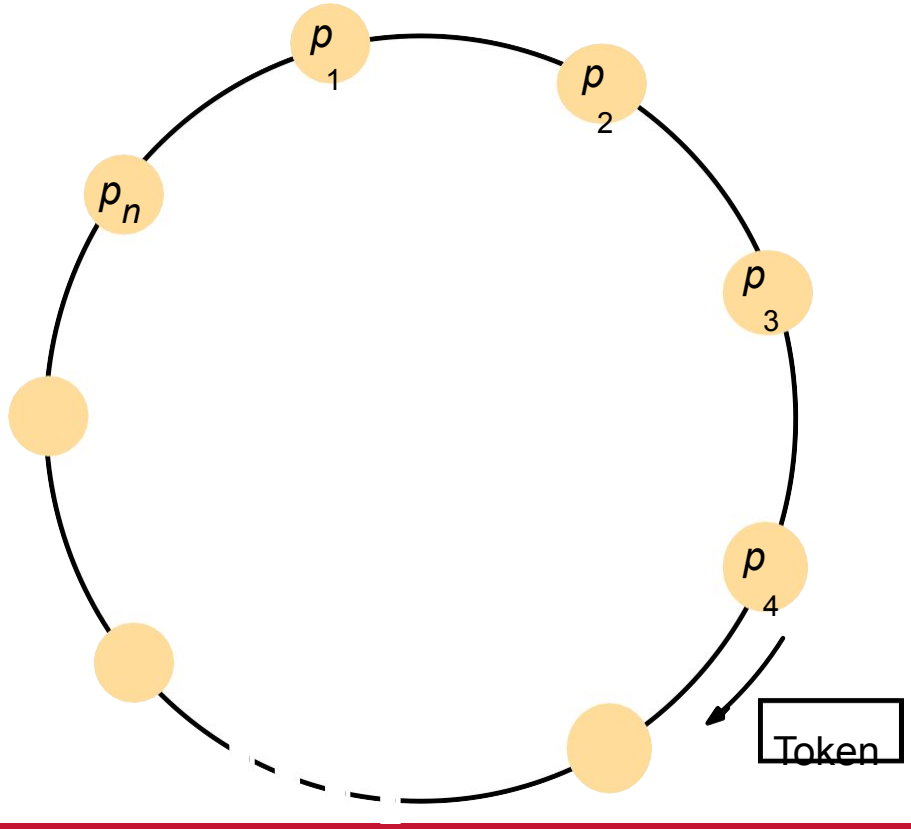


Exclusión mutua

- **Servidor central**: Evaluación:
  - Consumo de **ancho de banda**: requiere **envío** de mensaje **incluso cuando nadie tiene el testigo**
  - **Retraso** del cliente: **ida y vuelta** de los mensajes del proceso anterior más la **concesión** de entrada a la sección crítica
- **Anillo**: consigue la exclusión mutua **sin procesos adicionales** (servidores)
  - **Cada proceso** está únicamente **conectado a otro proceso**
  - Se pasa **un testigo de un proceso al siguiente** (mediante mensaje)
  - Si un proceso **no quiere entrar en la sección crítica** le **pasa el mensaje** al siguiente proceso. Si requiere **el testigo lo retiene hasta que acabe**
- **Anillo**: evaluación
  - Consumo de **ancho de banda**: **consumo continuo** ya que el testigo se intercambia constantemente
  - **Retraso** del cliente: entre **cero** y el **número de procesos** dentro del anillo

Exclusión mutua

- **Anillo**:

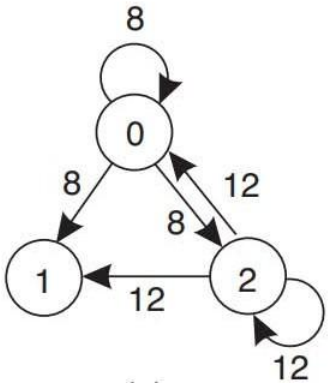


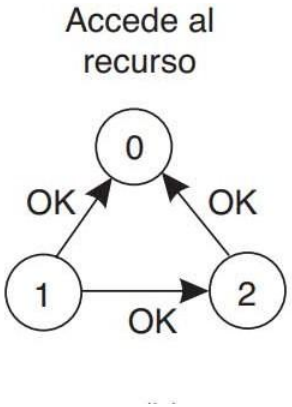
Exclusión mutua

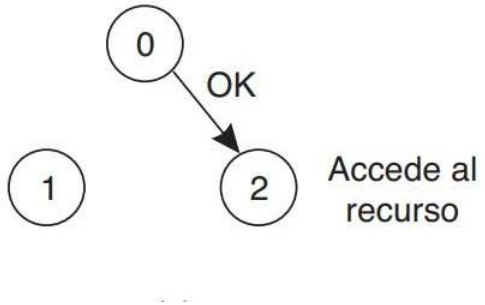
- **Ordenación de eventos** : **entra** en la sección crítica el proceso que tenga la **marca de tiempo menor**, es decir, el que lleva más tiempo esperando. Cuando **acaba** envía un **mensaje de OK a todos los afectados** para que pueda entrar el siguiente proceso en la sección crítica
  - **Ordenación de eventos**: Evaluación
    - Consumo de **ancho de banda**: **elevado**, envía marcas temporales continuamente
    - **Retraso** del cliente: **ida y vuelta** de los mensajes del proceso anterior más la **confirmación de finalización** (OK) del proceso anterior y el **envío** y **comprobación de las nuevas marcas de agua** y **concesión** de entrada a la sección crítica

Exclusión mutua

- **<u>Ordenación de eventos</u>** :







Exclusión mutua

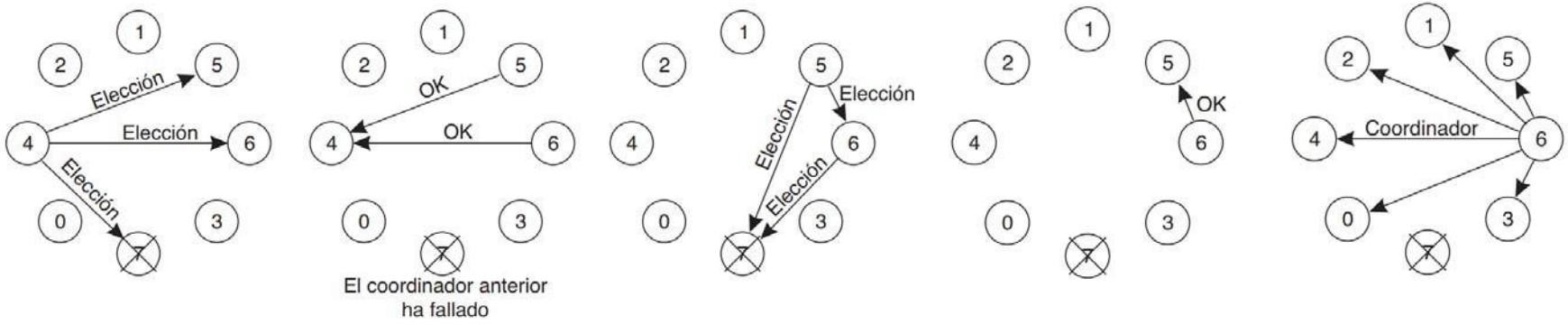
- **Algoritmo de Maekawa**: parte de la idea de que **no es necesario que todos los procesos estén conformes** para que un proceso determinado acceda a un recurso
  - El algoritmo propone una **votación**. Si un proceso **consigue mayoría** de votos favorables se le **permite acceder al recurso**
  - El problema es que **si se produce un empate de votos** o todos los procesos desean acceder simultáneamente a los procesos, se produce una situación de **estancamiento**
  - Existen **alternativas** basadas en este algoritmo que evitan estas situaciones, pero tienen **mayor complejidad**

Exclusión mutua

- **<u>Algoritmos de selección de coordinador</u>**:
  - Bully:
    1. P envía un mensaje de elección a todos los procesos con números superiores
    2. Si ningún proceso responde, P gana la elección y se convierte en el coordinador
    3. Si uno de los procesos superiores responde, toma el mando. El trabajo de P está hecho

Exclusión mutua

- **<u>Algoritmos de selección de coordinador</u>**:
  - Bully:



Exclusión mutua

- **<u>Algoritmos de selección de coordinador</u>**:
  - Anillo:
    1. Cuando cualquier proceso advierte que el coordinador no funciona, elabora un mensaje de elección que contiene su propio número de proceso y envía el mensaje a su sucesor
    2. Si el sucesor falla, el remitente lo salta y se dirige al siguiente miembro del anillo, hasta que localice un proceso en ejecución
    3. El mensaje regresa al proceso que inició todo. Ese proceso reconoce este evento cuando recibe un mensaje entrante que contiene su propio número de proceso
    4. El tipo de mensaje cambia a COORDINADOR y circula una vez más, esta vez para informar a todos de quien es el coordinador

Exclusión mutua

- **<u>Algoritmos de selección de coordinador</u>**:
  - Elecciones en sistemas de gran escala: varios nodos deben seleccionarse
    - En este caso se denominan superpuntos

      1. Los nodos normales deben tener acceso de baja latencia a los superpuntos
      2. Los superpuntos deben distribuirse uniformemente
      3. Debe haber una proporción predefinida de superpuntos, relativa al número total de nodos

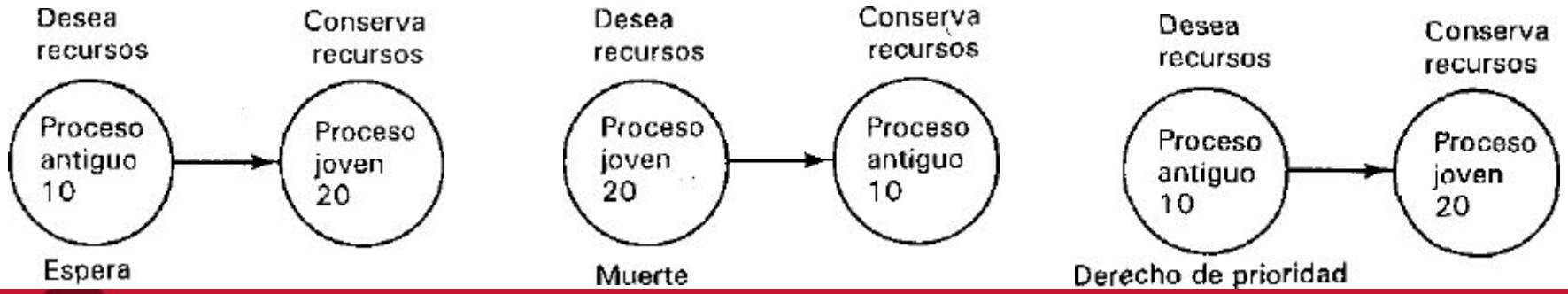
  - Ejercicio: Definir un algoritmo que tenga en cuenta todos los requisitos anteriores

Bloqueo

- Existen distintos **tipos de bloqueo** en los sistemas distribuidos:
  - **Bloqueos de comunicación**: se **satura el canal** cuando varios procesos intentan enviar **mensajes simultáneamente**
  - **Bloqueo de recursos**: se intenta acceder a un **recurso no disponible u ocupado**
- Estrategias de **gestión de bloqueos**:
  - **Prevención del bloqueo**. Ejemplo: **Sobredimensionamiento**, técnicas de prevención
  - **Ignorar el bloqueo**
  - **Detectar e intentar recuperar** el sistema
    - Detección de bloqueos **centralizada**
    - Detección de bloqueos **distribuida**: Chandy-Misra-Haas

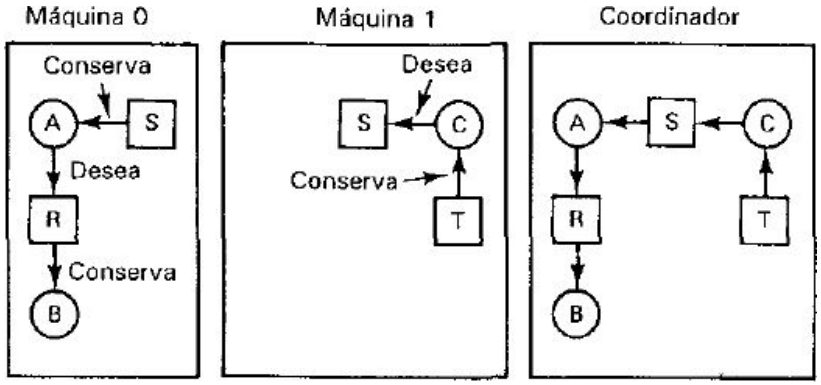
Bloqueo

- Técnicas de **<u>prevención de bloqueo</u>**: intentar que el **BLOQUEO sea IMPOSIBLE**
  - Dar **prioridad** a procesos que **llevan más tiempo** esperando un determinado recurso
  - Dar **prioridad** a los procesos que **llevan menos tiempo** esperando que se les otorgue el recurso
  - **Destruir los procesos** que desean acceder a un determinado **recurso ocupado**
  - Establecer **tiempos de espera si el recurso está ocupado**



Bloqueo

- **Detección de bloqueos centralizada**:
  - **Cada esclavo** dispone de una gráfica de **los recursos que desea** y los que **conserva**
  - El coordinador o **maestro debe detectar** si más de un proceso **desea un recurso a la vez** y si se pueden **generar bloqueos** en procesos **dependientes**.
    El coordinador se encarga de **introducir retrasos** o **secuenciaciones** en la asignación de los recursos
  - Este sistema **no es útil si no se utilizan marcas temporales**



Bloqueo

- **Detección de bloqueos distribuida**: Chandy-Misra-Haas
  - Se genera un **mensaje espacial de exploración** que consta de :
    - el **proceso recién bloqueado**
    - el **proceso que envía el mensaje**
    - el **proceso receptor del mensaje**
  - Se envía el **mensaje espacial de exploración** cuando sospechamos que el **sistema está bloqueado**.
  - **Si llega al ORIGEN es que existe un CICLO** y se confirma el **BLOQUEO**
  - Si se **confirma el BLOQUEO** el proceso origen del **mensaje se "suicida"**
  - Si dos procesos envían mensajes espaciales y se confirman **DOS bloqueos** siempre se **suicida el proceso que está en un CICLO MAYOR**

Bloqueo

- **<u>Detección</u> de bloqueos <u>distribuida</u>**: Chandy-Misra-Haas
  - Mensaje espacial de exploración con ciclo