Desarrollo de Sistemas Distribuidos

Tema 3
Coordinación

Contenidos

1. Tiempo lógico
2. Algoritmos Distribuidos
   - Exclusión Mutua
   - Elección

Tiempo Lógico

- Cuestiones de tiempo importantes en Sist. Distribuidos por:
  1. **Medida** que deseamos obtener con precisión para:
     a) **Sincronización externa:** cuándo ocurrió un evento concreto (a qué hora sucedió, p.ej. transferencia bancaria). Para ello es necesario sincronizar la hora de la máquina donde ocurrió el evento con algún **reloj o fuente externa autorizada** (relojes atómicos que transmiten por radio terrestre o satélite, o red telefónica)
     b) **Sincronización interna:** se trata de obtener las mismas referencias de tiempo o intervalo entre dos eventos ocurriendo en dos computadoras diferentes **conociendo precisión**, para un instante dado (p.ej., tiempos de transmisión de un mensaje entre máquinas → dos marcas de tiempo: una en origen y otra en destino)
  2. **Problemas lógicos** debidos a la distribución (p.ej., para mantener la consistencia en un gestor de transacciones bancarias, auditorías…)

- **Evento**: Acción que parece ocurrir indivisiblemente (p.ej. envío de mensaje)
- El **orden de la ocurrencia de eventos** puede ser **crítico** en aplicaciones distribuidas (p.ej., servidor de datos replicados)

Diapositiva 123

M1 10.3 Colouris, pág 371

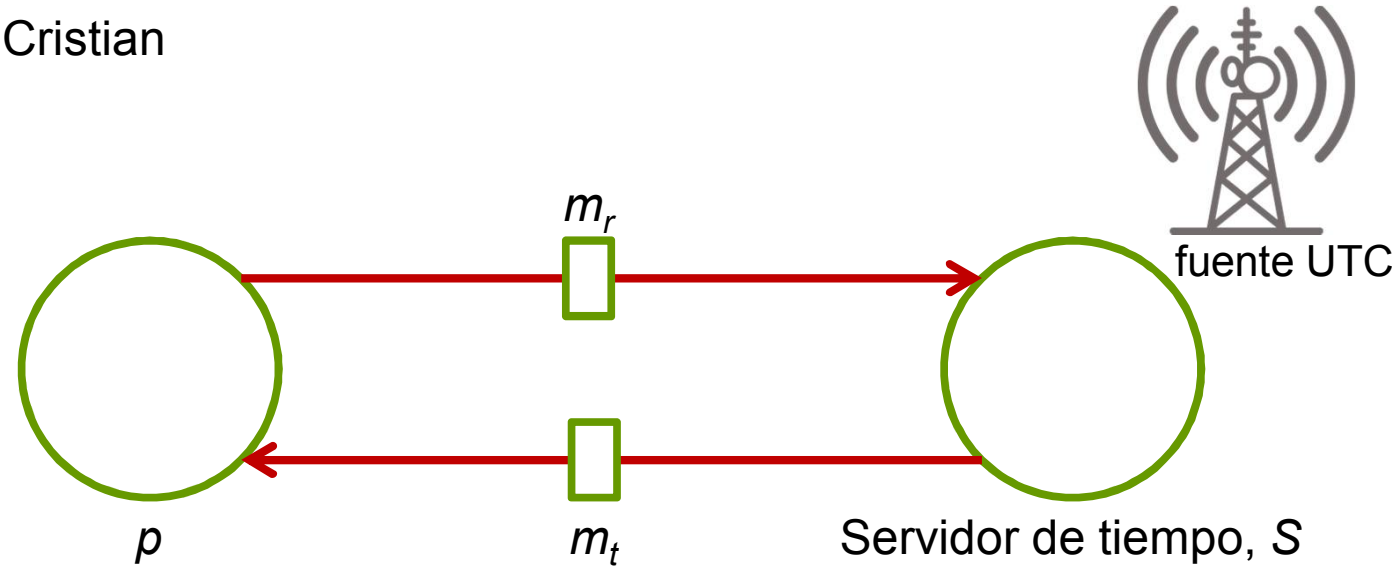
Manolo; 21/04/2015

Tiempo Lógico

- Requisitos y tipos de aplicaciones:
  - **Centralizadas**: Sólo necesitan conocer el **orden de los eventos**, con lo que basta asociar un **reloj** (tiempo absoluto) o **contador** (tiempo relativo) a cada evento
  - **Distribuidas**:
    - Conocer el desplazamiento relativo del tiempo (reloj) de una máquina con respecto a otra, e idéntica velocidad del pulso → casi imposible
    - Otra opción es que exista un reloj físico compartido (sistemas síncronos)
    - Servidor de tiempo sobre peticiones (sistemas asíncronos):
      - Existe un método (**Cristian**) para sincronizar relojes que se basa en el tiempo universal coordinado (estándar internacional) y en la existencia de un servidor de tiempo. **Problema**: fallo del servidor, o de una replica de éste o impostor (que responda a los mensajes *multicast*)
    - ¿Contador? ¿centralizado o distribuido?

Tiempo Lógico

Método Cristian



$$T_{round} = T_{recibir\_mt} + T_{enviar\_mr}$$

- Asumiendo iguales tiempos de envío y recepción, *p* puede fijar su reloj a: $t + T_{round}/2$

Tiempo Lógico

- **Relación de orden** (*ocurrió-antes)*:
  - **Esquema de ordenación** de eventos basado en dos puntos:
    1. Si dos eventos ocurren en el mismo proceso, entonces ocurren en el orden que se observan
    2. Si se envía un mensaje, entonces el evento asociado al envío **ocurre antes** que el evento de recepción de dicho mensaje
  - *Lamport* generalizó estas dos relaciones en una **relación de orden causal** denominada *ocurrió-antes* ($\rightarrow$):
    1. Si $\exists p: x \xrightarrow{p} y$ (en p) entonces $x \rightarrow y$
    2. $\forall m \in Mensajes,\ send(m) \rightarrow receive(m)$
    3. Siendo $x, y, z \in Eventos$: $x \rightarrow y$ e $y \rightarrow z$ entonces $x \rightarrow z$

Tiempo Lógico

- **Relojes lógicos:**
  - Mecanismo simple que propuso *Lamport* para capturar numéricamente la relación *ocurrió antes*
  - Un **reloj lógico** es un contador software que se incrementa monótonamente:
    - C_p : nota el reloj lógico *C* del proceso *p*
    - $C_p(a)$: nota la marca de tiempo del evento *a* en el proceso *p*
    - $C(b)$: nota la marca de tiempo del evento *b* en cualquier proceso donde haya ocurrido

Tiempo Lógico

- **Relojes lógicos:**
  - Para capturar la relación *ocurrió-antes,* los procesos actualizan sus relojes lógicos y transmiten sus valores en los mensajes:
    1. C_p se incrementa antes de cada evento que ocurre en *p*
    2. Cuando un proceso *p* envía un mensaje le añade el valor $t = C_p$
    3. Cuando un proceso *q* recibe un mensaje entonces:
       - computar $C_q = max\ (C_q, t)$ y
       - aplicar *acción 1* antes de marcar el evento *receive(m,t)*
  - Fácil demostrar que si $a \rightarrow b$ entonces $C(a) < C(b)$
  - Extensión a relación de orden total:
    - $C(a) < C(b) \Leftrightarrow Cp(a) < Cq(b) \vee (\ Cp(a) = Cq(b) \wedge p < q\ )$

Algoritmos Distribuidos

- **Exclusión Mutua**:
  - Cuando **no existe núcleo central local** para basar la exclusión mutua en variables u otras facilidades compartidas
  - **Ejemplos**:
    - Existencia de servidores o recursos que no tienen mecanismos de sincronización incorporados
    - Coordinación distribuida en servicios replicados o distribuidos
- **Elección**:
  - Método para escoger un único proceso que realice un **rol concreto**
  - **Ejemplo**: elección de servidor replicado primario cuando el anterior falla

Algoritmos Distribuidos

- Exclusión Mutua:
  - Requisitos básicos y comunes:
    - Propiedades de **seguridad y vivacidad**
    - Adicional: **Orden causal** en la entrada a la sección crítica
  - **Soluciones**:
    1. Servidor centralizado
    2. Algoritmo distribuido basado en relojes lógicos
    3. Algoritmo basado en anillo

Algoritmos Distribuidos

- Exclusión mutua con **servidor centralizado**:
  - Para **entrar en sección crítica** se envía una petición al servidor y se espera la respuesta (***testigo*** o ***token***)
  - El servidor **encola peticiones** cuando no dispone del *testigo*
  - Cuando un proceso **sale de la sección crítica** envía un mensaje de liberación (devuelve el *testigo*), si el servidor tiene mensajes de petición encolados le envía el *testigo* al primero de ellos y lo saca de la cola

Algoritmos Distribuidos

- Exclusión mutua con **servidor centralizado**:
  - El servidor se puede convertir en un **cuello de botella**
  - El servidor es punto crítico de fallo
  - Hay que **regenerar el testigo** si el cliente que lo tiene **falla**

Algoritmos Distribuidos

- Exclusión mutua con **algoritmo distribuido basado en relojes lógicos:**
  - **Idea básica**: los procesos que desean entrar en la **sección crítica** envían un mensaje *multicast* a los otros *n-1* procesos. Un proceso puede entrar si todos los demás le responden, es decir, la obtención del *testigo* requiere ***n*** mensajes.
  - **Suposiciones**:
    - Los procesos **conocen las direcciones** de los demás
    - Paso de mensajes **fiable**
    - Cada proceso mantiene su **reloj lógico**

Algoritmos Distribuidos

- **Algoritmo distribuido basado en relojes lógicos** (proceso P_i):
- **Inicialización:**
  - *estado := LIBERADO*
- **Obtención del toquen:**
  - *estado := INTENTANDO;*
  - Envío selectivo de petición a los demás procesos*;*
  - T_i *:=* marca de tiempo de la petición*;*
  - ***wait until*** *(*número de respuestas recibidas = *(n-1));*
  - *estado := EN_SECCION_CRITICA;*
- **Recepción de una petición <T_j,P_j> en P_i (i ≠ j):**
  - ***if*** *(estado=EN_SECCION_CRITICA or (estado=INTENTANDO and* $(T_i,P_i)<(T_j,P_j)$ *))*

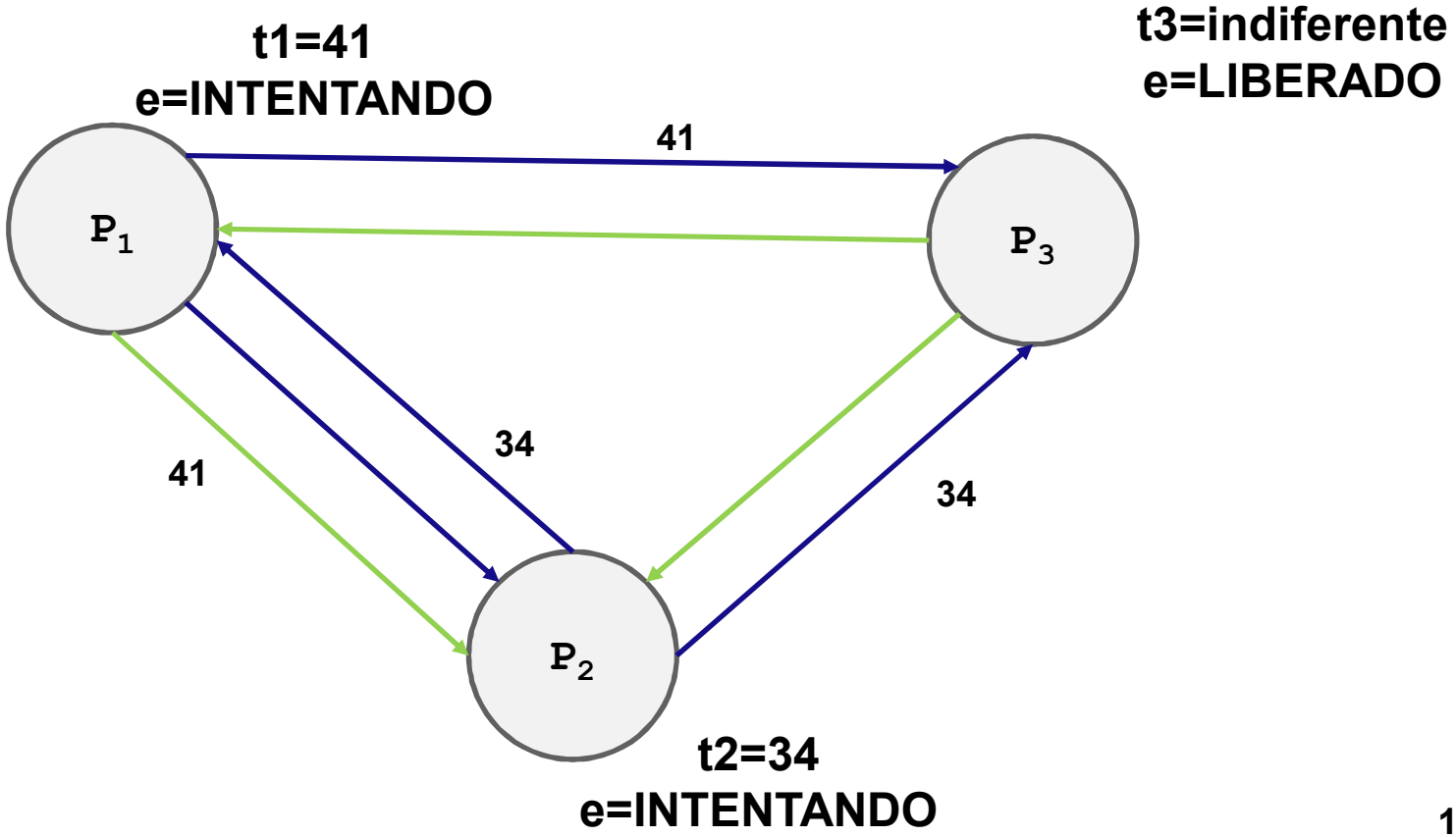
    then encolar petición de P_j*;*

    else actualizar reloj a (***máx{T_i,T_j}+1***) y enviar mensaje de respuesta a P_j*;*
- **Liberación del Testigo:**
  - *estado := LIBERADO;*
  - Enviar mensaje de respuesta a todas las peticiones encoladas y eliminarlas*;*

Si aún así, coinciden, se cederá al que tenga menor id de proceso

Algoritmos Distribuidos

- Ejemplo:



Algoritmos Distribuidos

- Exclusión mutua con **algoritmo distribuido basado en relojes lógicos:**
  - Obtener el testigo requiere *?* mensajes
  - Es bastante **costoso**
  - Cualquier proceso es punto crítico de fallo
  - Cada proceso recibe peticiones y envía respuestas, por tanto, el **cuello de botella** puede ocurrir

Algoritmos Distribuidos

- Exclusión mutua con **algoritmo basado en anillo:**
  - Los procesos se configuran en un anillo lógico (cada proceso conoce la dirección de sus vecinos)
  - El *testigo* circula en una sola dirección y el proceso que lo tiene puede acceder a la sección crítica; en caso contrario ha de esperar
  - **Suposiciones**:
    - Cada proceso conoce la dirección de su vecino por la derecha
    - Paso de mensajes **fiable**

Algoritmos Distribuidos

- Exclusión mutua con **algoritmo basado en anillo:**
  - Obtener el testigo requiere máximo *?* mensajes
  - El testigo está **continuamente circulando**
  - Si un proceso falla se ha de **reconfigurar el anillo**
  - **Regenerar testigo** si se pierde
  - **No** es posible asegurar el cumplimiento de la relación *ocurrió-antes*

Algoritmos Distribuidos

- **Elección**:
  - Se trata de escoger un único proceso de un conjunto de ellos, por ejemplo, debido al fallo de otro proceso
  - Principal **requisito** → que el proceso sea **único** (con mayor identificador) incluso si varios solicitan la **elección** simultáneamente
  - **Soluciones**:
    1. Algoritmo del valentón
    2. Algoritmo basado en anillo

Algoritmos Distribuidos

- Elección con el **algoritmo del valentón**
- **Requisitos:**
  - Los miembros del grupo se conocen, aunque puede haber caido más de un proceso aparte del coordinador
  - Paso de mensajes fiable
- Tres tipos de mensajes:
  - **Elección**: Para anunciar una elección
  - **Respuesta**: Se envía como respuesta a un mensaje de elección
  - **Coordinador**: Se envía para anunciar el id del nuevo proceso coordinador
- Costoso: se requieren hasta ? mensajes

Algoritmos Distribuidos

- Pasos **algoritmo del valentón**:
  1. Un proceso comienza una elección cuando detecta que un proceso ha fallado → Envía un mensaje de **"elección"** a los procesos con id más alto que él mismo
  2. Espera un mensaje de **"respuesta"**
  3. Si no llega el mensaje de respuesta, se proclama coordinador y envía un mensaje **"coordinador"** a todos los procesos con id más bajo que él; en otro caso, espera un tiempo a que llegue un mensaje **"coordinador"** del proceso elegido; si éste no llega comienza una nueva elección
  4. Si un proceso recibe un mensaje **"coordinador"** graba el identificador contenido en dicho mensaje
  5. Si recibe un mensaje de **"elección"** devuelve un mensaje de **"respuesta"** y comienza una elección (a menos que ya haya iniciado una)
  6. Cuando se restablece un proceso que había fallado, comienza una nueva elección. Si tiene el id más alto será el nuevo coordinador junto con el actual hasta que éste reciba el mensaje **"coordinador"**

Algoritmos Distribuidos

- Elección con el **algoritmo basada en anillo**
- **Requisitos:**
  - Los procesos están organizados en anillo lógico, aunque sin coincidir el orden con su identificador
  - Paso de mensajes fiable y los procesos no fallan durante la elección
- Los procesos indican su participación, tipos de mensajes:
  - **Elección**: Para anunciar que hay una elección en curso
  - **Coordinador**: Se envía para anunciar el id del nuevo proceso coordinador
- Costoso: se requieren hasta ? mensajes

Algoritmos Distribuidos

- Elección con el **algoritmo basado en anillo**
- **Pasos**:
  - Inicialmente, cada proceso está marcado como **no-participante**. Cualquiera puede comenzar la elección → se marca como **participante** y envía un mensaje de elección con su id de proceso a su vecino por la derecha
  - Si recibe un mensaje de **"elección"**, compara su id con el del mensaje:
    - Si es mayor que el del mensaje →
      - Si está marcado como **no participante,** sustituye el id del mensaje de elección por el suyo y lo envía
      - Si está marcado como **participante**, no envía el mensaje
    - Si es menor → pasa el mensaje de **"elección"** recibido a su vecino
    - En cualquier caso, si no lo estaba, se marca como **participante**
  - Si el *id* recibido es el del propio receptor (igual) → es el proceso con mayor id y se convierte en **coordinador**. Se marca como **no-participante** y envía un mensaje de **"coordinador"** a su vecino con su *id*
  - Si se recibe un mensaje de **"coordinador"**, se marca como **no-participante** y reenvía dicho mensaje a su vecino