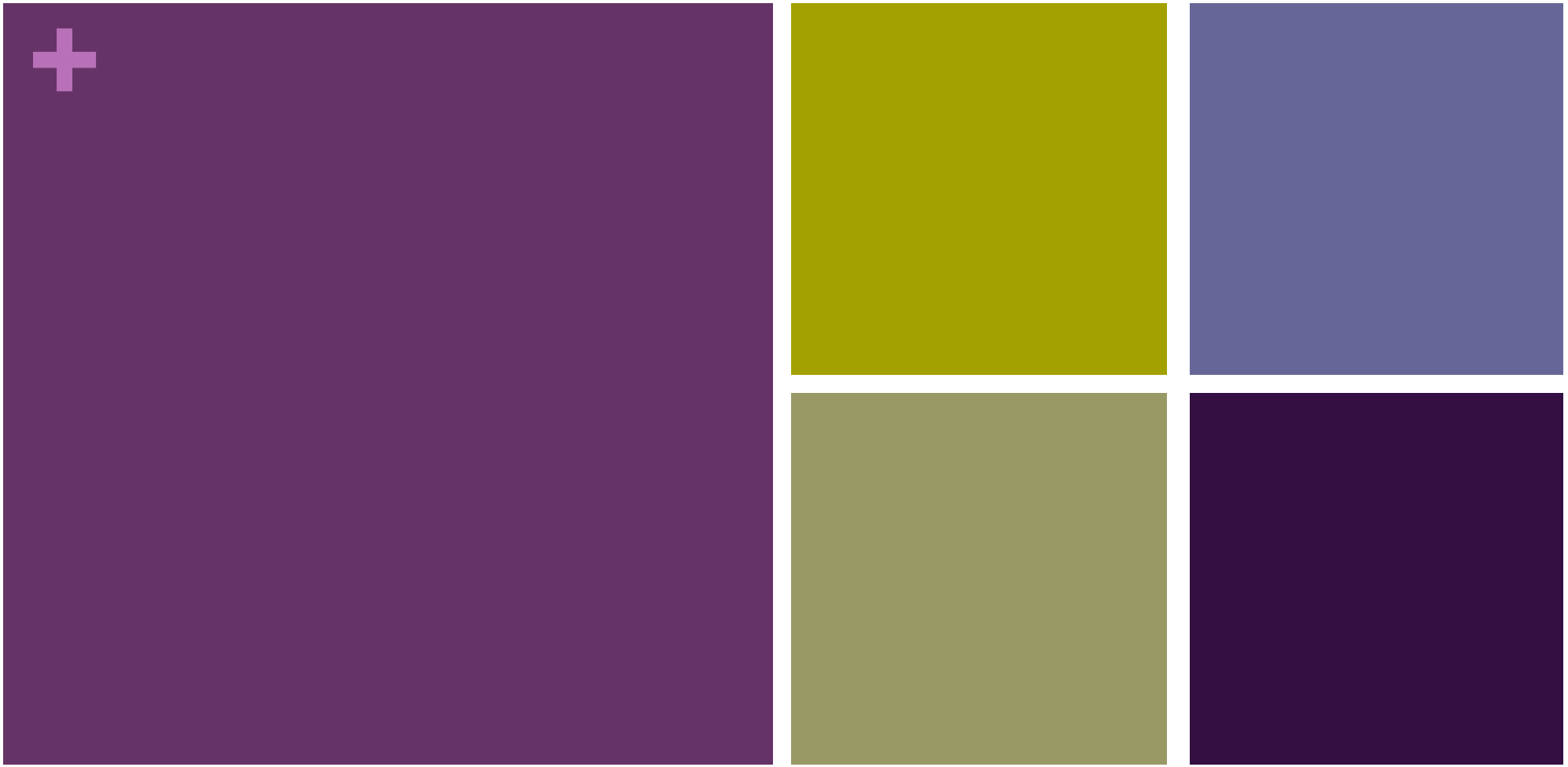


Algoritmo de Ricart y Agrawala

Sistemas Distribuidos

Rodrigo Santamaría

Objetivo

Implementar un conjunto de procesos distribuidos que autorregulen el acceso a una zona de exclusión mutua común

Descripción

- Usaremos el algoritmo de Ricart y Agrawala
  - Tema 6 – Exclusión mutua distribuida
  - Ricart, G.; Agrawala, A.K.: An Optimal Algorithm for Mutual Exclusion in Computer Networks, *Communications of the ACM*, Vol. 24, No. 1, pp. 9–17, 1981
- Las marcas temporales se obtendrán mediante el tiempos lógicos de Lamport
  - Tema 5 – Tiempo y relojes lógicos
- Como requisito auxiliar necesitamos un algoritmo tipo NTP
  - Tema 5 - Sincronización

Diseño

Despliegue

- La práctica consiste en 6 procesos java en tres nodos distintos (2 procesos por máquina)
- Se deberán desplegar los procesos en todas las máquinas desde un único nodo.

Diseño

Proceso

- Todos los procesos ejecutan el mismo programa, garantizando tiempos aleatorios independientes
  - Mediante una llamada a *sleep*, simulan la realización de un cálculo de duración aleatoria, distribuida uniformemente entre 0.3 y 0.5s
  - Luego entrarán en una sección crítica común de gestión distribuida (SC)
  - Permanecerán en ella de 0.1 a 0.3s
  - Repetirán el cálculo + estancia en SC 100 veces
  - Terminarán la ejecución de manera ordenada

Diseño

Algoritmo R&A + Lamport (N procesos)

En la inicialización (p_i)

estado = LIBERADA;

$C_i = 0$;

cola vacía;

Entrada en la SC (p_i)

estado = BUSCADA;

$T_i = C_i$

Multidifusión de la petición $<T_i, p_i>$ de entrada en SC

Espera hasta que (n° de respuestas = (N-1));

estado = TOMADA;

$C_i = C_i + 1$ // LC1

Al recibir una petición $<T_j, p_j>$ en p_i

$C_i = \max(C_i, T_j) + 1$ // LC2

si (*estado* = TOMADA o

(*estado* = BUSCADA y $(T_i, p_i) < (T_j, p_j)$*))

pon en *cola* la petición, por parte de p_j

si no

responde inmediatamente a p_j

Salida de la SC (p_i)

estado = LIBERADA;

responder a todas las peticiones en *cola*;

cola vacía;

*$(T_i, p_i) < (T_j, p_j)$ implica que $T_i < T_j$ o que $T = T_j$ y $p_i < p_j$
Para ello, los identificadores de proceso deben ser comparables (p. ej. $p_1 < p_2$)

Diseño

logs

- Cada vez que un proceso entra en la SC, escribe en un fichero local de log una línea:
  - `Px E` *`tiempo`*
    - *x* es el identificador de proceso (del 1 al 6)
    - *tiempo* es el número de milisegundos transcurridos desde el 1 de enero de 1970, según el reloj de la máquina local (`System.currentTimeMillis()`)
- Cuando abandone la SC, imprimirá
  - `Px S` *`tiempo`*
    - Mismas especificaciones que para la entrada

Diseño

Comprobación

- Al finalizar la ejecución, deben unirse los ficheros de log y verificar que no ha habido violación de la sección crítica
- Como los relojes de cada máquina no están sincronizados, debemos
  - Calcular mediante un algoritmo similar a NTP los desvíos de las máquinas respecto a una de ellas, que se tomará como referencia, y el error cometido en la medida del desvío
- La comprobación y estimación de los desvíos se hará mediante un proceso *supervisor* (puede ser uno de los 6 procesos, o un proceso aparte), que se ejecutará en una de las máquinas (*máquina de referencia*)

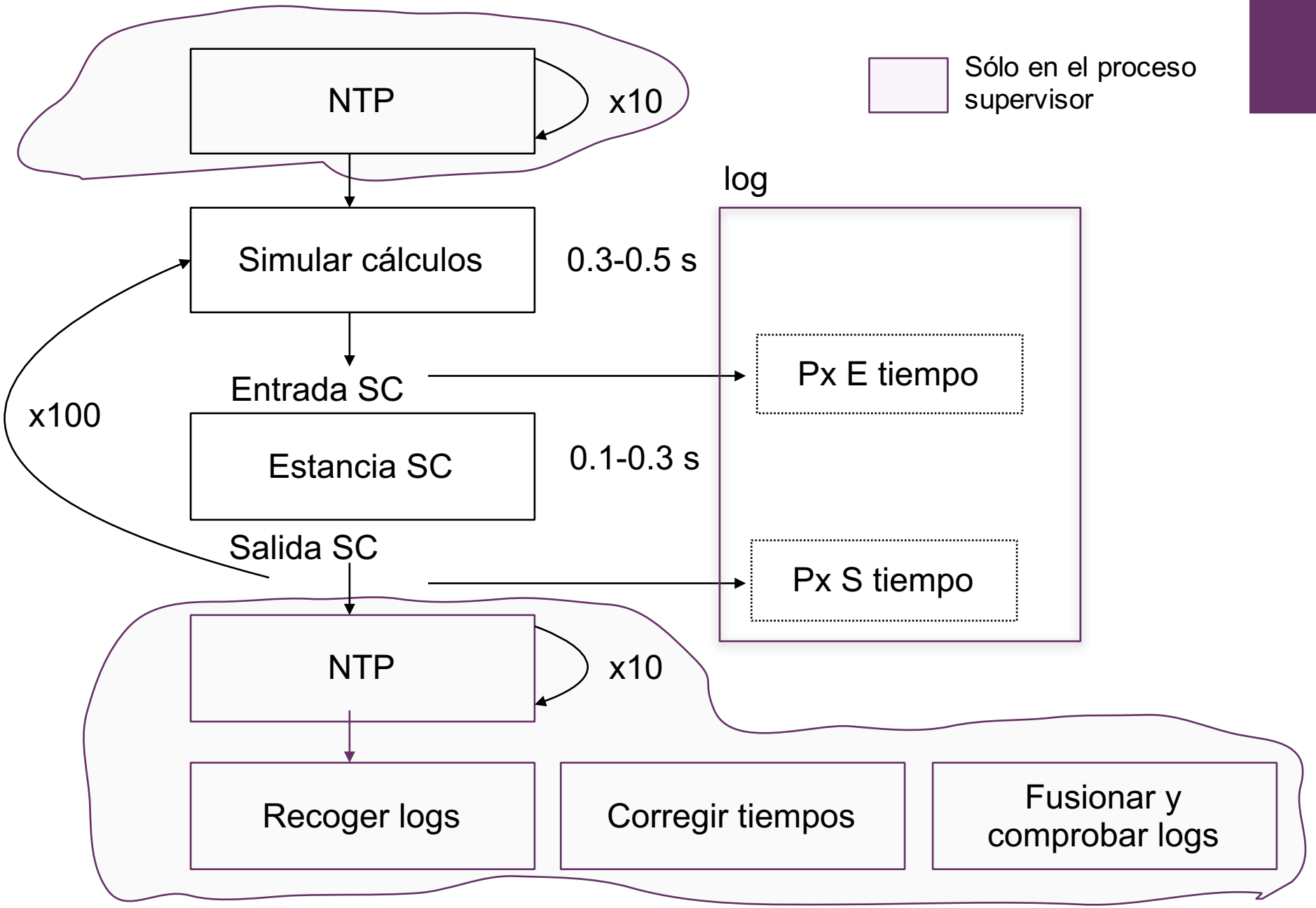
Diseño

Algoritmo "NTP"

- Se ejecutará 10 veces al principio y 10 veces al final del proceso *supervisor*, para estimar el desplazamiento (*offset*) y retardo (*delay*) cometido.
  - Tomaremos un par $<o_1, d_1>$ de la ejecución al inicio y otro $<o_2, d_2>$ de la ejecución al final, correspondientes a la iteración de menor delay de cada tanda de 10.
    - Usaremos como estimación $<o, d>$=media($<o_1, d_1>$,$<o_2, d_2>$)
    - Opcionalmente, se puede estimar $<o, d>$ con el algoritmo de Marzullo* utilizando los 20 pares.
- La comprobación sólo admitirá la simultaneidad dentro de la sección crítica si el tiempo de colisión es inferior al error obtenido en la estimación del desvío de los relojes

*En el tema 5.-tiempos (diap.27). Una buena explicación del pseudocódigo en Wikipedia

Diseño: resumen



Detalles

Requisitos

- El programa se hará utilizando REST
  - Debe funcionar en Windows y/o Linux del laboratorio, según el despliegue acordado
- Las prácticas se realizan **en parejas**
- Plazos de entrega y defensa en Studium
- Una vez pasado el plazo de entrega, la práctica no se puede modificar
- La detección de copia implica suspender automáticamente la asignatura

Detalles

Entrega

- Deben entregarse los ficheros fuente del programa, comprimidos
  - Un fichero por clase java utilizada (sólo los .java)
  - Scripts de lanzamiento
  - Cualquier otro documento que consideréis oportuno
    - .jar, informes, etc.
- La entrega se realizará por **Studium** (ambos miembros de la pareja deben realizar la entrega)
  - El nombre del fichero comprimido debe ser *Apellido1Nombre1Apellido2Nombre2*
    - Apellido1 y Nombre1 se refieren al primer apellido y nombre del primer miembro de la pareja (según orden alfabético del primer apellido)
      - Apellido2 y Nombre2 al primer apellido y nombre del otro

Detalles

Evaluación

- Para obtener un 5, basta con que funcione correctamente en Windows o Linux en el laboratorio
- Se considera correcto:
  - Que realice bien el algoritmo, sin provocar *interbloqueo* ni *inanición.*
  - *Que lo haga siguiendo los requisitos definidos*
  - La comprobación de logs no debe detectar violaciones de la SC

A partir de ahí, se valorará la calidad del trabajo:

- Documentación del código fuente
- Principios de ingeniería (modularidad, claridad, etc)
- Defensa del trabajo (evaluación individual*)

*NOTA: Una práctica correcta puede suspenderse si el/los estudiantes no son capaces de defenderla satisfactoriamente

Recomendaciones

Programar por secciones

- Programar sección crítica y escritura en logs
  - Prueba en local, con 2/3 procesos
- Incluir comunicación REST
  - Como en la fase 1
  - Prueba con 2 ordenadores y 4/6 procesos
- Incluir NTP
  - Muy parecido a la fase 2
- Recogida (scp) y fusión de logs (cat+sort)
- Comprobación SC
  - shell (awk) o java (BufferedReader)
  - O usar: http://vis.usal.es/rodrigo/documentos/aso/Comprobador.java

Bibliografía

- G. **Coulouris**, J. Dollimore, T. Kindberg and G. Blair. *Distributed Systems: Concepts and Design (5th Ed)*. Addison-Wesley, 2011
  - Sección 15.2, páginas 637-639
- Ricart, G.; Agrawala, A.K.: An Optimal Algorithm for Mutual Exclusion in Computer Networks, *Communications of the ACM*, Vol. 24, No. 1, pp. 9–17, 1981
  - Disponible en http://vis.usal.es/rodrigo/documentos/sisdis/papers/Ricart1981.pdf

FAQ

Frequently Asked Questions

FAQ

¿Cada proceso es un servidor o un cliente?

- Cada proceso deberá tener ambos aspectos
  - Enviará mensajes a otros procesos
    - Por ejemplo, para entrar o conceder acceso a la SC
  - Atenderá peticiones de otros procesos
    - Por ejemplo, peticiones de entrada en la SC o recepción de permisos de entrada

FAQ

¿Cómo selecciono el proceso al que quiero invocar?

- Con REST y Tomcat sólo podemos invocar métodos de una clase, sin embargo, hablamos de tener dos instancias de la misma clase Proceso en ejecución que ofrecen servicios. ¿Cómo puedo acceder una en concreto?

- Hay varias opciones:
  - Lanzar varios servidores, escuchando en distintos puertos
    - Proceso 1: http://ip:8080/Proceso/servicio
    - Proceso 2: http://ip:8081/Proceso/servicio
  - Utilizar como servicio de cara al exterior una clase @Singleton Despachador que mantenga un array con los dos procesos, acepte el id del proceso como argumento en la url, y derive la petición al proceso correspondiente
    - Proceso 1: http://ip:8080/Despachador/servicio?id=proceso1
    - Proceso 2: http://ip:8080/Despachador/servicio?id=proceso2
  - O cualquier otro método que se os ocurra, mientras cumpla con las condiciones del ejercicio

FAQ

¿Cómo lanzo los procesos si son a la vez servicios?

- De alguna manera, el servicio debe tener conocimiento del estado del proceso o procesos a los que representa
- Se puede implementar un servicio de 'arranque' de procesos que inicie/ponga en ejecución los procesos indicados.