Práctica Obligatoria

-Julia García Vega y Jorge Pozo Zapatero-

Introducción

En este informe se va a explicar como se ha realizado la práctica obligatoria que consiste en realizar la implementación de un conjunto de procesos en distintos nodos de ejecución para que autorregulen el acceso a una zona de exclusión mutua.

Se comentarán los distintos archivos que se utilizarán en el proceso de ejecución de la práctica de manera detallada.

Resumen de las clases

Clase Cliente

Comenzaremos comentando la clase Cliente, esta será el aspecto de los procesos encargados de enviar mensajes a otros procesos. Además uno de los procesos tendrá más funciones, siendo este el proceso supervisor que se encargará de realizar el algoritmo NTP y coordinar el comienzo de todos los procesos para la iteración del bucle de entrada y salida de la sección crítica autorregulada.

Como parámetros de entrada de esta clase, tendremos primero la id que tendrá nuestro proceso. Como tenemos que invocar 6 procesos las ids irán del 1 al 6. También guardaremos la dirección de la máquina además del puerto por el que escuchará la parte del servidor del proceso en cuestión, que será a través del que recibamos las peticiones de los otros procesos. El orden de estas direcciones siempre es primero la del proceso y luego la del otro proceso en la misma máquina, a continuación la del proceso id 1 e id 2, y por último las de la otra máquina.

En primer lugar, obviamente, iniciaremos las variables que vamos a utilizar en la clase.

Entraremos ahora en una sección donde los procesos, salvo el proceso con id 1 que será el proceso supervisor, construirán la uri para comunicarse con el proceso supervisor, es decir, el de id=1. A continuación todos llamarán al servicio “esperarSupervisor” de la parte del servidor del proceso supervisor que servirá para esperar a que todos los procesos se hayan iniciado y esperen todos juntos a que el

proceso supervisor realice las funciones que tiene que realizar antes de comenzar con la entrada y salida de la sección crítica de gestión distribuida.

Hacemos que estos procesos que esperen para que el proceso supervisor realice el algoritmo NTP para poder estimar luego el desplazamiento y retardo cometido y que de esta forma empiecen todos a la vez a realizar el bucle de la sección crítica.

Una vez el proceso supervisor ya ha obtenido el mejor par con el algoritmo NTP simplemente construye su uri con la dirección de su propia parte de servidor, puesto que es aquí donde están los procesos esperando, y llama al servicio "esperarSupervisor" donde puede que tenga que esperar si alguno de los procesos no ha llegado, en caso de que ya hayan llegado todos (los 6 incluido el propio proceso supervisor) simplemente desbloquea todos los procesos.

Ahora que ya el proceso supervisor ha dado paso a todos los procesos, todos estos continuarán con la ejecución del programa. Crearán en caso de que no exista el fichero local de log: "log.txt". Una vez realizado esto simplemente comenzarán con el algoritmo de Ricart y Agrawala.

Todos los procesos comenzarán por invocar el servicio de sus propios servidores de "inicio" donde simplemente se inicializarán las variables relacionadas a la sección crítica.

De seguido, ya entran en el bucle para entrar y salir de la sección crítica. Donde comenzarán durmiendo entre 0.3 y 0.5 segundos, para simular un cálculo, y continuarán llamando al servicio "busco" de su propio servidor para cambiar el valor de las variables para indicar su búsqueda de la sección crítica.

Ahora que los procesos ya quieren entrar en la sección crítica tendrán que realizar las peticiones al resto de procesos mediante las direcciones que les hemos pasado por argumento. Realizarán las 5 peticiones a los servicios de "petición" de los distintos servidores.

Dentro de estos servicios se quedarán bloqueados hasta que les permitan entrar en la sección crítica, es por esto que cuando recibimos la respuesta de todos ya podrán invocar el servicio "tomo" de su propio servidor. Que hace referencia a que entrarán en la sección crítica. Continuará realizando el correspondiente sleep dentro de la sección crítica para simular la realización de cálculos. Al entrar a la sección crítica y antes de salir escriben en el log la información correspondiente (id+entrada/salida+tiempo actual).

Por último antes de volver a iterar el bucle simplemente mediante la invocación del servicio "salgo" de su servidor notificará que ha salido de la sección crítica. Con este servicio, se cambiará el valor de las variables relacionadas con la sección crítica y

se notificará a aquellos procesos que estuvieran esperando a la respuesta de este proceso de que ya ha salido de la sección crítica y él les permite continuar.

En último lugar el proceso supervisor (id=1) realizará de nuevo el algoritmo de NTP y utilizará los dos mejores pares obtenidos antes y después del bucle de la sección crítica para quedarse con la media.

Clase ParNTP

Esta clase sirve simplemente para guardar los valores de los pares de NTP (offset y delay). Y para poder acceder a ellos.

Clase Servidor

Respecto a las clases simplemente nos falta comentar la clase del servidor. Se ejecutará uno por cada proceso que invoquemos, es decir 6 en total.

Este servidor ofrecerá servicios tanto al proceso del que es parte como al resto de procesos.

El servidor guardará también una serie de variables principalmente para poder realizar el algoritmo de Ricart y Agrawala.

Todos los servicios que ofrece van a ser utilizados unicamente por uno de los procesos salvo el servicio “petición” y el servicio “esperarSupervisor” que ofrecen los servicios a todos los procesos.

El primero de los servicios “pedirTiempo” simplemente lo utilizará el proceso supervisor para llevar a cabo el algoritmo de NTP. Este servicio devolverá los tiempos obtenidos.

El servicio “esperarSupervisor” que ya hemos explicado su funcionalidad levemente hace esperar a los procesos hasta que todos hayan llegado a invocar a este servicio.

Continuamos con el servicio “inicio” donde cuando lo llaman los procesos en sus respectivos servidores simplemente inicia las variables necesarias para llevar a cabo el algoritmo de Ricart y Agrawala.

El servicio “busco” sirve para actualizar las variables para saber que el proceso busca entrar en la sección crítica.

El servicio “peticion” maneja las peticiones del resto de procesos para entrar en la sección crítica.

Por último los servicios “tomo” y “salgo” que sirven para actualizar los estados cuando el proceso entra en la sección crítica y para notificar de que se ha salido de la sección crítica respectivamente.

Clase CorregirTiempos

El objetivo de esta clase es corregir los tiempos recogidos en el log.txt de in procesos según el offset de la máquina de ese proceso respecto a la máquina del proceso de id 1. Como argumentos se pasa primero el nombre de la carpeta que contiene los ficheros que debe estar situada en la carpeta descargas, a continuación el nombre del fichero log y por último el offset.

Para realizar esto, se crean dos objeto de File: uno para el log.txt existente y otro para un fichero corregidolog.txt nuevo en el cual se escribirán los tiempos corregidos por el offset.

El fichero log.txt mediante objetos FileWriter y BufferedWriter se lee línea a línea hasta su fin. Por cada línea, se separan los datos que contiene para obtener el tiempo que se corrige restando el offset. A continuación, mediante objetos FileReader y BufferedReader, se escribe lo leído del fichero log.txt pero con los tiempos corregidos. Para imprimir el tiempo que es un double con el formato correcto se ha utilizado las clases DecimalFormatSymbols y DecimalFormat. Por último, se cierran los objetos FileWriter, BufferedWriter, FileReader y BufferedReader.

Scripts utilizados

Lanzar.sh

Mediante este script lanzaremos todos los servidores y clientes que necesitamos desde un único ordenador. Nos ayudaremos del comando ssh para ejecutar órdenes remotas en los dos ordenadores remotos.

Como dice el enunciado se lanzarán 2 procesos en cada nodo.

Este script necesita de 4 parámetros: el nombre de la carpeta ubicada en descargas donde se encuentran los archivos de la práctica (también tiene que haber en esa ubicación el zip correspondiente a esa carpeta con el mismo nombre) y las 3 direcciones ip de los dispositivos siendo primera la local.

Mediante el comando ssh accederemos a las máquinas remotas utilizando las direcciones ip que hemos pasado por parámetro, gracias a este comando podremos mover ficheros entre nuestra máquina local y las remotas y ejecutar comandos en estas también. Además mediante el fichero ssh/authorized_keys nos permitirá acceder a este utilizando las claves públicas y privadas generadas anteriormente con el comando ssh-keygen.

A continuación, transferimos el zip con todos los archivos a los otros dos ordenadores mediante el comando scp y con el comando ssh y unzip descomprimiremos estas carpetas en los ordenadores remotos. De esta forma ya los tres ordenadores contendrán los mismos archivos necesarios para la ejecución del programa.

Continuamos lanzando los servidores tanto en la máquina local como en las otras dos máquinas remotas utilizando Tomcat 9.0. Por cada máquina se lanzan dos Tomcat distintos que escuchan uno por el puerto 8080 y otro por el puerto 8081. En la carpeta WebApps de ambos Tomcat se encuentra el archivo war que contiene nuestro servidor. Estos servidores se lanzan ejecutando para ambos Tomcat catalina.sh start que se encuentra en la carpeta bin de Tomcat.

Y por último lanzamos 2 clientes, a partir del jar que contiene las clases cliente y parNTP, en cada máquina con los parámetros adecuados que serían el id y las direcciones de los 6 distintos nodos y los puertos por los que escucharán sus servidores como se ha explicado en la clase cliente anteriormente.

comprobar.sh

Utilizaremos este script simplemente para comprobar si ha habido errores en el acceso a la sección crítica.

Este script se necesita de 8 parámetros. Los cuatro primeros son iguales que los del script lanzar.sh: el nombre de la carpeta ubicada en descargas donde se encuentran los archivos de la práctica (también tiene que haber en esa ubicación el zip correspondiente a esa carpeta con el mismo nombre) y las 3 direcciones ip de los dispositivos siendo primera la local. Los otros cuatro corresponden a los offsets y delays: primero el offset y delay de la primera máquina remota que se pasa como parámetro y luego el offset y delay de la otra máquina remota.

Al ejecutarlo, primero transferimos de los ordenadores remotos sus respectivos archivos logo.txt y los guardamos en el ordenador local. A continuación, compilamos CorregirTiempos.java y lo lanzaremos con los ficheros de log1.txt y log2.txt que

hemos creado localmente a partir del ficheros de log de las otras máquinas remotas y con el respectivo offset.

Una vez se tienen todos los logs corregidos, los fusionamos, los ordenamos según el tiempo y los comprobaremos con la clase Comprobador.java ofrecida por el profesorado de la asignatura.

Y por último, paramos todos los servidores de tomcat utilizados mediante el script catalina.sh y con el argumento stop.

Conclusión

Se ha conseguido realizar el objetivo propuesto de realizar la implementación de 6 procesos que autorregulen el acceso a una zona de exclusión mutua común mediante la utilización del algoritmo de Ricart y Agrawala. Además de la corrección de los tiempos mediante el algoritmo NTP y la comprobación de que no haya habido errores en los accesos de estos procesos a la sección crítica mediante los ficheros logs generados y su comprobación.

Además esta implementación y puesta en marcha se puede llevar a cabo desde un único equipo sin tener que interactuar directamente con los otros dos nodos de ejecución en los que se ejecutan los procesos.