Programación Concurrente - Práctica final

El objetivo es poner en práctica los conocimientos adquiridos sobre programación distribuida y concurrente.

La práctica consta de dos partes:

La primera parte implementa un servidor de ficheros concurrente que espera la conexión de un cliente al que enviar contenido.

La segunda parte implementa una aplicación en la que el intercambio de información se realiza directamente entre los propios clientes..

Funcionamiento del programa

CLIENTE

Inicia la sesión del usuario, pidiéndole a éste el nombre para poder hacer. Después le ofrece dos opciones: consultar el nombre de todos los usuarios conectados y la información que poseen, y descargar información.

El proceso de descarga será un proceso a parte, permitiendo que el usuario realice otras acciones, e incluso otras descargas, al tiempo que se descarga la primera información.

Además, el cliente es capaz de emitir información compartida, actuando como propietario de una información solicitada por otro cliente.

Al término de la aplicación se comunica el fin de sesión al servidor, permitiendo así que éste actualice apropiadamente su base de datos.

SERVIDOR

Al iniciarse la aplicación, se lee un fichero “users.txt” con la información de los usuarios registrados en el sistema así como los datos relativos a éstos.

A partir de este momento, el servidor atiende de forma concurrente todas las peticiones que realizan los clientes conectados al sistema. Dichas peticiones consisten en solicitar la búsqueda de usuarios conectados consultando la base de datos, y solicitar la descarga de información que ha solicitado el cliente.

Para hacer ésto, gestiona el inicio de la comunicación p2p entre los clientes. Una vez establecida la conexión se desentiende de la comunicación.

Por último, al finalizar sesión, se actualiza apropiadamente la base de datos.

Puntos críticos

Las secciones críticas del programa se encuentran en las tres tablas de información compartida que tenemos en el Server.

Hemos detectado conflictos al escribir, leer y borrar información en dichas tablas, identificando el problema de lectores y escritores.

Para solventar estos conflictos sin romper la propiedad de exclusión mutua hemos utilizado dos **Semaphores**, de la siguiente manera:

Por una parte, agrupamos las acciones de escribir y borrar, ya que consideramos borrar como una "no escritura", y en ambas acciones se modifican valores en la tabla. Estos dos procesos se gestionan utilizando el mismo Semaphore.

Por otro lado tenemos la lectura, gestionada con el segundo Semaphore.

Por último, también se emplean **Monitors**, haciendo uso de la palabra reservada "**synchronized**" para garantizar que no haya varios Threads intentando acceder al mismo tiempo a los recursos de memoria compartidos, bien sea para escritura/borrado o lectura de los mismos.

Integrantes del grupo: **Manuel Nevado y Sofía Capmany**