Prácticas 2019/2020, *convocatoria extraordinaria*

SISTEMAS DISTRIBUIDOS

El objetivo principal del proyecto es diseñar un **sistema cliente-servidor que permita la descarga de ficheros**. La implementación de este proyecto permitirá al alumno trabajar, **mediante ZeroC Ice**, los siguientes aspectos:

- Transparencia de localización
- Manejo de canales de eventos
- Despliegue de servidores

Arquitectura del proyecto

El sistema estará formado por cinco tipos de componentes: *senders*, encargados del envío de ficheros; *transfers*, para la gestión de la transferencia de cada archivo; *receivers*, empleados para la recepción de archivos; *clientes*, que solicitarán ficheros; y canales de eventos para la comunicación de estados entre componentes.

Descripción del sistema

El sistema estará compuesto por un *cliente* llamado *FileDownloader con* una factoría de *receivers*, un servidor con una factoría de *transfers* y otro servidor con una factoría de *senders*. El *cliente*, que recibirá como argumentos el nombre de los ficheros, solicitará una transferencia de N ficheros al servidor donde están ubicados los *transfers*. Este creará un *transfer* para controlar la transferencia, que a su vez creará una pareja *receiver-sender* para cada uno de los ficheros. Cuando todas las parejas estén listas el cliente iniciará la transferencia, es decir, el envío entre las parejas. Los *receivers* notificarán al *transfer* cuando hayan terminado, y este destruirá al *receiver* que envía la notificación y a su *sender* compañero. Si todas las parejas han terminado, el *tranfer* informará al cliente (estará a la espera), que destruirá el *transfer* y terminará su ejecución.

También será importante el control de fallos, al menos de los que pueden tener lugar por acciones del cliente, para garantizar el correcto funcionamiento del lado del servidor. Por ejemplo, en el caso de que se hagan solicitudes de ficheros que no existen la transferencia se daría por fallida, incluso cuando solamente uno de los ficheros solicitados no existiese, destruyendo todos los componentes relacionados con la transferencia e informando al cliente de lo sucedido.

Senders

El *sender* es el componente encargado del **envío de un fichero** y es **creado bajo demanda mediante una factoría de objetos**. Su funcionamiento consiste en lo siguiente:

- Es creado para el envío de un fichero concreto por una factoría de objetos.

- `Sender* create(string fileName) throws FileDoesNotExist;`
  devuelve un objeto *sender* al que se le ha asignado el fichero `fileName`, siempre que exista.

- Envía los datos de su fichero cuando se lo solicitan.
  - `string receive(int size)`
    manda un chunk del fichero del tamaño indicado.
  - `void close()`
    cierra el fichero.
- Es destruido por su *transfer* al finalizar el envío.
  - `void destroy()`
    elimina al *sender* del adaptador y termina su ejecución.

Receivers

El *receiver* es el componente encargado de la **recepción de un fichero** y es **creado bajo demanda mediante una factoría de objetos en la aplicación cliente**. Su funcionamiento consiste en lo siguiente:

- Es creado para la recepción de un fichero concreto por una factoría de objetos.
  - `Receiver* create(string fileName, Sender* sender Transfer* transfer);`
    devuelve un objeto *receiver* al que se le ha asignado un compañero *sender* y un fichero `fileName`. Se indica el *transfer* que lo creó para que pueda ser notificado al terminar.
- Solicita el envío de datos a su compañero *sender*.
  - `void start();`
    inicia la recepción del fichero.
  - `void peerFinished(PeerInfo peerInfo);`
    notifica a todos los *transfers* que el fichero está listo.
- Es destruido por su *transfer* al finalizar la recepción.
  - `void destroy()`
    elimina al *receiver* del adaptador y termina su ejecución.

Transfers

El *transfer* es el componente encargado de **gestionar la transferencia de ficheros**, y **son creados bajo demanda mediante una factoría de objetos**. Su funcionamiento consiste en:

- Es creado por una factoría de objetos para una transferencia concreta de N ficheros.
  - `Transfer* newTransfer(ReceiverFactory* receiverFactory);`
    devuelve un objeto *transfer.* Necesita `receiverFactory` para la creación de *receivers* en el lado del cliente.
- Una vez el cliente dispone de su objeto *transfer:*
  - `void createPeers(FileList files)`
    crea una pareja *receiver-sender* por cada fichero que se desea transferir.

    - `void destroyPeer(string peerId)`
      destruye la pareja *receiver-sender* que se corresponde con un ID dado.

- Es destruido por el cliente que lo creó al finalizar la transferencia, es decir, cuando no quedan archivos por transferir.
    - `void transferFinished(Transfer* transfer);`
      notifica a todos los clientes que la transferencia del objeto `transfer` ha finalizado.
    - `void destroy()`
      elimina al *transfer* del adaptador y termina su ejecución.

Ejecución y evaluación

Deberán lanzarse el *registry* (es necesario implementar transparencia de localización), la factoría de *transfers* y la factoría de *senders*, almacenando los archivos que pueden transferirse en un directorio *files/* en el raíz del proyecto. Se ejecutarán varios clientes de forma simultánea y se comprobará que pueden solicitar transferencias de ficheros y que estos son recibidos sin problema. En caso de que se produzca algún error durante cualquiera de las transferencias se asegurará la correcta destrucción de los componentes relacionados con la transferencia fallida.

El .zip entregado ha de contener:

- `file_downloader.py`: programa cliente.
- `client.config`: configuración del cliente.
- `sender_factory.py`: factoría de objetos *sender*.
- `senders.config`: configuración del servidor de *senders*.
- `transfer_manager.py`: manager de transferencias, factoría de objetos *transfer*.
- `transfers.config`: configuración del manager de transferencias.
- `icebox.config`: configuración de *icebox*.
- `icestorm.config`: configuración de *icestorm*.
- `registry.config`: configuración del *registry*.
- `trawlnet.ice`: interfaz del sistema.
- `README.md`: contendrá el enlace al repositorio, los nombres de los alumnos que componen el grupo y el manual de usuario con la especificación de cómo lanzar el sistema.
- `run_client.sh`: script para la ejecución del cliente.
- `run_server.sh`: script para la ejecución del lado del servidor.
- `Makefile`: reglas para la ejecución del sistema.

Los scripts de ejecución y el Makefile serán proporcionados por los profesores. Serán utilizados para la puesta en marcha del sistema, por lo que este deberá ajustarse al despliegue que realizan evitando en la medida de lo posible modificaciones por parte del alumno, salvo en aquellos apartados en los que se especifique lo contrario.

También podrán entregarse librerías auxiliares (`utils.py`, por ejemplo).

En caso de que se haya presentado el proyecto de la convocatoria ordinaria los alumnos deberán crear una rama llamada `extra` en el repositorio utilizado en esa primera convocatoria, donde irán subiendo los cambios de la entrega de la convocatoria extraordinaria durante su desarrollo.