Sistemas Distribuidos – Laboratorio Parte 1

Rev 2

El objetivo principal del proyecto es **desarrollar un sistema distribuido basado en varios servicios.** Para ello, se implementará un sistema de gestión de archivos de forma remota similar a One Drive de Microsoft o Google Drive de Google. El sistema estará dividido en tres servicios diferentes. Cada alumno tendrá que implementar sólo 1 de dichos servicios.

Los objetivos específicos de la práctica son:

- ✓ Familiarizarse con la invocación a métodos remotos.
- ✓ Diseñar algoritmos tolerantes a fallos comunes en sistemas distribuidos.
- ✓ Fomentar el trabajo en equipos multidisciplinares.

Estructura general

El servicio como base.

Cada servicio consistirá en un programa escrito en Python 3 que presentará una interfaz de objeto Ice con la que podrán interactuar otros servicios y aplicaciones de cliente. En éste primer entregable, los servicios **no necesitarán interactuar entre ellos**.

Cada servicio del sistema se implementará con un servidor Ice que tendrá tantos objetos como sean necesarios para la implementación del mismo.

Servicio de autenticación.

Este servicio será el encargado de la gestión de los usuarios del sistema y de la autenticación de los mismos. Desde una aplicación cliente se podrá crear un usuario, y una vez creado, podremos usar las credenciales para realizar una autenticación en el sistema, o bien para eliminar dicho usuario.

Por ahora, la autenticación no estará integrada con el resto de los servicios, con el fin de que pueda ser desarrollado y probado de manera totalmente independiente, pero para la segunda entrega, la autenticación será necesaria para utilizar todos los demás servicios.

Los usuarios deben ser persistentes en el sistema; es decir, los usuarios que hayan sido creados y sus credenciales deben estar almacenados en algún almacenamiento externo (archivo de texto en el sistema de archivos, base de datos, etc). Queda a elección del alumno la implementación de dicha persistencia.

Interfaz ::IceDrive::Authentication. Casos de uso.

La interfaz se define de la siguiente manera:

```
interface Authentication{
   User* login(string username, string password) throws Unauthorized;
   User* newUser(string username, string password) throws UserAlreadyExists;
   void removeUser(string username, string password) throws Unauthorized; // Hide
UserNotExist to avoid showing too much info
   bool verifyUser(User *user); // checks if the proxy is created by a valid instance of
Authentication, not if the credentials are still valid.
};
```

1) **Crear un usuario:** la aplicación invoca al método "newUser" con el nombre del usuario y su contraseña. Si el usuario ya existe, se lanzará la excepción UserAlreadyExists. Si el usuario no existe, se añadirá a la persistencia y se creará un objeto que cumpla con la interfaz User, que representará al usuario activo y que tendrá una validez de 2 minutos (ver la interfaz User más adelante).
2) **Eliminar usuario:** la aplicación invoca al método "removeUser" utilizando las credenciales del usuario. Si las credenciales son inválidas o el usuario no existe, se lanzará la excepción Unauthorized. Si las credenciales son correctas, se eliminará el usuario y, además, se eliminarán los objetos User que puedan estar añadidos al adaptador de objetos que representen a dicho usuario.
3) **Autenticar usuario:** la aplicación invoca al método "login" con las credenciales de usuario. Si no son correctas, se lanzará la excepción Unauthorized. Si las credenciales son válidas, se creará un objeto "User" que representará a dicho usuario y con una validez de 2 minutos.
4) **Verificar usuario:** esta operación únicamente verificará que el objeto User recibido corresponde a un objeto válido de este servicio. Esta operación en ningún caso comprobará si el usuario es activo o las credenciales han caducado: **únicamente se comprobará que el objeto existe en el adaptador de objetos local**. La razón de ser de esta operación es la comprobación de que un objeto User es válido y no un objeto User creado por cualquier otra aplicación con fines maliciosos.

Interfaz ::IceDrive::User. Casos de uso

La interfaz User es la representación de un usuario autenticado en el sistema. La interfaz es la siguiente:

```
interface User{
   string getUsername();
   bool isAlive();
   void refresh() throws Unauthorized, UserNotExist;
};
```

1) **Recuperar el nombre del usuario:** Para saber el nombre de usuario al que este objeto representa, se invocará al método "getUsername". Este método deberá devolver el nombre de usuario **siempre**, independientemente de que el objeto "User" siga estando activo o no.
2) **Validez de la autenticación:** Cuando el objeto "User" sea creado, ya sea al crear un usuario o al hacer "login", las credenciales tendrán una validez de 2 minutos. Durante ese periodo, el método "isAlive" debe devolver "verdadero", y una vez pasado el periodo de validez, deberá devolver "falso". Es importante tener en cuenta para saber el periodo de validez lo explicado en el siguiente caso de uso.

3) **Renovación de la autenticación:** como se enuncia anteriormente, la validez de los objetos "User" es únicamente de 2 minutos desde que son creados. Para poder extender la validez del objeto, se puede invocar al método "refresh". Dicho método fallará con Unauthorized o UserNotExist dependiendo de si las credenciales han sido modificadas o el usuario ha dejado de existir. En el resto de los casos, el método se ejecutará y las credenciales serán renovadas por 2 minutos más.

Servicio de directorio.

Este servicio permite crear una estructura de árbol en la que cada nodo puede almacenar enlaces a ficheros de manera similar a como funciona un sistema de ficheros de disco. El servicio se implementa mediante dos interfaces: ::IceDrive::Directory y ::IceDrive::DirectoryService.

Los datos manejados por este servicio han de ser persistentes, es decir: cuando el servicio se inicia, debe recuperar el estado que tenía antes de detenerse. Queda a elección del alumno utilizar la opción de persistencia que prefiera, ya sea mediante el uso de librerías de terceros o mediante implementación propia. Es muy recomendable que el directorio donde se almacene el estado pueda ser configurado mediante alguna propiedad y si ésta no se defina se use algún valor por defecto como por ejemplo el *current working directory*.

Interfaz ::IceDrive::DirectoryService. Casos de uso.

La interfaz se define en slice de la siguiente manera:

```
interface DirectoryService {
    Directory* getRoot(string user);
};
```

Esta interfaz tiene únicamente el método de acceso al servicio, los casos de uso que se deben contemplar son los siguientes:

1) Se obtiene un directorio raíz de un usuario que nunca lo ha solicitado anteriormente: en ese caso, se creará un directorio raíz nuevo y se devolverá. El servicio no necesita verificar que el usuario sea válido.
2) Se obtiene un directorio raíz de un usuario que lo solicitó anteriormente: se retorna el directorio que se creó. No es necesario verificar que el usuario sea válido.

Interfaz ::IceDrive::Directory. Casos de uso.

Se define la interfaz de la siguiente forma:

```
interface Directory {
    Directory* getParent() throws RootHasNoParent;
    Strings getChilds();
    Directory* getChild(string childName) throws ChildNotExists;
    Directory* createChild(string childName) throws ChildAlreadyExists;
    void removeChild(string childName) throws ChildNotExists;

    Strings getFiles();
    string getBlobId(string filename) throws FileNotFound;
    void linkFile(string fileName, string blobId);
    void unlinkFile(string fileName) throws FileNotFound;
};
```

Los casos de uso positivos (los negativos no se especifican, pero se pueden extraer atendiendo a las definiciones de los métodos:

1) Se obtiene la lista de ficheros del directorio mediante *getFiles()*, que devuelve una lista de cadenas en las que cada cadena es un fichero del directorio. Si no hay ficheros, se devuelve una lista vacía.
2) Se obtiene el *blobId* (una cadena) de un fichero mediante su nombre y el método *getBlobId()*.
3) Se crea un nuevo fichero especificando un nombre y su *blobId* utilizando el método *linkFile()*. No es necesario verificar que el *blobId* es válido.
4) Se elimina un fichero indicando su nombre y llamando a *unlinkFile()*.
5) Se obtiene la lista de cadenas que contienen los nombres de subdirectorios mediante *getChilds()*. Si el directorio no tiene ningún subdirectorio se obtiene una lista vacía.
6) Se obtiene un directorio hijo mediante su nombre y la función *getChild()*.
7) Se crea un nuevo directorio mediante un nombre invocando al método *createChild()*. El *parent* del nuevo directorio debe ser el directorio actual.
8) Se elimina un directorio hijo mediante su nombre y el método *removeChild()*.
9) Se obtiene el directorio superior mediante *getParent()*. Si el directorio no tuviese uno de más alto nivel (nodo raíz) la función deberá retornar valor nulo (o *None* en Python).

Servicio de almacenamiento.

Es el servicio que permite almacenar el contenido de los ficheros del sistema de archivos remoto. Al contenido de un fichero se le denomina *blob* (***b**inary **l**arge **ob**ject*). El servicio indexará cada blob con un identificador único que se obtendrá al almacenar un blob nuevo. Además, el servicio debe ahorrar espacio, de manera que cuando se suba un archivo que ya exista, no deberá volver a almacenar el nuevo blob y simplemente devolverá el identificador del blob previamente almacenado. La forma más sencilla para lograr esto es utilizar sumas hash como el MD5 o el SHA256 como identificador. De esta forma se puede saber si un blob ya está almacenado, simplemente recalculando su hash. La opción implementada por el cuerpo docente utiliza **SHA256** para calcular el identificador de un blob.

Además, el servicio mantendrá el número de veces que un blob se encuentra "enlazado" en el sistema de ficheros. Cuando un blob deje de estar enlazado por el sistema de archivos, automáticamente deberá eliminarse.

El servicio se realiza mediante dos interfaces: ::IceDrive::DataTransfer y ::IceDrive::BlobService. La primera interfaz se usa para gestionar las transferencias de ficheros en ambos sentidos (subida y descarga de blobs) y la segunda implementa el servicio en sí.

Este servicio es persistente, de manera que su estado debe guardarse antes de finalizarse y recuperarse antes del inicio. Es muy recomendable que se pueda definir mediante una propiedad el directorio que se utilizará para almacenar los archivos de persistencia y los propios blobs.

Interfaz ::IceDrive::DataTransfer. Casos de uso

La interfaz es la siguiente:

```
interface DataTransfer{
    Bytes read(int size) throws FailedToReadData;
    void close();
};
```

Una instancia de este objeto es creada por el agente que va a realizar el envío de datos. Para lo cual, deberá disponer de un descriptor de fichero o de otro tipo de variable con la misma semántica. El

destinatario de los datos irá solicitando bloques de bytes hasta que la transferencia se complete. Se detecta que la transferencia finalizó porque el bloque de bytes leído tiene menor longitud que el bloque solicitado.

1) El destinatario de la transferencia solicita el siguiente bloque de bytes mediante *read()*. La función retorna un bloque de bytes del tamaño solicitado.
2) El destinatario de la transferencia solicita el siguiente bloque de bytes mediante *read()*. Se retorna un bloque de bytes de menor tamaño si ya no hay más datos para leer.
3) El destinatario de la transferencia ya no desea seguir con la transferencia y finaliza llamando a *close()*. Esta operación debe eliminar la instancia de ::IceDrive::DataTransfer correspondiente.

Interfaz ::IceDrive::BlobService. Casos de uso

Interfaz BlobService:

```
interface BlobService{
    void link(string blobId) throws UnknownBlob;
    void unlink(string blobId) throws UnknownBlob;

    String upload(DataTransfer* blob) throws FailedToReadData;
    DataTransfer* download(string blobId) throws UnknownBlob;
};
```

Es la clase de entrada al servicio. Los casos de uso normales del servicio son:

1) Subir un nuevo blob: el usuario creará un objeto ::IceDrive::DataTransfer que pasará al método *upload()*. Este método, cuando finalice la transferencia, devolverá el *blobId*.
2) Descargar un blob: dado un *blobId*, la función *download()* devolverá una instancia de ::IceDrive::DataTransfer para permitir la descarga del mismo.
3) Se incrementa el número de veces que un blob se encuentra enlazado llamando a *link()* y pasando el *blobId* correspondiente.
4) Se decrementa el número de veces que un blob se encuentra enlazado llamando a unlink() y pasando el *blobId* correspondiente. Cuando el blob no se encuentre enlazado, se eliminará.

Entrega

La fecha de entrega para el primer entregable será el 10 de diciembre de 2023 y consistirá en un enlace a un repositorio del código en GitHub a través de la plataforma de CampusVirtual. Los profesores de prácticas de la asignatura deben tener acceso a dicho repositorio, que debe ser privado para evitar plagios.

El repositorio deberá incluir todo el código fuente tanto del servicio asignado al alumno como de aquellos programas de prueba que haya podido utilizar, así como código de pruebas unitarias, ficheros de configuración, scripts, etc

Durante las sesiones de prácticas se proporcionará acceso a los estudiantes a un repositorio plantilla donde se explicará detenidamente los ficheros que serán necesarios para la realización de la entrega.

Documentación

No se solicitará documentación adicional, pero se recomienda encarecidamente incluir en el repositorio tantos ficheros con información sobre el desarrollo. También se consultará el historial de commits en Git.

Defensa

La práctica se corrige a través de un correctos automático que realizará las pruebas pertinentes a los servicios, es habitual que este sistema puntúe a la baja porque es muy sensible a los defectos de forma (por ejemplo, si el ejecutable tiene como shebang “/usr/bin/python3” en lugar de “/usr/bin/env python3”) por lo que la nota inicial podría ser menor que en una corrección manual donde esos defectos son ignorados. Quien desee solicitar una defensa de práctica después de la publicación inicial de notas podrá hacerlo comunicándolo a su profesor de prácticas mediante correo electrónico. Salvo excepciones, la defensa se realizará online y en ella el alumno demostrará manualmente el funcionamiento de su práctica.

El profesorado también podrá solicitar una defensa “manual” cuando estime oportuno.