SDIS 2015/2016 - 2nd Semester
Project 1 -- Distributed Backup Service Interface

---

1. Introduction

These notes specify the interface that you will have to develop for your Distributed Backup Service. Basically, this interface will allow us to invoke the sub protocols provided by that service to backup, restore and delete files, as well as to reclaim space being used by the service.

The main reason for this specification is to make it straightforward to test your implementation of the service, and therefore reduce the time required for that test in the two classes following the submission of your service. Past experience has been that if we leave the interface to you, some groups take forever to setup their services wasting everybody's time.

The idea is that you will have to develop a small testing application with a command line interface (CLI) that triggers the execution of the different subprotocols depending on its input arguments. For that, you will have to define a simple protocol that will allow the testing application to communicate to one peer that will play the role of initiator of the subprotocol being tested.

These notes have three additional sections. In the next section, we define the CLI. In the following section we make some suggestions regarding the implementation of this interface. Finally, in the last section we make a some recommendations to take into account in the implementation of your service, with its testing in mind.

2. Testing Application Interface

As mentioned above, the testing application must have a command line interface. I.e. it should be invoked as follows:

```
$ java TestApp <peer_ap> <sub_protocol> <opnd_1> <opnd_2>
```
where:

<peer_ap>
: Is the local peer access point. This depends on the implementation. (Check the next section)

<sub_protocol>
: Is the sub protocol being tested, and must be one of: `BACKUP`, `RESTORE`, `DELETE`, `RECLAIM`. In the case of enhancements, you must append the substring `ENH` at the end of the respecive subprotocol, e.g. `BACKUPENH`

<opnd_1>
: Is either the path name of the file to backup/restore/delete, for the respective 3 subprotocols, or the amount of space to reclaim. In the latter case, the peer should execute the `RECLAIM` protocol, upon deletion of any chunk.

<opnd_2>
: This operand is an integer that specifies the desired replication degree and applies only to the backup protocol (or its enhancement)

E.g., by invoking:

```
$ java TestApp 1923 BACKUP test1.pdf 3
```

your `TestApp` is supposed to trigger the backup of file `test1.pdf` with a replication degree of 3. Likewise, by invoking:

```
$ java TestApp 1923 RESTORE test1.pdf
```

your `TestApp` is supposed to trigger the restauration of file `test1.pdf` that was previously replicated.

3. Testing Application Implementation

The testing application and your peers are different applications, and they should communicate by exchanging messages. Essentially, for testing purposes, the testing application is a client and the peers are servers. Therefore, you should define a client/server protocol between the testing application and a peer, and the testing application and the peer must implement the respective role of that protocol.

You can use whatever "transport protocol" you deem appropriate, e.g. UDP, TCP or even RMI. However, your choice of transport protocol will affect the syntax of the access point used in the invocation of the testing app.

If you use either UDP or TCP, the format of the access point must be `<IP address>:<port number>`, where `<IP address>` and `<port number>` are respectively the IP address and the port number being used by the (initiator) peer to provide the testing service. If the access point includes only a port number (with or without `':'`), then you should assume that the initiator server runs on the local host, i.e. the same host as the testing application.

If you choose to implement this interface using RMI you should use the name of the remote object providing the "testing" service.

To avoid modifying the definition of the command line of the peer, i.e. adding yet another argument, you should use the first argument not only as the identifier of the peer but also as the peer access point for the testing service. E.g. if you choose to use either UDP or TCP as transport protocol, then the port number on which the peer will provide the testing service will also be the peer's identifier. If you choose RMI, then you should specify the name of the remote object on which the peer will provide the testing service. Note that the name of a remote object in RMI is an arbitrary string, therefore you can use a string of digits, if your implementation of the peer requires the implementation to be numbers.

4. Testing Considerations

In this section we make a few considerations regarding the testing/grading of the four subprotocols.

4.1 Backup Subprotocol

According to the specification above, the testing application receives the name of a file to be backed up with some replication degree. The expected output of this command is the replication of all the chuncks that makeup the file to be replicated.

Neither this document nor the document with the protocols specification specifies the division of labor between the replication application and the initiator peer. That is, we do not specify which of these two applications should do the splitting of the file in chunks. It is up to you to decide whether the splitting of a file is done by the testing application or by the peer itself.

Remember that a peer must never store chunks of the files it backups.

4.2 Restore Subprotocol

Again, the specification above does not specify whether the restoration/recreation of a file should be performed by the testing application or the peer server itself. It is up to you to decide.

You should also avoid triggering the restoration of a file automatically upon deletion of a file. Not only this requires some additional work, but also may not be always appropriate.

Conversely, the restoration of a file should be performed whenever requested by the testing application to the initiator peer, independently of the existence of the original file. To avoid naming conflicts, you may wish to use either prefix and/or suffix strings in the name of the restored file.

4.3 File Deletion Subprotocol

Although the file deletion subprotocol is supposed to support the deletion of a file, you should not trigger the deletion subprotocol automatically when a file is deleted. Instead, the file deletion subprotcol should be triggered explicitly by the testing application.

4.4 Space Reclaim Subprotocol

Upon invocation of space reclaim suprotocol via the testing application, your implementation should delete enough chunks so as to ensure that it does not use more disk spaced than allowed by the owner. Again, whether the deletion of the chunks should be done by the testing application or by the peer, is up to you to decide. Nevertheless, we expect the peer to initiate the space reclaim subprotocol.

So as not to constrain testing, you should not impose artificial limits on the amount of space to reclaim. E.g. it should be possible to reclaim all the space that was previously reserved for the backing up of chuncks.