Project 2: A Simple File Service Using Remote Procedure Calls

Due: October 23rd, 2015 23:59:59pm

In this project, you will create a simple file service using the Apache Thrift remote procedure call framework.

I have provided a file, `fileservice.thrift`, that defines several basic operations that can be conducted on a remote file server.

The project consists of three parts, which are described below.

1 Compile the interface definition file

I have provided an Apache Thrift interface definition file, `fileservice.thrift`. It can be downloaded from Blackboard. You can use Thrift to compile interface definition file to either Java, C++, or Python (Thrift also supports other languages, but these are not available on CS computers).

I have compiled and installed Thrift under my home directory on CS Department computers. These computers are located in G7 and Q22, or you can also remotely access them by SSH'ing into `remote.cs.binghamton.edu`. Source code as well as compiled files are located under `/home/yaoliu/src_code/thrift`. Other relevant files can be found in `/home/yaoliu/src_code/local`. To use Thrift installed under my home directory, you need to set the environment variable PATH:

```
$> bash
$> export PATH=$PATH:/home/yaoliu/src_code/local/bin
```

You can also install Thrift on your personal computer. You can download the Thrift source code at `http://thrift.apache.org/download`. To compile the source code and install, follow the instructions listed in `http://thrift.apache.org/docs/BuildingFromSource`.

Thrift can be used to create service stubs in a language of your choice from a `.thrift` interface definition file. As an example, the Thrift command to compile this project's IDL file to Java is as follows:

```
$> thrift -gen java fileservice.thrift
```

In `http://thrift.apache.org/tutorial`, you can find example clients and servers written in many of the languages that Thrift supports. This example code can be found in the Thrift source tree under the `tutorial` directory[1]. You should read the tutorial code in the language of your choice (one of Java, C++, and Python) before beginning work on the remainder of the project.

Information about the basic concepts and features of Thrift are available at `http://thrift.apache.org/docs/concepts` and `http://thrift.apache.org/docs/features`. Although I have provided the IDL file for you, you should also check `http://thrift.apache.org/docs/idl` for details about the Thrift interface definition language and data types and services supported by Thrift.

[1]For example, the complete Java tutorial code is at: `https://git1-us-west.apache.org/repos/asf?p=thrift.git;a=tree;f=tutorial/java/src;hb=HEAD`, the complete C++ code is located at: `https://git1-us-west.apache.org/repos/asf?p=thrift.git;a=tree;f=tutorial/cpp;h=3485c6249c6ca90ef58aace69d4e31223037712a;hb=HEAD`

2 Extend the server-side method stubs generated by Thrift

When Thift processes the interface definition file, `fileservice.thrift`, it generates method stubs corresponding to the defined services. You must implement methods corresponding to the following server-side stubs:

writeFile given a name, owner, and contents, the corresponding file should be written to the server. Meta-information, including the filename, creation-time, update-time, version (start from 0), owner, content length, and content hash (use the MD5 hash) should also be stored at the server side. If the filename does not exist on the server, a new file should be created with its version attribute set to 0. Otherwise, the file contents should be overwritten and the version number should be incremented.

readFile if a file with a given name and owner exists on the server, both the contents and meta-information should be returned. Otherwise, an exception should be thrown.

listOwnedFiles given an user name, all files owned by the user should be listed. If the user does not exist, then an exception should be thrown.

Every time the server is started any filesystem or other persistent storage that it used should be initialized to be empty. If your server stores files in the file system, it should use only the current working directory from where it was run. There is no need to implement directory structure.

The server executable should take a single command-line argument specifying the port where the Thrift service will listen for remote clients. It should also use Thrift's TBinaryProtocol for marshalling and unmarshalling data structures.

3 Write a client program

You must also write a client program to call the remote methods that you created in Part 2. The client program should take five command line arguments as input. The first two arguments should be the host and port of the remote Thrift server. The final three arguments (occurring in no particular order) are as follows:

--operation [operation], where operation can be "read", "write", and "list", and should invoke the respective remote procedure call

--filename [filename], should be used as input to specify either a local file to be written to the remote system, or a remote filename to be read (if the operation is "list", then this argument should not be used)

--user [user], should specify the owner of the file for the current operation

After the client program receives the response from the remote procedure call, it should use a separate Thrift transport and protocol to write the returned structure to the standard output. (Hint: Under Python, use TFileObjectTransport and TJSONProtocol. Under Java, use TIOStreamTransport and TJSONProtocol.) **This output step is important! The client should also write any exceptions thrown by the remote method in JSON format to the standard output**.

An example of how the client program should operate is provided below:

Input:

```
$> ./client localhost 9090 --operation read --filename example.txt --user guest
```

The output is in the form of a Thrift-generated JSON[2] structure (do not try to manually generate this yourself):

```
{"1":{"rec":{"1":{"str":"example.txt"},"2":{"i64":1379202165425},
"3":{"i64":1379202165425},"4":{"i32":0},"5":{"str":"guest"},"6":{"i32":17},
"7":{"str":"7bdbc2db770bf3dc4f78ab4fbf88831a"}}},"2":{"str":"example contents\n"}}
```

More examples: Start the server at `remote01.cs.binghamton.edu`:

```
$> ./server 9090
```

Run the client at `remote05.cs.binghamton.edu`:

```
$> ./client remote01.cs.binghamton.edu 9090 --operation write --filename ex_file1 --user guest
{"1":{"i32":1}}

$> ./client remote01.cs.binghamton.edu 9090 --operation write --filename ex_file2 --user guest
{"1":{"i32":1}}

$> ./client remote01.cs.binghamton.edu 9090 --operation list --user guest
["rec",2,{"1":{"str":"ex_file1"},"2":{"i64":1441391699104},"3":{"i64":1441391699104},
"4":{"i32":0},"5":{"str":"guest"},"6":{"i32":17},"7":{"str":"dd75ce3f5c3f89878459545c697f87de"}},
{"1":{"str":"ex_file2"},"2":{"i64":1441391722853},"3":{"i64":1441391722853},"4":{"i32":0},
"5":{"str":"guest"},"6":{"i32":17},"7":{"str":"1f600d1084e1455bb7b309a50dd97085"}}]

$> ./client remote01.cs.binghamton.edu 9090 --operation list --user nonexist
{"1":{"str":"User nonexist does not exist."}}

$> ./client remote01.cs.binghamton.edu 9090 --operation read --filename ex_file1 --user guest
{"1":{"rec":{"1":{"str":"ex_file1"},"2":{"i64":1441391699104},"3":{"i64":1441391699104},
"4":{"i32":0},"5":{"str":"guest"},"6":{"i32":17},"7":{"str":"dd75ce3f5c3f89878459545c697f87de"}}},
"2":{"str":"example content1\n"}}

$> ./client remote01.cs.binghamton.edu 9090 --operation read --filename ex_file2 --user guest
{"1":{"rec":{"1":{"str":"ex_file2"},"2":{"i64":1441391722853},"3":{"i64":1441391722853},
"4":{"i32":0},"5":{"str":"guest"},"6":{"i32":17},"7":{"str":"1f600d1084e1455bb7b309a50dd97085"}}},
"2":{"str":"example content2\n"}}

$> vi ex_file1

$> ./client remote01.cs.binghamton.edu 9090 --operation write --filename ex_file1 --user guest
{"1":{"i32":1}}

$> ./client remote01.cs.binghamton.edu 9090 --operation read --filename ex_file1 --user guest
{"1":{"rec":{"1":{"str":"ex_file1"},"2":{"i64":1441391699104},"3":{"i64":1441391833690},
"4":{"i32":1},"5":{"str":"guest"},"6":{"i32":17},"7":{"str":"c2907ca1ae7a101fb58a093a081d7b4c"}}},
"2":{"str":"modified content\n"}}

$> ./client remote01.cs.binghamton.edu 9090 --operation list --user guest
["rec",2,{"1":{"str":"ex_file1"},"2":{"i64":1441391699104},"3":{"i64":1441391833690},
"4":{"i32":1},"5":{"str":"guest"},"6":{"i32":17},"7":{"str":"c2907ca1ae7a101fb58a093a081d7b4c"}},
{"1":{"str":"ex_file2"},"2":{"i64":1441391722853},"3":{"i64":1441391722853},"4":{"i32":0},
"5":{"str":"guest"},"6":{"i32":17},"7":{"str":"1f600d1084e1455bb7b309a50dd97085"}}]
```

Note: When accessing `remote.cs.binghamton.edu`, you are actually redirected to one of the 16 `REMOTE` machines ({remote00, remote01, ..., remote09, remote10, ..., remote15}.cs.binghamton.edu) using DNS redirection. So to test your implementation, you need to make sure your client is contacting the server using the correct host name or public IP address.

[2] http://en.wikipedia.org/wiki/JSON

4 How to submit

To submit the project, you should first create a directory whose name is "your BU email ID"`-project2`. For example, if your email ID is `jdoe@binghamton.edu`, you should create a directory called `jdoe-project2`. You should put the following files into this directory:

1. Your source code.
2. A `Makefile` to compile your source code into two executables, which should be named `client` and `server`. (It is okay if these executables are bash scripts that call the Java interpreter, as long as the command line arguments follow the format described in Parts 2 & 3.)
3. A `Readme` file describing the programming language you are using, your implementation details, and sample input/output.

Compress the directory (e.g., `tar czvf jdoe-project2.tgz jdoe-project2`) and submit the tarball (in `tar.gz` or `tgz` format) to Blackboard. Again, if your email ID is `jdoe@binghamton.edu`, you should name your submission: `jdoe-project2.tar.gz` or `jdoe-project2.tgz`. Failure to use the right naming scheme will result in a 5% point deduction.

Your project will be graded on the CS Department computers `remote.cs.binghamton.edu`. If you use external libraries in your code, you should also include them in your directory and correctly set the environment variables using absolute path in the Makefile. If your code does not compile on or cannot correctly run on the CS computers, you will receive no points.