CS378 Socket Programming Lab: Distributed File System

Due: Nov 10, 10pm

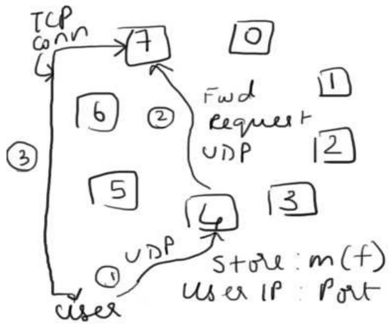
Context:

Many of you may have used Google Drive or DropBox. These systems provide an abstraction of cloud based storage. That is, a user can store files onto a file-system 'somewhere' in the network and retrieve them. This can be done in a device-independent and location independent fashion, which provides seamless access anywhere in the world to the file system across a user's many devices: desktop, laptop, lab machine, smart-phone, tablet, etc.

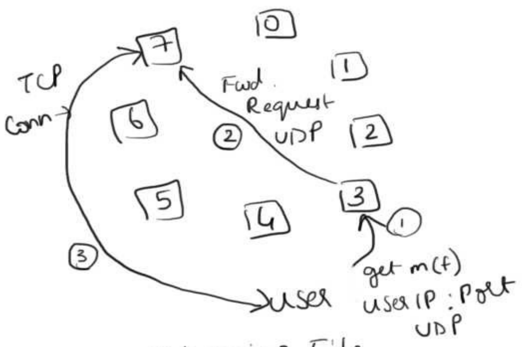
FileMesh:

In this project, we are going to implement a rudimentary network file-system, called FileMesh.

- In FileMesh, there is a logical mesh of 'nodes'. These nodes implement the distributed file system. A node is nothing but a process with appropriate network socket communication mechanisms.
- Each node has a UDP 'control' port on which it listens.
- Nodes are numbered *0, 1, … (N-1)*
- To begin with, each node knows:
    - The number of nodes in the mesh *N*
    - Each node's number
    - Each node's IP address and control port
- Each node also has its own local file-system 'folder' which is used to store files of 'end-users'.
- Files are stored in the distributed file system in a pseudo-random fashion as follows:
    - Given a file *f*, compute its 128-bit md5sum = *m(f)* (you can use the program md5sum for this, or an appropriate library)
    - File *f* is stored in node number *[m(f) % N]*
- The rest of the operation is best explained through examples



File storing



Retrieving File

File storing example:

- Step s1: A user program (outside of the mesh) initiates a store request to an arbitrary node in the mesh (say node 4 in this example). The store request packet contains the request type (i.e. store), *m(f)*, as as well as an IPaddr:port. This IPaddr:port corresponds to the user program's own IP address and TCP port, on which it is listening, ready for an incoming TCP connection, which happens in step s3 in this example. Note that in step s1, the file contents do NOT go over the network, only the md5sum!

- Step s2: Node 4 computes *m(f) % 8*, which say is *7*. It then sends the request to node 7.

- Step s3: Node 7 checks that *m(f) % 8* is indeed its own id. Then it makes an outgoing TCP connection to the given IPaddr:port in the request. The user program then uses this TCP connection to transmit the contents of file *f*. Node 7 stores the file in its folder, under the name *m(f)*.

File retrieving example:

This is very similar to the file storing

- Step t1: The user program (same user program but can represent a different user or same user but in another location) makes a file get request to an arbitrary node in the mesh, say node 3. The contents of this get request are similar to the store request.

- Step t2: Node 3 computes m(f)%8, which say is 7 again. Node 3 then passes this request to node 7.

- Step t3: Node 7 checks m(f)%8 is its own id. Then it makes an outgoing TCP connection to the given IPaddr:port in the request. It then sends the file contents from its folder, over this TCP connection. The user program stores this file locally as file with name m(f).

What you need to implement

1. The ‘FileMeshNode’ program
   a) This must take one argument, which is the node id
   b) This must read a configuration file (passes information about all nodes that are part of the mesh; line no indicates node no) FileMesh.cfg in the current directory which has the format:
   ```
   IPaddr0:port0    folder0
   IPaddr1:port1    folder1
   and so on...
   ```
   c) The first line corresponds to node 0, the next line to node 1, and so on.
   d) Note that during initial testing, you can run all nodes on the same machine
2. The ‘User’ program which can initiate both store and get requests
   a) This program must have an appropriate interface (command-line, or stdin, or GUI) to take necessary arguments to achieve the store or get functionality. For example, you should be able to specify if it’s a store or get operation, the path where the file is or where it is should be stored, which node in the mesh it should contact etc.
   b) Note that the focus here is on socket programming, so a crude (but usable) stdin or command-line user interface is fine
3. Optional: a bash script which, given FileMesh.cfg, will start up nodes corresponding to IP-addresses which belong to ‘this’ machine.

General Instructions:

1. This lab is to be done in **groups of three students.**
2. You are **not allowed to exchange code snippets or anything** across groups. In case of any doubts, you are allowed to consult any Internet resource, books, or the instructor. While it is alright to read on general socket programming on the Internet, and look at code examples, this should be only for the purpose of understanding. Do all the coding yourself, do not copy or cut-paste code from any website. You can use only socket programming for these projects. Do not use any sophisticated libraries downloaded from the web. Please understand the spirit behind all these and follow them strictly.

3. You can use C/C++ for coding, **no other language**. Although Java/Perl and other languages may provide a better interface, C/C++ are bare-bones, and hence appropriate for a networking course.

4. Give sufficient time for testing the code, do not cram everything to the last day. Remember that in industry, people spend more time testing than coding!

5. Commenting and indenting are important. Comment *during/before* coding, not at the end. For every variable/function you should have a comment. Use appropriate code comments for explaining the logic where necessary.

6. Provide documentation for your code. This should explain the code structure in terms of directories, files, how to compile etc. Detailed instructions for submitting the documentation along with code are at the end of the document.

7. Pay attention to the variable names, function names, file names, directory names, etc. Make sure they are intuitive.
8. Refer to http://java.sun.com/docs/codeconv/html/CodeConvTOC.doc.html for more instructions on coding conventions. Although this link is for Java, many of the principles are applicable for any programming language.
9. We will evaluate all of the above aspects of your code, not just whether or not the code runs.

Submission guidelines

Read the below instructions *carefully* and follow them meticulously.

Organizing your submission

- All relevant files should be under one directory. This directory should be named after the roll numbers of the three students.
- Within the directory, you should have a file called "README.txt" or "README.html" or "README.pdf" (only text, HTML, or PDF formats allowed for this). The following contents are required in the README file:
    - **List of relevant files:** Give the list of relevant files including all source files and configuration files which *you* have written. Of course you need not include things like stdio.h which someone else has provided for you. Specifically *do not* have any irrelevant, old, or temporary files which clutter the directory. Clean-up before submission.
    - **Compilation instructions:** How should one go about generating the executable from your source files? Give the actual set of commands which someone can cut-paste from the README in order to compile. Provide such instructions for each executable file you have to generate.
    - **Configuration file(s):** If your code uses any configuration file(s), describe the format of such file(s) clearly. Also include in the directory some example configuration files.
    - **Running instructions:** You may be generating more than one executable. Describe logically what each executable does. In addition, for each such executable file, how should one run it? What are the command line arguments (describe each argument)?
- You may create any number of sub-directories (multiple levels too if you want) within your main directory. The README file within the main directory should describe everything: all files/directories within any sub-directory too.
- **Commenting:**
    - Each source file should describe in the beginning, in a comment, what that source file contains logically.
    - Make sure to name variables, functions, file names with intuitive names wherever possible. In addition, provide a comment for each variable/function describing it logically. You need not do this for very trivial variables/functions; use your common sense judgment. The overall objective is that a third person should be able to easily understand what that variable/function does logically.

- Please also comment sections of the code which will help in understanding the logical flow of the code. For example, comments for a loop can describe any non-obvious invariant involved.

How and when to submit

- In the final submission, you have to tar-gzip or zip the main directory and submit a single file. Make sure to tar-gzip or zip from the *parent directory* of this directory, not from within this directory itself.
- Submit the code via moodle before the deadline.

The real test

- Save your submission somewhere and look at it after say, a semester. Does your README help? Do your comments help in understanding what you did in the code? This is the real test! Of course, since we have to grade you much before that, we will look at your code, comments, and documentation as described above.

Demo & Evaluation

We will have a project evaluation session per group. During this evaluation session, you will have to show a demo of your project (show even if it is not fully working) on the machines provided by us. I will also ask you questions to test your understanding of the code and its working. I will also evaluate the cleanliness of your coding, documentation, and commenting. I expect all members of the group to be present during this evaluation session.

During the evaluation session, be sure to arrive at least 15 minutes before your slot, setup your demo using the code that was submitted by the deadline. Ensure it works correctly. No debugging will be permitted during evaluation.

1. I will provide the configuration file based on the above format.
2. Your program should work with any number of nodes in the mesh (upper-limit say 10)
3. The IP addresses within the configuration file could belong to different machines. Say 3 node processes may run on machine with address IP1, while 4 may run on machine with address IP2 and 2 may run in machine with address IP3.
4. You will have to do a few stores and a few retrievals. These operations could be executed on any machine. The necessary detail i.e. which machine to run the program on, which mesh node (id) should it contact first and the path of the files will be specified during testing.