Robert Judka
CS550
Programming Assignment 2

# Manual

## Requirements

This system was developed in C++11 and compiled with the g++ compiler and a Linux subsystem. Running this will not work on a Windows (possibly not MacOS either) subsystem as it does not have the required directory interface.

## Build

In ‘src/’, running

*make all*

will generate the super_peer and leaf_node executables. It will also create the node directories (and more for using the full configuration files) used in this manual and the log directory where the system logs could be found (this may fail if you are not within the ‘src/’ directory).

## Execute

To start a super peer, each in a separate terminal run

*./super {id} {path_to_configuration_file}*

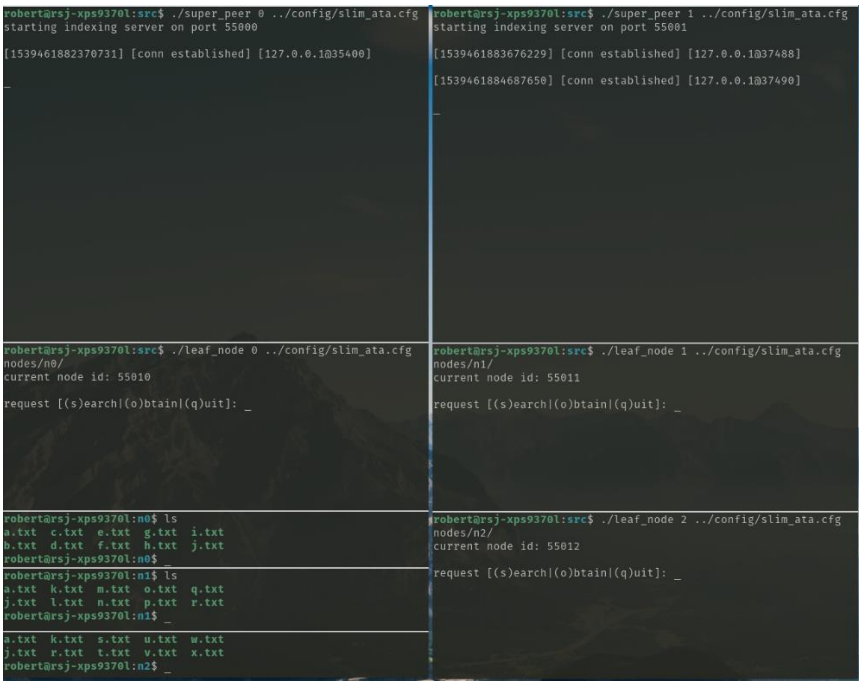
To start a leaf node, each in a separate terminal run

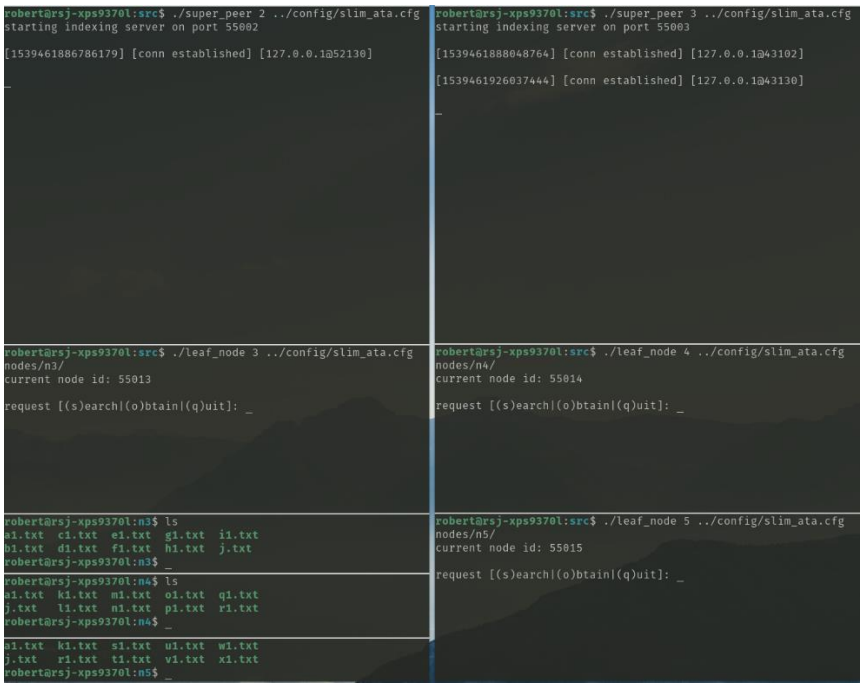
*./leaf_node {id} {path_to_configuration_file} {path_to_files_directory}*

## Demo

For convenience, a ‘slim_ata.cfg’ configuration file was provided for this demo (found in ‘config/’) which defines 4 super peers and 6 leaf nodes (look in the configuration file to get the id for each). This configuration file defines the connections between all the super peers and leaf nodes. Also, 6 peer directories were prepopulated with files. You can run these 6 leaf nodes (in 6 separate terminates) with ‘nodes/n0/’, ‘nodes/n1/’, nodes/n2/’, ‘nodes/n3/’, nodes/n4/’, and ‘nodes/n5/’ as their directories. NOTE these directories will only exist if using the build procedure from above and stay within the ‘src/’ directory.

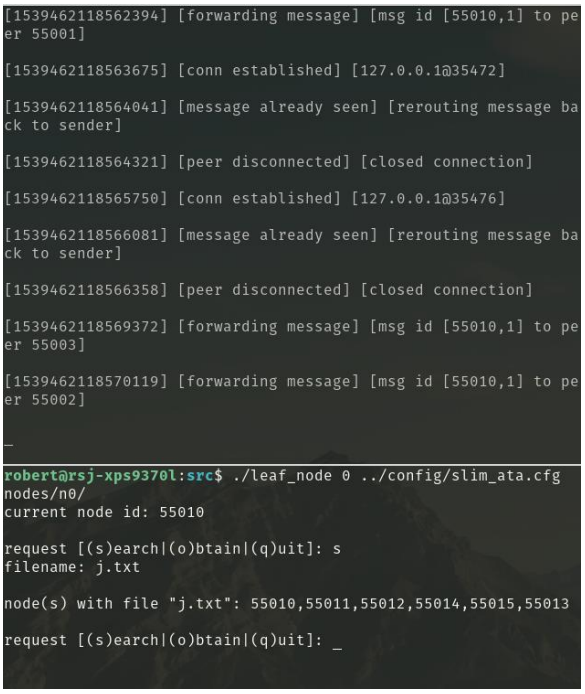
This will be the state of your system after executing the 4 super peers and 6 leaf nodes:

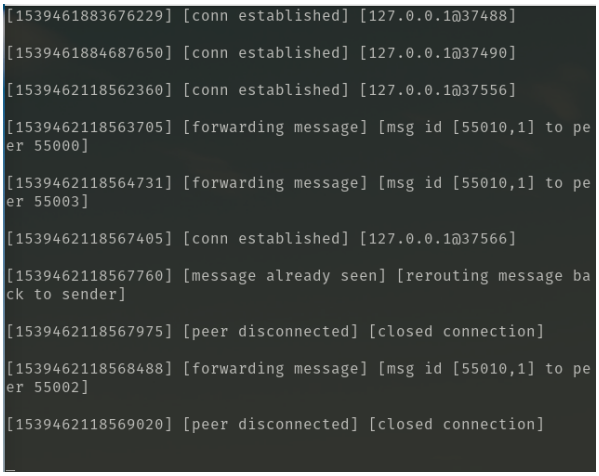



*more detailed outputs can be viewed in the ‘logs/’ directory

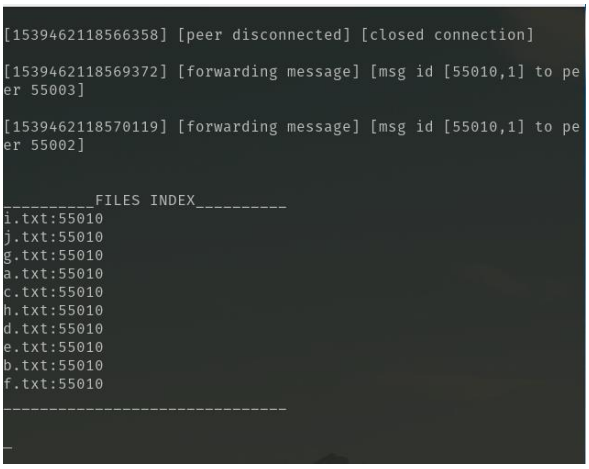
To search for a specific file in the network, we enter ‘s’ for ‘search’ into the command line in leaf node 0 and then enter the file we want to search for. In this example, we search for ‘j.txt’:

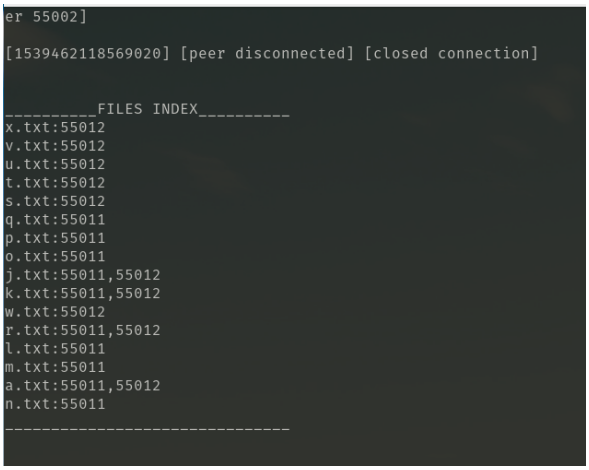



Looking at the first screenshot, we can see that ‘j.txt’ is owned by all the leaf nodes, and so all the leaf node ids are shown (note the leaf node id is different than the id used to initialize the leaf node). Included is also a snippet from one of the other super peers and how the request was broadcasted to other peers.

To see a list of all registered files, a *hidden* request ‘l’ could be made through a leaf node. Here we can see the contents of the *_files_index* on super peer 0 and super peer 2:

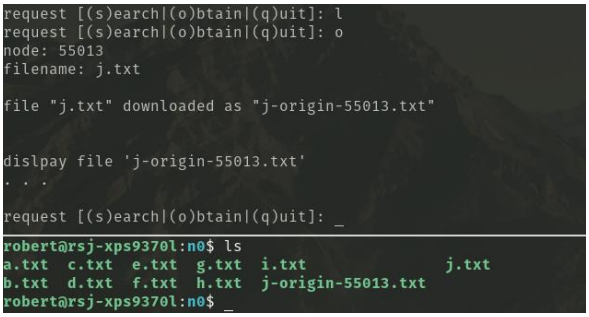



We can see all the files registered to the each of the leaf nodes, and the files which are shared among the leaf nodes.

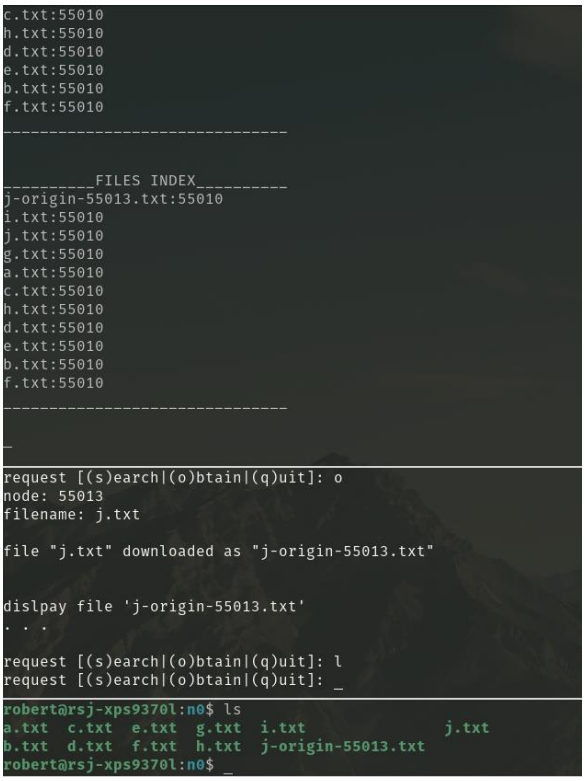
*more detailed outputs can be viewed in the ‘logs/’ directory

Next, to download a file, we can choose one of the ids we just learned and the file to download into our directory. To do this, we first enter ‘o’ for ‘obtain’ into the command line, then the leaf node’s id (in our example we entered ‘55013’), and finally, the file to download, ‘j.txt’:



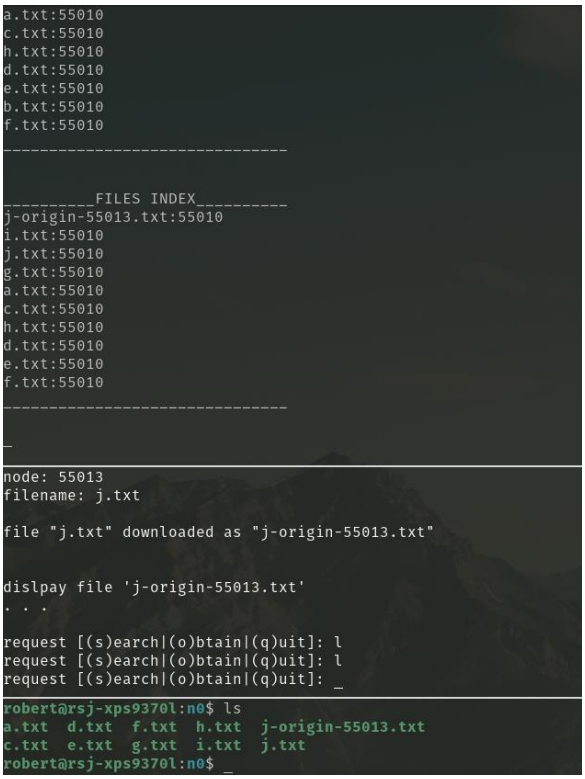

We see we have gotten the message saying ‘j.txt’ was successfully downloaded (saved in our directory as ‘j-origin-55013.txt’ since we already had a file named ‘j.txt’) and attempted to *display* the file (‘. . .’ symbolizes the content of the file). We can also see on the bottom terminal pointing to our directory, that by running the ‘ls’ command, ‘j-origin-55013.txt’ is now within our directory.

We can also see, by entering ‘l’ again into the command line, that the *_files_index* has been updated to include our new file ‘j-origin-55013.txt’:

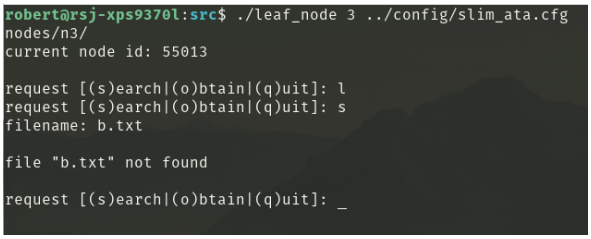



*more detailed outputs can be viewed in the ‘logs/’ directory

Now say we don't want the file 'b.txt' in our directory anymore. After running the 'rm b.txt' command on the bottom terminal pointing to our directory, and entering 'l' again into the command line, we see that that *_files_index* no longer contains our file 'b.txt':



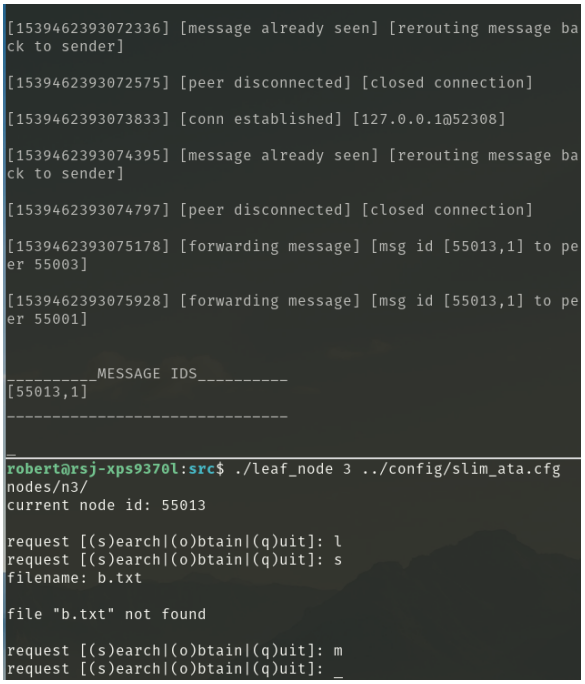

To check that other super peers also function properly, we will move to super peer 2's group and enter 's' into the command line for leaf node 3 to search for the file 'b.txt':

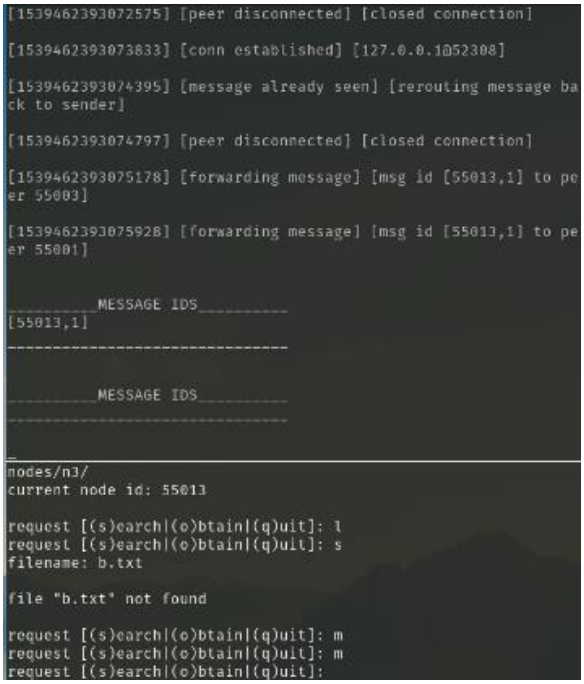



Since we just removed 'b.txt' from leaf node 0 and it was the only node to own that file, we received the message stating the file could not be found.

*more detailed outputs can be viewed in the 'logs/' directory

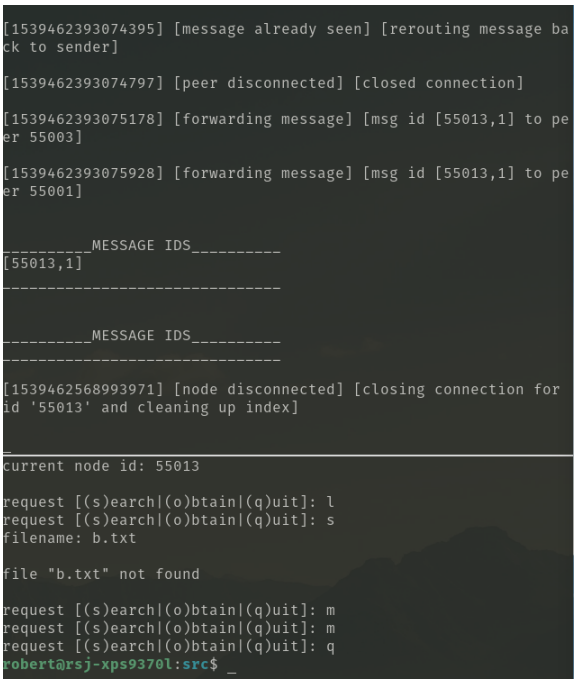
Now, to ensure messages are being handled correctly in the super peers, we will enter the *hidden* request ‘m’ in the leaf node to see the current messages being managed by the super peer:




In the super peer, we can see the request we just made is being tracked (left). Then, after waiting 2 minutes and entering ‘m’ again, we can see the super peer removed the old message from its list (right).

Finally, to ensure closed leaf node connections are handled properly, we enter ‘q’ into the command line:




*more detailed outputs can be viewed in the ‘logs/’ directory