Robert Judka
CS550
Programming Assignment 3

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_peer {id} {path_to_configuration_file}

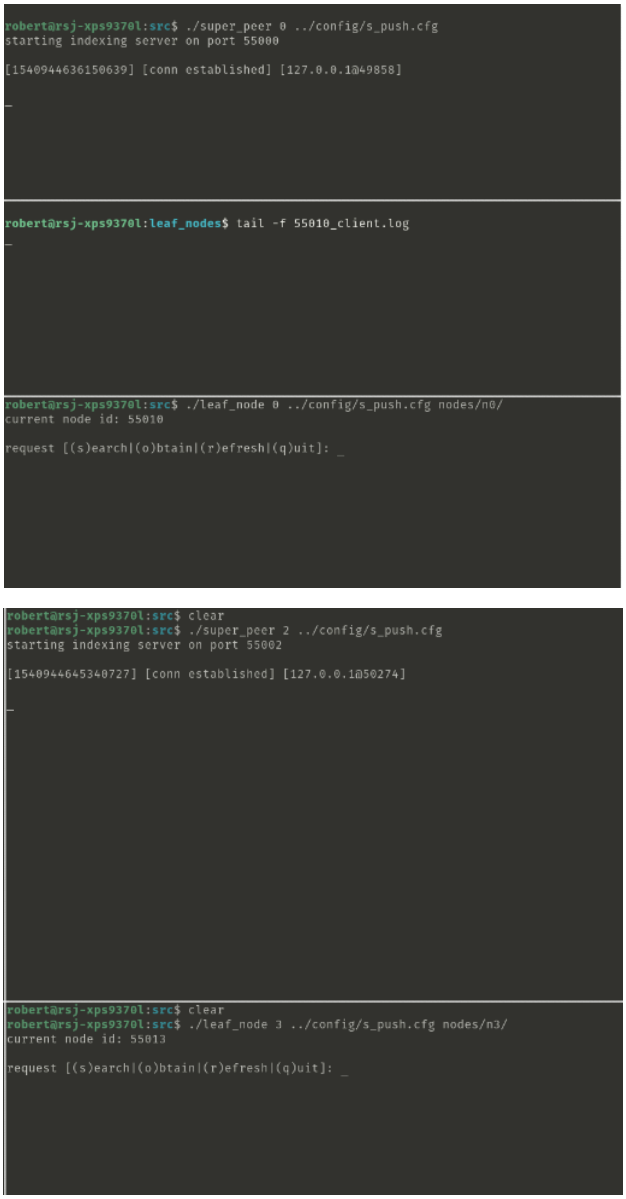
To start a leaf node, each in a separate terminal run

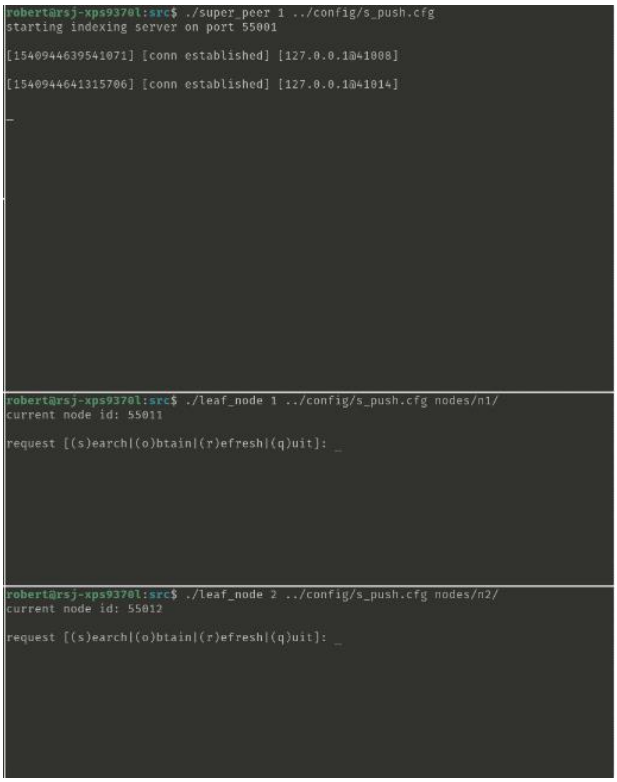
./leaf_node {id} {path_to_configuration_file} {path_to_files_directory}

Demo

For convenience, a ‘s_push.cfg’ configuration file was provided for this demo (found in ‘config/’) which defines 3 super peers and 4 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, 4 peer directories were prepopulated with files. You can run these 4 leaf nodes (in 4 separate terminates) with ‘nodes/n0/’, ‘nodes/n1/’, nodes/n2/’ and ‘nodes/n3/’ 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 3 peers and 4 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 node 0 and then enter the file we want to search for. In this example, we search for ‘j.txt’:

```
robert@rsj-xps9370l:src$ ./leaf_node 0 ../config/s_push.cfg nodes/n0/
current node id: 55010

request [(s)earch|(o)btain|(r)efresh|(q)uit]: s
filename: j.txt

node(s) with file "j.txt": 55010,55011,55012,55013

request [(s)earch|(o)btain|(r)efresh|(q)uit]: _
```

We can see that ‘j.txt’ is owned by all the nodes, and so all the node ids are shown (note the leaf node id is different than the id used to initialize the node).

To see a list of all registered files, a *hidden* request ‘l’ could be made through a node. Here we can see the contents of the *_files_index* on peer 1:

```
[1540944848225456] [peer disconnected] [closed connection]

[1540944848225746] [peer disconnected] [closed connection]


__________FILES INDEX__________
x.txt:55012
v.txt:55012
u.txt:55012
t.txt:55012
s.txt:55012
q.txt:55011
p.txt:55011
o.txt:55011
j.txt:55011,55012
k.txt:55011,55012
w.txt:55012
r.txt:55011,55012
l.txt:55011
m.txt:55011
a.txt:55011,55012
n.txt:55011
______________________________

_
```

We can see all the files registered to the each of the nodes, and the files which are shared among the nodes (like for example, ‘j.txt’).

*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 node’s id (in our example we entered ‘55013’), and finally, the file to download, ‘j.txt’:

```
node(s) with file "j.txt": 55010,55011,55012,55013

request [(s)earch|(o)btain|(r)efresh|(q)uit]: o
node: 55013
filename: j.txt

file "j.txt" downloaded as "j.txt"


dislpay file 'j.txt'
. . .

request [(s)earch|(o)btain|(r)efresh|(q)uit]: _
```

```
robert@rsj-xps9370l:n0$ clear
robert@rsj-xps9370l:n0$ ls remote/
j.txt
robert@rsj-xps9370l:n0$ _
```

We see we have gotten the message saying ‘j.txt’ was successfully downloaded (saved in our remote files directory) 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.txt’ is now within our directory.

Now say we want to edit the original ‘j.txt’ from node 3. To simulate a change, we quickly opened the file and resaved it, essentially updating its last modified time. This will be recognized by the node as an update and will thus broadcast a message to all other nodes to remove their old versions. Node 0, who just downloaded that file, will see a message that the file was removed since it was modified:

```
robert@rsj-xps9370l:leaf_nodes$ tail -f 55010_client.log
! [SRCH] [j.txt] [55010,55011,55012,55013]

! [OBTN] [55013/j.txt]

[1540945087466186] [file download] file download successful

[1540945160678617] [removing file] remote file "j.txt" modified

! [RMV] [55013/j.txt]

_
```

```
robert@rsj-xps9370l:n0$ clear
robert@rsj-xps9370l:n0$ ls remote/
j.txt
robert@rsj-xps9370l:n0$ ls remote/
robert@rsj-xps9370l:n0$ _
```

Note, to avoid any sort of *STDOUT* issues with the client cli, we opted to log the removals to a client’s log, which could be viewed in real time using the ‘tail -f’ command. Using the ‘ls’ command again, we can also see ‘j.txt’ was successfully removed from our directory.

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

To see how we can refresh a file, we first download a new file. In this example, we'll be downloading file 'a1.txt' from '55013':

```
. . .

request [(s)earch|(o)btain|(r)efresh|(q)uit]: o
node: 55013
filename: a1.txt

file "a1.txt" downloaded as "a1.txt"


dislpay file 'a1.txt'
. . .

request [(s)earch|(o)btain|(r)efresh|(q)uit]: _
```

We'll then simultaneously (or as close to it as possible) edit the original 'a1.txt' from node 3 and refresh node 0's cached copy of 'a1.txt':

```
robert@rsj-xps9370l:n3$ vim local/j.txt
robert@rsj-xps9370l:n3$ vim local/a1.txt
robert@rsj-xps9370l:n3$ _
```

```
. . .

request [(s)earch|(o)btain|(r)efresh|(q)uit]: r
node: 55013
filename: a1.txt

file "a1.txt" updated to version 1540944415


dislpay file 'a1.txt'
. . .

request [(s)earch|(o)btain|(r)efresh|(q)uit]: _
```

```
! [OBTN] [55013/j.txt]

[1540945087466186] [file download] file download successful

[1540945160678617] [removing file] remote file "j.txt" modified

! [RMV] [55013/j.txt]

! [OBTN] [55013/a1.txt]

[1540945335151188] [file download] file download successful
```

We can see that the refresh updated node 0's version of 'a1.txt' while not removing it from the node's remote directory. **The results from this may vary, however, as this demo is using the *PUSH* consistency method where stale files get invalidated almost immediately after an update takes place, which means a refresh might not finish downloading before the file gets invalidated (and thus removed) or the refresh occurs on the original file.** To more easily see a refresh occur, you could try using one of the other consistency methods (*PULL FROM NODES* or *PULL FROM PEERS*).

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