Report Replication

Alexander Michielsen, Group: **#b10197**

This part of the lab was carried out completely by Louis and me.

I will go over the three phases of replication and how we have implemented them.

Prerequisites

Before we dive into the functionality itself, there are some systems that are needed beforehand.

File System

The file system is an object on each node that acts as a bookkeeper. It contains a map of all files together with the custom FileParameters class, containing the parameters of the file.

They are:

-Whether the file has been downloaded

-Whether the file is locked

-On which node the file is replicated (the owner)

-On which node the file is local

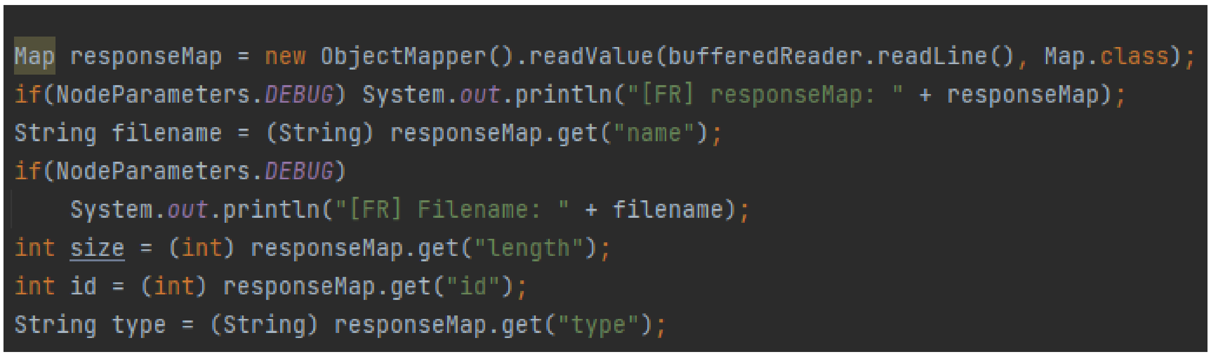
-Which node has locked the file

On top of that, functionality has also been implemented on the file system that can return the list of local files, replicated files and downloaded files.

Receiving Files

In order to receive files an extra thread capable of receiving files is created. This class constantly listens for files being sent (Server socket on port 5044).

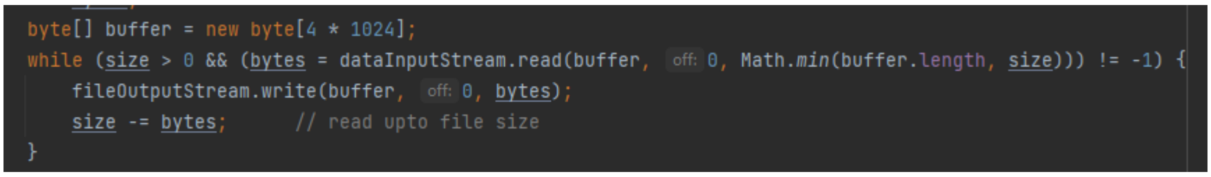
This socket first waits for a json object containing all the metadata of the file being sent.



```java
Map responseMap = new ObjectMapper().readValue(bufferedReader.readLine(), Map.class);
if(NodeParameters.DEBUG) System.out.println("[FR] responseMap: " + responseMap);
String filename = (String) responseMap.get("name");
if(NodeParameters.DEBUG)
    System.out.println("[FR] Filename: " + filename);
int size = (int) responseMap.get("length");
int id = (int) responseMap.get("id");
String type = (String) responseMap.get("type");
```

Through this, it knows what the file name should be, how long the file is and what the ID (hash) of the file is.

Once this has been received correctly, the receiver responds with an acknowledge to the sender. This one starts sending the file so the receiver can receive it.



```java
byte[] buffer = new byte[4 * 1024];
while (size > 0 && (bytes = dataInputStream.read(buffer, off: 0, Math.min(buffer.length, size))) != -1) {
    fileOutputStream.write(buffer, off: 0, bytes);
    size -= bytes;      // read upto file size
}
```

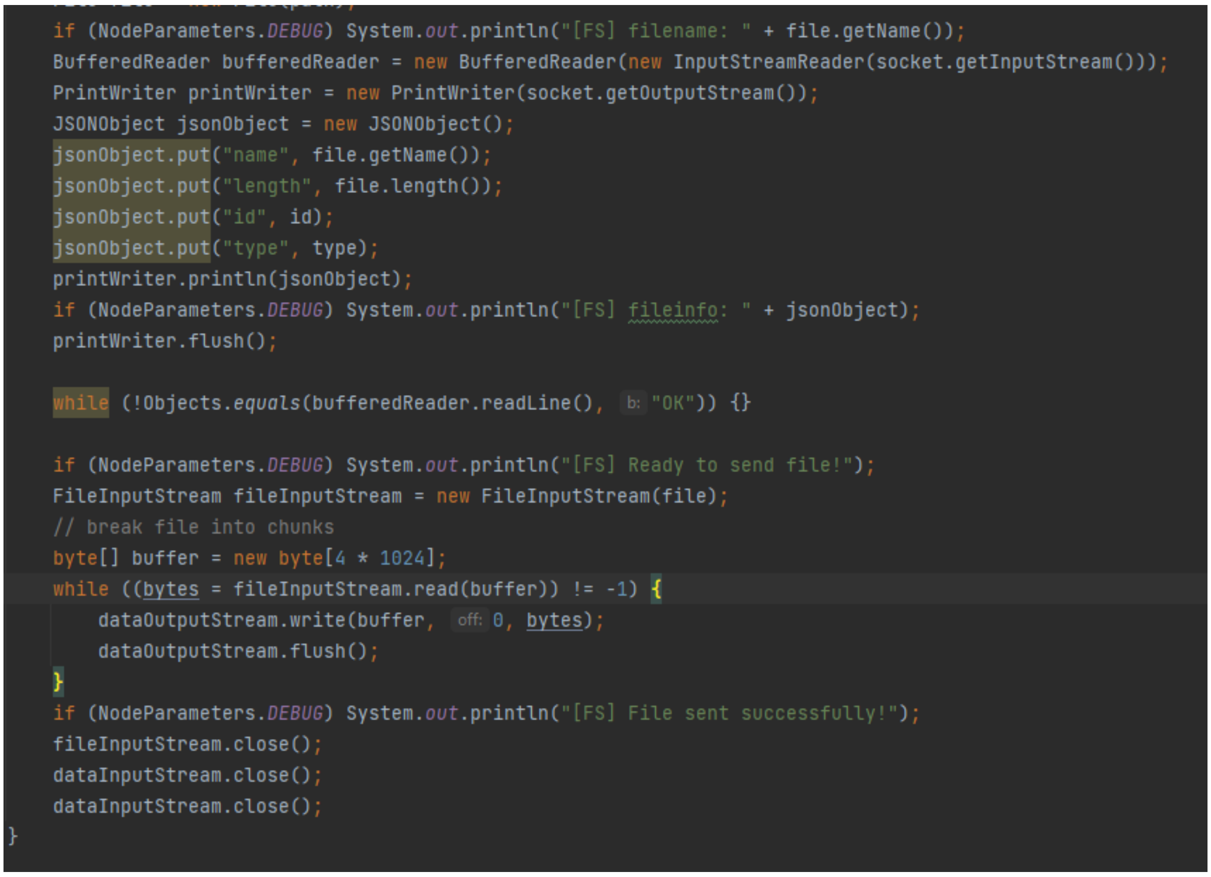
Receiver receiving the file itself

The file is added to the file system afterwards with the correct settings.

Sending Files

In order to send files, a separate file sending class has been implemented. This works using a basic socket on the same port as the receiver.

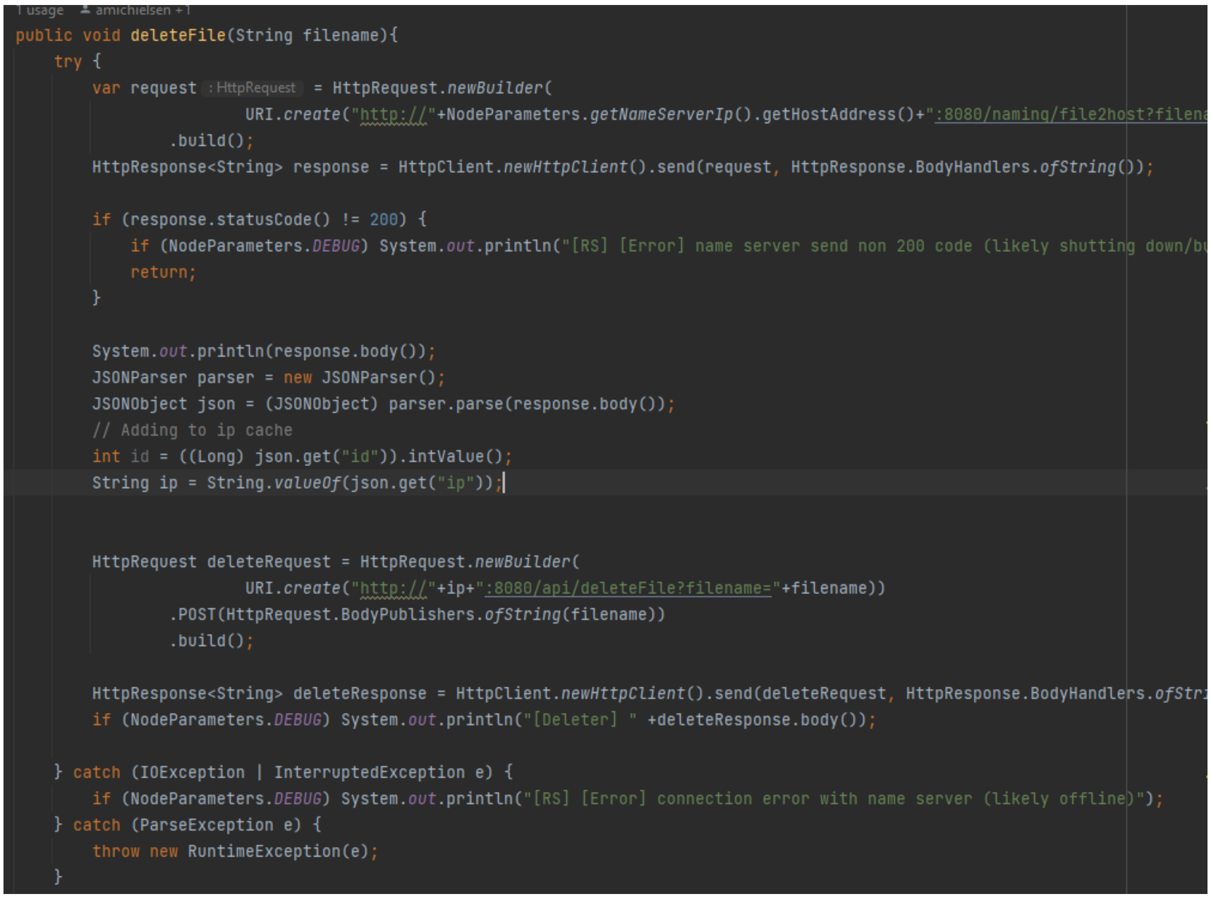
This class first sends the metadata of the file as a json object (just like the receiver expects). After the receiver acknowledges that it has received the metadata, the sender sends the file itself.



```
    if (NodeParameters.DEBUG) System.out.println("[FS] filename: " + file.getName());
    BufferedReader bufferedReader = new BufferedReader(new InputStreamReader(socket.getInputStream()));
    PrintWriter printWriter = new PrintWriter(socket.getOutputStream());
    JSONObject jsonObject = new JSONObject();
    jsonObject.put("name", file.getName());
    jsonObject.put("length", file.length());
    jsonObject.put("id", id);
    jsonObject.put("type", type);
    printWriter.println(jsonObject);
    if (NodeParameters.DEBUG) System.out.println("[FS] fileinfo: " + jsonObject);
    printWriter.flush();

    while (!Objects.equals(bufferedReader.readLine(), b: "OK")) {}

    if (NodeParameters.DEBUG) System.out.println("[FS] Ready to send file!");
    FileInputStream fileInputStream = new FileInputStream(file);
    // break file into chunks
    byte[] buffer = new byte[4 * 1024];
    while ((bytes = fileInputStream.read(buffer)) != -1) {
        dataOutputStream.write(buffer, off: 0, bytes);
        dataOutputStream.flush();
    }
    if (NodeParameters.DEBUG) System.out.println("[FS] File sent successfully!");
    fileInputStream.close();
    dataInputStream.close();
    dataInputStream.close();
}
```

File Deleter

The File Deleter class that takes care of a local file that has been deleted. It will figure out which file owned the file (thanks to the naming server) and then sends a rest request telling the owner to delete the replica because the local file has been removed.



```
1 usage   amichielsen +1
public void deleteFile(String filename){
    try {
        var request :HttpRequest  = HttpRequest.newBuilder(
                        URI.create("http://"+NodeParameters.getNameServerIp().getHostAddress()+":8080/naming/file2host?filena
                .build();
        HttpResponse<String> response = HttpClient.newHttpClient().send(request, HttpResponse.BodyHandlers.ofString());

        if (response.statusCode() != 200) {
            if (NodeParameters.DEBUG) System.out.println("[RS] [Error] name server send non 200 code (likely shutting down/bu
            return;
        }

        System.out.println(response.body());
        JSONParser parser = new JSONParser();
        JSONObject json = (JSONObject) parser.parse(response.body());
        // Adding to ip cache
        int id = ((Long) json.get("id")).intValue();
        String ip = String.valueOf(json.get("ip"));


        HttpRequest deleteRequest = HttpRequest.newBuilder(
                        URI.create("http://"+ip+":8080/api/deleteFile?filename="+filename))
                .POST(HttpRequest.BodyPublishers.ofString(filename))
                .build();

        HttpResponse<String> deleteResponse = HttpClient.newHttpClient().send(deleteRequest, HttpResponse.BodyHandlers.ofStri
        if (NodeParameters.DEBUG) System.out.println("[Deleter] " +deleteResponse.body());

    } catch (IOException | InterruptedException e) {
        if (NodeParameters.DEBUG) System.out.println("[RS] [Error] connection error with name server (likely offline)");
    } catch (ParseException e) {
        throw new RuntimeException(e);
    }
```

Folder watchdog

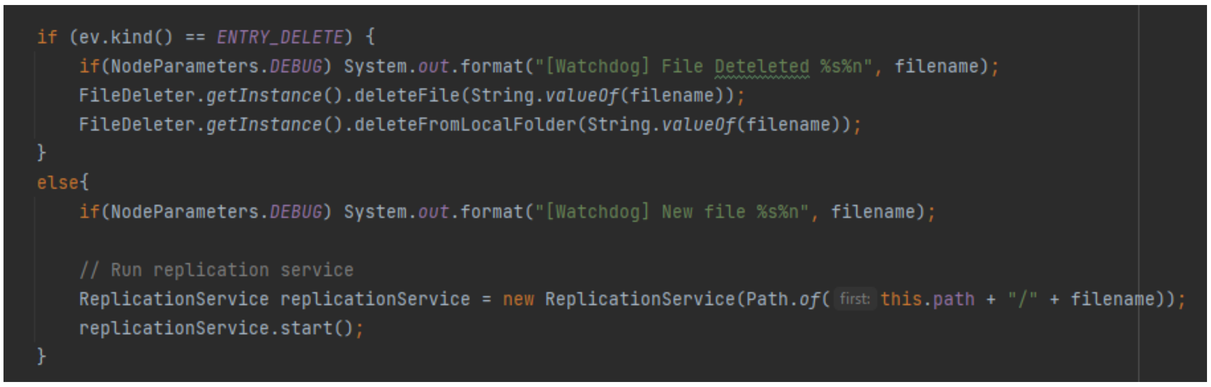
In order to detect changes or additions in the local folder, a watchdog class was implemented.

This thread only listens for events happening in the folder. It was implemented using the Watch Service in the Java nio library.

If a file was added, modified or deleted appropriate action is taken.

For deletion, the file is deleted from the disk and also from the file system. In addition, the file is also deleted from the nodes with the replica.

If it’s a new or modified file the replication service is started to send the file to the correct node.

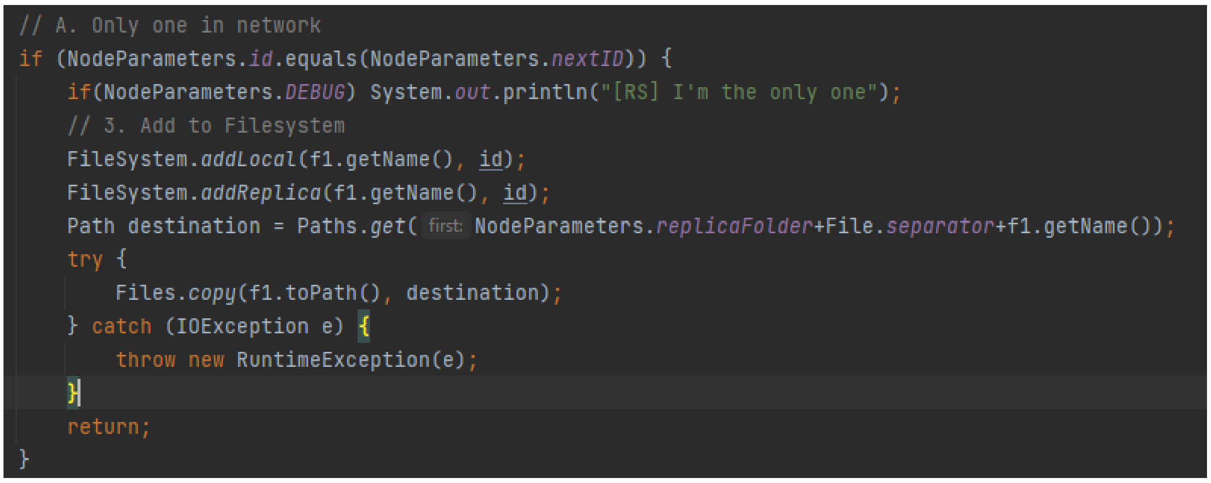


```java
if (ev.kind() == ENTRY_DELETE) {
    if(NodeParameters.DEBUG) System.out.format("[Watchdog] File Deteleted %s%n", filename);
    FileDeleter.getInstance().deleteFile(String.valueOf(filename));
    FileDeleter.getInstance().deleteFromLocalFolder(String.valueOf(filename));
}
else{
    if(NodeParameters.DEBUG) System.out.format("[Watchdog] New file %s%n", filename);

    // Run replication service
    ReplicationService replicationService = new ReplicationService(Path.of( first: this.path + "/" + filename));
    replicationService.start();
}
```

Replication Service

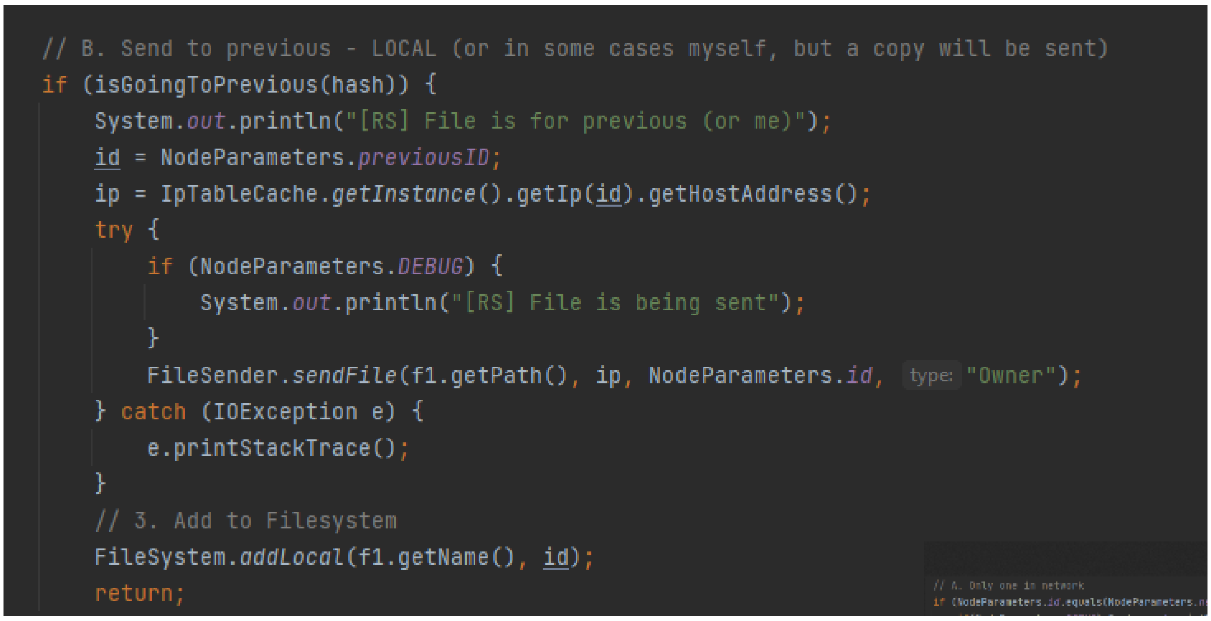
The replication is a class that does exactly what it says in a couple of steps.

It first checks whether the node is the only one in the system. In that case the replica is saved in its own replica folder. (If another nodes comes along to which the file should belong, the sync agent will sort things out).



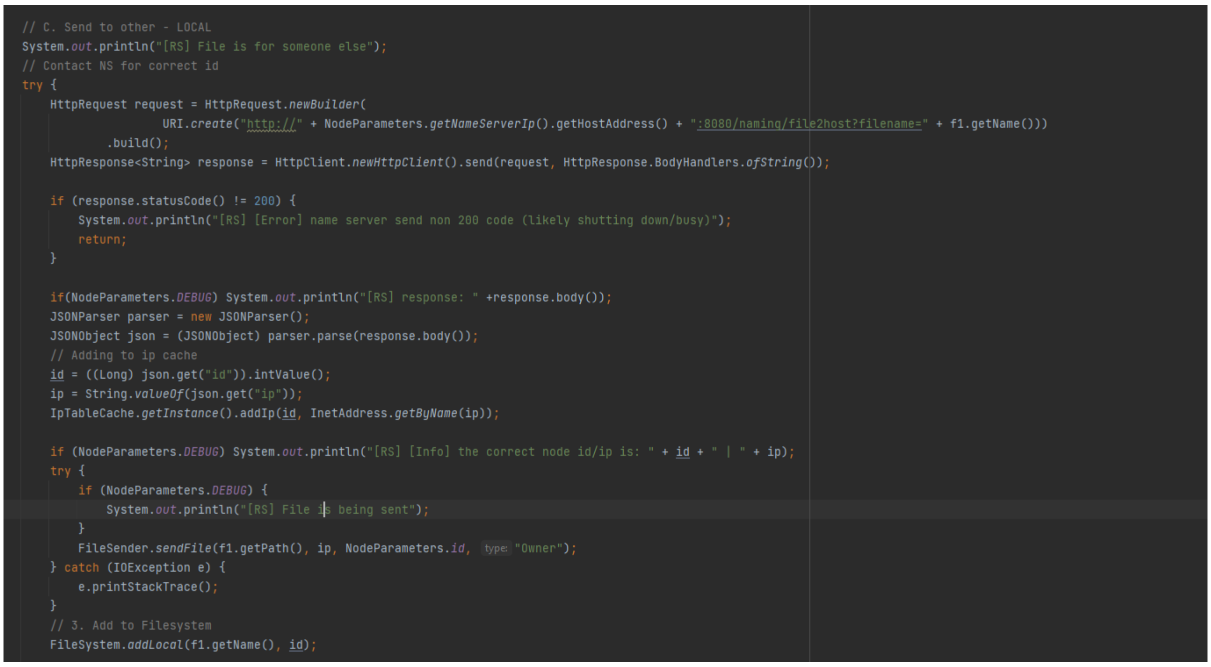
```java
// A. Only one in network
if (NodeParameters.id.equals(NodeParameters.nextID)) {
    if(NodeParameters.DEBUG) System.out.println("[RS] I'm the only one");
    // 3. Add to Filesystem
    FileSystem.addLocal(f1.getName(), id);
    FileSystem.addReplica(f1.getName(), id);
    Path destination = Paths.get( first: NodeParameters.replicaFolder+File.separator+f1.getName());
    try {
        Files.copy(f1.toPath(), destination);
    } catch (IOException e) {
        throw new RuntimeException(e);
    }
    return;
}
```

Otherwise if there are multiple nodes in the system, first it is checked whether the file should be for the previous neighbour (this can be done without the naming server) or for myself. This is done based on the hash value of the file. In both cases, the replica is sent to the previous neighbour.



```java
// B. Send to previous - LOCAL (or in some cases myself, but a copy will be sent)
if (isGoingToPrevious(hash)) {
    System.out.println("[RS] File is for previous (or me)");
    id = NodeParameters.previousID;
    ip = IpTableCache.getInstance().getIp(id).getHostAddress();
    try {
        if (NodeParameters.DEBUG) {
            System.out.println("[RS] File is being sent");
        }
        FileSender.sendFile(f1.getPath(), ip, NodeParameters.id,  type: "Owner");
    } catch (IOException e) {
        e.printStackTrace();
    }
    // 3. Add to Filesystem
    FileSystem.addLocal(f1.getName(), id);
    return;
```

In all other cases (the file is for another node), the filesender is used to send the file to the correct location. For this to happen, the node that should “own” the file is fetched from the naming server. Then, the file is sent.



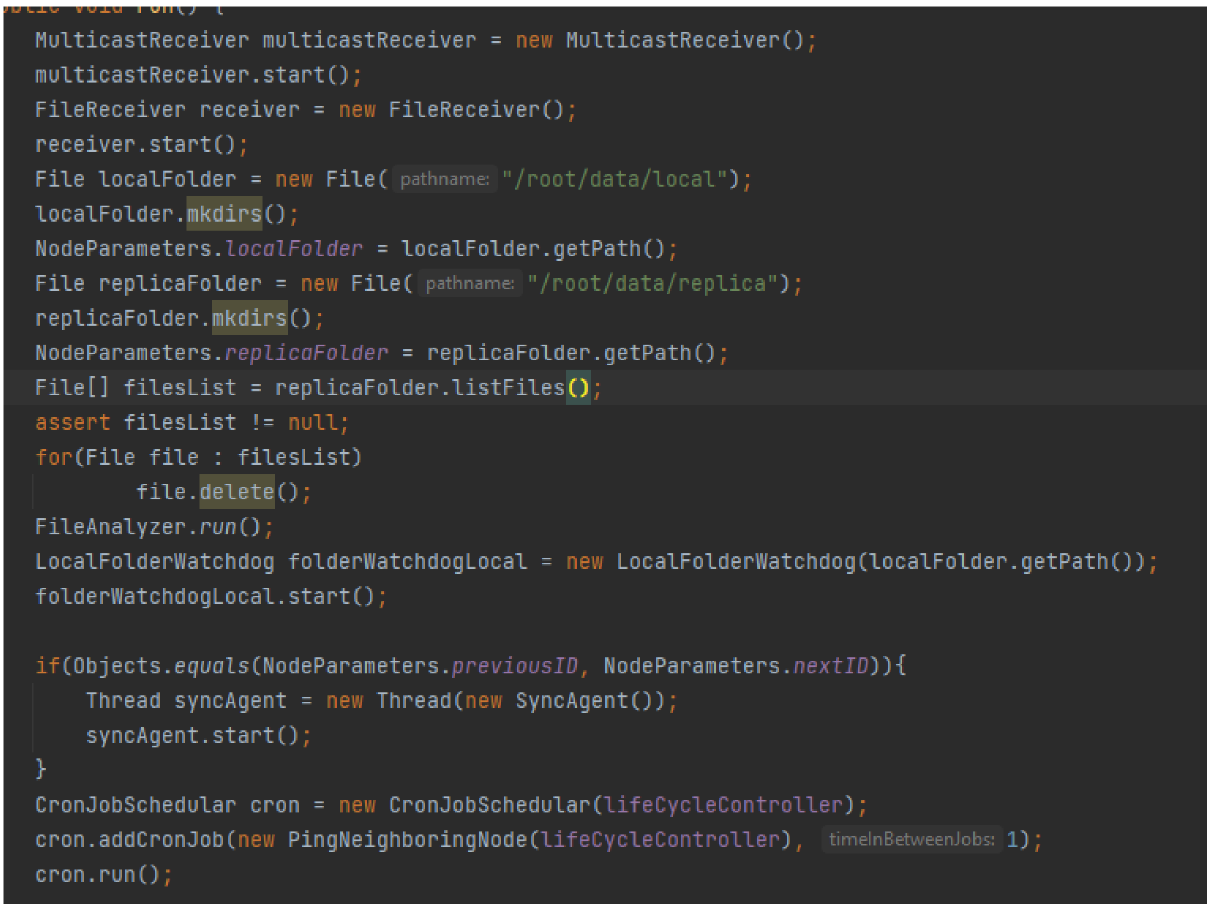
```java
// C. Send to other - LOCAL
System.out.println("[RS] File is for someone else");
// Contact NS for correct id
try {
    HttpRequest request = HttpRequest.newBuilder(
                URI.create("http://" + NodeParameters.getNameServerIp().getHostAddress() + ":8080/naming/file2host?filename=" + f1.getName()))
            .build();
    HttpResponse<String> response = HttpClient.newHttpClient().send(request, HttpResponse.BodyHandlers.ofString());

    if (response.statusCode() != 200) {
        System.out.println("[RS] [Error] name server send non 200 code (likely shutting down/busy)");
        return;
    }

    if(NodeParameters.DEBUG) System.out.println("[RS] response: " +response.body());
    JSONParser parser = new JSONParser();
    JSONObject json = (JSONObject) parser.parse(response.body());
    // Adding to ip cache
    id = ((Long) json.get("id")).intValue();
    ip = String.valueOf(json.get("ip"));
    IpTableCache.getInstance().addIp(id, InetAddress.getByName(ip));

    if (NodeParameters.DEBUG) System.out.println("[RS] [Info] the correct node id/ip is: " + id + " | " + ip);
    try {
        if (NodeParameters.DEBUG) {
            System.out.println("[RS] File is being sent");
        }
        FileSender.sendFile(f1.getPath(), ip, NodeParameters.id,  type: "Owner");
    } catch (IOException e) {
        e.printStackTrace();
    }
    // 3. Add to Filesystem
    FileSystem.addLocal(f1.getName(), id);
```

In all of the above cases, the file system is updated accordingly.

Now onto the three phases of replication

Startup

After discovery and bootstrap, a couple of things happpen:



```java
MulticastReceiver multicastReceiver = new MulticastReceiver();
multicastReceiver.start();
FileReceiver receiver = new FileReceiver();
receiver.start();
File localFolder = new File( pathname: "/root/data/local");
localFolder.mkdirs();
NodeParameters.localFolder = localFolder.getPath();
File replicaFolder = new File( pathname: "/root/data/replica");
replicaFolder.mkdirs();
NodeParameters.replicaFolder = replicaFolder.getPath();
File[] filesList = replicaFolder.listFiles();
assert filesList != null;
for(File file : filesList)
        file.delete();
FileAnalyzer.run();
LocalFolderWatchdog folderWatchdogLocal = new LocalFolderWatchdog(localFolder.getPath());
folderWatchdogLocal.start();

if(Objects.equals(NodeParameters.previousID, NodeParameters.nextID)){
    Thread syncAgent = new Thread(new SyncAgent());
    syncAgent.start();
}
CronJobSchedular cron = new CronJobSchedular(lifeCycleController);
cron.addCronJob(new PingNeighboringNode(lifeCycleController),  timeInBetweenJobs: 1);
cron.run();
```

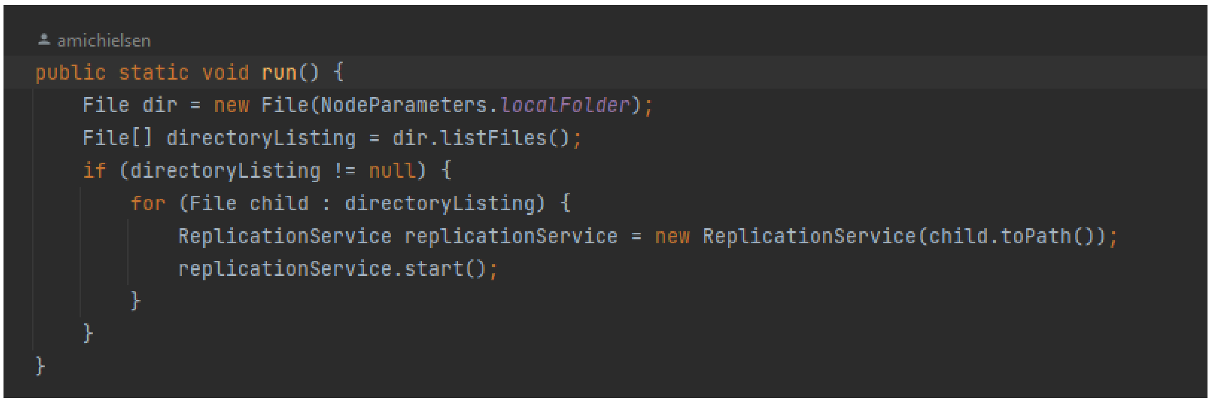
-The filereceiver is started

-A local folder is created if this wasn’t the case yet

-A replica folder is created if this wasn’t the case yet

-The replica folder is cleared so no old files can interfere with the system.

-An analyser is run scanning the local folder and replicating the files to the correct nodes using the replication service mentioned above.


amichielsen
```java
public static void run() {
    File dir = new File(NodeParameters.localFolder);
    File[] directoryListing = dir.listFiles();
    if (directoryListing != null) {
        for (File child : directoryListing) {
            ReplicationService replicationService = new ReplicationService(child.toPath());
            replicationService.start();
        }
    }
}
```

-The watchdog is started for the local folder.

Update

Updates are handled through the watchdog mentioned above.

Shutdown

This was probably the most complex part of the replication service.

Firstly the sequence from previous sessions are carried out.

-The node removes itself from the naming service

-The node updates the configuration of its neighbours.

Then the replication starts.

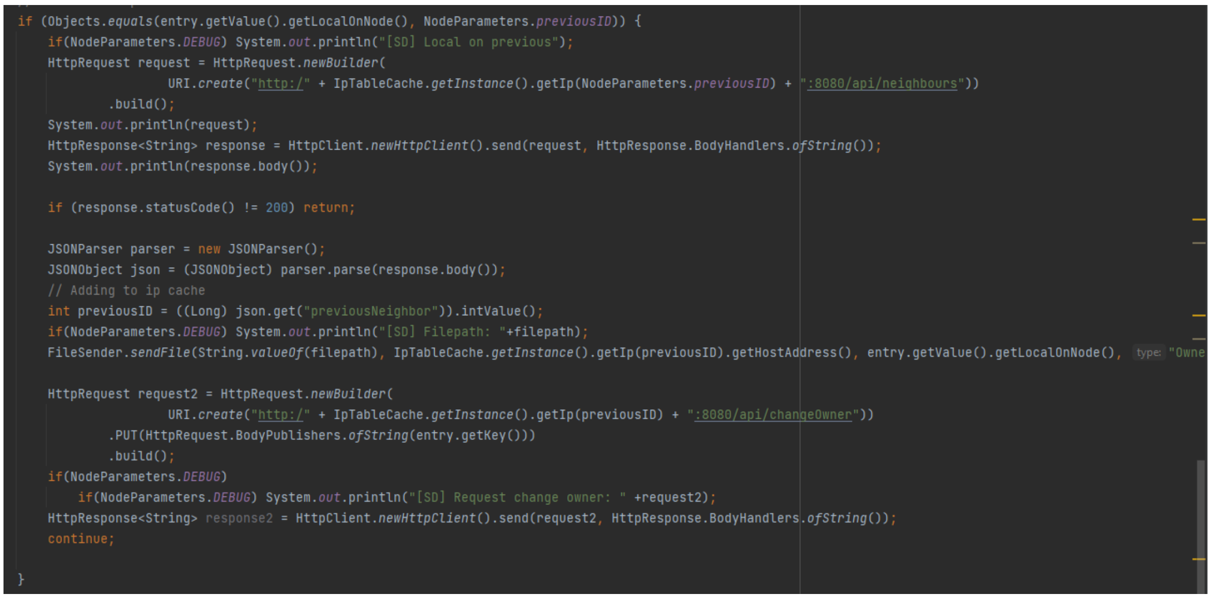
First it sends the all its replicated files to the previous node except for when the previous node is the local node of the file.

To achieve this, the process goes through the list of all replicated files and checks its parameters (localOnNode).

If the file is local on the previous one, the previous neighbour of the previous neighbour is fetched.

With this info, the file is sent to the previous of the previous using the File Sender class mentioned above.

Afterwards, a REST request is also sent letting the receiver know that it is the new owner of the file and that it should update its File System accordingly.

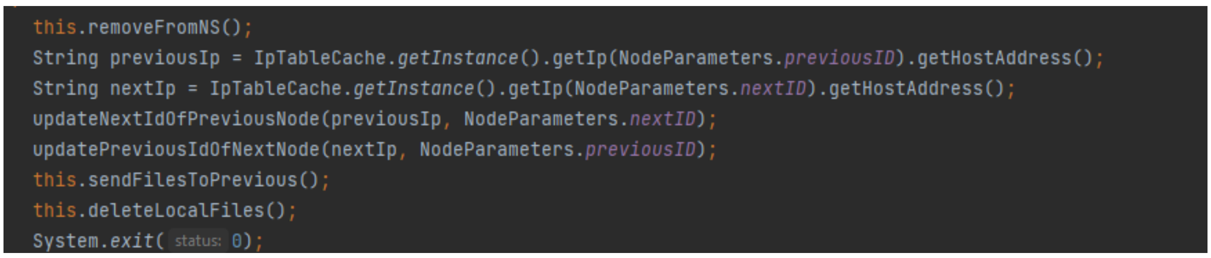


```java
if (Objects.equals(entry.getValue().getLocalOnNode(), NodeParameters.previousID)) {
    if(NodeParameters.DEBUG) System.out.println("[SD] Local on previous");
    HttpRequest request = HttpRequest.newBuilder(
                    URI.create("http:/" + IpTableCache.getInstance().getIp(NodeParameters.previousID) + ":8080/api/neighbours"))
            .build();
    System.out.println(request);
    HttpResponse<String> response = HttpClient.newHttpClient().send(request, HttpResponse.BodyHandlers.ofString());
    System.out.println(response.body());

    if (response.statusCode() != 200) return;

    JSONParser parser = new JSONParser();
    JSONObject json = (JSONObject) parser.parse(response.body());
    // Adding to ip cache
    int previousID = ((Long) json.get("previousNeighbor")).intValue();
    if(NodeParameters.DEBUG) System.out.println("[SD] Filepath: "+filepath);
    FileSender.sendFile(String.valueOf(filepath), IpTableCache.getInstance().getIp(previousID).getHostAddress(), entry.getValue().getLocalOnNode(), type: "Owne

    HttpRequest request2 = HttpRequest.newBuilder(
                    URI.create("http:/" + IpTableCache.getInstance().getIp(previousID) + ":8080/api/changeOwner"))
            .PUT(HttpRequest.BodyPublishers.ofString(entry.getKey()))
            .build();
    if(NodeParameters.DEBUG)
        if(NodeParameters.DEBUG) System.out.println("[SD] Request change owner: " +request2);
    HttpResponse<String> response2 = HttpClient.newHttpClient().send(request2, HttpResponse.BodyHandlers.ofString());
    continue;

}
```

If the file is not local on the previous one, the same sequence is done but for the previous node.

After sending the files around, the node shutting down will let the owners of the its local files know that they can remove their files. This process is the same as when a local file is deleted (mentioned before).

Finally, the system exits and the program stops running.



```java
this.removeFromNS();
String previousIp = IpTableCache.getInstance().getIp(NodeParameters.previousID).getHostAddress();
String nextIp = IpTableCache.getInstance().getIp(NodeParameters.nextID).getHostAddress();
updateNextIdOfPreviousNode(previousIp, NodeParameters.nextID);
updatePreviousIdOfNextNode(nextIp, NodeParameters.previousID);
this.sendFilesToPrevious();
this.deleteLocalFiles();
System.exit( status: 0);
```

Whole process of shutdown

Tests

Now onto some tests.

First, I added some files to a node to test whether its files are spread correctly.

The files were the following

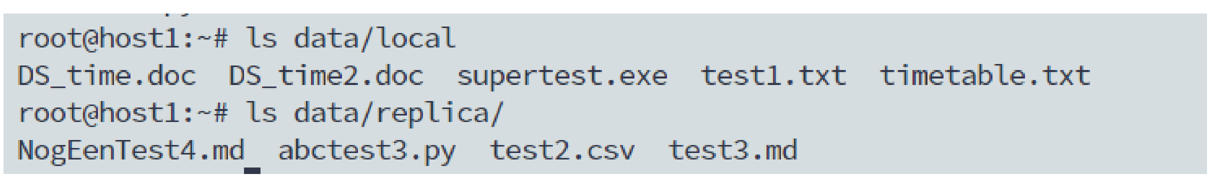
Files

Filename	Hash	Should be on node (in order)	Local Node: hash (nodenumber)

Filename	Hash	Should be on node (in order)	Local Node: hash (nodenumber)
test2.csv	7047	1, 2, 3	30988 (2)
abctest3.py	16488	1, 2, 3	27131 (3)
NogEenTest4.md	26153	4, 1	23274(4)
test3.md	7626	1, 2	23274(4)
test1.txt	7040	2, 3, 4, 1	2077 (1)
1zw.bx	27973	3, 4, 1, 2	23274(4)
timetable.txt	1406	2, 3, 4, 1	2077 (1)

test1.txt should be replicated to node 1 according to the hash value but because it is also local there, it is transferred to node 2. Same for NogEenTest4.md but then on node 4 and should go to node 1.

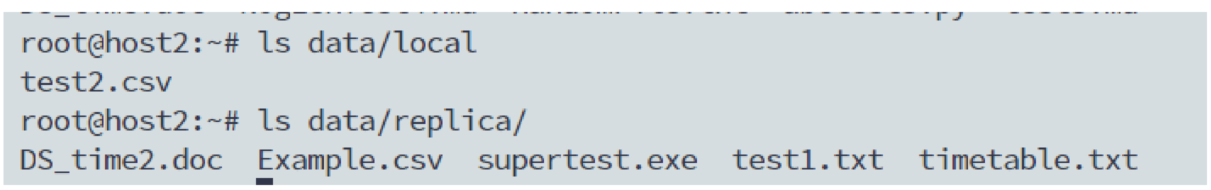
I checked all nodes to see if this happened correctly. I show the local and replica folder for each node.



```
root@host1:~# ls data/local
DS_time.doc  DS_time2.doc  supertest.exe  test1.txt  timetable.txt
root@host1:~# ls data/replica/
NogEenTest4.md  abctest3.py  test2.csv  test3.md
```

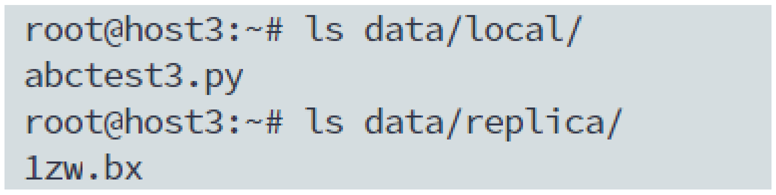
The files on host 1 are already correct! (The other files are from my peers testing our program)

On node 2 we should expect test1.txt and timetable.txt in replica which is the case (also other local files on node 1 who should belong to node 1 but are transferred because they are local on 1).



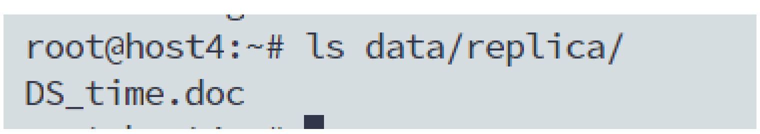
```
root@host2:~# ls data/local
test2.csv
root@host2:~# ls data/replica/
DS_time2.doc  Example.csv  supertest.exe  test1.txt  timetable.txt
```

On node 3 we expect 1zw.bx



```
root@host3:~# ls data/local/
abctest3.py
root@host3:~# ls data/replica/
1zw.bx
```

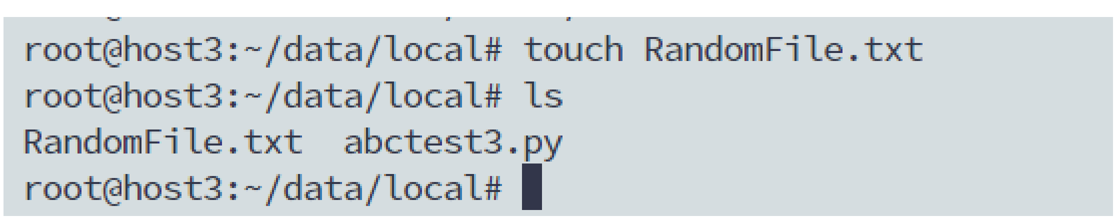
And on node 4 we expect nothing in the replication folder.



```
root@host4:~# ls data/replica/
DS_time.doc
```

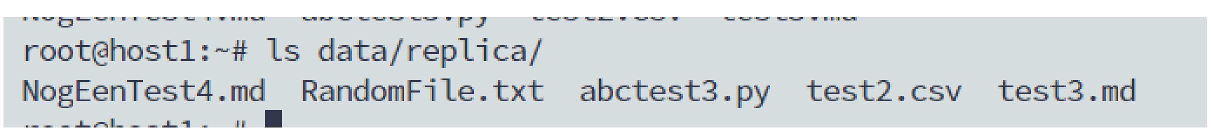
The DS_time.doc file is not mine but when we calculate its hash value: 25864. We notice that it in fact belongs to node 4. More prove that the system actually works.

We add a new file to the system: ‘RandomFile.txt’ to node 3.



```
root@host3:~/data/local# touch RandomFile.txt
root@host3:~/data/local# ls
RandomFile.txt  abctest3.py
root@host3:~/data/local#
```

Its hash value is 19108 so it should go to node 1. Which is the case.

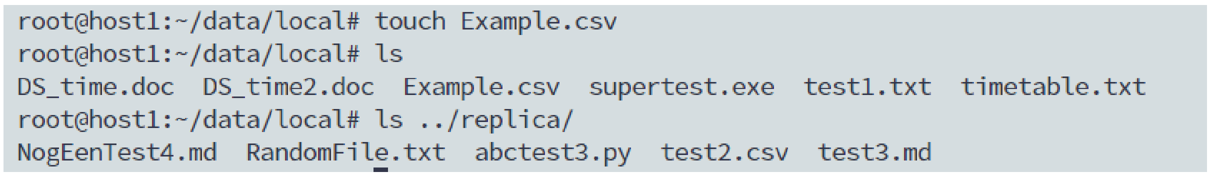


```
root@host1:~# ls data/replica/
NogEenTest4.md  RandomFile.txt  abctest3.py  test2.csv  test3.md
```

Then we add a file whose hash points to the node that stores the file locally.

‘Example.csv’.

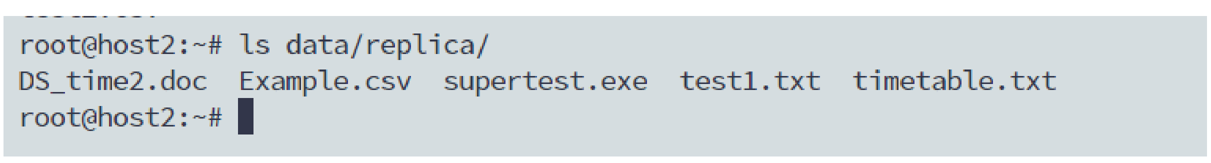
The hash for this file is 14277 (so it should go to node 1). We add this file to node 1.



```
root@host1:~/data/local# touch Example.csv
root@host1:~/data/local# ls
DS_time.doc  DS_time2.doc  Example.csv  supertest.exe  test1.txt  timetable.txt
root@host1:~/data/local# ls ../replica/
NogEenTest4.md  RandomFile.txt  abctest3.py  test2.csv  test3.md
```

We see that the file doesn’t show up in the replicated folder of node 1.

Instead, if we go to the previous neighbour of node 1, node 2:

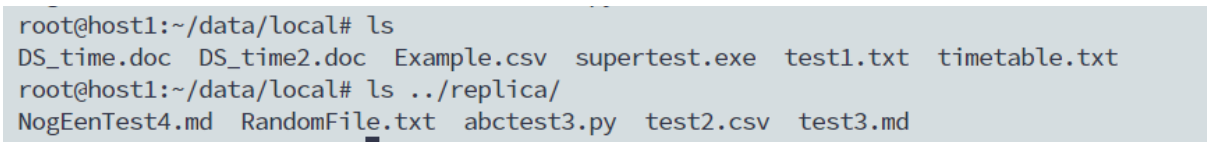


```
root@host2:~# ls data/replica/
DS_time2.doc  Example.csv  supertest.exe  test1.txt  timetable.txt
root@host2:~#
```

The file is replicated here!

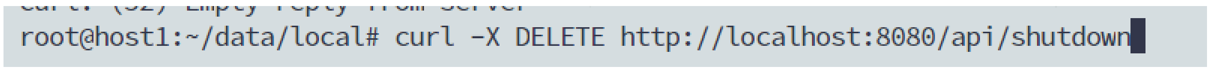
As a final test, we are going to shut down one node.

Let’s say node 1 whose local and replica files can be seen in the screenshot



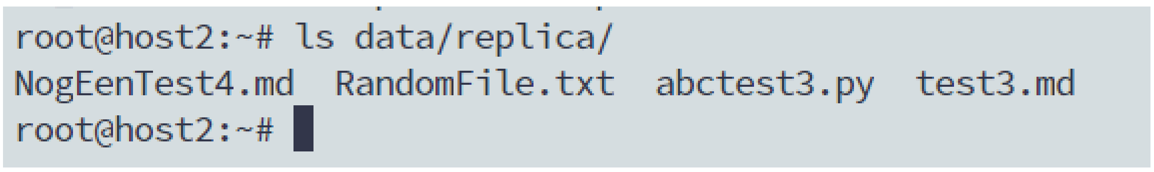
```
root@host1:~/data/local# ls
DS_time.doc  DS_time2.doc  Example.csv  supertest.exe  test1.txt  timetable.txt
root@host1:~/data/local# ls ../replica/
NogEenTest4.md  RandomFile.txt  abctest3.py  test2.csv  test3.md
```

We shut this node down:



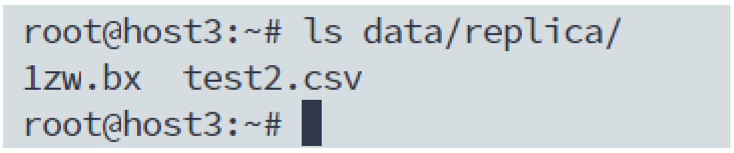
```
root@host1:~/data/local# curl -X DELETE http://localhost:8080/api/shutdown
```

On node 2 (previous neighbour) we see that the files that were replicated on node 1 are replicated to this one



```
root@host2:~# ls data/replica/
NogEenTest4.md  RandomFile.txt  abctest3.py  test3.md
root@host2:~#
```

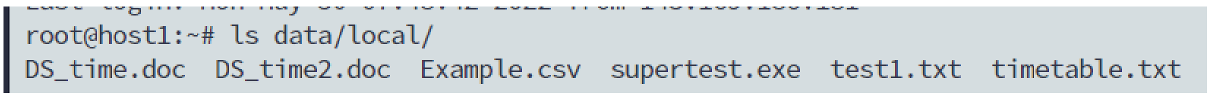
Except for 1, test2.csv which is local on node 2. This one went to the previous of the previous node 3:



```
root@host3:~# ls data/replica/
1zw.bx  test2.csv
root@host3:~#
```

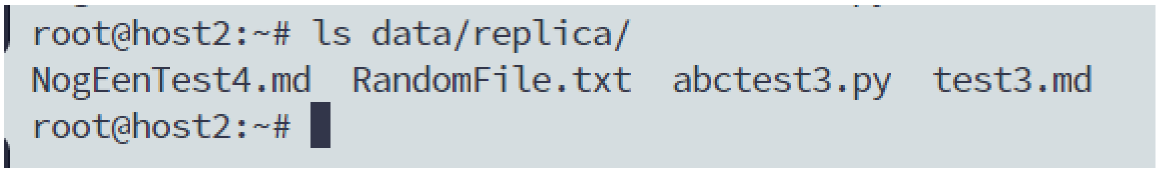
On top of that, the files that were replicated on this, file local on node 1 are gone:

The local files were:

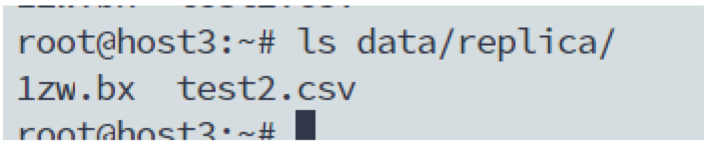


```
root@host1:~# ls data/local/
DS_time.doc  DS_time2.doc  Example.csv  supertest.exe  test1.txt  timetable.txt
```

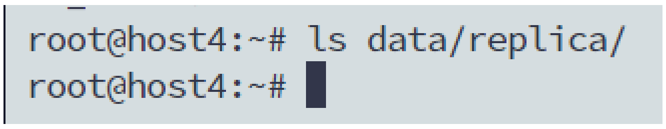
Let’s look in all folders on all nodes (it can be compared with above screenshots). No node 1 local file is to be found in any of these folders.



```
root@host2:~# ls data/replica/
NogEenTest4.md  RandomFile.txt  abctest3.py  test3.md
root@host2:~#
```



```
root@host3:~# ls data/replica/
1zw.bx  test2.csv
root@host3:~#
```



```
root@host4:~# ls data/replica/
root@host4:~#
```