SE424: Distributed Systems	**Recitation 6**
Semester 1 5785	**9 Dec 2024**
Lecturer: Michael J. May	**Kinneret College**

Flat Naming: Building Hierarchical Location Service

We discussed the Hierarchical Location Service (HLS) of Globe in the context of *flat naming systems*. In this recitation, you will develop a tool which lets you explore the functionality of the HLS concept and implement concrete methods for flat name resolution. A screen shot of the tool you will be developing is shown in Figure 1.

Listening address The tool listens for requests on its port and IP address and contacts its father on the HLS tree at the address given in the text box labeled "Father IP". If the node is a root, the "Root" check box should be checked to indicate that there is no parent.

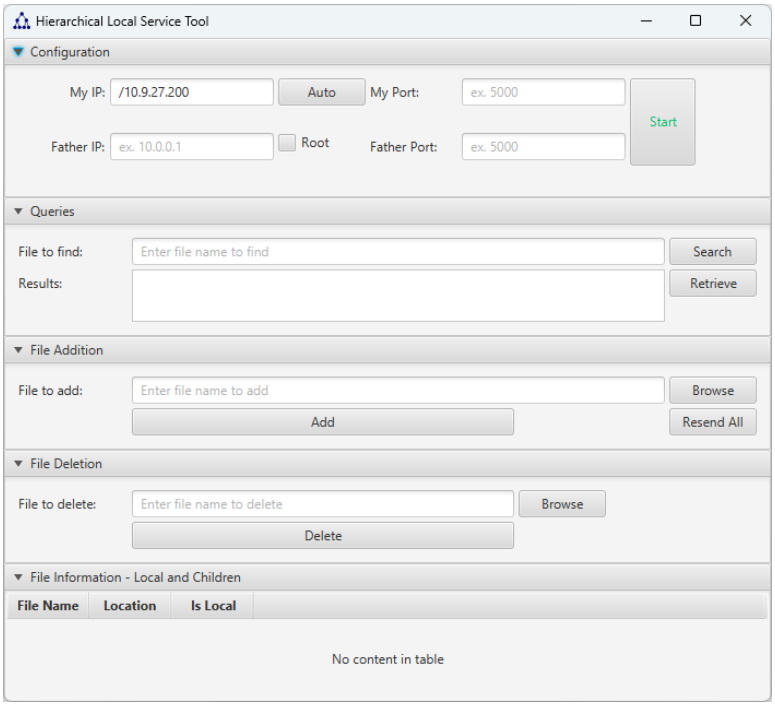


Figure 1: HLS Tool Screen Shot

1 Background: Searching for Files

The *Queries* section of the tool allows you to search the HLS tree for a file by name. There are two general techniques for resolving names when multiple steps are required for the complete resolution process: *recursive resolution and iterative resolution.*

Recursive Name Resolution implies that if A requests information about an object from B and B isn't aware of the complete answer, B will send requests on behalf of A until it has a complete answer. Afterwards, B will send the final answer to A.

Iterative Name Resolution implies that if A requests information about an object from B and B isn't aware of the complete answer, B will send A an *immediate* response with the partial information that

it has. *A* will use the partial information to make further requests until it discovers the final answer.

Which technique to choose is a design decision which affects how the HLS lookup works - either recursively or iteratively. Iterative lookup makes the client more complex and the server simpler. Recursive lookup makes the client simpler and the server more complex.

Search Conclusion Another design decision in designing the tool is deciding when a search action ends. Remember that all searches begin locally and may ascend or descend the hierarchical tree as necessary to discover the results. There are two reasonable approaches for how to do search. The first approach is simpler; the second is more complicated.

First Hit: The searching node begins looking locally for the desired file. We may distinguish three cases:

(a) The file is found locally. The local result is shown in the Results list and the search ends.

(b) The file is not found locally, but is found in the file information table at one or more of the child subtrees. If there is more than one option, one subtree is chosen at random and the child is queried either recursively or iteratively, depending on how the system is built. The final result from the subtree (*i.e.* where a copy of the file is found locally) is shown in the Results list and the search ends.

(c) The file is not found locally or in the file information table. A query is sent to the father of the node. The process continues either iteratively or recursively, depending on how the system is built.

In this case, the Results list will normally have only one result or will show a *not found* result if the file isn't found. We will discuss more about *not found* below in Section 2.

All Locations In this approach, the searching node will attempt to find all copies of the desired file. The searching process therefore involves three stages.

(a) If the file is found locally, its location is shown in the Results list.

(b) If the file is found in the file information table at one or more of the child subtrees, the node searches all subtrees which hold a copy of the file and discovers the nodes which have local copies. All resulting nodes (*i.e.* where a copy of the file is found locally) are shown in the Results list.

(c) In all cases (except when the node is the root), the node sends a request to its father. The process continues at the father either iteratively or recursively, depending on how the system is built.

In this case, the Results list may have more than one result shown for each search. If the file is not found anywhere, a *not found* result is shown. We will discuss more about *not found* below in Section 2.

2 HLS Protocol

Each node includes a multithreaded listener with a "cancel" ability, a data table which can store information about local files and those which are located at child subtrees, and logic for client requests.

The nodes implement the following protocol for communication:

1. `LOOKUP fileName` - This command is sent to the node to find the file. The response is either the word "NotFound" or a semicolon separated list of locations, possibly including the word "Local". Some sample responses are:
   - `NotFound`
   - `Local`
   - `10.0.0.4:5000`

- `Local;10.0.0.4:5000;10.0.0.5:4000`

Regardless of how you implement search, an entry of "Local" implies that the node sending the response has a local copy of the requested file. When an IP address and port appear in the response, the implication depends on whether you have implemented iterative or recursive name resolution.

- With iterative name resolution, an IP address and port (*e.g.* `10.0.0.4:5000`) are a hint that the responding node thinks that the listed IP address and port (`10.0.0.4:5000`) has a local copy of the file. The IP address and port (`10.0.0.4:5000`) might refer to the responding node's father, the root, or one of the responding node's children. The node performing the search will only be certain of the location of the file when it gets a "Local" response.
- With recursive name resolution, every IP address and port in the response refers to an actual location where the file is located. That means that if A hears `10.0.0.4:5000` from B, B has contacted `10.0.0.4:5000` and gotten a "Local" response or B has talked to someone else who has done so.

"NotFound" is a special response, sent only by a node which is certain that the file is not to be found anywhere in the system. Its use differs based on whether you use iterative or recursive name resolution.

- With iterative name resolution, *only the root* can respond with a "NotFound" response. It will send that response when it is queried for a file not found in its local store or in any of the subtrees in the system.
- With recursive name resolution, "NotFound" will be sent back to any node which queries for a file which is not found. As in the iterative case, only the root can authoritatively declare that a file isn't found anywhere. After the root has responded "NotFound" to a query, the recursive resolvers all return "NotFound" along the return path to the original searcher.

2. `ADD fileName IP:Port` - This command is sent to add information about a record found in a child of the sender or itself. The provided IP address and port indicate the IP address and listening port of the node which is sending the message. The sender either holds the file itself or is propagating a message from one of its subtrees. The message causes a new record to be added if needed. If the record is new, an update message is propagated to the father (unless the node is the root).

   If the file to be added already exists in the recipient's table under the sender's IP address and port, the message is ignored. Otherwise, the information is added to the node's file table.

3. `DELETE fileName IP:Port` - This command is sent to update the node to delete a file that previously was added by the sender. The message is sent by a node when it deletes a local copy or receives a deletion message from one of its subtrees and the deletion needs to be propagated. The message causes the record for the file name at the given IP address and port to be removed at the recipient to reflect the deletion. If the deletion needs to be propagated to the father (*i.e.* it's the last entry in the table for the specific file), it is propagated to the father (unless the node is the root).

   If the file to be deleted doesn't exist in the recipient's table under the sender's IP address and port, the message is ignored.

2.1 Additional Messages

Note that there is no message for joining as a child of a node. This is intentional as the messages above suffice for searching, adding, and deleting. We could add JOIN, LEAVE, or other message to the protocol to make it richer. The outcome would then be that the parents are aware of all of their children and can take action if a child goes away.

3 GUI: File Deleting and Adding

As shown in Figure 1, the tool has areas for adding and deleting files.

When adding a file, the user selects the local location of the file using an Open File Dialog. The added file is then added to the local file list. A message is sent to the father node if necessary.

The Resend All button sends ADD messages to the node's father for all files in the node's table. It's useful when a node has been offline for a while or if a node changes its father.

A file can be deleted from the tool only if it's a local file. In that case, the record for the file is removed from the local file list. A message is sent to the father node if necessary.

4 GUI: File Information Table

The information stored in the File Information Table shown in Figure 1 includes information about two kinds of files - those stored locally and those stored at children.

Rows where the field "Is Local" has the value "True" refer to files which are stored locally at the node. For them, the location must be a local directory on the computer.

Rows where the field "Is Local" has the value "False" refer to files which are stored at children of the node. For them, the location must be a child of the computer in the form of an IP address followed by a colon and a port number (*e.g.* `10.0.0.4:5000`).

A sample file information table is shown in the following table:

File Name	Location	Is Local
test1.txt	C:\myfiles\	True
test1.txt	10.0.0.4:5000	False
test1.txt	10.0.0.5:5000	False
test2.pdf	C:\myfiles\	True
test3.doc	10.0.0.5:5000	False

Note that in the above example, `test1.txt` is stored at multiple locations. File `test1.txt` is stored locally at path `C:\myfiles\test1.txt`, at one subtree which begins at 10.0.0.4:5000, and at another subtree which begins at 10.0.0.5:5000. Both 10.0.0.4:5000 and 10.0.0.5:5000 are children of the node which stores the table. Recall that `test1.txt` might not in fact be stored directly at either 10.0.0.4:5000 or 10.0.0.5:5000, but rather at descendants of them. In fact, `test1.txt` might be stored multiple times at nodes on the subtrees beginning at 10.0.0.4:5000 and 10.0.0.5:5000.

File `test2.pdf` is stored only locally at the node. File `test3.doc` is stored at the subtree beginning at 10.0.0.5:5000. Again, it might be stored at a descendant of 10.0.0.5:5000 or 10.0.0.5:500 itself and there may be multiple copies of `test3.doc` in the subtree.

5 GUI: Retrieve File

If there are results from a search, clicking on the "Retrieve" button will retrieve the file from the specified location. The user will be given a chance to select where the file is to be downloaded to.

© Michael J. May

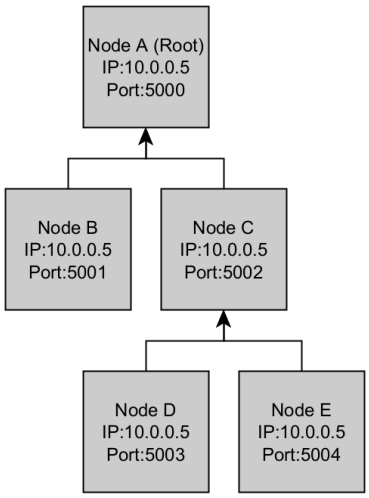


Figure 2: HLS sample topology

6 A Sample Topology

To give an example of how the tool behaves, let's consider a sample topology shown in Figure 2

The topology shows 5 nodes and their associated IP addresses and ports. In this case, the root node is the one with IP address 10.0.0.5:5000. It has two children (ports 5001, 5002). Node 5002 has two more children (ports 5003, 5004). After connecting the instances together and adding some files, we get the states shown in Figures 3 and 4.

The files are shown in the local indexes. Each row in the file information table shows a single file which may be located locally (true) or remotely (false).

7 What to do

7.1 Step 1: Initial implementation

Your first task is to implement the HLS protocol and functionality in a multithreaded peer-to-peer application. The main elements are:

1. Name resolution mechanism (file search) using iterative or recursive lookup. When the file is located (a Local answer is received), the user must be shown at least one IP address and port at which is can be found. See above in Section 1 for a discussion about how to decide when a search is completed.
2. State maintenance functionality: adding and deleting files.

You may choose to implement the HLS tool using the sockets and threads technique we learned or using RMI.

7.2 Step 2: File Download

Once you have completed the above parts, add the following additional functionality to make the tool more interesting.

Add the following message to the protocol:

1. `RETRIEVE fileName` - This command is sent to ask the server to send back the file listed. If the file is not found locally, the response sent back is the word "Not". If it is found, the word "Found" is sent as a string followed by the complete contents of the file. Ideas for sending the file include breaking it into 4096 byte chunks and sending them as `byte[]` over a *InputStream* connection on a separate socket connection or as base64 encoded strings over a `PrintWriter` connection.

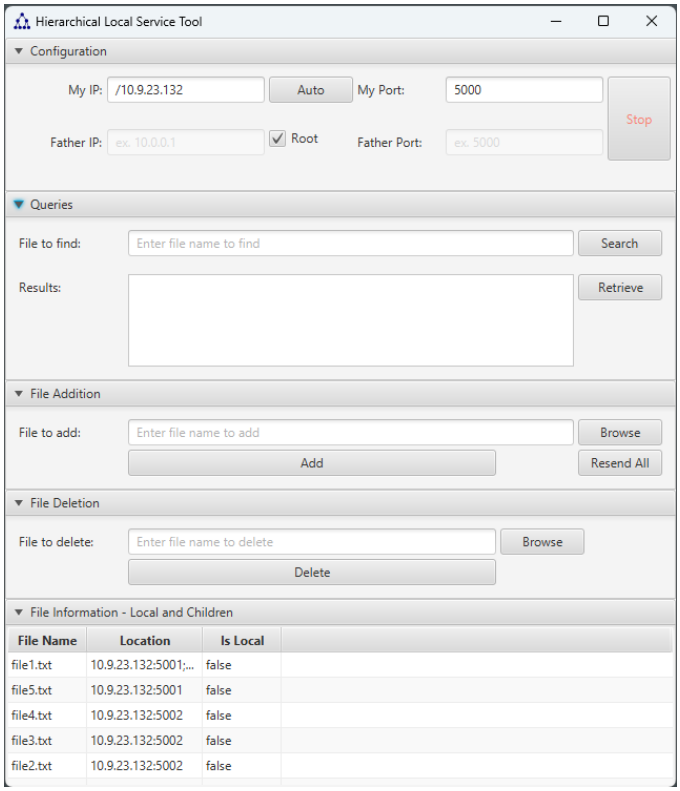
When the file is returned, open a *Save file as...* dialog box to let the user select where to save the file.

8 Additional User Interface Requirements

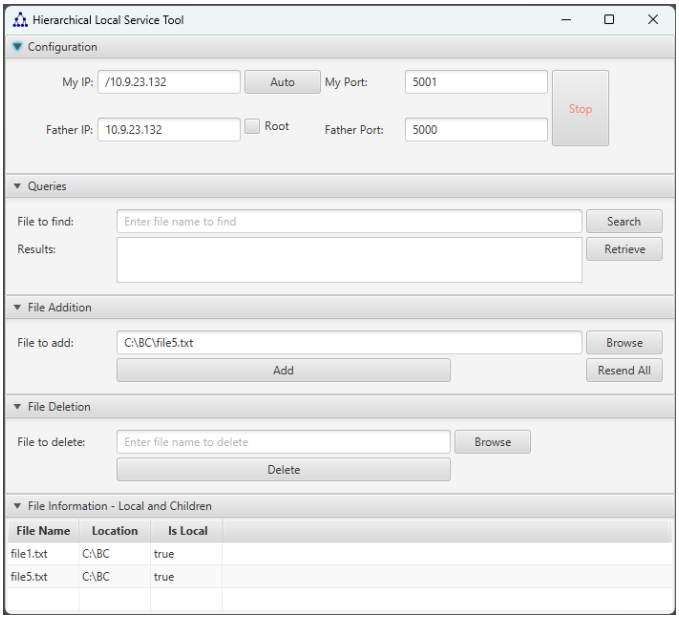
The tool and its GUI must support the following requirements:

1. Robustness: The tool must perform error handling, not crashing due to unhandled exceptions.
2. Responsiveness: The tool must enable the user to interact with the user interface when listening or processing. This implies that network communications must not be done on the same thread as the user interface.

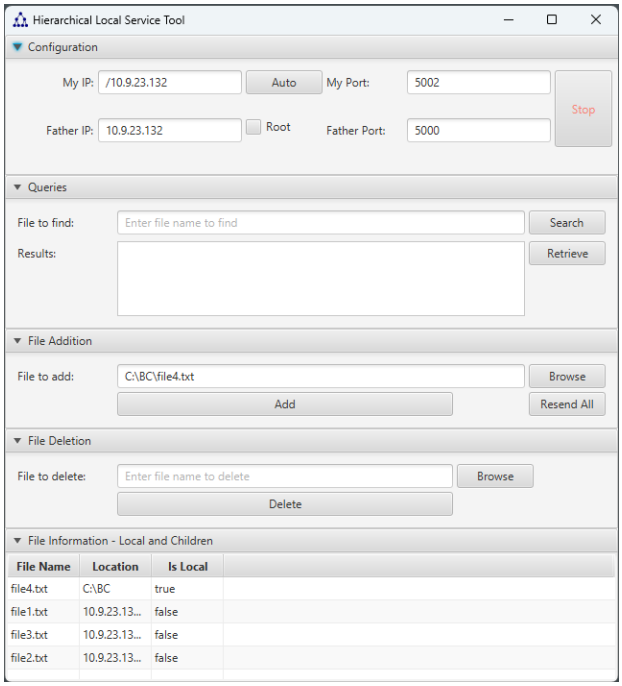
© Michael J. May



(a) Root (5000) Node A State

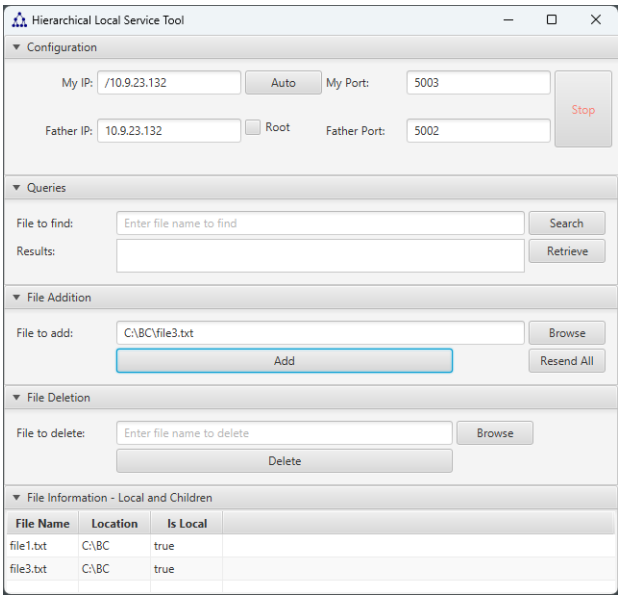


(b) Child (5001) Node B State

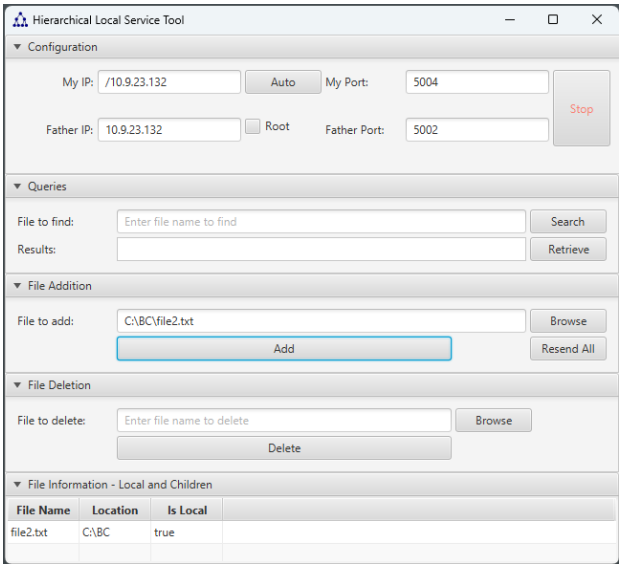


(c) Child (5002) Node C State

Figure 3: Nodes in the sample topology



(a) Grandchild (5003) Node D



(b) Grandchild (5004) Node E State

Figure 4: Nodes in the sample topology