Operating Systems and Networks

Network File System

Simply put - implement a simple network (or rather, a distributed) file system *from scratch*.
Image

Deadline: 11:59PM, 19th November, 2023

Github Classroom

We will use GitHub classroom for the projects. The instructions for registering your team are given below. Please read all the instructions before proceeding.

1. One teammate has to create the team. They need to click on this link. This opens a page of registered teams. Your team should not exist here at this moment. The next two instructions are for the same teammate.
2. Create a new team with only the team number allotted as the name. The team number can be found in the moodle announcement. So if your team number is 10, the team's name should be 10. Nothing more, nothing less.
3. On the next screen, accept the assignment. The other members can now join this team.
4. The remaining members now need to click on this link. This takes you to the list of registered teams. Join your team, which should have your team number as the name.

Please use the repo that gets created to work on your project.

Introduction [0 Marks]

The **major components** that you'll need to implement are:

1. *Clients*: Clients represent the systems or users requesting access to files within the network file system. They serve as the primary interface to interact with the NFS, initiating various file-related operations such as reading, writing, deleting, and more.
2. *Naming Server*: The Naming Server stands as a pivotal component in the NFS architecture. This singular instance serves as a central hub, orchestrating communication between clients and the storage servers. Its primary function is to provide clients with crucial information about the specific storage server where a requested file or folder resides. Essentially, it acts as a directory service, ensuring efficient and accurate file access.

3. *Storage Servers*: Storage Servers form the foundation of the NFS. These servers are responsible for the physical storage and retrieval of files and folders. They manage data persistence and distribution across the network, ensuring that files are stored securely and efficiently.

Within the NFS ecosystem, clients enjoy a suite of **essential file operations**, enabling seamless interaction with the network file system:

1. *Writing a File/Folder*: Clients can actively create and update the content of files and folders within the NFS. This operation encompasses the storage and modification of data, ensuring that the NFS remains a dynamic repository.
2. *Reading a File*: Reading operations empower clients to retrieve the contents of files stored within the NFS. This fundamental operation grants clients access to the information they seek.
3. *Deleting a File/Folder*: Clients retain the ability to remove files and folders from the network file system when they are no longer needed, contributing to efficient space management.
4. *Creating a File/Folder*: The NFS allows clients to generate new files and folders, facilitating the expansion and organization of the file system. This operation involves the allocation of storage space and the initialization of metadata for the newly created entities.
5. *Listing All Files and Folders in a Folder*: Navigating the NFS structure becomes effortless for clients as they can retrieve comprehensive listings of files and subfolders within a specified directory. This feature aids in efficient file system exploration and management.
6. *Getting Additional Information*: Clients can access a wealth of supplementary information about specific files. This includes details such as file size, access rights, timestamps, and other metadata, providing clients with comprehensive insights into the files they interact with

The **exact specifics** of the operations that you'll need to implement are elaborated below.

Specifications

1. Naming and Storage Servers

1.1 INITIALIZATION [60 MARKS]

The process of initializing the Naming Server (NM) and Storage Servers (SS) sets the foundation for the entire network file system. Here are the key steps involved in starting the system:

1. *Initialize the Naming Server (NM)*: The first step is to initialize the Naming Server, which serves as the central coordination point in the NFS. It is responsible for managing the directory structure and maintaining essential information about file locations.
2. *Initialize Storage Server 1 (SS_1)*: Each Storage Server (SS) is responsible for storing files and interacting with both the Naming Server and clients. Initialization of SS_1 involves several sub-steps:
   - Upon initialization, SS_1 sends vital details about its existence to the Naming Server. This information includes:
     a. IP address: To facilitate communication and location tracking.
     b. Port for NM Connection: A dedicated port for direct communication with the Naming Server.
     c. Port for Client Connection: A separate port for clients to interact with SS_1.
     d. List of Accessible Paths: A comprehensive list of file and folder paths that are accessible on SS_1.
3. *Initialize SS_2 to SS_n*: Following the same procedure as SS_1, additional Storage Servers (SS_2 to SS_n) are initialized, each providing their details to the Naming Server.
4. *NM Starts Accepting Client Requests*: Once all Storage Servers are initialized and registered with the Naming Server, the Naming Server begins accepting client requests. It becomes the central point through which clients access and manage files and directories within the network file system.

1.2 ON STORAGE SERVERS (SS) [120 MARKS]

The Storage Servers play a crucial role in the network file system and are equipped with the following functionalities:

Adding new storage servers: New Storage Servers (i.e., which begin running after the initial initialization phase) have the capability to dynamically add their entries to the Naming Server at any point during execution. This flexibility ensures that the system can adapt to changes and scaling requirements seamlessly.

Commands Issued by NM: The Naming Server can issue specific commands to the Storage Servers, including:

1. Create an Empty File/Directory: The Naming Server can instruct a Storage Server to create an empty file or directory, initiating the storage of new data.
2. Delete a File/Directory: Storage Servers can receive commands to delete files or directories, thereby freeing up storage space and maintaining data consistency.
3. Copy Files/Directories: Storage Servers can copy files or directories from other Storage Servers, with the NM providing the relevant IP addresses for efficient data transfer.

Client Interactions: Storage Servers also facilitate client interactions by providing the following functionalities:

1. Read a File: Clients can request Storage Servers to retrieve and send the content of a specific file, allowing seamless data access.
2. Write to a File: Clients can send data to Storage Servers, which is then written to the designated file. This operation ensures data updates and modifications are correctly stored.
3. Get Size and Permissions: Clients can query Storage Servers to retrieve essential file information such as size and access permissions, aiding in file management.

1.3 ON NAMING SERVER (NM) [30 MARKS]

Storing Storage Server data: One of the fundamental functions of the Naming Server (NM) is to serve as the central repository for critical information provided by Storage Servers (SS) upon connection.

Client task feedback: Upon completion of tasks initiated by clients (as described in Specification 2), the NM plays a pivotal role in providing timely and relevant feedback to the requesting clients.

2. Clients [50 Marks]

Clients initiate communication with the Naming Server (NM) to interact with the NFS. Here's how this interaction unfolds:

Directory Mounting: [Old: Clients establish contact with the NM by pinging it with the mounted directory path. When a client requests a resource at a specific location within the NFS (e.g., "~/../mount/dir1/dir2…"), the NM undertakes a comprehensive search across all registered Storage Servers (SSs), encompassing the given directory structures - "dir1/dir1" - to locate the requested resource.] This specification has been removed (you may implement it if you wish to). Here's how you do it now - say, the client passes `READ dir1/dir2/file.txt` to the NM -> the NM looks over all the accessible paths in SS1, SS2, SS3… then sees that the path is present in SSx -> The NM gives relevant information about SSx to the client.

Functionalities to implement are:

1. Reading, Writing, and Retrieving Information about Files:

- Client Request: Clients can initiate file-related operations such as reading, writing, or obtaining information by providing the file's path. The NM, upon receiving the request, takes on the responsibility of locating the correct SS where the file is stored.
    - NM Facilitation: The NM identifies the correct Storage Server and returns the precise IP address and client port for that SS to the client.

Direct Communication: Subsequently, the client directly communicates with the designated SS. This direct communication is established, and the client continuously receives information packets from the SS until a predefined “STOP” packet is sent or a specified condition for task completion is met. The “STOP” packet serves as a signal to conclude the operation.

2 Creating and Deleting Files and Folders:

- Client Request: Clients can request file and folder creation or deletion operations by providing the respective path and action (create or delete). Once the NM determines the correct SS, it forwards the request to the appropriate SS for execution.
- SS Execution: The SS processes the request and performs the specified action, such as creating an empty file or deleting a file/folder.
- Acknowledgment and Feedback: After successful execution, the SS sends an acknowledgment (ACK) or a STOP packet to the NM to confirm task completion. The NM, in turn, conveys this information back to the client, providing feedback on the task’s status.

3 Copying Files/Directories Between Storage Servers:

- Client Request: Clients can request to copy files or directories between SS by providing both the source and destination paths. The NM, upon receiving this request, assumes responsibility for managing the file/folder transfer between the relevant SS.
- NM Execution: The NM orchestrates the copying process, ensuring that data is transferred accurately. This operation may involve copying data from one SS to another, even if both source and destination paths lie within the same SS.
- Acknowledgment and Client Notification: Upon the successful completion of the copy operation, the NM sends an acknowledgment (ACK) packet to the client. This ACK serves as confirmation that the requested data transfer has been successfully executed.

NOTE: Marks awarded here are essentially for the requests to the NM. Actual implementation of the requests by the NM and SSs have been assigned points in Specification 1.

3. Other features

3.1 MULTIPLE CLIENTS [80 MARKS]

Concurrent Client Access: Your NFS design accommodates multiple clients attempting to access the Naming Server (NM) simultaneously. To ensure a smooth experience, the NM responds to client requests with an initial ACK to acknowledge the receipt of the

request. If this initial ACK is not received within a reasonable timeframe, the client may display a timeout message. Importantly, the NM should not block while processing operations specified in parts 2 and 3 of Specification 2. To achieve this, an initial ACK and a final ACK from the relevant Storage Server (SS) can be employed, allowing the NM to handle other requests between these acknowledgments.

Concurrent File Reading: While multiple clients can read the same file simultaneously, only one client can execute write operations on a file at any given time.

3.2 ERROR CODES [20 MARKS]

Error Handling: Define a set of error codes that can be returned when a client's request cannot be accommodated. For example, the NM should return distinct error codes for situations where a file is unavailable because it doesn't exist and when another client is currently writing to the file. Clear and descriptive error codes enhance the communication of issues between the NFS and clients.

3.3 SEARCH IN NAMING SERVERS [80 MARKS]

Efficient Search: Optimize the search process employed by the Naming Server when serving client requests. Avoid linear searches and explore more efficient data structures such as Tries and Hashmaps to swiftly identify the correct Storage Server (SS) for a given request. This optimization enhances response times, especially in systems with a large number of files and folders.

LRU Caching: Implement LRU (Least Recently Used) caching for recent searches. By caching recently accessed information, the NM can expedite subsequent requests for the same data, further improving response times and system efficiency.

3.4 REDUNDANCY/REPLICATION [100 MARKS]

Failure Detection: The NM should be equipped to detect Storage Server (SS) failures. This capability ensures that the NFS can respond promptly to any disruptions in SS availability.

Data Redundancy and Replication: Implement a redundancy and replication strategy for data stored within the NFS. This strategy involves duplicating every file and folder in an SS in two other SS (once the number of SS exceeds two). In the event of an SS failure, the NM should be able to retrieve the requested data from one of the replicated stores. However, at this stage, only read operations should be allowed.

SS Recovery: When an SS comes back online (reconnects to the NM), the duplicated stores should be matched back to the original SS. Additionally, no new entries should be added to the list of accessible files and folders in the NM during this process.

Asynchronous Duplication: Every write command should be duplicated asynchronously across all replicated stores. The NM does not wait for acknowledgment but ensures that data is redundantly stored for fault tolerance.

3.5 BOOKKEEPING [20 MARKS]

Logging and Message Display: Implement a logging mechanism where the NM records every request or acknowledgment received from clients or Storage Servers. Additionally, the NM should display or print relevant messages indicating the status and outcome of each operation. This bookkeeping ensures traceability and aids in debugging and system monitoring.

IP Address and Port Recording: The log should include relevant information such as IP addresses and ports used in each communication, enhancing the ability to trace and diagnose issues.

SOME POINTERS:

Use TCP sockets

You may use any *POSIX C library* (opengroup link)

As always, decompose the problem and write *modular code*

Start _early _- we really can't stress this enough

Cite your resources if you take any ideas or code

Lastly, make necessary *assumptions*

ADDITIONAL HOW DO I START?

Define the calls. For example, define the format of the functionality of the read call for the client as `READ path` and then divide the work amongst yourselves with each team member implementing either the client, the naming server or the storage server code.

Identify things that can be decoupled. For example, the specification where an SS can join the NM at any given moment of execution doesn't depend on neither the client or the SS (Assuming you have figured out how SSs attach themselved at the beginning of the execution). One team member can implement this while the other thinks of ways to implement caching (You probably won't get merge conflicts as you'll be changing different parts of the naming server code)

Remember figuring things out is just as important as coding the implementation in this assignment. Not everyone needs to be programming at once. One could design the redundancy attribute and start coding it up later.

Here are a few **resources** you could read **before starting**:

A few slides on the topic: https://cs.gmu.edu/~setia/cs571-F02/slides/lec9.pdf

The Apache Hadoop DFS: https://www.databricks.com/glossary/hadoop-distributed-file-system-hdfs

CMU slides on Distrbuted File Systems: https://web2.qatar.cmu.edu/~msakr/15440-f11/lectures/Lecture19_15440_MHH_14Nov_2011.ppt

Rutgers university's resources: https://people.cs.rutgers.edu/~pxk/rutgers/lectures/l-

dfs.html

Resources you could use for the implementation:

Handling multiple clients: https://www.geeksforgeeks.org/handling-multiple-clients-on-server-with-multithreading-using-socket-programming-in-c-cpp/

Handling multiple clients without multithreading - https://www.geeksforgeeks.org/socket-programming-in-cc-handling-multiple-clients-on-server-without-multi-threading/

(This section will be updated on specific requests in the doubts document)