Project README

Overview

This project implements a distributed file system with a client-server architecture. It includes a Naming Server, Storage Servers, and a Client. The Naming Server manages metadata and coordinates file operations, while Storage Servers handle actual file storage and retrieval. The Client interacts with the Naming Server to perform various file operations.

Assumptions

1. Maximum size of file path is 256 characters.
2. Any accessible path (file or folder) is available only on one server to remove ambiguity.
3. File names shall not contain commas, exclamations, and other special characters (excluding `/`).
4. No file name or path shall contain the substring `--`.
5. File names should have some extension (e.g., `.txt`).
6. No two files across all Storage Servers shall have the same absolute path.
7. Input to the Storage Server: series of files followed by "STOP".

Dependencies

- `trash-cli`: This tool must be installed to handle file deletions safely. You can install it using the following command:

```
pip install trash-cli
```

Constants

PROFI

`nfs_project/common/utils.h`

- `MAX_FILE_PATH_SIZE`: 256
- `MAX_CONNECTIONS`: 10
- `MAX_CLIENTS`: 10
- `BUFFER_SIZE`: 4096
- `MAX_FILES`: 64
- `FAIL`: 1
- `SUCCESS`: 0

`nfs_project/naming_server/naming_utils.c`

- `MAX_CHILDREN`: 256
- `CACHE_SIZE`: 8

Usage

Client Commands

Read the content of a file.

```
read <file_path>
```

Write data to a file with synchronous confirmation.

```
write <file_path> --sync
```

Get information about a file.

```
info <file_path>
```

Stream audio from a file.

```
stream <file_path>
```

Create a new file.

```
create <file_path> <file_name>
```

Copy a file from source to destination.

PROFI

```
copy <source_path> <dest_path>
```

List files in a directory.

```
list <dir_path>
```

Delete a file.

```
delete <file_path>
```

Executable Files

- `client`: The client executable to interact with the Naming Server and perform file operations.
- `storage_server`: The storage server executable to handle file storage and retrieval.
- `naming_server`: The naming server executable to manage metadata and coordinate file operations.

Functions

`nfs_project/clients/client.c`

- `connect_to_server(const char *ip, int port, int *sock)`: Connect to the Naming Server.
- `read_file(int sock, const char *file_path)`: Read a file from the Storage Server.
- `copy_file(int sock, const char *source_path, const char *dest_path)`: Copy a file.
- `write_file(int sock, const char *file_path)`: Write data to a file.
- `get_file_info(int sock, const char *file_path)`: Get information about a file.
- `stream_audio_file(int sock, const char *file_path)`: Stream audio from a file.
- `create_file(int sock, const char *file_path, const char *file_name)`: Create a new file.
- `delete_file(int sock, const char *file_path)`: Delete a file.
- `list_files(int sock, const char *dir_path)`: List files in a directory.
- `print_usage(char *program_name)`: Print usage instructions.
- `main(int argc, char *argv[])`: Main function to handle client commands.

`nfs_project/storage_server/storage_server.c`

- `get_local_ip_address()`: Get the local IP address.
- `scan_directory(const char *homedir)`: Scan a directory and populate accessible paths.
- `start_storage_server(void* params)`: Start the storage server and register with the Naming Server.
- `handle_nm_request(void* args)`: Handle requests from the Naming Server.
- `process_nm_request(int nm_port)`: Process requests from the Naming Server.
- `handle_create(int nm_sock, char *path1, char *path2)`: Handle "CREATE" command to create a file or directory.
- `handle_delete(int nm_sock, const char *path)`: Handle "DELETE" command to delete a file or directory.
- `handle_copy(int nm_sock, const char *source_path, const char *dest_path)`: Handle "COPY" command to copy a file or directory.
- `handle_client_request(void* socket_ptr)`: Handle client requests.
- `client_connection(int client_port)`: Handle client connections.
- `handle_sigint(int sig)`: Handle SIGINT signal.
- `process_ss_request(int ss_port)`: Process requests from other storage servers.
- `handle_ss_request(void* args)`: Handle requests from other storage servers.
- `main(int argc, char const *argv[])`: Main function to start the storage server.

`nfs_project/naming_server/naming_server.c`

- `handleCtrl(int signum)`: Handle Ctrl+C signal.

- `start_naming_server(int port)`: Initialize the Naming Server.
- `handle_client(void *socket_desc)`: Handle each client connection in a new thread.
- `main(int argc, char *argv[])`: Main function to start the Naming Server.

nfs_project/storage_server/file_manager.c

- `read_file(const char *file_path, int client_socket)`: Read a file.
- `write_file(const char *file_path, int client_socket, char* global_nm_ip, int global_nm_port, char* global_ss_ip, int global_ss_port)`: Write data to a file.
- `copy_to_file(const char *file_path, int client_socket)`: Copy data to a file.
- `paste_to_folder(const char *file_path, int client_socket)`: Paste data to a folder.
- `get_file_info(const char *file_path, int client_socket)`: Get information about a file.
- `stream_audio_file(const char *file_path, int client_socket)`: Stream audio from a file.
- `create_file(const char *file_path, const char *file_name, int client_socket)`: Create a new file.
- `copy_file(const char *source_path, const char *dest_path, int client_socket)`: Copy a file.
- `delete_file_or_directory(const char *path, int client_socket)`: Delete a file or directory.

nfs_project/naming_server/naming_utils.c

- `find_in_cache(const char *path)`: Find an entry in the cache.
- `add_to_cache(const char *path, const FILEINFO *file_info)`: Add an entry to the cache.
- `remove_from_cache(const char *path)`: Remove an entry from the cache.
- `create_trie_node()`: Create a new trie node.
- `create_trie()`: Create a new trie.
- `update_file_info(Trie *trie, const StorageServerInfo *ss_info, const char *token)`: Update file information in the trie.
- `update_file_info_fromdir(Trie *trie, const FILEINFO *ss_info, const char *token)`: Update file information from a directory in the trie.
- `find_file_info(Trie *trie, const char *file_path)`: Find file information in the trie.
- `delete_file_info(Trie *trie, const char *file_path)`: Delete file information from the trie.
- `find_directory_info(Trie *trie, const char *dir_path)`: Find directory information in the trie.
- `traverse_trie(TrieNode *node, char *path, int depth, int client_socket)`: Traverse the trie and list files.
- `list_files(Trie* trie, const char *dir_path, int client_socket)`: List files in a directory.
- `copy_trie_node(const TrieNode *node)`: Deep copy a trie node.
- `find_trie_node(Trie *trie, const char *path)`: Find the trie node corresponding to a given path.
- `insert_trie_node(Trie *trie, const char *path, TrieNode *new_node)`: Insert a trie node at a specified path.

- `copy_directory_trie(Trie *trie, const char *arg1, const char *arg2)`: Deep copy the trie of one directory to another.

Compilation

To compile the project, use the provided `compile.sh` script:

```
chmod +x compile.sh
./compile.sh
### Running the Servers

To start the Naming Server:

```sh
./naming_server <client_port> <storage_server_port>
```

To start a Storage Server:

```
./storage_server <NM_IP> <NM_PORT_to_connect> <CLIENT_PORT>
<NM_PORT_TO_RECEIVE>  <ss_port_to_connect_other_ss> <HOMEDIR>
```

To start the Client:

```
./client <naming_server_ip> <naming_server_port>
```
```