Steps to Execute:

Rename the macro MSGQ_PATH in all files to an existing location in order to create the message queue.

```
#define MSGQ_PATH "/mnt/e/Academics/3-1/IS F462/ass_1_group/q2"
```

First compile the following files:

1. m_server.c
2. d_server.c
3. client.c

Copy the executable file of D_server.c to different directories (these will be the local directories of D).

Open terminals for m_server, client, d_server<n> (Where n is the number of D servers in the system) and execute the files to make all servers in running state.

Now, enter the commands in the terminal which is running the executable of client.c

Data Structures used in M:

1. CHUNK_SERVER: which stores the PID's of 3 D servers storing it.

```
struct chunk_server {
  int chunk_id;
  int server1;
  int server2;
  int server3;
};
```

2. FILE_CHUNKS: which store the filename, pointer to array of chunks and the number of chunks present for the file.

```
struct file_chunks {
  char filename[200];
  struct chunk_server * chunks;
  int num_chunks;
};
```

3. FILES: an array for files present in metadata server.

```
struct file_chunks * files;
```

Implementation:

The supported commands are:

1. ADD_FILE /path/to/file/filename

   It is assumed that this file will be already present on the disk. When we try to create a file, we are actually loading its metadata and creating chunks for the file in the D servers. The following steps happen when we ADD a file using this command:

   1. Client sends a message to the M_Server to create a file.
   2. M_Server sends a response if the request was successful.
   3. Client divides the file into chunks inside a loop. For each chunk:
      a. Client sends a request to the M_server “ADD_CHUNK”. M_server queries the D_servers for availability.
      b. The list of first 3 D_servers to respond to M_Server’s query is sent to Client to ADD_CHUNK along with the absolute chunk_id to be used.
      c. Client sends the contents of the file to the corresponding D_server.
      d. The D_servers store the contents in a file names as the absolute chunk_id that the M_server provided.

2. CP /path/to/src/filename /path/to/dest/filename

   The steps executed are:

   - Client sends a request to M_server to move the file.
   - M_server first checks if the filename was added using ADD_FILE earlier. If not, it returns an error.
   - Then, M_server loops through all the chunks of the file. For each chunk, it sends a COPY_CHUNK message to the 3 D_Servers which stores it, along with the new absolute chunk_id for the new chunk.
   - D_Servers copy the contents of the chunks character by character.

3. MV /path/to/src/filename /path/to/dest/filename

   The steps executed are:

- Client sends a request to M_server to move the file.
- M_server first checks if the filename was added using ADD_FILE earlier. If not, it returns an error.
- M_server then renames the original filename with the new filename in it's metadata structure.

4. RM /path/to/file/filename

   The steps executed are:
   - Client sends a request to M_server to delete the file.
   - M_server first checks if the filename was added using ADD_FILE earlier. If not, it returns an error.
   - Then, M_server loops through all the chunks of the file. For each chunk, it sends a DELETE_CHUNK message to the 3 D_Servers which stores it.
   - D_servers use the remove() function to delete the chunk from local directory.
   - In the end, M_server frees the memory used in storing the metadata and renames the file as "DELETED". It is assumed that the user will not try to load any file named "DELETED" into the server.

5. EXEC ON <pid of D server> <command>
   It is assumed that the client knows the correct process id of the D_Server to which the command needs to be sent. The Client sends the command to the D_server in the form of a message and the D_server executes and displays the result on it's terminal.

Comments were added in the code from time to time but it was not possible to describe each step due to the limited time frame.