Assignment - I (Problem 2)
(Network Programming)

Utkarsh Dwivedi (2018A7PS0131P)
Bikash Jena (2018A7PS0181P)

In this file storage system, there is a metadata server (M), data servers(D0, D1, ..) and a client(C) as mentioned in the problem statement.

General information about the implementation:

- There are three message queues : **client_mq**, **m_server_mq** and **d_server_mq**, which respectively are used by the corresponding processes for reading messages while it can be written on by any of the other two remaining types of servers.
- There are three semaphores : **s_client**, **s_m_server** and **s_d_server**. The semaphores ensure synchronicity in any action for interprocess back and forth communication amongst servers. During transmission, the semaphore of the sending server is blocked and that of the receiving process is unlocked.
- There is a shared memory **d_servers_pids** that is shared by the starting process start.c, client.c and m_server.c. It stores the d_server_ids against corresponding pids.
- To ensure only d_server accesses the d_server_mq at a time, each d_server is by default indefinitely paused, and have signal handlers for SIGQUIT, SIGUSR1 and SIGUSR2. It is used by any other server to communicate with that specific d_server, as pid of that d_server is determined by d_server_pids and it is used to send signal to that d_server to wake it so that it becomes ready to receive message in d_server_mq. Hence, only the required d_server receives the message.
- Each d_server creates a new directory where it stores chunks as files with chunk_id as filenames.

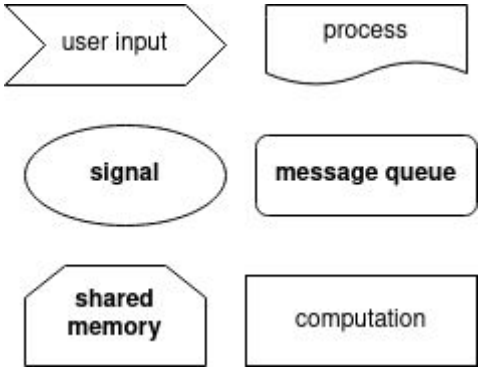
Specific information about the implementation:

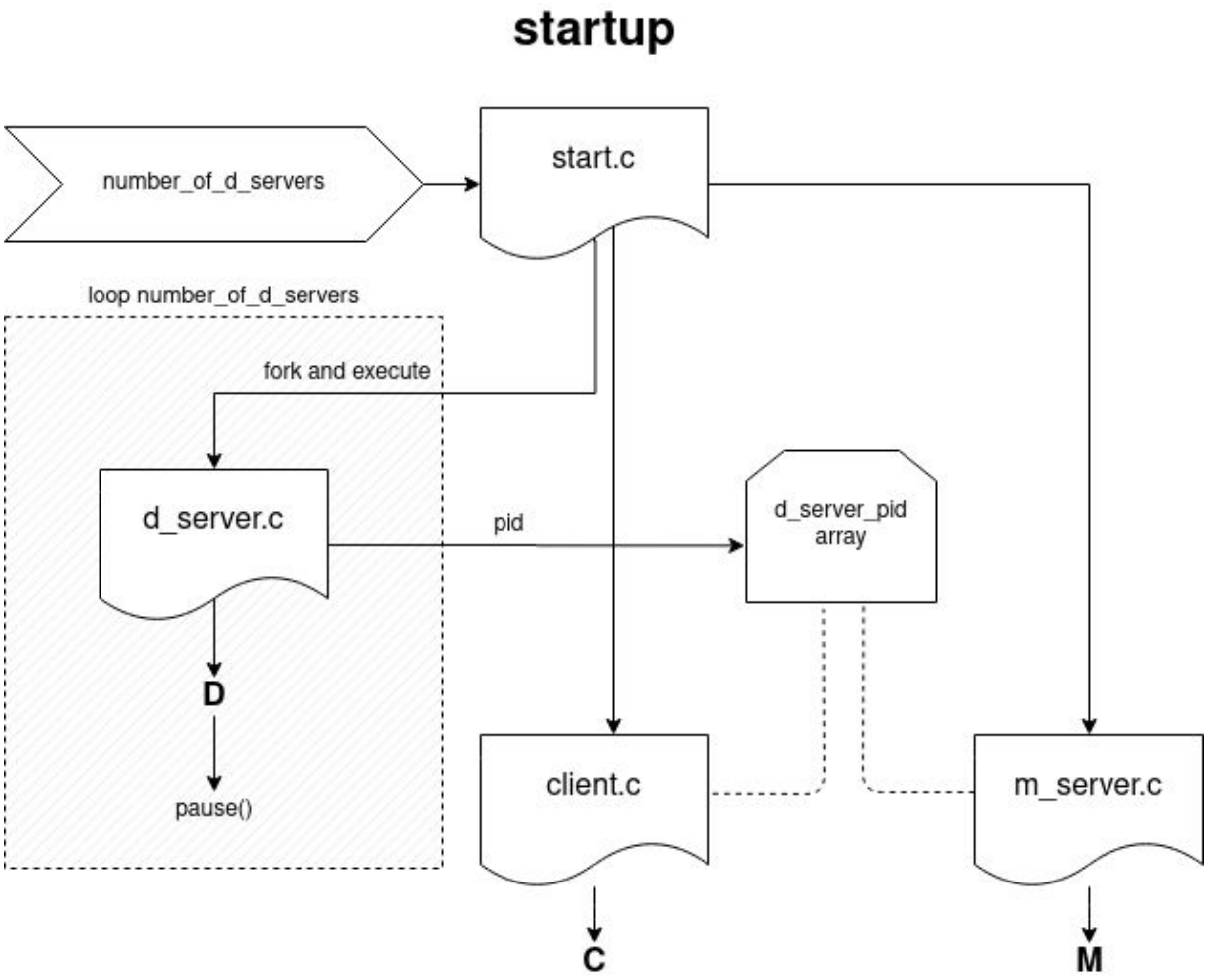
MAX ALLOWED CHUNK SIZE < 30 bytes
MAX NUMBER OF D SERVERS < 50
MAX LENGTH OF DIRECTORY/FILE NAME < 20 bytes
MAX NUMBER OF IMMEDIATE FILES IN A DIRECTORY < 10

Input from the client: The client can submit requests for adding, copying, moving, and removing a file from the file system. The input formats of these requests are given in the respective descriptions in this assignment.

File structure in m_server.c: There are two structs “dir” and “file” containing the required attributes like the name for directories and files.
Each directory (dir) contains an array of pointers to other directories present in it along with an array containing the files inside it. All directories emerge from root(/) directory in a tree-like structure.
Each file contains a pointer to its parent directory, an array of chunks with specific chunk size.

Startup: The process start.c initialises ipc mechanisms and has the responsibility of closing them and also deleting d_server directories. It also ensures the other programs are compiled.





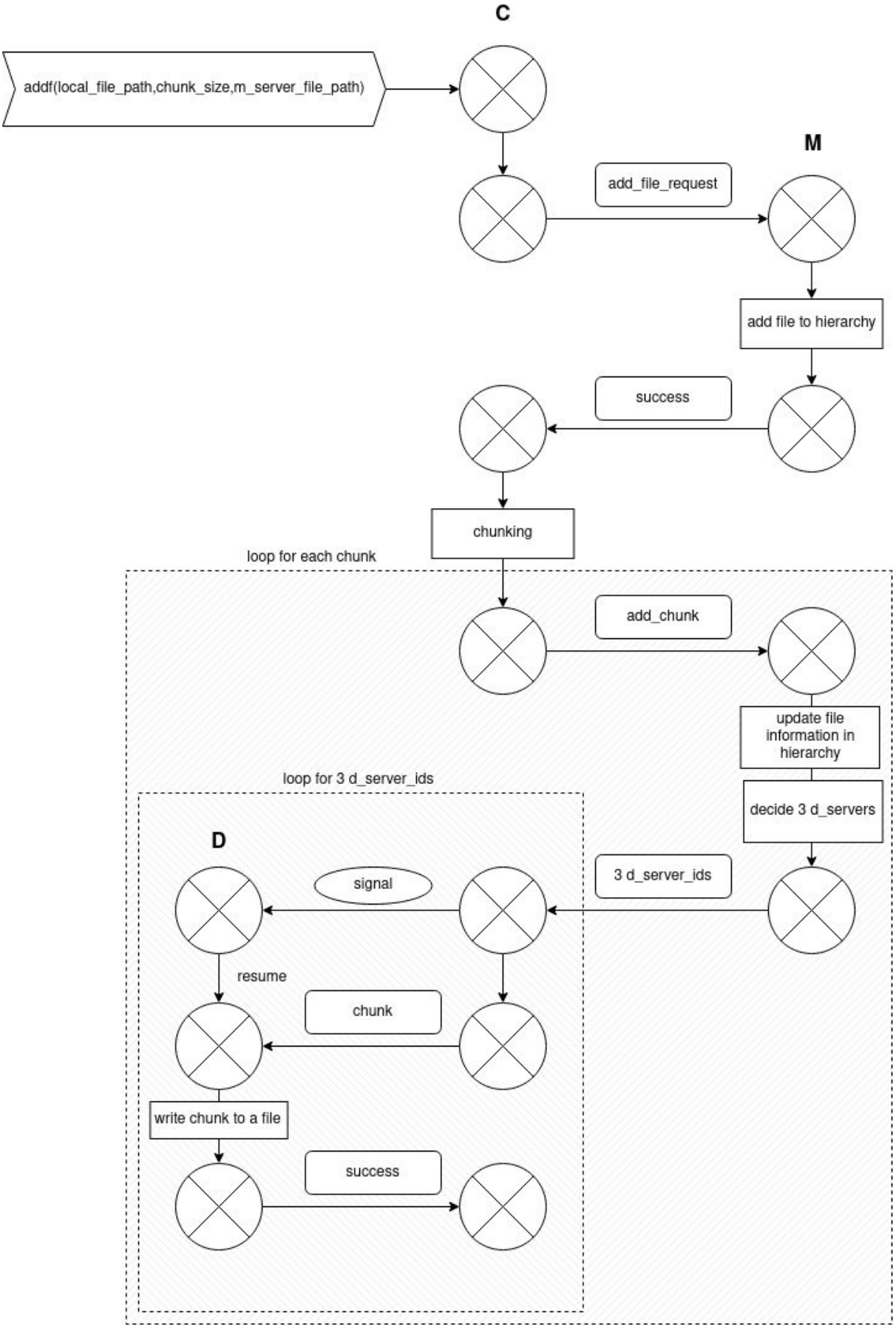
For running the program (m_server and d_servers)
$ gcc start.c -ipthread -lrt
$ ./a.out
For starting any client
$ ./client *num_of_d_servers*

NOTE: Use SIGINT (Ctrl+C) to terminate the program, start.c handles it and ensures that all ipc mechanisms are closed and d_server directories are deleted.

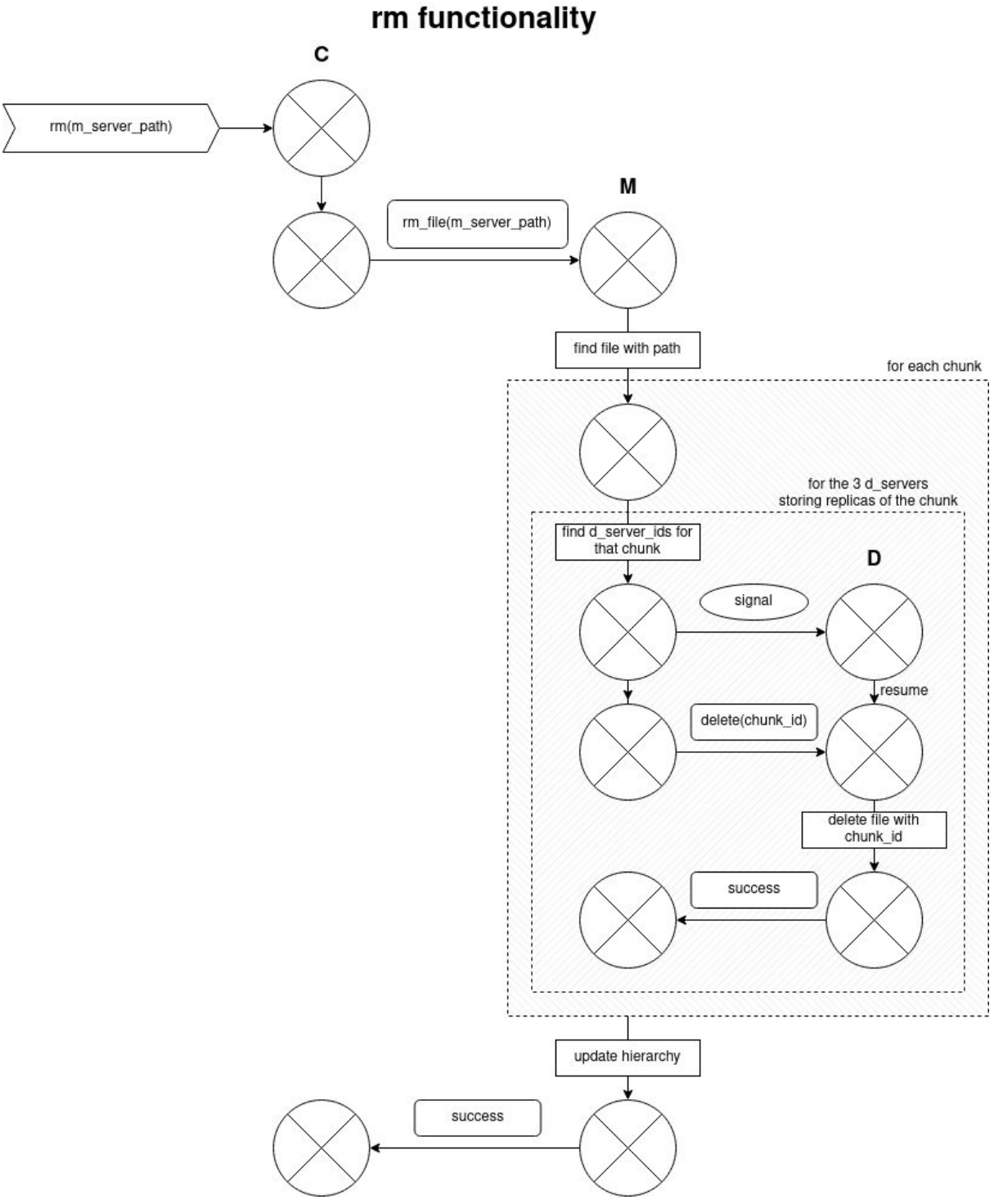
Client requests for file:
The requests from the client server to the metadata server are accomplished through message queues for the following commands. Each server process has its own semaphore synchronized to each other so that only the intended server gets the message.

- **addf(source file path,chunk size,destination path)**: With the help of this, a file can be added to the implemented file storage system. The client server sends the chunks to the metadata server and all the affected directories' attributes are changed on the metadata server. On successful execution, the metadata server notifies the client. Then using a message queue, each chunk is delivered to the required data server by the client.

addf functionality



- **mv(source file path, destination file path)**: In this command, the file pointers and the parent directory pointer of the file are changed on the metadata server. The attributes are accordingly updated like the number of files, etc.
- **rm(file_path_in_m_server):**



- **cp(*source_filepath*, *dest_file_path*):** This works in very much the same way as rm functionality. The only difference is that instead of

providing a single path in delete(chunk_id) in rm, there are two arguments new_chunk_id and old_chunk_id, for the d_server to copy the old chunk with the newly supplied new_chunk_id. Also, in each iteration of chunk, the chunk information is copied to a new file in m_server with the only values changed is the chunk_id.

- **Direct commands from client (format: d_server_id command):** In this simply a signal is used to the d_server_id server waking it up for receiving a message while other d_servers are blocked. The message contains the command to be executed which the d_server does. The output is stored in a temporary file which the client prints and then deletes.