Design Document for Q2

Name: Samina Shiraj Mulani Name: Jithin Kallukalam Sojan
ID: 2018A7PS0314P ID: 2017A7PS0163P

Deliverables: client.c, m_server.c, d_server.c

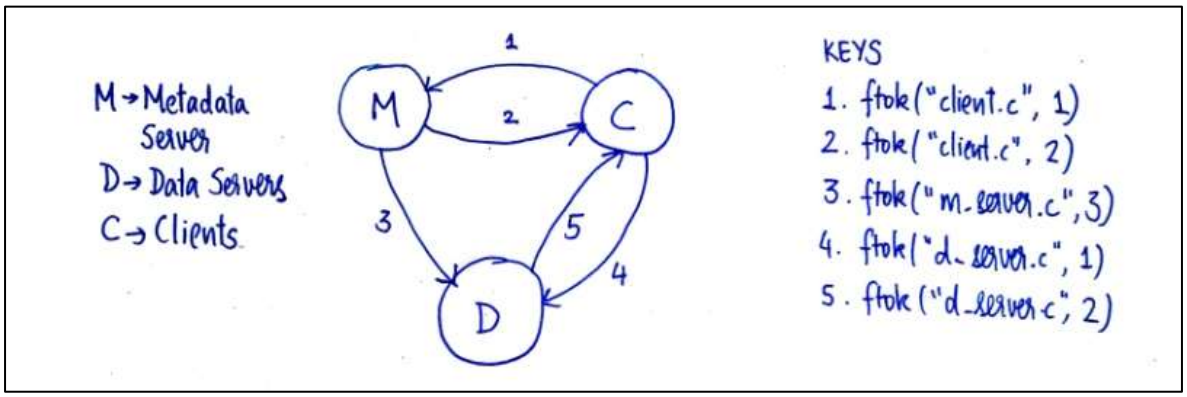
Let the metadata server be called M, the data servers (D1, D2, .., DN) and the client, C.

Assumptions and understandings

- None of the processes are terminated by sending a signal through bash.
- File hierarchy stored by M is an in-memory structure.
- Any file path is case-sensitive and consists of only alphanumeric characters and the symbols '.' And '/'. A filename cannot end in '/', but has to begin with '/' (indicating root).
- A file can be added in the file hierarchy and consequently, D, only if the file is an existing file on the system.
- Maximum chunk size is 2000 bytes. (This includes a last '\0' character).
- Maximum size of file path or command entered by client is 100 bytes.
- Maximum size of result of command returned to C by D is two times max chunk size (4000 bytes).
- Collision between keys of the message queues are unlikely.
- There can be multiple clients and they all interact with the same file system.
- There are N data servers, where N is entered as a command line argument when running m_server and d_server. It is assumed that these two arguments enteed separately are the same (to be taken care of by the user running these files).
- N data servers are created via forking in d_server.c.
- Data servers store the chunks as files within their own directories (./d1/,./d2/, etc) by the name of respective chunk ID. A copy command is understood as replication of the same chunk in terms of content but assigning it a new chunk ID and hence a different file name. Additionally, copy of a chunk is assumed to be created in the same D server of the original chunk.
- File names "client.c", "m_server.c" and "d_server.c" are available before hand and thus are defined as macros in the relevant programs.
- File permissions are not taken into account on creation.
- The move, copy and remove features take in input like a simple command from the user (ex: cp /a/b /a/d), which is parsed at M to get the paths of source and destination files.
- The order of running the programs is metadata server, followed by data server and then client. (So that message queue opening/creation is done correctly).

Design

M, C and D communicate via message queues. N is the number of data servers.
Following is a figure depicting the various message queues. Separate message queues for back and forth communication have been used for easier visualisation and simpler message data structures.



There is a single queue from C to M (1) and from M to C (2). M sends messages to D servers via queue 3. Messages from C to D servers is facilitated by queue 4 while those from D to C, by queue 5.
Description of the message queues (structure of messages, usage, pathname and proj id used to generate key) can be found in a later section. Only message queues 3 and 4 have IPC_NOWAIT specified at the data server end, as it should be able to handle requests from C and M. For the other queues, messages are either responses or requests initiated by client, which are handled one at a time.

Metadata server M uses a trie structure to store the file hierarchy. Trie has been used for efficient lookup time, albeit the penalty on storage. The end of a file path is marked with an integer called notEnd (0 indicates the end while 1 indicates that it is not the end of file path). The last node, apart from having end set to 0, also stores a dynamic array of a structure called chunkInfo, which holds the chunk ID and corresponding address of D servers the chunk is stored on. The size of this dynamic array is also stored. Chunk IDs are generated using rand() and an array keeps track of chunk IDs already used so that a newly generated chunk ID does not coincide with one created before. File existence is checked via traversal through trie. Deletion of file is performed by deletion of key in trie. File path validity involves checking for characters other than '.','/' and alphanumeric in the file path string. Additionally, the file path name cannot end in a '/' and always begins with a '/'. A check in insert() is also performed so that the file added is valid after adding to trie. (example1: if /dog is an existing file, /dog/a cannot be added) (example2: if /a/b/c is an existing file, /a/b cannot be added).

Client is offered options to add file, copy a file, move a file, remove a file, execute command on a chunk and exit the program. Copy, move and remove are entered as simple commands which are parsed at M. For example, if user wants to remove a file '/dog', they select the option to remove file from the menu and enter 'rm /dog', i.e., (rm <filepath>).

N data servers are created by forking. Each has a corresponding directory that is created if not already existing at the start of execution of program. These directories store file chunks by chunk IDs. Address of a D server is the type of the message it accepts in queues 3 and 4. The N D servers have types ranging from 1 to N (parent having type1 and last child having typeN).

Functionalities

1. Add file – User enters full file path of an existing file on his/her system. Next, the desired file path (to be stored in file hierarchy of M) is entered. Chunk size is specified by the user. The client then sends a message to M, which adds the file to hierarchy if file path is valid. Next, the file is opened in client and is divided into chunks. Request to add chunk is sent to M, which assigns the chunk a unique ID, 3 D servers where it will be stored, and adds both the aforementioned information in the file hierarchy. Status is returned which contains the chunk ID and 3 types of the message to be sent to queue 4.
   File is then divided into chunks in the client and sent to the addresses received from M, along with the chunk ID. If chunk is stored, each D sends a status message back to the client that requested the addition of chunks.

2. Copy – User enters input of the form cp <source> <destination>. Message is sent to M, which modifies the file hierarchy appropriately if source and destination are valid file paths, destination does not already exist, and if source file exists. Status about this modification is sent back to C. New chunk IDs for the replica destination are generated. These new IDs and source file chunk IDs are sent to relevant D servers. Multiple messages are sent if number of chunks of the file on a D server exceeds 100. Subsequently, a message with one of its variable ‘size1’ set to -1 is sent to relevant D servers to let them know the to be copied chunk IDs have been sent. Those D servers which are able to carry out the copy command successfully, print command executed to console. Errors, if any, are also printed. The status of the copy is sent back to specific client which waits for M responses from the N D servers. Here, M is the number of D servers that in total have all chunks of the source file the client specified. (As different chunks of the same file can be on two different D servers).

3. Move - User enters input of the form mv <source> <destination>. Message is sent to M, which modifies the file hierarchy appropriately if source and destination are valid file paths, destination does not already exist, and source file exists. Modification status is sent back to C.

4. Remove – User enters input of the form rm <filepath>. If file path is valid and exists, it is removed from the trie structure. Status is sent back to C. Next, chunk IDs of the file are sent to those data servers which have the relevant chunk(s), and these chunks are deleted. Status of deletion is printed to console.

5. Command to D – Client specifies a command as input. The filename from this command is extracted and both are sent to M. M returns the list of chunk IDs and the address of the data servers these chunks are present in. (The last message to C from M has a variable ‘size’ to -1 so that client knows that M has sent all the chunk related information). Next, C sends the command and chunk ID to one of the D servers that has this chunk. D executes the command on specified chunk locally and sends the result back to the client that requested it. C prints the result and then continues to send the commands to relevant D servers till all chunks of the file are exhausted.

6. Exit – Client can exit at any time.

Message Queues

Message Queue 1
Metadata server uses path “client.c” and proj ID 1 to create the key to the queue.
Structure of message

```
struct ctom{
    long mtype;
    int pid; //pid of client sending message
    char forc[100]; //file path or command
};
```

Type	Purpose	pid	forc
1	Request to add file	Process ID of client	Full file path
2	Request to add chunk	Process ID of client	Full file path
3	Request to copy file	Process ID of client	CP command (with src and dst)
4	Request to move file	Process ID of client	MV command (with src and dst)
5	Request to remove file	Process ID of client	RM command (with full file name)
6	Request chunk information	Process ID of client	Full file path

Message Queue 2
M uses path “client.c” and proj ID 2 to create the key to the queue.
Structure of message

```
 struct mtoc{
    long mtype;
    int statFor; //secondary status
    int d1,d2,d3;
    int CID;
    int status;
    struct chunkInfo CIDs[100];
    int size;
};
```

Mtype for all these messages is process ID of client requesting a service from M.

Purpose	statFor	d1	d2	d3	CID	status	CIDs	size
Status of request to add file	1	-	-	-	-	0 success 1 fname invalid 2 file exists	-	-
Response after add chunk request sent	2	Addr of D1	Addr of D2	Addr of D3	Chunk ID	-	-	-
Response to CP	3	-	-	-	-	0 success 1 src does not exist	-	If status==0, number of D

						2 src or dst not valid 3 cmd is wrong		servers file is on
Response to MV	4	-	-	-	-	0 success 1 src does not exist 2 src or dst not valid 3 cmd is wrong	-	-
Response to RM	5	-	-	-	-	0 success 1 src does not exist 2 src or dst not valid 3 cmd is wrong	-	-
Give chunk information	6	-	-	-	-	0 success 1 fname invalid 2 file does not exist	Chunk IDs + Addr of Ds	Size of CIDs if sending chunk information. -1 if all information has been sent

Message Queue 3
These are created by meta data server and uses “m_server.c” and proj ID 3 to generate keys.
Structure of message

```
1.  struct mtod
2.  {
3.      long mtype;
4.      int op;
5.      int pid;
6.      int chunks1[100];
7.      int size1;
8.      int chunks2[100];
9.      int size2;
10. };
```

Type of messages are addresses of relevant D servers. Pid field is the process ID of the client sending the request to M.

op	Purpose	chunks1	size1	chunks2	size2
1	To copy chunks	Old chunk IDs of chunks to be copied	Size of chunks1 array, is -1 if all chunk IDs sent by M	New chunk IDs to be created	Size of chunks2 array
2	To remove chunks	Chunk IDs of chunks to be removed	Size of chunks1 array	-	-

Message Queue 4
Created by D servers using pathname “d_server.c” and proj ID 1 to generate keys.
Structure of message

```
struct ctod{
    long mtype;
    int CID;
    int pid;
    char cmd[100];
    char chunk[MAX_CHUNKSIZE];
    int size;
};
```

Pid is process ID of C sending request to D. Type is address of D server.

Purpose	CID	cmd	chunk	size
Chunk of data of file to be stored on D	ID of chunk to be stored on D	-	Data in file chunk	Size of this chunk in bytes
Command to be executed on a specific chunk of D	ID of the chunk on which cmd is to be executed	The command to be executed	-	-1

Message Queue 5
Created by D servers using pathname “d_server.c” and proj ID 2 to generate keys.
Structure of message

```
struct dtoc{
    long mtype;
    int status;
    int CID;
    char result[2*MAX_CHUNKSIZE];
    int resSize;
};
```

Mtype is the process ID of the client that requested a service from D.

Purpose	status	CID	result	resSize
Status after adding chunk	0 success 1 error	Chunk ID of chunk added	-	-
Result of cmd to D	0 success 1 error	Chunk ID of chunk on which command was executed	Holds the result	Size of result