Design Document for Q2

Name: Samina Shiraj Mulani
ID: 2018A7PS0314P

Name: Jithin Kallukalam Sojan
ID: 2017A7PS0163P

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

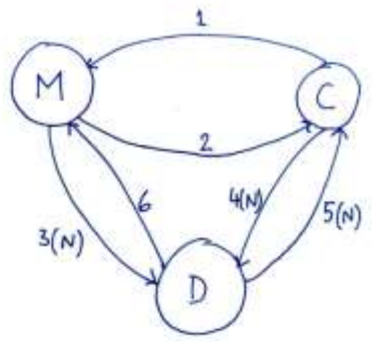
Let the metadata server be called M, the data servers (D1, D2, .., DN) and the client, C.

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 '/'.
- 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.
- Maximum size of file path or command entered by client is 100 bytes.
- Chunk ID is known to user (client) as it is displayed to him/her when adding a file or copying a file. Thus, the commands sent by client to D servers specifies a chunk ID on which command is to be executed. This command is executed on all servers that C is aware of at a point in time.
- There are N data servers (N declared as a configurable macro in m_server.c and d_server.c). The client has address of only 3 data servers at maximum at any point in time (received when M sends it to C on requesting an addition of chunk).
- 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.
- 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 client, followed by data server and then meta data server. (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.



There is a single queue from C to M and from M to C. There are N queues from M to D. Queue 6 is from the last child in d_server.c to M. This is for the purpose of sending the process IDs of all N D server processes to M in the start. This enables M to send these pids to C when C requests to add a chunk. Message queues 4 and 5 connect C and D. Here, N is at maximum 3 (only for queues 4 and 5).
Description of the message queues (structure of messages, usage, pathname and proj id used to generate key) can be found in the next section. Only message queues 3 and 4 have IPC_NOWAIT specified.

Metadata server M uses a trie structure to store the file hierarchy. The end of a file path is marked with an integer end (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 chunk IDs and the size of this dynamic array. 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. A check in insert() is also performed so that 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 sent to client is understood as being the pathname and proj id (which generates a key) of the message queue connecting client and data server (there are 2 such queues for each data server – 4 and 5 in the diagram). The pathnames are fixed to the two paths ("m_server.c" and "d_server.c", also called p and q), which is known my M and D (and is sent to client). <p,pid> is used to generate key for queue 4 and <q2,pid> is used to generate key for queue 5.

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, adds it in the file hierarchy, and returns this ID along with addresses of 3 D servers. (Address is combination of path and process ID of data server which is used to generate a key for the message queue).
   File is then divided into chunks in the client and sent to the addresses received from M (message queues are opened), along with the chunk ID. If chunk is stored, each D sends a status message back to C.

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 all N D servers. Those which are able to carry out the copy command successfully, print command executed to console. Errors, if any, are also printed.

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 all data servers, and these chunks are deleted. Status of deletion is printed to console.

5. Command to D – Client specifies a command as input, with chunk ID as filename. This command is sent to the D servers C knows. If successful, status is set accordingly and result is sent to C.

6. Exit – Client can exit at any time.

Message Queues

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

```c
struct ctom{
    long mtype;
    char forc[100]; //file path or command
};
```

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

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

```c
struct mtoc{
    long mtype;
    char p[100],q[100];
    int d1,d2,d3;
    int CID;
    int status;
    int newCIDs[100];
    int size;
};
```

Type	Purpose	p	q	d1	d2	d3	CID	status	newCIDs	size
1	Status of request to add file	-	-	-	-	-	-	0 success 1 fname invalid 2 file exists	-	-
2	Response after add chunk request sent	Path1	Path2	Pid of D1	Pid of D2	Pid of D3	Chunk ID	-	-	-
3	Response to CP	-	-	-	-	-	-	0 success 1 src does not exist 2 src or dst not valid 3 cmd is wrong	New chunk IDs of copied chunks	Size of new CIDs array
4	Response to MV	-	-	-	-	-	-	0 success 1 src does not exist 2 src or dst not valid 3 cmd is wrong	-	-
5	Response to cmd to D	-	-	-	-	-	-	0 success 1 error	-	-

Message Queue 3(N)
These are created by data servers and uses “client.c” and pid+10 to generate keys.
Structure of message

```c
struct mtod
{
    long mtype;
    int chunks1[100];
    int size1;
    int chunks2[100];
    int size2;
};
```

Type	Purpose	chunks1	size1	chunks2	size2
1	To copy chunks	Old chunk IDs of chunks to be copied	Size of chunks1 array	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(N)
Created by D servers using pathname “d_server.c” and own process ID.
Structure of message

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

Type	Purpose	CID	cmd	chunk	size
1	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
2	Command to be executed on a specific chunk of D	-	The command to be executed	-	-

Message Queue 5(N)
Created by D servers using pathname “m_server.c” and own process ID.
Structure of message

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

Type	Purpose	status	CID	result
1	Status after adding chunk	0 success	Chunk ID of chunk added	-
2	Result of cmd to D	0 success 1 error	-	Holds result

Message Queue 6
Created by last child in D using “client.c” and 6 to generate key. Only used to send a single message in the start of execution to let M know the process IDs of D servers.
Structure of message

```c
struct dtom{
    long mtype;
    int d_pids[N];
};
```

Type	Purpose	d_pids
1	To let M know of pids of D servers	List of N process IDs