BigFS

A File System to store large files on multiple systems. Data blocks are downloaded and uploaded to different FileDataServers in parallel. Commands such as `cat`, `cp`, `mv`, `ls` and `rm` are supported.

Getting Started

Compilation

Use `make` to compile the files. It will generate 3 files,

- `fns` (Executable for FileNameServer)
- `fds` (Executable for FileDataServer)
- `bigfs` (Executable for Client)

Running the FileNameServer

```
$ ./fns <port number>
```

For example, to run the server at port `5555`,

```
$ ./fns 5555
```

Running the FileDataServer

```
$ ./fds <port number>
```

For example, to run the server at port `8081`,

```
$ ./fds 8081
```

Running the Client

```
$ ./bigfs <path to config file>
```

For example, if the config file is `config.txt` in the same directory,

```
$ ./bigfs config.txt
```

Format for the config file

The config file should have the following format,

```
<FNS IP> <FNS PORT>
<k = Number of FDS>
<FDS0 IP> <FDS0 PORT>
<FDS1 IP> <FDS1 PORT>
..
..
..
<FDS(k-1) IP> <FDS(k-1) PORT>
```

For example,

```
127.0.0.1 5555
3
127.0.0.1 8081
127.0.0.1 8082
127.0.0.1 8084
```

Commands

`bigfs` is used to represent the remote file system. So, Copying into `bigfs` is equivalent to uploading the the BigFS and copying from `bigfs` is equivalent to downloading from the BigFS.

cp - Usage Examples

To copy the file `movie.mp4` to a remote directory `movies` with name `movie1.mp4` ,

```
$ cp movie.mp4 bigfs/movies/movie1.mp4
```

To copy the file `movie1.mp4` from remote directory `movies` to local directory `localmovies` with name `movie1.mp4`

```
$ cp bigfs/movies/movie1.mp4 localmovies/movie1.mp4
```

To copy all contents of the local directory `localmovies` recursively to root directory of BigFS,

```
$ cp -r localmovies/ bigfs/
```

To copy the remote directory `movies` and all its contents recursively to local directory `downloads`,

```
$ cp -r bigfs/movies downloads/
```

Note:

- *`cp` does not support copying files from one remote folder to another remote folder.*
- *copying files within local filesystem is supported.*

mv - Usage Examples

To move a file `lecture10.pdf` from the remote directory `study` to remote directory `acads` with name `lecture10.pdf`

```
$ mv bigfs/study/lecture10.pdf bigfs/acads/lecture10.pdf
```

Note:

- *Moving directories is supported.*
- *Moving files from local file system to remote file system and vice versa is not supported.*
- *Moving files from one local directory to another local directory is supported.*

rm - Usage Examples

To remove a remote file `song1.mp3` from the remote directory `music`,

```
$ rm bigfs/music/song1.mp3
```

To remove the complete `music` remote directory recursively,

```
$ rm -r bigfs/music
```

Note: *`rm` on local files is supported.*

ls - Usage Examples

To view the files and folders in root remote directory,

```
$ ls
```

To view the files and folders in a remote directory named `music`,

```
$ ls bigfs/music
```

Note: *`ls` on local directories is supported.*

cat - Usage Examples

To display the file `client.c` in remote directory `code`,

```
$ cat bigfs/code/client.c
```

Note: *`cat` on local files is supported.*

exit

To exit from the client application,

```
$ exit
```

Working

Temporary Directory

All intermediate files at client side are stored in a temporary directory named `tmp` which is created automatically in the current working directory.

Storage at a FileDataServer

All files are stored in a directory named `bigfs` with a unique name, generated by the FileDataServer. These files are the parts of different big files uploaded by a client application.

Storage at a FileNameServer

All files are stored in a directory named `bigfs` with path of each file same as the remote path given by the user to the client application. These files do not hold the actual content, instead, these contain the information about the UIDs of parts of the actual file stored across multiple FileDataServers.

Format of a file in FNS

A file contains `#` separated UIDs of different parts of that file. For example, for a remote file `bigfs/movies/m1.mp4`, we will have a information file at the location `bigfs/movies` with the name `m1.mp4` in the FileNameServer having data as

```
uo9btk7tksdmoe0#df3a167rlvg5k4c#oj8fxus2ba9udwd#f3aonoz1rr1xs1q
```

(4 UIDs separated by `#`, which gives information about 4 parts of the file in various FileDataServers.)

Spliting a File

Declaration of the splitFile function,

```
char *splitFile(char *filePath, int *numParts);
```

- Before spliting a file, an alpha-numeric random string of `10 characters` is generated as a basename for the temporary file.
- 1MB blocks are created for the file and are stored in the temporary directory.
- Basename of the file is returned by the split function and number of parts is stored in the `*numParts`.

For Example, for a file of 3.4MB, suppose the 10 character random string is `pi2ow8xbshsxitc`, then it will have its parts named as `pi2ow8xbshsxitc_0`, `pi2ow8xbshsxitc_1`, `pi2ow8xbshsxitc_2` and `pi2ow8xbshsxitc_3`.

Joining parts of a File

Declaration of the joinFile function,

```
char *joinFile(char *baseFilePath, int numParts);
```

For a file with basename as `pi2ow8xbshsxitc` and numParts as `3`, it joins the parts with name `pi2ow8xbshsxitc_0`, `pi2ow8xbshsxitc_1`, `pi2ow8xbshsxitc_2` and saves the file with name as `pi2ow8xbshsxitc` in the temporary directory.

Uploading a part to a FileDataServer

Declaration of the uploadPart function,

```
char *uploadPart(char *fpath,int FDS_idx);
```

Uploads the file at path `fpath` to the FDS at index `FDS_idx` and returns a unique id generated by the FDS.

Steps for the Client

- Compute the size of the file to be uploaded. (Suppose it is, `1231 bytes`)
- Write to the socket, `PUT <size>`. (Here it will be, `PUT 1231`)
- Read the unique id sent by the server.
- Write the contents of the file to the TCP socket.

Downloading a part from a FileDataServer

Declaration of the downloadPart function,

```
int downloadPart(char *uid, char *tgtPath,int FDS_idx);
```

Downloads the file with unique id `uid` on FDS at index `FDS_idx` to the location `tgtPath`.

Steps for the Client

- Write to the socket, `GET <uid>`.
- Read the size of the file sent by the server. (`-1` is received if the file is not found)
- Write to the socket `READY`.
- Read the file data sent by the server and save it to `tgtPath`.

Removing a part from a FileDataServer

Declaration of the removePart function,

```
int removePart(char *uid, int FDS_idx);
```

Deletes the file with unique id `uid` on FDS at index `FDS_idx`.

Steps for the Client

- Write to the socket, `DEL <uid>`.
- Read the success message `DONE` sent by the server.

Adding an entry to the FileNameServer

Declaration of the newFnsEntry function,

```
int newFnsEntry(char *remote_path, char **uidarr);
```

Adds the uid information contained in `uidarr` for a remote file at `remote_path` to the FileNameServer.

Steps for the Client

- Generate the `#` separated UID string and store into a buffer.
- Write to the socket, `PUT <size>` (where `<size>` is the size of the buffer).
- Read the message `READY` sent by the server.
- Write the contents of the buffer to the server.

Retrieving an entry from the FileNameServer

Declaration of the getFnsData function,

```
char **getFnsData(char *remote_path);
```

Gets the `#` separated UID list from the FNS for the remote file at `remote_path` and returns an array of UIDs.

Steps for the Client

- Write to the socket, `GET <remote_path>`.
- Read the size of the file sent by the server.
- Write to the socket `READY`.
- Read the file data sent by the server, split it, convert into array and return.

Deleting an entry from the FileNameServer

Declaration of the removeFnsEntry function,

```
int removeFnsEntry(char *remote_path);
```

Removes the entry for the remote file at `remote_path` from the FNS.

Steps for the Client

- Write to the socket, `DEL <remote_path>`
- Read the success message `DONE` sent by the server.

Moving an entry in the FileNameServer

Declaration of the moveFnsEntry function,

```
int moveFnsEntry(char *srcPath, char *destPath);
```

Moves the file information at remote path `srcPath` to the remote path `destPath`.

Steps for the Client

- Write to the socket, `MOV <srcPath> <destPath>`.
- Read the success message `DONE` sent by the server.

Getting ls output from the FileNameServer

Declaration of the getLs function,

```
char **getLs(char *remote_path);
```

Gets the `ls` output of the remote directory at `remote_path`, converts it into an array and return.

Steps for the Client

- Write to the socket, `LS <remote_path>`.
- Read the output from the socket.

Uploading a File to BigFS

Declaration of the uploadFile function,

```
int uploadFile(char *rpath, char *fpath);
```

Uploads the local file at `fpath` to remote path `rpath`.

Steps

- Split the file into 1MB blocks and get value of `numParts`.
- Fork `num_FDS` number of child processes, each simultaneously uploading its set of parts to its FDS.
- Any part `k` goes to FDS with index `k % num_FDS`.
- After all parts get uploaded, UID data is sent to the FileNameServer.

Downloading a File from BigFS

Declaration of the downloadFile function,

```
char *downloadFile(char *rpath);
```

Downloads the remote file at `rpath` to a temporary location and return the local path.

Steps

- Get the UID information from the FNS.
- Fork `num_FDS` number of child processes, each simultaneously downloading its set of parts.
- Any part `k` is downloaded from a FDS with index `k % num_FDS`.
- After all parts get downloaded, they are joined and the path of the final file is returned.

Limitations

- When entering a command, filepath should not contain spaces.
- Changing the number or order of FileDataServers in the config file will result in corruption of old files on the server.
- Maximum length currently supported for the output of `ls` command is `2047 bytes`.
- Trying to copy files larger than `64 GB` may have undefined behaviour.