Design Details

Our task starts with a “Library” we built . In the Library directory, there are two files “Shell.h” and corresponding “Shell.cpp” . Shell.h declares a class we need for our basic shell operations , “MyShell” and Shell.cpp implements the class .

File : `Shell.h` and `Shell.cpp ( Defined is "Library" Directory )`

Class Name : `MyShell`

Class Variables : `function` , `argument` , `documentation_directory` , `list_file` , `help_file` , `result` : **`string`** AND `quit` : **int**. `function` is for function name we use for corresponding command , `argument` is if there any argument (or simply NULL) , `documentation_directory` is for the directory name the documentation is stored , `list_file` is the file name containing the list of commands , `help_file` is the file name containing the help , `result` for the output returned and `quit` (=1) for indicating whether user want to quit or not .

Class Functions :

1. `bool copyfile( const char src[] , const char dest[] ) :` Opens a file name src specified to copy and creats file dest for destination . It is a private function called from function cp .
2. `bool cd() :` Changes directory to the argument name . Uses system function `chdir(newDirectory)` for this purpose .
3. `void clr() :` clears the screen . Uses system function `system()` for the purpose .
4. `bool cp() :` copies the source file to the destination file . Uses `copyfile()` function we implement earlier .
5. `bool del() :` deletes the specified file . Uses `remove(filename)` for this purpose .
6. `bool dir() :` lists files and directories under a directory or current directory otherwise . Firstly , It opens the directory using `opendir(directoryName)` returning a pointer to the directory `"directoryPointer" .` Then read the directory using `readdir(directoryPointer)` and print .
7. `void echo() :` echo or print the `argument .`
8. `void environment() :` prints `environment variables .` Uses the `extern` variable defined in linux system header.

9. `void help() :` shows help content . Opens `help_file` and prints the content .
10 .`void host() :` prints `host` information . Uses `gethostname()` function for this purpose .
11 . `void list_command() :` lists command supported by the shell . Reads list_file and prints to the console . Uses `list_file` and shows it's contents .
12 . `void pwd() :` print working directory . Uses `system()` function defined in `stdlib.h .`
13. `void touch() :` touches a file . Means creates the file if not already exist . Uses `system()` function defined in `stdlib.h .`
14 . `MyShell() :` constructor for the class . Initializes all the private variables it needs.
15. `string Execute( string function , string argument ) :` Takes the function and argument as parameter . Processes the command with parameter and returns result as string .
16. `int isQuit() :` returns the value of variable `“quit” .`

File : `myshell.cpp ( defined in "MyShell" Directory )`

Description : We define the main function for our shell program here . It is only for using in Desktop environment . As the shell allows some batch file , first It checks whether any batch file is supplied to it or not . If so , it scans the command line argument and search for the file specified . We allow 1 batch file in our shell .If not , It waits for user's input to the command line .

First It determines the current directory to create a linux-like environment "`user@MyShell : /home/current`" something like that .If user enters the command and press ENTER , it then starts scanning the input . It eliminates all the starting and trailing spaces and distinguishes the `argument` from `function` .

Creates an instance of `MyShell` defined in `Shell.h` and pass the function argument to the shell's `execute()` function . Gets the result and prints to the standard output .

For batch file , it tokenizes all the lines using `strtok()` defined in `cstring.h` , it gets and pass each individual line as a command . Others procedures are just same .

We additionally create a `error()` function here to simplify the error and exit process .

File : `myserver.cpp ( defined in "MyServer" Directory )`

Description : We do have 2 major functions here , `dostuff()` and `main()` . In `main()` function , we setup the server settings . First we get the `portno` from the command line parameter . This is the `portno` we want to listen from the client . In order to connect with this server , the clients must have to connect through this port . Socket creation and binding is done through the `socket()` and `bind()` function . We define a marco named *`BACK_LOG`* , which indicates maximum number of client we want to permit waiting to our server . This is done through `listen()` function . `signal()` function is used for avoiding the *`Zombies`* created by the exited clients . It will read from client process through the `read()` function and write back through `write()` function defined in `unistd.h` .

The client request is accepted by an `accept()` function . We do a `fork()` here . The parent process just closes the client socket id . In the child process we do the necessary task for the corresponding client . We call the `dostuff()` function here . It will listen for client request (that is requested command) and run on it's local shell . It will then pass the output to the client .

The server will run on an **`INFINITE`** process . We must have to press **`Ctrl-Z`** to terminate the server process .

File : `myclient.cpp ( defined in "MyClient" Directory )`

Description : There is one major function in this file , the `main()` function . Additionally we have the `error()` function only to handle the error condition smoothly . This function requires 2 command line parameters , 1 the `server name` or `IP` and the `portno` to connect the server . It creates a socket for itself using the `socket()` function . And then sends request to the server through `connect()` function . If the server fails to accept the request , `connect()` will fail and the program will terminate .

After a successful connection , it will be able to "`talk`" with the server and "`listen`" from it respectively through `write()` and `read()` function . It simply reads a line of command (with starting and trailing spaces) and just pass to the server unchanged . The elimination of spaces , processing input and other stuffs are just left to the server to complete . Gets result from server and prints to the standard output console . A `quit` command will terminate the client .

<u>File :</u> `help.txt and list_command.txt ( defined in "Documentation" Directory )`

<u>Description :</u> Contains help files . For command help and list-command , we use them . list_file and help_file string of MyShell(declared in /Library/Shell.h) class indicates the list_file and help_file names respectively .