Name: Joseph Hany Boulis
ID: 900182870

Design Document

To build this simple shell, you have to follow the instructions I followed in each of these 6 steps:

1. In order to make the shell exit upon entering the shell “exit” command, you should check whether the entered line is the word exit or not. If so, exit(0), else do nothing.
   - `if (strcmp(line, "exit") == 0) exit(0); //exit`
2. In order to make the shell start with a default directory as the home directory of the current user, you should change the current directory to the home directory. For changing the directory, we use the following command:
   - `chdir(dir_to_go_to)`

   In order to get the home directory, we use the following command:
   - `getenv("HOME")`

If the user only entered “cd” without any other directory path, you should change the current directory to be the HOME directory.

Likewise, if the user wants to change the current directory to another one, he/she enters the full path of the desired directory and we repeat the previous command upon seeing “cd” as the first two characters in the input line, followed by the path of the file.

3. In order to make the shell run commands with arguments. You should only pass the path and argv variables to the execve function as follows:
   - `execve(path,argv,0);`
   - The path variable is the program full path of the shell command. I.e “cd Desktop” will have a full path of /bin/cd.
   - Argv is simply the shell command arguments parsed based on the space delimiter.

The execve replaces the whole process upon successful completion, but it will not be replaced if it was not executed successfully.

4. In order to make the shell implement output redirection:
   a. You should first detect that there is a ‘>’ sign that indicates the existence of an output redirection. You should do so using the following strchr function:
   ```
   if(strchr(command,'>')){
         exec_redirection(command,1); // 1 here indicated
   that it is an output redirection not an input
   }
   ```
   b. After that, you should parse the command based on the delimiter ‘>’. Then, fork and inside the child perform the following:
   ```
         int fd1 = creat(argv[1], 0644) ;
         dup2(fd1, STDOUT_FILENO);
   ```

```
                    close(fd1);
```

This will create a file and make its input the output of the previous command that will be executed inside it.

5. In order to make the shell implement input redirection:
    a. You should first detect that there is a ‘<’ sign that indicates the existence of an input redirection. You should do so using the following strchr function:
    ```
    if(strchr(command,'<')){
          exec_redirection(command,0); // 0 here indicated
    that it is an input redirection not an output
    }
    ```
    b. After that, you should parse the command based on the delimiter ‘<’. Then, fork and inside the child perform the following:
    ```
          char es[BUFFER_LEN];
          strcpy(es,argv[1]);
          es[strlen(argv[1])-1]='\0';
          int fd0 = open(es, O_RDONLY);
          dup2(fd0, 0);//STDIN_FILENO);
          close(fd0);
    ```

This will simply open a file and put it as the input of the stream and put its output as the input of the first command.

6. In order to combine both, you should prioritize ‘>’ then look check for ‘|’ then check the existence of ‘<’.

7. In order to implement multiple pipes, you have to follow the following steps:
    a. Detect that the given command has “|” sign in it.
    b. Separated the command by the “|” delimiter while calculating the number of commands and feed it to a function and call it exec_multi_pipes.
    c. Exec_mulit_pipes should include a while loop that keeps on iterating over the piped commands.
    d. Before the loop, you should initialize an array of file descriptors.
    e. For each command, you should fork and inside the child do the following:
        i. If this is the first command, you should close the input of the first file descriptor to ensure that the loop will not halt, waiting for an input from a pipe rather than taking input form the user,
        ii. If this is the last command, you should close the output of the last file descriptor to ensure that the loop will not halt, waiting to output in a pipe rather than outputting to the screen.
        iii. For the rest of the commands (between the last pipse and the first pipe), you should connect the input of the pipe with the previous pipe and the output of the pipe to be the execution of the current command. Then, close all the opened pipes.

        iv. In the parent of eac fork, you should wait close to all the opened pipes and wait for the rest of the children to die.

8. In order to support any combination of the above, you should do the following:
    a. First search for ‘>’ in the entered command, if you find it, you should search for the pipes then for ‘<’ and execute them respectively.
    b. Second search for a pipe, if you find it, then search for ‘<’, if you find it execute it and continue.
    c. Third search for ‘<’ if you find it just execute it.

9. In order to define variables and assign them to constants or to the output of other commands, you should do the following:
    a. First check for the “=” sign, if you found it, you should assign whatever is found after it to the variable defined before the “=” sign. However, you should check whether there is a tick sign or not. If you found a tick sign, then you should treat whatever is after the “=” sign as a command, if not, you should treat it as a constant.
    b. Second search for “echo” keyword if you find it, then print whatever is saved inside the environment for the input variable to the screen.