SimpleShell Implementation

Group Details

Group - 103
Members : { Arjun Tandon <2022095>, Chahat Ahuja <2022138> }

Application Design

1. The `main()` function initiates the global `HISTORY` variable, saves the original `STDIN` and `STDOUT` by using the `dup()` command
2. A signal-`SIGINT` handler method is mentioned before calling the `shell()`function
3. `shell()` function:
    1. It first calls `restore_stdio()` each time it is called since they could be modified if the user had added any `pipe` commands earlier
    2. It takes input via using the `text` helper methods
    3. It checks whether the command is `exit` or `SHELL_NAME file.sh` or a regular command
    4. In each of the case it calls a custom wrapper for these functions which calls `cleanexec(char *command)` which itself is a wrapper for `pid_t execute command(char *command)`
    5. It frees the text objects that were `malloc`-ed and returns `0` to the main function.
4. The star of the show is `execute_command` functions, other functions are generally wrappers for it
    1. First the command string is tokenized by using `strtok()` by setting `|` as the separater
    2. If there was no pipe delimiter, then normal execution is done
    3. In normal execution the string is separated or rather tokenized by using `strtok()` function by using whitespaces as separator

4. Now each if we had `@usr: a b c` it will be converted to an `exec_argv` array that would be `{"a", "b", "c", NULL}`
5. Now execvp will be called
6. If the command had pipes, then first the string would be tokenized on basis of pipes and the normal execution would be done (tokenization on basis of whitespaces) for each of the substrings (separated by pipes).

Limitations

1. To run a `.sh` file, the user must write `SHELLNAME filename.sh` to execute the shell script in the fixed syntax, so it's output can't be piped out.
2. Using interrupts like `Ctrl+Z` to perform tasks (like push the process into the background) aren't provided.
3. `cd` command doesn't work. You can use `mkdir` or pass path of a file to a command and that works but the. The reason for this is that the exec function will execute a program, whereas `cd` is a shell-specific inbuilt functionality that needs to custom implemented.
4. Using `^` (up arrow key) to get the previous command doesn't work. That would require custom input handling for special characters like `^` , `v` etc. to parse the previous commands via history.