Lab one
Shell and system calls

- **Prepared by :**

Mira Samir Ragheb (75).

• Introduction :

- A Unix shell is a command-line interpreter provides a traditional user interface for the Unix operating system and for Unix-like systems. The shell can run in two modes: **interactive and batch**. In the shell interactive mode, the shell program starts, which displays a prompt (Shell>) and the user of the shell types commands at the prompt.
- In the shell batch mode, you start the shell by calling your shell program and specifying a batch file to execute the commands included in it. This batch file contains the list of commands (on separate lines) that the user wants to execute.
- Commands submitted by the user may be executed in either the foreground or the background. User appends an & character at the end of the command to indicate that your shell should execute it in the background.

• Code Organization :

The code is divided about 14 modules as follows: -

1) The **main module** starts by checking the argument main parameter to determine whether it is an interactive mode or batch mode or invalid arguments.

2) The **Initializer module** allocates the suitable data structure , change the working directory to HOME and fills the file directories with the suitable paths to search for commands.

3) If it is interactive mode :- the shell goes for a while true loop to enable the user enter the command. If it is batch mode the **FileProcessing module** reads line by line from the batch file. If end of file or exit command has been encountered it returns to interactive mode. However, **BufferReader module** reads from the user through *fgets()* system call if it is interactive mode.

4) After reading the line and checking whether it is within the maximum range or not , The **LineParser** role comes which is to parse the line and separate it by spaces , quotes , expressions ,…etc.

5) The **command handler** takes the parsed line to handle it like :-

a) Check whether “$” variable exists or not and if exists it replaces its value from *user defined shell table* or *environment variables* if exists.

b) Check whether it is valid expression or not (eg x=5 , export y=”shell”) by invoking The **Expression module**. If it is valid It determines whether it is user shell variable (stores it at user variables table by using **The variables module**) or environment variable (stores it using *putenv()* system call by using **EnviromentVariables module**).

6) If it is not expression The **command executer module** role takes place which is :-
   a) Check whether it is a command executed by parent process (eg. exit , cd and clear) commands that executed by The **SpecialCommandExecuter** module or needs forking child process.
   b) If the command needs forking a child it forks a child process to execute the command by using the *(execv())* system call.
   c) It loops for the available paths to determine the path of the command and if it is not found the command marked as undefined.
   d) If the command ends with (&) the process goes background and the parent does not wait for the child termination and if not the parent waits for the child process to terminate.
7) After the child process terminates it sends *SIGCHILD* to its parent to allow The **Logger module** to write that "child process terminated" to the *shell_logger_file* which is found in the user HOME dir.
8) The **History Handler module** role is to record each instruction entered by user or read from the batch file *to shell_history_file* which is found in the user HOME dir.
9) If the user types exit or press (ctrl-D) the shell exits.
10) Finally, **ShellColor module** holds the colors used for shells.

• Main Functions :

Function	Explanation
void start_shell_loop()	The main function that loops while true invoke the buffer reader to reads command and invokes the parser then the command handler module to handle command and execute it.
void read_line(char* line)	Reads line from buffer and stores it in char* line
int parse_line(char** list, char* line)	Parses the line and separates it by delimiters, stores the parsed tokens inside char** list and returns its size.

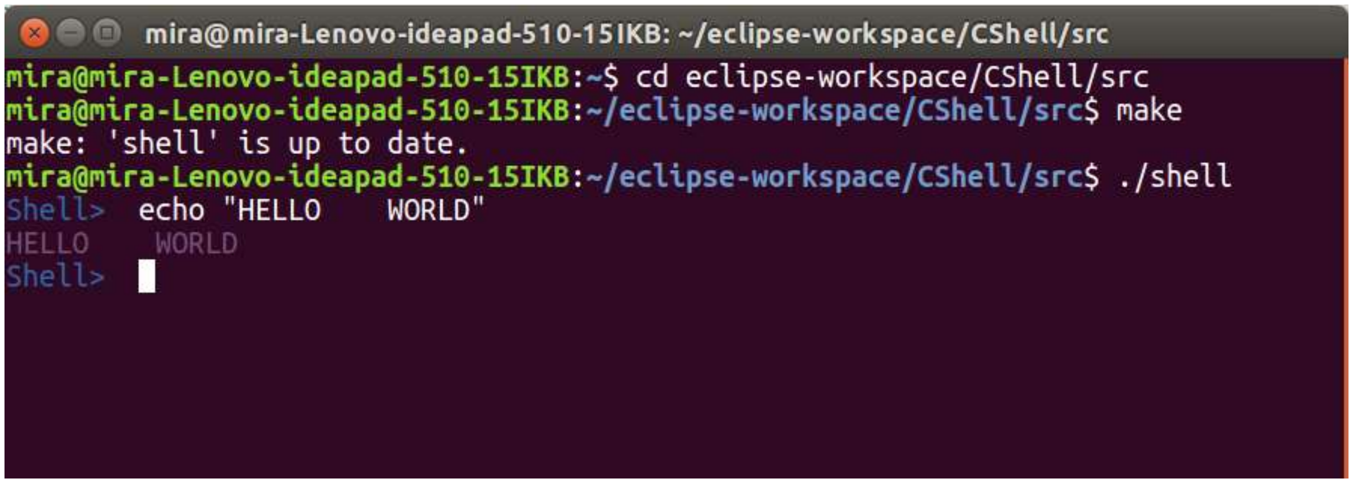
void append_to_history (char line)*	Appends to the history file the user command.
void append_to_log ()	Appends to the log file the “child is terminated” statement
*void handle_command (char** parsed_command , int size , char** file_directories)*	Checks whether the command is comment line , expression or else to invoke the command executer.
*void handle_variables (char** parsed_command , int size)*	Replaces variables ($) by its value whether it is user defined variable or environment variable.
*void handle_params (char** parsed_command , int size , char** final_params)*	Checks whether the parsed parameters is quoted or not and determine whether to remove the quotes or not.
*int evaluate_expression (char** parsed_command , int size)*	Checks whether the expression is user defined variable or environment variable If it is user defined invokes the variable handler to store it inside the table.
*void execute_command (char** parsed_command , int size , char** file_directories)*	If it is special command it invokes the special command method to execute it and if not it forks a child process to call execv system call to execute the command.
*void execute_cd (char** parsed_command , int size)*	Replaces (~) if exists by the HOME env variable and calls chdir system call to change the directory.
*execute_echo(char** parsed_command , int size)*	Calls printf to print the parameters as the user entered it.
Char find_path (char** parsed_command , int size , char** file_directories)*	Checks whether the user changed the PATH environment variable or not and if not it loops for the given file_directories to find the command path , returns NULL if not found.

• Features :

- The shell supports *printenv* command to print all environment variables even if it is changed by user.
- The shell handles echo command like the linux shell :-
  If the user stores the variable like v=” Hello   World”
  echo $v ➔ Hello World   but echo “$v” ➔   Hello   World

• How to compile and run :

- Change current directory to the directory which holds the src code.
- Type make to compile the make file (but it is updated so you can skip this step).
- Type ./shell to run the shell.
- If you want to execute batch mode type the file path as an argument like ./shell /home/username/test


mira@mira-Lenovo-ideapad-510-15IKB: ~/eclipse-workspace/CShell/src

```
mira@mira-Lenovo-ideapad-510-15IKB:~$ cd eclipse-workspace/CShell/src
mira@mira-Lenovo-ideapad-510-15IKB:~/eclipse-workspace/CShell/src$ make
make: 'shell' is up to date.
mira@mira-Lenovo-ideapad-510-15IKB:~/eclipse-workspace/CShell/src$ ./shell
Shell>  echo "HELLO    WORLD"
HELLO    WORLD
Shell>  
```