CS 3031

OPERATING SYSTEMS LAB

IMPLEMENTING SIMPLE SHELL

Group Assignment 1

Aman Verma (CS11B002)

Bhupesh Raj (CS11B006)

Ritvik Jaiswal (CS11B031)

PURPOSE

The purpose of this project is to implement a simple shell on Linux and Minix platforms. Ubuntu was the preferred OS for the Linux platform. The shell mimics **bash** in its functionality.

AVAILABLE RESOURCES

We are provided with 2 files – **shell.l** and **myshell.c**. The file **shell.l** provides a function **get_line()** to parse the command line and separate the command and its options.

Other available resources include systems calls of which the following were used –

1. **fork()** – Forks a new process from the current process
2. **exit()** – Exits the current process and returns to parent process
3. **execvp()** – Executes the command given as first argument along with options given as second argument
4. **wait()** – Waits for the child process to finish execution before resuming
5. **dup()** – Redirects stdin/stdout to a file or pipe
6. **close()** – Closes the file descriptor given as argument
7. **pipe()** – Creates a pipe between the parent and child processes

DESIGN

The project was designed in a modular fashion with different functions handling different cases.

The following functions were defined –

1. **void sig_handler (int signo)**

   Checks for the key combination ^C. This sends the signal SIGINT upon which the child process is terminated. The shell remains intact. **signo** is the signal which in this case is SIGINT.

2. **int normal (char **args)**

   Executes the commands which contain neither pipes nor input/output redirections. For eg. **ls –l**. args are the command line arguments. This function returns 0 if it exits normally, else -1.

3. **int output_redirection (char **args, int i, int n)**

   Forks a child, closes standard output and duplicates output to file. Parent process waits for child to finish execution before resuming. args are the command line arguments, i refers to the position of output redirection on the command line, n is the total number of arguments. This function returns 0 if it exits normally, else -1.

4. **int input_redirection (char **args, int i, int n)**

   Forks a child, closes standard input and duplicates input from file. Parent process waits for child to finish execution before resuming. args are the command line arguments, i refers to the position of input redirection on the command line, n is the total number of arguments. This function returns 0 if it exits normally, else -1.

5. **int pipes (char **args, int i, int n)**

   Multiple child processes are created from a single parent. First process writes output into the first pipe. Second process takes input from first pipe and writes output into the second pipe and so on. The last process writes output to the last pipe from which the parent process reads. args are the command line arguments, i refers to the position of piping symbol on the command line, n is the total number of arguments. This function returns 0 if it exits normally, else -1.

6. **int deciding (char **args)**

   Chooses which function to execute based on return value – 0 : normal, 1 : input_redirection, 2 : output_redirection, 3 : pipes. args are the command line arguments. This function returns the value 0, 1, 2 or 3 as stated above.

7. **int place (char **args)**

   Computes the positions in the command line of the input/output redirection symbols and pipes. args are the command line arguments. This function returns the position of any of the special symbols >, < or | on the command line.

TESTING

Testing involved trying out different cases like no pipes, multiple pipes, pipes with input/output redirection, simple input/output redirection and so on. Incorrect commands were tested to see if the program crashed, and if so, the error was identified and rectified.