S. HARISH KOUSHIK KUID: 2800977
SRAVYA, KUID: 2842319

Project 1: Quite a Shell (Quash)

Quash is a command line interpreter for use in linux systems. The basic functionality of Quash is similar to some popular command line interpreters like bash, tcsh, ksh. A while loop is run indefinitely to keep the shell active. Quash supports reading the commands interactively from shell prompt and from a file. This feature is implemented using the function readline () taken from the GNU readline library.

FEATURES IMPLEMENTED IN QUASH:

- **Run executables without arguments (10)**

When the program starts it reads the input from the function readline () which is taken from the GNU repository. The input is then compared with a set of commands implemented in the execute() function. If the input doesn't match then it checks in the path mentioned in execvp().

- **Run executables with arguments (10)**

In this case input arguments are checked using the splitargs(). If the arguments contain the special characters like '& or < or > or |' then the respective functionality is carried out.

- **set for HOME and PATH work properly (5)**

set for HOME and PATH is done using the functions **getenv()** and **setenv().** These are done when the input encounters the first argument as set then it is compared with the function set string in the execute function and when the string matches the set command is executed.

- **exit and quit work properly (5)**

Using the string comparison function, the input string is compared to "exit". If a match is found then the corresponding functionality is implemented. The "quit" is implemented in a similar way.

- **cd (with and without arguments) works properly (5)**

Using cd command we can change the current working directory to the directory specified and if no argument is given, it changes to the directory mentioned in the HOME environment variable. Implemented it using a function changedir() to change the directory. If the function returns -1 then it is displayed as Invalid Directory path. cd command uses the chdir() Application Program Interface call.

- **PATH works properly. Give error messages when the executable is not found (10)**

Quash searches the directories in the environment variable PATH if the executable is not specified in the absolute path format and if no executable file is found, quash prints an errormessage. PATH is taken from the current environment's PATH variable.

- **Child processes inherit the environment (5)**

The fork() command is used to create the child process. When the child process is created it inherits the parent's address space and environment including the parent processes functionality. To make the child execute a different functionality we use exec. In our Quash code we have used execp(). This functionality is mainly found while implementing pipes.

- **Allow background/foreground execution (&) (5)**

The executables are by default executed in the foreground. The input is constantly monitored by the chksymbol() function to check if it has any special symbol like '&' which indicates background execution.

- **Printing/reporting of background processes, (including the jobs command) (10)**

Commands without '&' are assumed to run in foreground. When the command is running in the background, quash prints: "[JOBID] PID running in background" to let user know the status of the process. When a background command finishes, quash prints completion messages as finished.

- **Allow file redirection (> and <) (5)**

I/O redirection is implemented in quash. The '<' character is used to redirect the standard input from a file. The '>' character is used to redirect the standard output to a file. Quash supports reading commands interactively (with a prompt) or reading a set of commands stored in a file that is redirected from standard input. Implemented these functionalities using chksymbol() function. The functionality of redirection has been implemented in in_redirection() and out_redirection() functions.

- **Allow (1) pipe (|) (10)**

Pipes are used send the output of one process to another process. File descriptors are created to read in and write through the pipe. We have implemented this functionality using the function runpipe().

- **(Bonus) kill command (5)**

The input is compared against the string "Kill" and if it is found then the a call to the function getjobjobid() is called, this is a structure of type jobstructure which returns the job id to the calling function. This job id is then returned to the built in kill function and the process gets killed eventually.

TESTING

We tested our script with grader script provided and we also made sure that all the possible scenarios are covered. The test cases gave us some positive results. Below table shows us the table of the manual test results

Testcase	Manual Testing
1) Run executables without arguments (10)	Pass
2) Run executables with arguments (10)	Pass
3) set for HOME and PATH work properly (5)	Pass
4) exit and quit work properly (5)	Pass
5) cd (with and without arguments) works properly (5)	Pass
6) PATH works properly. Give error messages when the executable is not found (10)	Pass
7) Child processes inherit the environment (5)	Pass
8) Allow background/foreground execution (&) (5	Pass
9) Printing/reporting of background processes, (including the jobs command) (10)	Pass
10) Allow file redirection (> and <) (5)	Pass
11) Allow (1) pipe (|) (10)	Pass
12) Supports reading commands from prompt and from file (10)	Pass
Bonus Question	
13) Support multiple pipes in one command. (10)	
14) Kill command (5)	Pass

We have not implemented multiple pipes and hence tests based on it will not be successful.

ADDITIONAL FEATURES

1) Added history to store the commands that were given as input. This gives us the flexibility to use arrow keys for command navigation. This has been implemented using the readline.h headers in the GNU repository

2) Added signals to handle Interrupts

ACKNOWLEDGEMENTS:

- Prof. Heechul Yun for making us understand the concepts
- GTA, Amir Modarresi, for guiding us during the course of the project

THANKS