Sean Stout
Chinmay Ratnaparkhi

EECS 678
Project 1
March 7, 2015

Quash is a simple program designed to simulate the shell. It continuously listens for user input, reads and executes commands. The overall architecture includes an infinite loop prompting the user for input, a parser which breaks the input down into commands or keywords and the execution blocks which run the commands.

Features

Run executables without arguments (10)

Any executable file located in the same directory as the current working directory of Quash can be executed simply by typing its name.
A user can also enter the absolute path of the executable file to run it.
For example : /home/cratnapa/EECS678/project1/a.out
If only the name of the executable file is provided and the file is not present in the current working directory, Quash looks for the file in all the directories specified in PATH. If the file is not present in any of these directories an error message is printed. Quash continues to listen for user input.

Run executables with arguments (10)

The first argument that is not recognized as a built in command is interpreted as the name of an executable file. The words that follow the name of the executable file are considered to be the arguments to the file.
For example : gedit myFile.txt

'set' for HOME and PATH work properly (5)

set requires the user to enter a path, that is assigned to either of the variable names (HOME or PATH). The user must enter a valid absolute path for HOME in the following format:

```
> set HOME=/home/cratnapa/
```

The user can enter multiple valid absolute paths for PATH, as it can hold more than one addresses. The paths must be separated by colons, for example:

> set PATH=:/usr/sbin:/usr/lib64/alliance/bin

At any time, these **values can be viewed** by entering the following command
> get paths

‘exit’ and ‘quit’ work properly (5)

Exit and quit are built into quash as special keywords. The core of Quash is wrapped in an infinite while loop which keeps it listening for input. when ‘exit’ or ‘quit’ is recognized, a break(); statement is executed which brings the control out of the loop and subsequently terminates the execution.

Exit and quit stop the execution of quash, although if there are processes running in the background, they are not affected by the termination of Quash. They run their course and report their independent completion after finishing execution.

‘cd’ (with and without arguments) works properly (5)

chdir() is used to change the working directory when the cd command is used.

1. **cd with no arguments** uses the getnv() on the environment variable HOME to change the working directory to the HOME path set by the user.
2. **cd with valid directory name** changes the working directory to the provided directory and reports the new location.
3. **cd with invalid directory name** does not change the working directory and simply reports the current location.
4. **cd with ..** changes the working directory to the parent directory of the current directory and reports the new location.

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

When an executable file name is entered, i.e. when the entered value is not recognized as any special command, Quash first checks for the executable file in the current working directory. If the file is not present, it looks for it in the paths included in the PATHS environment variable.
If the file is found in one of the addresses in PATH, it is executed. If it is not present at any of the addresses, an error message is printed on the screen.

Child processes inherit the environment (5)

If a child processes is forked out from Quash to run a given assignment, it inherits the environment variables HOME and PATH. The command execvpe is used to pass in the environment to the child process.

As expected, if the inherited environmental variables are modified in the child process, the environmental variables of the parent process remain unaffected.

Allow background/foreground execution (including the jobs command) (10)

A process can be run in background using the &. There must be a space between the process name and the &. For example:

```
> ./a.out  &
```

The parent process does not wait for the background process and continues execution. If the process is run in foreground, i.e. without the '&' then Quash waits for it to finish to resume execution.

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

An array is used to keep records of all the background processes in execution. Every time when a new background process is created, the global Job ID variable is incremented. The process ID of the background process is added to the array.
When the background process finishes its execution, the exit_handle() function is signaled with the completion and the process is then removed from the array.
At any time, active background processes can be viewed by entering the 'jobs' command.

```
> jobs
```

Allow file redirection (> and <) (5)

The file redirection is achieved by using the freopen() function to set the stdin and stdout values of the processes. For example, a command such as below :

```
> ls > list.txt
```

would be recognized as a file redirection situation by the '>' in it. The result of execution of the ls command will then be written into list.txt by setting list.txt for stdout.
Similarly, the '<' symbol will set the command following it to be stdin for the command preceding to it.

Allow 1 pipe (10)

When commands are entered with a pipe in between them, the strtok() function is used to chop the input string into two parts, the command before the pipe and the command after the pipe.
The first command is executed as an independent process and its output is written to the writing end of the pipe. Then the second command is executed as another independent process with its stdin coming from the reading end of the pipe. Hence the output of the first process is used as the input of the second process by using pipe.

Supports reading commands from prompt and from file (10)

This is achieved by redirecting the provided file to stdin.
When all the commands are executed from within the file, it is necessary to return the control to the keyboard, so the user can continue interacting with Quash. This is achieved by calling freopen("/dev/tty", "r", stdin); which makes Quash start listening to keyboard input.

Bonus

Kill sends signals to the background processes (5)

PIDs of the processes can be viewed with the 'jobs' command. The kill command can then be used to send signals to a specific process using its pid. The kill command has the following format :

> kill SIGNAL PID

Following 7 signals are supported :

1. SIGABRT - Abnormal termination.
2. SIGFPE - Floating point exception. For example, division by 0.
3. SIGILL - Illegal instruction
4. SIGINT - Interrupt signal
5. SIGSEGV - Segmentation violation signal
6. SIGTERM - Request to terminate the execution
7. SIGKILL - Request to terminate the execution

Extra features

1. **Current location**
   Quash prints the current working directory location on the following command
   > where

2. **Environment state**
   Quash prints out the current settings for the environment variables such as HOME, PATH and ROOT upon entering the following command
   > get paths

3. **cd to a folder with space in its name**
   If a directory name has a space in it, Quash can still navigate to it without a problem. For example :
   > cd Documents/Homework/Assignment 4/

4. **GCC**

   One can compile a *.c file by entering the following command

   > gcc <name>.c

5. **Copy, move, rename and delete files**

   One can use all the standard shell commands for the following functionality

   a. mv currentpath newpath -- move a file to a new location
   b. mv filename newname -- rename a file
   c. cp filename newname -- make a copy of the existing file
   d. rm filename -- delete an existing file