Timothy Frese
Dawson Conway
EECS 678
March 7, 2015

Project 1 Report

The objective of this projective was to create a shell with similar functionality to bash. This shell needed to be able to run executables with command line parameters, search for executables in the environment variable PATH if it was not specified in the absolute path format, allow for foreground and background executions, support the built-in function set to set the value of a variable in the environment, support the built-in function cd to change the current working directory, support the built-in function quit or exit to exit the quash, support the built-in function jobs to print all of the currently running background processes, support I/O redirection, support the pipe (|) command, support reading commands from a prompt, and support reading a set of commands from a file.

Our solution is based around a while loop that continually parses the user's input, turns it into a command, executes that command, then checks the status of jobs running in the background. Commands are represented by a struct that contains the name of the function, an array of the arguments, the file descriptor for the input, the file descriptor for the output, the pid (if it is a background process), and a pointer to the next command in the chain of commands. Our parse function is able to turn the line of user input into one or more of these structs and pass it along to the execute function. The execute function first checks whether the command is a built-in function, if it is, the command is run and then the program returns to main. Next it checks if the command is to be run in the background, if it is, the program forks, executes the command in the child while the parent returns to main. Otherwise the command is run and then the program returns to main.

When a function is not specified in the absolute path format, we utilize the realpath() function to first determine whether or not the function exists. If it is not a relative or absolute path, we begin searching in the directories in the environment variable PATH.

Our solution allows for foreground and background processes. This is accomplished by when parsing the input, setting a variable name background to 1 before passing it to the execute function. The execute function then checks if the job is to be run in the background. If it is, the job is added to an array of commands to be run in the background. The function checkjobs loops through this array and checks for jobs that have terminated and cleans them up.

The built-in functions set, cd, quit or exit, and jobs are supported in our solution. Set uses setenv to set a value of a variable in the environment. If set is called with no assignment, the environment variable is printed instead. The cd function changes the directory to the input. This can be relative or absolute. Passing no input causes the directory to change to HOME. Quit or exit simply stops all processes and exits quash. Jobs loops through all the currently running background jobs and prints each of them out.

I/O redirection is supported. Each command holds file descriptors for the input and output of that function. If a < is present, the following filename is opened is opened and will be used as input by the first command. If > or >> is present, the following file is opened (as overwrite or append respectively) and used for output of the last command. If there are |s each one delimits a command. Pipe is called for each | and the write end of the pipe is used for the output of the command before the pipe, and the read end is used for the input to the command after the pipe. If pipes are used (actually in any case) each command is run in it's own process. The parent process waits for all it's children. In the case of background execution the parent of the command processes is a child of the main process, and the main process does not wait for it.

Reading commands in from a file works the same as reading in from stdin, as long checks are made for EOF.

Quash is fully capable of performing each specific function described in the project assignment. Additionally, our solution is capable of supporting not just one, but multiple pipes. This document details how our solution matches all the requirements of this project and the extra features it supports.