Quash Report

Tim Fox 2911208

Tanner Strickler 2370044

Build instructions:

make

./quash

\*follow prompts from there

Our run executable without arguments feature runs an executable with out any arguments passed in. We implemented this by finding the path to the command either in the working directory or by one of the paths in PATH env. We check if the executable exists and if it does then we pass it into execvp which executes the executable.

Our run executable with arguments feature is able to run an executable and also pass in arguments after it delimited by white space. We were able to implement this feature by taking the command line and parsing the command from the arguments. We then find the path of the command either in the current working directory or follow one of our paths from PATH env. We then take our arguments and turn them into a char star array. We then pass the command and the arguments array into execvp and that runs the program.

Our quash child processes inherit the parent process environment. This happens through the execvp command. We tested it by calling quash inside of quash and using our pwd command to see that the paths were identical.

Our program is able to execute programs in the background. We use the ‘&’ to denote when we want to run a program in the background. We check the command line input for an ‘&’ and if it exists we send our parsed command to the run in background function. We erase the & and handle the command with the arguments we passed in as described above. However, we fork the child and have the child run the command and the parent will create a job for our job list. We will insert a job id, the pid, and the command that was run. We will then push the job into the job list. We then have a while loop that waits for the child but allows the parent to proceed. We print out the job information and tell the user when running in background and when it is finished running in background.

We are able to set HOME and PATH variables in our quash. We first check to see if the user is trying to set HOME or PATH. If not we do not allow any other variables to be set in this fashion. If it is we check if the environment is already set to what we want to set it to by using get env and comparing. If not then we use setenv to set environment to our desired location.

Our cd function checks to see if we added a destination after our cd command. If no destination is specified then we change directory to our HOME environment whatever we have it specified to. If we have a faulty home directory path then we will output an error. If we do enter a destination after the cd command then we will change the directory by using chdir and passing in the destination as the argument.

In our main function we set a flag running to true and if the user enters the word “quit” or “exit” then we set the flag running to false and exit our while loop which closes our quash.

Our jobs command prints all of our running jobs in our job list. We first flush jobs which means we remove all the jobs in our list that have been terminated and their job\_running flag has been set to false with our kill function. Then we simply output our array of jobs with job id, pid, and command.

Our Quash program allows for I/O redirection. We can overwrite standard input with the ‘<’ character and overwrite standard output with the ‘>’ character. We check to see if the two characters are present or if one or the other characters are present. We then build inputfile and / or output file paths based on the right side of the characters. We take the left side of the character as the command. We then fork the process and use freopen read or write whether we are reading or writing. Then we handle the command with that input or output specified.

Our Quash program allows for the use of ‘|’ pipe commands. We can pipe std output from the left hand of the pipe to std input to the right hand of the pipe. We implemented this by breaking down each pipe with a before\_pipe and after\_pipe string. We then create a pipe with an array of 2 elements. We fork a process and dup2 the pipe and overwrite stdout. We run the command and move on to the second process where we use dup2 to overwrite std input and run the second process. We the close the pipes to cleanup any errors there may be. We handle multiple pipes the same way.

Our quash supports reading commands stored in a file that is redirected from standard input. We do this by inside our main function using get line at the beginning, and if it is not an empty string, then we run the commands that are stored in the text file. If it is an empty string then we enter the interactive quash terminal and prompt the user to enter commands to execute or quash functionality to execute.

Our quash supports killing processes based on job id. We loop over the jobs and match the job id with the job id that was specified and if they match then we call kill and pass in the pid of the job with the job id. We then set the job\_running flag to false so that when we flush jobs, those killed jobs will be erased out of our job list erase.