Sean Hannah, Eric Chanthorabout
EECS 678
Project #1 Report
03/06/14

Implementation:

Our Quash shell program begins by running a while loop that takes in each character entered in by the user and loop this, concatenating each character to a sting,until the user enters in a new line. Once a new line is read, the program will then tokenize the string created by the input by spaces and then store them into an array to be compared later. After the string is tokenized, the shell will go to a large if-else block to determine which feature Quash should run. Quash will keep running through user input and running the desired features until the user enters the terminating token that will be mentioned below.

Features:

1. In general, to run an executable with or without arguments, Quash goes to the default case of the if-else block and begins a new job in the foreground with the tokenized string. The executable process is then forked and added to the job list as a child of the main process. If arguments are present, Quash will run them with the executable.
2. To set HOME and PATH, Quash checks for the token “set” as the first token and then checks for “HOME” or “PATH” inside the string argument after set. If they are found, Quash uses setenv() to change the appropriate path or will return an error if they are not found.
3. To quit, Quash simply checks for the user input “quit”, “exit”, or “q” as the first token and then uses exit(EXIT_SUCCESS) to terminate the while loop in the main function.
4. To change the directory, Quash checks for the first token to match to “cd”. If it does, then Quash will use chdir() to change the directory to the second input token. If there is only the one token, “cd”, then Quash will default to setting the directory to HOME by using chdir(getenv(”HOME”)). Any incorrect path entered will return an error.
5. As an executable is run, as mentioned in section 1, Quash will search through PATH to see if the executable is found. If not, then Quash will return an error and abort the process.
6. To make a child process inherit the environment, Quash puts the requested job in the foreground by changing its status and using tcsetpgrp() to make it take over the environment while we want it to.
7. In Quash, if a & is read after an executable or “bg” is read as the first token, then the process will be pushed to the background and added to the job list by our beginJob() method.
8. To check on the current list of background processes, the user needs to enter “jobs” into the command line and Quash will report the current list with each job's information from the job structure.
9. If “<” is read from the token list, then Quash will create two list of tokens from both sides of the operator and will fork the process and use dup2() and execvp() to pipe the output of the right side to the operators left side. If the “>” token is read from the token list, Quash will do the same process as above, but pipe the results of the left side of the operator into the right output file.
10. If “|” is read from the token list, Quash will create two list of tokens from both sides of the operator and will do a similar process above to pipe information from the command of the first list to the command of the second list. This project currently only works properly with one pipe.
11. Since Quash uses strings and tokenizes them for input, the program is able to process user commands when prompted and can read lines from a file and execute them in order.

Testing:

To test Quash, we ran through each case in the if-else block in the main while loop to see if the general cases were being reached. Typing in “cd” or “cd /home/shannah/EECS_678” went to the proper directories. To test executables, we used xload, gedit, and files from previous labs that required arguments. When gedit was closed, control went back to Quash. These worked, so we used “bg” and & to push them into the background and then “jobs” to list them out. This worked fine, although there was issues with getting processes moved to and from the background and foreground. Using “ls” and “cd” inside Quash showed us that our PATH and SET environment variables were working properly and that using “set” changed them to the intended values. To test piping, we used “ls | head -3” to print the first 3 elements from ls. This matched the expected values compared to the Linux terminal. To test input/output redirection, we “tested ls > a.txt” which put the results of ls into a.txt and then used “head < a.txt” to print the first 10 elements of a.txt to the shell.