Adam Smith, Kendal Harland
Kulkarni
EECS 678

Project 1 Report

Implementation

Quash is centered around a while loop that accepts user input. While user input exists, Quash accepts input and parses it into arguments using *tokenize()*, a parsing function. *tokenize()* also determines whether the commands require pipes or file redirection. Once *tokenize()* has parsed the user input into a list of jobs, these jobs are sent to a function called *executeJobs()* where they are run.

Features

- Run executables without arguments (10)
    - Executables are run in *executeJobs()*. First, the executable is compared against a list of built-in commands. If none of these match, *execvpe()* is called and this searches the PATH for the executable.
- Run executables with arguments (10)
    - These are run the same way as described above. A list of arguments is also included and is passed to *execvpe()*.
- set for HOME and PATH work properly (5)
    - For all environment variables, we are using *getenv()* and *setenv()* to handle any updates.
- exit and quit work properly (5)
    - The *executeJobs()* function recognizes *exit* and *quit* as built in commands and exits the program if it encounters either.
- cd (with and without arguments) works properly (5)
    - changing directories is done using the cd() command. cd() uses the chdir() API call. If cd() returns an integer < 0, the target directory was not found - else the current directory is changed to the target directory.
- PATH works properly. Give error messages when the executable is not found (10)
    - PATH is taken from the current environment's PATH variable. execvpe() searches path for a specified executable if the command does not match one of quash's built in commands.
- Child processes inherit the environment (5)

    - This was implemented by using the *execvpe()* function. The third argument of this is *char ** environ*. This allows the child process to inherit the environmental variables of the parent process.
- Allow background/foreground execution (&) (5)
    - Jobs were implemented by creating a Job *struct* with various properties. One of these properties, *bool* bg, is true if the process is to be a background process. When a program is executed, a child process is always created. If it is to be a foreground process, the parent process waits for the child process to complete.
- Printing/reporting of background processes, (including the jobs command) (10)
    - Jobs uses a combination of hashmaps to associate a given job id with a process id and filename. These two maps are utilized when printing job information to the screen.
- Allow file redirection (> and <) (5)
    - It is determined in *tokenize()* whether or not file redirection is needed. If the parser encounters '>', then the outputFile in the Job *struct* is set to the next argument. Input files work in the same way. In *executeJobs()*, a check is run to see if that job has specified input or output files. If so, a function is called to redirect STDIN or STDOUT to the correct file. This was implemented using the *dup2()* method.
- Allow (1) pipe (|) (10)
    - This is handled the same way multiple pipes are handled.
- Supports reading commands from prompt and from file (10)
    - Our program reads from STDIN, so it can handle input from both prompt and file.
- (Bonus) Support multiple pipes in one command. (10)
    - Each set of commands (separated by a pipe) is put into a job struct. Each job maintains the list of arguments and the pid corresponding to that job. As the jobs are executed in succession, the main thread closes the write end of the last job to execute, while the jobs themselves close the read ends of their stdin descriptors.
- (Bonus) kill command
    - This was implemented by using the C *kill()* function. This function allows a process to be killed by passing its process ID.

Testing

Testing was pretty straight-forward. cd was tested with various directories including ~, home, ../, ../../, etc. Vim and grep were used to test executables. In order to ensure that background processes, jobs, and the kill function were working correctly, we opened gedit in the background, listed all jobs, and then killed the gedit process. File redirection was easy enough to test. We used commands like ‘cat text1.txt > text2.txt’. Testing pipes was similarly very easy. A couple examples were ‘ls -l | grep .txt’ and ‘cat test.txt | more | grep the’.