Devin Mullins & Christian Warner

Quash Project

EECS 678

3/25/14

Quash Implementation

**Running an executable**

Whether a job is being executed with or without arguments, file descriptors are first saved for STDIN and STDOUT to insure if file redirection occurs, that it goes back to the prompt. Next, file redirection is checked and set. A loop then goes through and breaks the command into characters to check if the path is absolute, local, or under the path variable. If the file exists on a given path, it will fork a child process and run execve() using arguments if given. The program will let the user know if there is an error executing or forking, or if there was a problem accessing the file.

**Set HOME and PATH**

Quash includes the ability to set the HOME or PATH variables. This is invoked when the program sees the word “set” as the first argument. It then checks the second variable, telling the user to enter “PATH=” or “HOME=” if the argument is either nonexistent or invalid. If it is “HOME=” or ”PATH=”, it will utilize setenv() to set the variable equal to the substring of the second argument located after the fifth character.

**Exit**

If the program sees the words “exit” or “quit”, it terminates.

**Change directory**

Upon seeing “cd” the program looks for an additional argument, if one is not found, it changes the directory to the variable “HOME” using getenv(). Otherwise, it will use the system call chdir(directory) to go to the targeted directory.

**Jobs**

When Quash sees the word “jobs”, it will use the function jobs() to print out a vector containing all background jobs. There is a job struct, keeping track of all relevant information. Data is inserted into the struct when the job is flagged as a background job.

**File redirection**

When Quash parses a command, it looks specifically for file redirection symbols, “<” and “>” , when these are seen, it simply sets a variable in the job struct to the string following the symbol. At the beginning of execution, if these variables are not equal to an empty string, it will change the STDIN\_FILENO and STDOUT\_FILENO to the designated files.

**Reading commands from a file**

When an input redirection symbol is seen as the first parameter, Quash will take that as the input, as compared to reading from the terminal. It handles reading from a file the same way it would redirecting input of any kind.

**Background processes**

During parsing, Quash also looks for an ampersand, setting a flag for a job when one is found, telling it to run in the background. When it is ran in the background, a signal handler is used to keep track of when a job is terminated.

**Environment inheritance**

The environments of parent processes are passed down in Quash by using execve() over other execution options.

**Piping**

Pipes are first detected when the command is being parsed. If it detects a pipe symbol, it will add a job to the vector. Initially, when executing a job, Quash initializes a pipe. When the process is forked, if there are enough arguments to support a pipe, Quash sets a write and read end to a pipe. At the end of execution, all pipes are closed.

**Kill**

Processes are killed in Quash by simply using the kill(string) command, which simply terminates a process based on its pid, by utilizing the system call kill().

**Testing**

Quash was tested by using all implemented commands followed by both a valid and invalid input. Different cases of invalidity were used where possible, including both nonexistent and inaccessible files. Simple output streams were used to essentially analyze each case, ensuring Quash works under all circumstances. Quash was also tested to the 4th level, as Quash can run itself, under itself, under itself, likely until system resources are completely used.