Sean Boeck, David Gier EECS 678

**Implementation**

**Project 1 Report**

Quash creates a clone of a shell by reading user input and executing commands. The program prompts the user upon running to enter in any normal command and then processes the string of input. Using the strtok() function Quash reads the inputted line word by word looking for special tokens to denote certain jobs using the parse() function. Once read the list of created jobs is then executed as commanded using the execute() function.

**Features**

* Run executables without arguments (10)
  1. o Executables can be run using the execute() command. If the current  argument given by the list of Jobs is not a pre-built function Quash runs it  as an executable.
* Run executables with arguments (10)
  1. o The first argument to miss the pre-built commands is taken to be an executable. Any others are taken to be arguments for the executable if they too are not pre-built commands.
* ‘set’ for HOME and PATH work properly (5)
  1. o Set works by using the system call ‘setenv’ and inputting the name (PATH  or HOME) and the path as parameters.
* ‘exit’ and ‘quit’ work properly (5)
  1. o ‘exit’ and ‘quit’ are pre-built commands that set exitbit to 1. This bit is used in the while loop that continually prompts the user for input. When set to 1 the loop ceases and the program can end.
* ‘cd’ (with and without arguments) works properly (5)
  1. o Uses the system call chdir() to change the directory. If there is no  argument it uses getenv() to find the path to home and uses that.  Otherwise it uses the given path as a parameter.
* PATH works properly. Give error messages when the executable is not found (10)
  1. o When a command is executed, it looks for the file in all places specified by PATH. PATH is set using the ‘set’ command.
* Child processes inherit the environment (5)
* Allow background/foreground execution (including the jobs command) (10)
  1. o Done by having the parent process wait for processes that are being run in the foreground but not for those being run in the background. All jobs are child processes that are overseen by these parent forks.
* Printing/reporting of background processes, (including the jobs command) (10)
  1. o The list of background jobs is kept in a Job array. When ‘jobs’ is called, the program checks each of these background processes to see if it’s still  running and prints its information.
* Allow file redirection (> and <) (5)

o This is done by using dup2() to change file input/output to  STDIN\_FILENO/STDOUT\_FILENO. When this job is finished the job input and output field are reset back to null so as to appear empty and not interfere with the next job.

* Allow 1 pipe (10)

o Same as multiple piping

* Supports reading commands from prompt and from file (10)

o Done by redirecting the file given to STD\_IN after seeing the ‘quash’ command.

* (Bonus) Support multiple pipes in one command (10)

o When the ‘|’ token is seen the number of jobs is incremented to show the current job will be piping into the next. In execute() if the number of jobs is greater than 1 then piping must have occurred. The number of pipes is equal to the number of jobs minus one. Each job is then linked in sequence so that each pipes into the next. Each pipe that is not last to writes to the next pipe id and each pipe that is not first reads from the current pipe id. After the final job finishes all pipes close if they have not already

* (Bounus) Kill comman (5)

**Testing**

Testing was done through using three files and a folder. HelloWorld.c outputted a simple phrase without taking arguments. Mimic.c took a single word as an argument and appended it to a string before returning the string. TestWorld.c was a clone of HelloWorld.c placed in a sub folder ‘bar’ in order to test ‘cd’’s functionality. Piping was tested by combining writer.c and reader.c. Input and output redirect was tested using the test1.c file in combination with Input.txt and Output.txt