Karl Schaller, CIS 3207, Project 2

Shell Program Description

A shell is an interactive program that allows the user to send commands to the OS and allows the OS to respond to the user. This is achieved through a command line where the user can enter a string of characters terminated by pressing the Enter or Return key and the OS can type lines of characters back to the screen.

In our program design, we first check if commands are to be read from a batch file specified at launch. If so, stdin is redirected to this file. Then, in the main loop, we prompt the user with myshell:~/CURRENT-DIRECTORY before reading input from stdin line by line until we reach EOF or a “quit” command. The input is tokenized by whitespace into a vector of arguments terminated by a NULL pointer.

The first argument, argv[0], is the command and is analyzed to see if it is one of the 8 built-in commands. If it is, the corresponding built-in is run with the given arguments. Output redirection is supported by dir, environ, echo, and help through fopen. Otherwise, the command is interpreted as if it is the name of an external executable, and, after a fork, is to be run by the child process with execvp().

If a piping symbol ‘|’ is included in an external command, the child forks again. In the child, stdout is redirected to the write end of the pipe and argv is changed to the first command. In the parent, stdin is redirected to the read end of the pipe and argv is changed to the second command. If redirection symbols are included instead, stdin or stdout are redirected to the specified files in the child process that will execute the command. If the command contains a trailing ampersand ‘&’, it is removed from the arguments before processing, the parent will not wait for the child to finish processing before printing the prompt and awaiting the next instruction.

Testing the built-in commands was simple. To test environ, I compared the output with the external command env. Error checking is included for cd and dir. The commands were all tested using various arguments and output redirection (both write and append).

External commands were tested extensively with redirection and piping. Ex: ls > out.txt, wc -m < out.txt, cat < out.txt >> file.txt, cat file.txt | wc -m. Error handling is also included for many parts that could result in an error: trying to open a file that cannot be opened, opening a directory that cannot be opened, fork() errors, pipe() errors, and exec errors.