Christopher Richard Bilger
CIS 3329 - Network Architectures

Shell Pseudo-Code

Parser

The parser will handle all text that is input from the user's keyboard. This will be done by a function that will take in a char *cptr and parse through the char *cptr with a space delimiter. The parsed output will be of the form char **cptr, which will be a pointer to a character array. The character array that is pointed to will contain a series of elements equal to the input typed by the user, excluding all spaces.

- Ex: myshell> ./myprogram 10 20 30
- Parser output: [“./myprogram”, “10”, “20”, “30”]

Batch Input versus Single Command Input

The first portion of code that will run will determine whether or not the shell program will execute as a normal shell, or whether it will be processing a text file containing a list of commands to be executed.

Batch Execution:

- In batch execution mode, the shell program will have to use the above parser to make an array of parsed commands that will then be run in series until completion and then the shell program will exit successfully and give control back to the terminal.

Single Command Execution:

- In single command execution mode, the same process will occur where the input gets parsed by the above parser; however, there will only be one command to be run. Once this command is run successfully, the shell program will not exit, but rather it will loop back and ask for another command to be input from the user.

Internal Commands That Need To Be Supported

- cd <directory>
- clr
- dir <directory>
- environ
- echo <comments>
- help
- pause
- quit

Christopher Richard Bilger
CIS 3329 - Network Architectures

External Commands That Need To Be Supported

- exit
- cd <directory>
- path <... arguments to be added to the system path>

Specifics That The Shell Will Handle

- I/O Redirection of input: <
- I/O Redirection of output: > and >> (difference is truncation vs file creation)
- Piping: |
- Parallel Commands: command1 & command2 & command3
- Background Execution: <commands and arguments> &
- Error Message: standard error message given out whenever an error is encountered

Logic Control Blocks

The main logic will be very similar to that of the previous assignment. The main.c file's main function will determine the mode of the shell program (single command execution vs batch execution).

1. Single Command Execution
    a. A while() loop will start by printing the shell name along with the current working directory. Ex: "myshell:/home/user/chris/ > "
    b. The loop will then wait for the user to enter the desired input. After entering input, the main loop will send the c-string to a function in the parser.c file that will parse the c-string and return an array of char *cptr in the form of char **cptr.
    c. After parsing the given input from the user, the shell will determine whether or not the given command is in the OS's "path" environment variable. If it is, then the code will be run directly. If it is not, then the shell will do one of two steps.
        i. The first step is to determine if the command is a filename in the current working directory. If it is then the program will be run directly.
        ii. The second step is to check whether or not the command is hard-coded into the shell program and if so, then run the code for that command. (Some commands must be hard-coded as per project requirements)
    d. Once the program is determined, then the parser will check for I/O redirections and/or pipes and will handle those accordingly.
    e. After all of this occurs and the given child process is running properly, the parent process will loop back to the beginning of the while() loop and continue until an exit condition is met. (Exit conditions include ctrl-d and the "exit" command)
2. Batch Input Execution:

a. A while() loop will run over the batch file, extract out each line individually from the batch file and use the parser.c file to parse the lines into char arrays of the form char **cptr.
b. A new loop, inside of the while loop, will then run over every single given command in the specified file, and run every single command in series until completion.
c. Once the nested loop finished the array of commands, the shell process will terminate with a successful exit code.

Library/System Calls

The system libraries will be used very frequently during the parsing, forking, and execution of child processes. The following is a list of some library/system calls that will be used in the shell program.

1. getcdw() from unistd.h
2. fork() from unistd.h
3. execX() from unistd.h (X will be changed with the necessary letters for each execution function call)
4. strtok() from string.h (The standard C library "string.h" will be used thoroughly in the parser.c file)

Test Plan

a. As far as individual modules are concerned, there will most likely be three. One will be parser.c which will handle the parsing of user input. Another will be main.c which will run the main while() loop to control the main logic of the shell. Lastly will be a tests.c file which will be used to handle all of the tests for each function that I implement myself.
b. The possible bugs for the parser.c file will have to be tested extensively to make sure that any user input will, at the very least, be handled and tokenized properly. The main.c file will, most likely, be the trickiest part of the shell program to debug because of the fact that it runs most of the core logic that the shell will be implementing. The tests.c file will not have as many opportunities for bugs to come about because it will simply test the functions that I make in order to check for consistency.
c. In order to test for possible bugs in each of the modules that I will have, I will need a feasible testing suite to get rid of them. To achieve this I will implement two main testing features. Firstly, I will generate as much user input from other people, as well as myself, to throw at the parser functions to try to break them. I will also check for "edge" cases so that I know all bases will be covered. Secondly, I will throw as many different combinations of commands, I/O redirections, etc. at the main loop to see that if the main shell program breaks, where, when, and how it breaks.

Christopher Richard Bilger
CIS 3329 - Network Architectures

Makefile/Documentation/Manuals

The makefile will compile all of the .c files into object (.o) files and then link all of those object files in the executable file, just like in the previous project. Overall documentation will include this file, the readme file and the manual files. The manual files will be a user-friendly guide to using the shell program and will provide feature listings and use cases.