1 Design

My shell implementation is named mysh. It is implemented as a shell loop which listens for lines delimited by the enter key. It parses lines from the user, separates them into space-delimited symbols, groups the symbol list into commands separated by redirection operators (—, ¿, ¿¿, ¡), and uses the fork and execve system calls to perform each command. The program sets STDIN and STDOUT of each child process to accomplish Linux redirection and piping.
The shell portion of the program sets signal handlers to ignore SIGINT and SIGQUIT system calls. It also listens for the ëxiẗkeyword, which prompts the program to exit. Every step of the main loop (iterating by while(1)) the program reads the current working directory, prints it, flushes STDOUT, then listens for user input. When the enter key is pressed, it reads everything from STDIN, lops off the final newline and sends the input to a symbol parser which separates the command into an array of individual symbols.
These symbols are then separated into subcommands. The first symbol in every subcommand is the name of the Linux program to run. The $PATH variable is searched through and the first path which contains a viable program file is passed to execve(), along with a formatted list of the rest of the options in the subcommand. If this subcommand is piping to another subcommand, being piped to by another subcommand, outputting to a file or getting input from a file, mysh sets its STDOUT and STDIN values temporarily to the necessary file to accomplish the pipe or redirection. It then returns the streams to their original file descriptors. Both of these operations are done via dup2().
Once execve has run all the subcommands in the command, control returns to the shell loop, which restarts.
This program does NOT implement the stream redirection operator x¿&y.

2 Worklog

Sunday July 19th 12:00pm - 4:00pm: Explore exec wrapper functions, build temporary program which executes argv input via execv and fork.
Monday July 20th 7:00am - 8:10am: Implement beginnings of subcommand parsing, supporting only redirection operators.
Monday July 20th 9:30am - 12:00pm: Iron out bugs in redirection operator code, begin on piping.
Monday July 20th 1:00pm - 6:30pm: Finish piping stuff.

Monday July 20th 6:30pm - 9:00pm: Get piping and all other redirection to work fairly seamlessly.
Monday July 20th 9:00pm - 10:00pm: Get shell portion of code to work. Correctly listen for user input and send lines to parser.
Monday July 20th 10:00pm - 11:15pm: Finishing touches. pwd on each new line. Swap out execv for execve very easily.
Monday July 20th 11:15pm: Work on worklog.

3 Challenges

This assignment was more open than the archive assignment, so it was challenging thinking of all the things a shell needed to do. It was challenging and fun thinking of the program's architecture and which functions would do which jobs. PATH searching and char** argument array helper functions were easy and very helpful. Overall, the most challenging part was planning and ironing out bugs in the piping and redirection code.

4 Questions

Point of assignment: To go over how shells work and how little they actually do on the system. To go over how to use execve() and its wrapper functions. To teach about how processes work on the system, and how forking can be used to make new processes.

Correctness testing: To test correctness I ran as many linux commands as I could think of. I ran them on both my machine and the os-class server. I tested different piping and redirection combinations.

Learn: I learned a lot about how shells work and what they do for the user. I also learned how processes are made. I learned how to manage my time and the importance of knowing how long different parts of a project will take.