**Operating systems write up 3**

This homework assignment was about creating my own shell that can implement simple commands such as ls, cat, head, etc., as well as implementing redirection, pipes, and signal handling for ^c.

The first thing I did for this was to make a parser that could separate a given line from its tokens. To do this I would keep on appending the inputs I got until the enter key was pressed. I made an array of chars where each group of chars without a white space had its own location. For example the group of char ‘hello’ would be location in foo[0]. Also in this function, the signal handler was located here. So whenever ^c was hit, it would catch the signal and kill the current process it was in without killing the entire process, which was the shell.

After receiving the input, it would go into the function called parser. This is where commands and tokens are separated. I used the function strcmp to see if any of the elements in the array were equal to the tokens. If they were the proper flags were set and it would be passed back to the main function.

In the main function, there were three states. One was for the pipe, one was for the redirect, and one was for a normal command. Also there was a condition to output the handler message.

If the condition was a normal command, then it’ll go to the run command function. In here it first looks to see if there’s an &. To do this properly I had to iterate through the arrays char by char so I had to use multidimensional arrays. If an & is found, then it sets the appropriate flag and takes out the & from the command. After this a new process is created and execvp is called in the child process, which executes the command with the given path.

If the condition was a pipe, then it goes to the run pipe command. In this function a pipe is initiated first. Then a new process is created. In the parent process, you close the unused end of the pipe, and execute the command. In the child process you close the unused end of the pipe, duplicate the read end into stdin and execute the command.

If the condition was a redirect, then you go to the function called run redirect. But before that happens the parser will have to parse through the entire statement to see which redirect symbol it is. Once that happens, the correct flags will be set. Inside the run redirect function, two pipes are firstly initiated. Then it checks to see if appending should happen when writing to a file. If so it opens the file with the mode append. Otherwise it opens the file with no append. Also it checks to see if we’re reading from a file or not. Then it creates a new process and checks the conditions to see which one to execute. After performing the appropriate functions for the specified tokens, it executes the command.

I was successfully able to implement all the features. Except for a few minor glitches. When I try to pipe, it would sometimes work, other times it would make it so I cannot see what I am typing into the command line, or sometimes it terminates my current shell. For example if I do ls –l | less, it might work, or it might terminate the TSH shell. Also I had the & working as an optional feature. To test the code, I just wrote to a file, read from a file, passed the output of a file to another program to test pipe, and I ran normal commands to see if it worked. Also to test the interrupt I enabled a while loop to keep incrementing a number after a certain command was typed. If I pressed ^c it would be able to exit out of that loop, without exiting out of the TSH shell.