Alexander Soe
asoe@ucsc.edu
1533280

Asg0 Writeup

Since my code was broken down into a couple of sections I think it is safe to say that I had a mixture of both unit testing and then whole system testing. My first test was simple as all I wanted to do was to get used to how the Linux commands worked as I was unfamiliar with them. I started testing with just one file and then multiple once I had that setup. Then afterwards I tried to test for errors so I would continue to try to "dog" multiple files with incorrect file names written in between to catch errors. Once I was able to print errors on incorrect file names as well as actual files, I started to try to implement "-". When I finally got "./dog -" to work correctly I would then do some whole system testing by interweaving "-" with either real file names or just random names not assigned to files to see if I could correctly get the write print statements for each condition. Similarly I tested directory inputs by first checking if "dog" could establish if a filename was a directory, as well as if multiple files were directories. I also made sure to check if I could interweave directory names with file names, non-file names, and "-". However, after figuring out directories, I started having trouble with my warn() not giving me the exact messages I wanted when a name was not a file. To test for this I did a whole lot of unit testing and whole system testing till I was safely able to remedy all the problems of the assignment.

The code handling the files involved reading a set amount of the file, writing it, then resetting the buffer to empty and reading/writing the next part of the file until read returned an "eof". On the other hand reading from standard input involved reading the user response and then printing. Standard input was run with an infinite while loop until the user entered an "eof" by pressing "ctrl-d". So with file handling, you read each part of the file starting from the

beginning and then writing when the buffer was full whereas with standard input, you need to read and write immediately to be ready for the next user input. I think the concept that encapsulates this the best is the idea of coherence and atomicity. Reading/writing from a file seems to be an example of coherence as the files are finite and each part of the file needs to be read and written from beginning to end. On the other hand reading from standard input is an example of atomicity as each thing read must be immediately written to keep up with the user's input as there is no end to the user's input until an "eof" is signaled.