Programming Assignment 2 Documentation

Basma Mahamid

CS 3377.0W1

Documentation of the Linux script and commands used to execute this project-

After the cpp source files of both programs are loaded into the directory im working in Linux. I used the command g++ followed by the name of the .cpp file I would like to turn into a linux executable file. I used a shortcut to change the name of this file from the default “a.out” to a more fitting name for the executable. I could have used “mv a.out filecnt” but instead in the same command line as converting the program to a linux executable file I added the option -o followed by the new name of the file. So my first two lines were:

g++ filecnt.cpp -o filecnt.cs2

g++ filedisp.cpp -o filedisp.cs2

(outside of the script I tested each of these executables individually without the pipeline in order to ensure they each work on their own)

Then to execute them I used the command “./” followed by the name of the first executable “filecnt.cs2” that searches and counts through the given directory. Followed by said directory as an argument which can be an absolute path or a relative path or any other way to access the directory normally in linux.

Then the pipeline is used followed by another “./” to execute the second file “filedisp.cs2” followed by the desired option to pipeline and filter through the standard input produced from the first file and display only the lines desired by the option given in the argument, “-d” to display the number of directories, “-f” to display the number of files, and “-b” to display the size of all the files in the directory in bytes. Like so:

./filecnt.cs2 <directory> | ./filedisp <option>

Afterwards, I also executed a few commands to execute the programs using relative paths in the directory argument to show this extra credit portion in action.

Documentation of filecnt.cpp-

This program accepts an absolute path of a directory as an argument and recursively traverses through this directory to count the total number of directories, files, and bytes occupied by the files.

I implemented multiple headers to complete this one of the most important being <dirent.h> which defines the data type DIR and the structure dirent amongst other necessary capabilities essentially to allow me to retrieve information about files and directories in Linux.

.

I added some debugging features throughout the code in order to catch where errors in my program may occur while I was writing it. I did this by implementing #define DEBUGGING 0. I can change the 0 to 1 when I want to catch these errors and can change it to 0 when I don’t want the debugging messages to show.

I then defined a struct dir\_info that declares variables for the count of directories, count, and file size. Then a prototype of the function filecnt with a pointer to the struct. The main program takes two arguments when ran in linux. The first argument is the program itself and the second is the directory I want to search through.

Lines 32-36 are in order to catch the error in the occasion that the program is executed with the incorrect arguments. Instead of crashing, it displays the error message that there are an invalid number of arguments as well as a usage message to tell the user how the command should be used ie. the executable followed by the name of the directory.

The next part is implementing the realpath() function which reads the path of the directory.

Then it outputs the amounts of each part needed to be counted along with which directory it was searching through by calling the first argument.

The main function utilized to count these variables is with the function filecnt with a struct that has a pointer to the directory path. It takes the argument- the directory path name and returns the info of that directory. In this, an array “newpath[]” is used to create a subdirectory path name.The directory is then opened and validation checker is implemented to where if the parent directory wasn’t able to be opened the first time it prints an error message. A while loop is implemented to go through and read all the files and sub-directories in the original parent directory recursively. I used if statements to check if the argument is not in the current directory or parent, then to see if it is a directory, which if it is would increment the count of directories by one. Then, it constructs a new path from the parent path.

In order to check the permissions of a directory and determine whether or not I have access to the directory and if it is accessible, I implemented an if statement with debugging features and used lstat() and this section of my code overall checks if the directory has execution privileges and if not, it will be skipped over instead of crashing the program. This was one of the major errors I ran into and took a while to solve. The program without this portion would work only with directories that all had executable permissions but not if there was a directory or subdirectory with some that did not have these permissions.

In lines 44-50 when reading the path of the directory I also implemented the extra credit portion which not only does it make it able to accept an absolute path to a directory, it also enables the user to give a relative path to a directory or any other form of input that would normally access a directory in linux such as ~ expansion, ‘.’ expansion, etc...

I then recursively called the function filecnt() so that once it goes through all the directories in the directory and counts them, if there is a subdirectory, it goes through the subdirectory the same way. It then does the same thing to count the files and for the file byte size.

It then closes the directory and returns the directories information with the counts of each of the variables for the output in the main function.

Documentation of filedisp.cpp-

This program takes two arguments, one being the program itself and the other being the option the user selects to specify what they want to find in the directory.

A validation statement is implemented to ensure that the command is executed properly in linux with the correct number of arguments and is followed by a usage statement to help the user write the command the proper way.

Then a while loop is implemented to loop through each line of the standard input to output the lines that contain a specific word according to the option given by the user that is sent to the argument of the program. I.e. Searches for “files” if “-f” if the option, “directories” if “-d” was the option, and “bytes” if “-b” was the option given.

A minor issue I encountered and found the solution to while programming was when the loop searched the standard input line by line for the lines containing the word “files” it also printed the line that displays the size of the files because that line also contains the word “files” in it. I fixed this by adding another condition in the if statement that only outputs the line with the number of files if it does not contain the word “bytes” and if it does it will skip over this line when asked for the line that displays the number of files only.