ELEC 377 Lab 4: Description of the Program

Name & Student ID: Shiyan Boxer (20106887) and Arsh Kochhar (20104779)
Date: Nov 16th, 2020

Overview of the Program

This program uses two shell scripts to analyze files. The first script extracts formatted comments from C source files. The second script does an analysis of the source code written in the previous labs.

Our Solution

Script 1

Script 1 extracts formatted comments from C source files. It does this by starting in the current directory and recursively searching for files that end in .c or .h. For each file it prints a bar equals sign and extracts all the blocks of line comments that start with a //+ and end with //-.

The script uses a find statement to find all files that end in .c or .h. The script adds bar equal signs by adding an echo statement before the echo $file command.

```
for file in `find . -name *.c -o -name *.h`
do
    echo
"================================================================
================================"
    sed -e '/\/\/+/,/\/\/-/p' -e 'd' $file | sed -e
's/\/\/[+-]*//' #removing comments and blocks of comments
done
```

Then, the script extracts the block of line comments between the //+ and //- by using the sed p command and setting the start pattern to //+ and end pattern to //-.

Lastly, it removes the leading comment delimiter in a separate sed command since the selected statements were extracted using the p command. It's added using a pipe to pipe output to another sed command that uses a single -e argument that removes the // followed by a + or -.

Script 2

The second script takes a path as an argument. It recursively finds all of the .c files and produces the following information:

1. For each source file that contains a main function, it displays the full path of the file and appends the number occurrences of printf and of fprintf as separate numbers.
2. For each module file, it lists the full path of the file and appends the number of lines that contain printk. If no main file is found, the script displays “no main files..”; if no module file, it displays “no module files...”.

First, it checks if the number of arguments is correct. The variable $# should contain 1 argument after the command line, if not then it prints an error message, The $0 was used in the error message for the name of the script.

```
if [ $# -ne 1]; then
    echo "Usage: $0 ProgStats"
```

Then it checks that the argument is in a directory by using an if-else statement. If the first argument is a directory, it runs the actions of the script. Else, it produces an error message.

```
if [ -d "$1"]; then #if the arg is a directory
.
.
.
else
    echo "The first arg is not a directory"
    exit $1
```

Then, it finds the main files using a for loop and a find statement. Within the for loop is an if-else statement that uses grep as a condition that looks for int main and echo the name of the file. A variable is used to indicate that at least one main file was found. After the loop, it checks the variable to see if any main files were found, and if not, print a message.

Count the number of lines that contain a pattern using the command grep -c. The output is assigned to 2 variables, one for printf, one for fprintf. These variables are added to the echo and use awk to format the output.

```
COUNTPRINTF=$(grep -wc "printf" $file -s) #searches for printf,
-wc used to count its occurrences
COUNTFPRINTF=$(grep -wc "fprintf" $file -s) #searches for
printf, -wc used to count its occurrences
```

Find the modules files using the same loop as finding the main files but instead it checks for the function init_module. Lastly, the script lines numbers for printk using the command grep -n which will print the line numbers in front of lines that match and the tr command to change the spaces to comma.

```
PRINTKLN=$(grep -wn "printk" $file -s | sed -e 's/:.*$//')
```