# CSC 412: Operating Systems

**Assignment 4 (25 points, due 5/5)**

Complete the assignment below. Make sure to include appropriate comments and error testing in your code.

1. (12 points) Create an ASCII data file called **test** containing several lines of arbitrary text. Type the UNIX command: **wc test** The program reports the number of lines, words and characters/bytes in the file, followed by its name. Write another program **cw** which produces an identical output. The program **cw** should use only UNIX system calls to open and read the file. But, to simplify programming, use **printf** to output the counts, the filename, messages, etc., on screen. Assume that a word is any sequence of characters that does not include a blank or a newline character.

Produce a **script** file containing the source code and several executions. Be sure to include test situations where a file to be processed could not be opened and where no command line argument is provided. If no command line argument is provided, the program reads from standard input. Include a test case in which the very first word in the file is preceded by blanks. Also include a test case where the very last word is not followed by a blank or a newline character.

For each test case, first show the contents of the file using **newcat**, then show the output of **ls -l**, **wc** and **cw** on the file. Here is an example of what is expected (If your **newcat** program does not work, you may use **oldcat** in its place):

[vanand@holly2]% newcat junk

Hello,

This is a test file

I have blank lines above and below.

[vanand@holly2]% ls -l junk

-rw------- 1 vanand faculty 65 Jan 15 17:07 junk

[vanand@holly2]% wc junk

5 13 65 junk

[vanand@holly2]% cw junk

5 13 65 junk

Note that **wc** and **cw** have identical output. In addition, the character count reported by **wc** and **cw** matches the file size.

2. (13 points) For this part you have to write a C program called **tinyshell** that behaves much like a shell; but your program has to accept and respond to the following four commands only, presented one at a time:

**world n** run the command called **world** and pass the argument ***n*** to it

**mars n** run the command called **mars** and pass the argument ***n*** to it

**newcat file** run the **newcat** program to display the **file** on standard output

**quit** terminate execution of the **tinyshell**

Thus, in addition to the **newcat** program constructed before, you should construct two small C programs, compile and set up two executable files for the two commands **world** and **mars**. Both commands simply print out messages on the screen:

**world n** print the message **hello world** on the screen ***n*** times

**mars n** print the message **HELLO MARS** on the screen ***n*** times

(If your **newcat** program does not work, you may use **oldcat** in its place.)

The **tinyshell** program works as follows: puts out a prompt **next command>** and then reads the command string typed by the user (use **scanf**). If the command typed by the user is not one of the above four, it responds with an error message and prompts the user again. If the command typed is **quit** the program is terminated. Otherwise, it reads, as a string, the remaining part of the command (i.e., the count ***n*** or the file name). Then it executes the command **world** or **mars** or **newcat** as required.

For this, **tinyshell** forks and the parent process immediately executes **wait**. The child process executes **execlp** to overlay itself with the code of **world** or **mars** or **newcat**. On its completion, the parent process resumes execution putting out a new prompt for the next command. Construct a complete **C** program to accomplish the above task, compile and test it. Create a **script** file that shows all source code files and a few typical runs of the program.