csci-e28
Assignment 3: stty-lite

Introduction

For this assignment you will write a program that implements a subset of the Unix **stty** command. In doing so, you will work with the tcgetattr/setattr() calls and learn about some of the settings of terminal devices.

The Terminal Driver

A computer terminal is a machine with a display unit and some input devices. The display unit is usually a video screen but could be a voice synthesizer or braille output device. A terminal usually has some sort of keyboard. On a Unix system, these terminal devices appear as files in the directory tree, but they are not disk files; they are device files.

When a user types, electrical signals from the keyboard are received by software in the kernel, and that software transfers the keystroke data to programs that read input from that terminal. But that software can do more than simply pass bits between the device and processes. For example, people sometimes make typing mistakes. These people would like to be able to press an `erase´ key to correct the errors before the program receives the input. The sofware sitting between the device and the process can help by providing additional processing.

The software that transfers data and provides additional processing is called a terminal driver. Terminal drivers usually accept keystrokes and assemble them into a line of text. When the user presses the `erase´ key, a character is removed from the line. When the user presses the return (or Enter) key, the terminal driver sends the line to the program. The program does not have to worry about the editing commands. Unless it wants to.

For another example consider the passwd program. This program prompts a user for a new password. It disables the usual echoing of characters. It does so by telling the tty driver not to echo characters.

The tty driver, then, is a piece of software. It is part of the kernel. The tty driver has for each terminal a collection of settings. These settings include what the erase character is, whether to echo characters, whether to wait for the Enter key or whether to pass characters along as soon as they show up.

The stty command

The stty command lets you see what the current settings are, and it lets you change the tty settings. Try it. With no arguments, stty lists the most popular tty settings. With the argument `-a´ stty lists a lot of info.

To change settings, call stty with the name of a setting. For example, to enable echoing, type: *stty echo* . Typing *stty -echo* turns off echoing. Read the manual page for a list of available options. You can set several options on one line as in

```
stty -echo onlcr erase ^X
```

which turns off echo, adds a carriage return with each newline, and sets the erase character to Ctrl-X.

Read the manual entry on stty and experiment with setting the erase character and the modes to things like -echo or -icrnl. When you try these experiments, you must use the standard Unix shell, *sh*, not your usual login shell. On nice, type *sh* to run this shell, and type *exit* to return to your usual shell. You might need to press Ctrl-J if you disable carriage return mapping. And you might have to type logout or exit if Ctrl-D seems to stop working.

How stty works

stty is an interface to the tcgetattr() and tcsetattr() functions that, respectively, gets and sets driver settings associated with standard input. When you call stty with no arguments, stty gets the current settings and

displays those settings in a form readable by people.

When you call stty with a setting, such as -icrnl, it converts that string into the appropriate bit in the status word and sets the tty driver. The convention is that a word like *canon* enables a mode, while the word with a leading dash, as in *-icanon* disables that mode.

The Assignment

Write a version of stty, called sttyl, that supports the following options:

1. *display settings* if sttyl is invoked with no arguments, it prints something more or less like the following. It does not have to be exact, it just has to include the speed, the intr, erase, and kill chars, the number of rows and cols, the settings for `icrnl, hupcl, echo, echoe, opost, icanon,` and `isig` , and any other ones you wish to add.

```
speed 9600 baud; rows 63; columns 80;
intr = ^C; erase = ^H; kill = ^U; start = ^Q;
stop = ^S; werase = ^W
-parenb hupcl cread
brkint -inpck icrnl -ixany onlcr
iexten echo -echoe -echok
```

2. *set erase and kill* sttyl should accept the arguments `erase´ and `kill´ . In each case, the argument after the word `erase´ or `kill´ is the character to make the erase or kill character. Try the real stty to see how this works. The standard version of stty also accepts control characters as the two-char combination of `^´ and a letter. Your program is not required to handle that.

3. *set other attributes* sttyl should accept and correctly handle the words: `icrnl, hupcl, echo, echoe, opost, icanon,` and `isig` . It should handle words with or without a leading dash. A leading dash turns off that setting, while no leading dash turns on that setting. Be careful about getting stuck in -icanon mode.

4. *multiple arguments* sttyl must accept multiple settings on the command line. See the example in the section `The stty commmand´ above for an example.

5. *error handling* sttyl should print the `invalid argument´ for arguments that it does not know.

6. *table driven* It is possible to write this program with a large number of *if(strcmp(..)) else if (strcmp(..))* blocks. Using that approach makes the code very long and tedious to update. Ten points of your score go to a table-driven solution. See the showtty.c program from lecture for a starting model.

7. *Clean Compile* Compile your program with `gcc -Wall` and make sure you correct every warning and error it reports. Many small bugs can be avoided by heeding these warnings.

Important: test your program using **sh**.

Overview: sttyl has to get the current settings for stdin, and if there are no args, then print the current settings. If there are args, it has to step through through the command line arguments and act on each string. When it has processed all the command line arguments, it has to write the new settings back to the driver.

Programming Note: Do not define tables (or any variable) in .h files

Define your table or tables in one of your C files. You could even create a file called tables.c if you like. Then, put the declarations for any tables in the files that refer to those tables *or* put the declarations in a header file. If you have any questions about the correct use of .h files, please ask.

What to Turn in

As usual, turn in (a) fully documented, easy-to-read source listing, (b) a sample run, and (c) a script of a clean compile, (d) a Makefile, (e) a Software Plan. Your sample run must include a run of the official test script. Run this by typing `~lib215/hw/stty/test.stty` Submit your work electronically: `~lib215/handin sttyl`