Programming Project: stty-lite

Introduction

For this assignment you will write a program that works sort of like the Unix **stty** command. In doing so, you will have a chance to work with the tcgetattr/setattr() calls and learn about some of the attributes of terminal devices.

The Terminal Driver

The terminal input/output devices like crt terminals, printers, and modems appear as files in the Unix directory tree, but they are really terminals. As such, they must have attributes that make them useful to people. 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.

This 'erase' key example introduces the idea of a tty driver. The driver is a piece of software that accepts keystrokes from the terminal and assembles them into a line. When the user presses the 'erase' key, the character is removed from the line. When the user is done, he or she presses the return key, and the terminal driver sends the line off to the program. The program does not have to worry about the editing commands. Unless it want 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 stores for each terminal a set of attributes. These attributes include what the erase character is, whether to echo characters, whether to wait for a newline 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, it lists the most popular tty settings. With the argument '-a' it lists a lot of info.

To change settings, you call stty with the name of a setting. For example, to enable echoing, you 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 olcuc erase ^X
```

which turns off echo, outputs lowercase letters in uppercase, 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. 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 if Ctrl-D seems to stop working.

How stty works

stty is an interface to the tcgetattr() and tcsetattr() call that gets and sets the driver settings associated with standard input. When you call stty with no arguments, it gets the current settings and displays them 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 `icanon` 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 intr, erase, and kill chars, and about eight other settings.

```
speed 9600 baud; evenp hupcl cread
intr = ^C; erase = ^H; kill = ^U; start = ^Q; stop = ^S;
brkint -inpck icrnl -ixany onlcr tabs
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 character can also be expressed as a "^" and a letter.

3. *set other attributes* sttyl should accept and correctly handle the words: `icrnl, onlcr, echo, echoe, olcuc, tabs, icanon,` and `isig` . It should handle these with or without a leading dash. A leading dash turns off that attribute, while no leading dash turns it on. 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 out the message 'unknown mode' for arguments that it does not know about.

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 in the text 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.

What to Turn in

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 `~COURSE/hw/stty/test.stty`