COS 350: Program #4: scroll [file]

Objectives: Working with terminal modes, timers and signals.

You will write a program very similar to *more*, except that your program will allow slow continuous scrolling. Your program will work in text mode within a window.

Your program should display the file specified on the command line, or read from *stdin* if no file is specified. When reading from stdin it is expected that the input has been redirected from a file or piped from another program.

It will start out, just like *more*, by displaying the first page of the file with a prompt on the bottom line. Pressing **space** will display the next screenful. However pressing **Enter** will start scrolling forward one line at a time at a regular interval (initial value of 2 seconds). Pressing **Enter** again will toggle scrolling off.

Here is the complete list of keys your program should respond to:

key	**action**
space	move forward 1 screenful immediately and stop scrolling
Enter	toggle scrolling on/off
f	scroll faster by 20%
s	scroll slower by 20%
q	quit

The goal of this assignment is to use timers and signal handlers. Don't structure your code like the book's "more" with the primary loop on reading the input file. That would get too complicated when dealing with lines that wrap. Instead, I suggest reading the entire file into a buffer when your program starts, and then whenever you need to print, just get the next output line(s) from the buffer. Your main loop should just wait for and process the keyboard input. When scrolling, your timer handler will be doing the work: erasing the old prompt, drawing the next line, and then redrawing the prompt. This will feel very unusual to write a program where you are not completely controlling the order of execution.

Requirements:

- Create a Makefile for your program. This is now a required part of the submission. Create this when you start writing your program so that you can use it throughout your development.
- Determine and use the window size at start up. Get the window size using:
  ioctl(1, TIOCGWINSZ, &winsizestruct);
  You do not need to adjust if the window size is changed while you are running.
- Use tcgetattr(), and tcsetattr() to set the terminal behaviors you want.
- You should have a prompt on the bottom line in reverse video. During scrolling this prompt should show some indication of the current scrolling speed.

  A bit of information about terminals is useful for creating this prompt. Special control sequences are used to control the terminal. Today's terminal emulators such as putty and

(OVER)

xterm are based on a very common terminal called the VT100. Here are a few of its control sequences, which you can use:

- `"\033[7m"` – turn on reverse video (note `\033` is the ESCape character)
- `"\033[1m"` – turn on bold (may appear as colored)
- `"\033[4m"` – turn on underlining
- `"\033[0m"` – turn off any special character formatting
- `"\033[0K"` – clear to the end of line (note capital K)
- `"\b"` – backspace moves the cursor 1 position to the left (does not erase)
- `"\n"` – Newline character moves down 1 line and to column 0. On the last line it causes the screen to scroll.

Thus to print your prompt: print the control sequence to turn reverse video on, print your prompt. And then print the sequence that turns off special character formatting.

To erase your prompt: print the correct number of backspace characters and then print the control sequence to clear to the end of the line.

- Long lines in the input file should wrap to multiple lines on the screen and should scroll one screen line per time step. (E.g. long line should not all appear in one chunk.)
- Tab characters can be output as tab characters or as the appropriate number of spaces, but you will need to think carefully about them when dealing with long lines.
- When the user exits your program, either with "q" or with Ctrl-C, the terminal should be reset to its original mode. (E.g. don't leave echoing off.)
- At the end of the file the prompt should indicate you have reached the end of the file, but your program should not exit until the users types 'q'.
- If the text to be scrolled is from stdin, you will need to open /dev/tty explicitly to gain access to the keyboard. Use open() on the keyboard device, and then use read() to read 1 char at a time from the keyboard. Do this after your code is otherwise fully debugged because this will cause confusion when using the debugger.
- Create a document describing your program and how to use it. Write this document on a word processor of your choice and organize and format it in the style of a man page. For an example see man more. Use your own words. Do not copy the wording from this assignment.

Tips & Suggestions:

- There are test files in the course directory named: **test.txt** and **extracredit.txt**
- Build and test this as you go. Write little parts that you can test.
- Keep it simple! You may put reasonable fixed limits on the sizes of lines and files.
- You will want to include the header files: termios.h, fcntl.h, signal.h, string.h, sys/time.h, sys/ioctl.h
- The normal behavior of the buffered I/O commands like printf() and putchar() is to buffer data in memory until a newline character is sent. Use fflush(stdout) to force any characters that were printed to be sent to the window.
- If you kill your program, your terminal may be left in a funny state. Use the Unix commands clear or reset to fix the terminal.
- Debugging a program that is explicitly manipulating the terminal is difficult. First you will need to tell the debugger to direct your program's output to another window. Then as you

are debugging you will need to switch your keyboard focus back and forth as you are entering commands either to the debugger or to your program.
  - For ddd, after starting ddd but before running your program click:
    view / Execution Window
  - For gdb you need to start an extra terminal that you will hijack for your output
    - In the extra terminal use the tty command to get the device. e.g. /dev/pts/2
    - Next type "sleep 10000" in that terminal so the shell using it quits paying attention
    - After starting gdb but before running your program tell gdb which terminal to use for output with a gdb command like: tty /dev/pts/2
- To save time you might choose to skip some non-critical cases that complicate the design such as wrapping long lines and tab characters. Look at the grading sheet to see how much these are worth.
- Your program might ignore some options, or display some cases incorrectly, however it should never crash.
- In the past some students have output more backspace characters than they needed to remove their prompt. This works on some terminal emulators where the extra backspaces at the start of a line are ignored, but on others (such as putty) these extra backspaces wrap to the previous line and cause problems with the program's output.

Extra Credit: 10 points

When you use *man* to examine man pages, these are displayed by *less* using underlining and bold (colored) text. The way they did this was interesting. Various terminals provide support for display attributes such as underlining and bold through extra escape sequences, but these escape sequences cannot be put into a text file because they are inconsistent across terminal types. Instead, in a text file they put sequences using backspace characters (\b) to represent typing over a character.

b \b b o \b o l \b l d \b d would be: **bold**
_ \b u _ \b n _ \b d _ \b e _ \b r _ \b l _ \b i _ \b n _ \b e would be: <u>underline</u>

less then interprets these sequences and turns on those attributes when displaying the characters.

For extra credit, implement this capability in your scroll program. Use those extra control sequences I told you about.

What to turn in:

Written report:

1. Your manual page
2. Please test your programs thoroughly and explain anything that is not working correctly. You will get more partial credit if you understand and explain that something isn't working than if you haven't even tested it enough to discover any flaws.
3. Your code
4. Your Makefile

(There is no example execution sequence for this assignment because it has dynamic behavior that can't be captured by typescript.)

Electronic submission:

- Before you submit your program, compile and test it on a **linux** machine in the lab.
- From a machine in the lab run the program "submit" to submit your files.
- Submit your source code (**scroll.c**) and any other source files, a **Makefile**, and your executable (**scroll**) to a directory named: **prog4**
- **Do not in any way combine, compress, zip, or tar your files!**

Grading:

10 points: man page
10 points: Initial screenful
10 points: Prompt
10 points: space key (page forward)
20 points: slow scrolling (Enter key: on/off)
10 points: f & s keys (faster/slower)
5 points: Line wrapping
5 points: Line wrapping with Tabs
5 points: q key (quit)
5 points: also accepts file from stdin
5 points: Makefile
5 points: Error Handling
10 points: Extra credit – underlining and bold

	Pts	Works	Mostly Works (explain)	Occasionally Works (explain)	Not implemented	Explanation
man page	20					
Initial screenful	5					
Prompt	10					
space key (page forward)	5					
slow scrolling (Enter key: on/off)	20					
f & s keys (faster/slower)	10					
Line wrapping	5					
Tabs handled correctly	5					
q key (quit)	5					
Also accepts file from stdin	5					
Makefile	5					
Error Handling	5					
Extra credit	10					