Framebuffer Graphics

ECE598: Advanced Operating Systems – Homework 9

Spring 2016

Due: Thursday, 12 April 2018, 2:00pm

This homework involves writing to the Pi's graphical framebuffer.
You may work in groups for this assignment.

1. **Download the homework code template**
   - Download the code from:
     `http://web.eece.maine.edu/~vweaver/classes/ece598/ece598_hw9_code.tar.gz`
   - Uncompress the code. On Linux or Mac you can just
     `tar -xzvf ece598_hw9_code.tar.gz`
2. **You will need an HDMI connector and monitor**
   - To test the graphics output of the homework, you will need to connect a monitor via an HDMI cable.
   - If you have trouble getting access to an HDMI monitor let me know.
   - Note: we still do not have keyboard support, so you will still need to have a serial connection to your pi to enter keypresses.
3. **Provided Code Changes**
   - The code that talks to the GPU is in `kernel/drivers/firmware/mailbox.c`
   - The code that implements low-level framebuffer drawing is in `kernel/drivers/framebuffer/framebuffer.c`
   - The code that implements the text console is in `kernel/drivers/framebuffer/framebuffer_console.c`
   - The default font used is `kernel/driver/framebuffer/c_font.h`
   - The `kernel/drivers/console/console_io.c` code has been modified to not only print output to the serial port, but also to send it to `framebuffer_console_write()` as well.
4. **Implement font printing. (4pts)**
   - Implement the `framebuffer_console_putchar()` function in `kernel/drivers/framebuffer/framebuffer_console.c`.
   - You may use the provided `framebuffer_putpixel(color,x,y)` routine to set pixels.
   - Remember from the lecture notes, the font is just a series of bytes, with each byte representing an 8-bit bitmap of whether a pixel is on or not.

     This will require some C array manipulation.
     So if the character being passed in is 'A' then you will find the beginning of the character's

bitmap in the `default_font` array found in `c_font.h`. So `default_font[65][0]` has the first line (there are y-height lines, for the provided font the characters are 16 lines high). Each byte you will need to break out into 8 bits:

```
0xa1 = 1010 0001 = "* *    *"
```

so you will have to have a loop (probably using shifts and masks) to break out the bits and then putpixels or not depending if the bit is set.

- The putpixel routine takes color, x, and y arguments (find where it is implemented to get the calling parameters). So you won't have to break down the colors or do the math to poke the framebuffer directly, putpixel will do it for you.
- Once it is working, when you boot the Pi while connected to the HDMI monitor it should display the boot messages to the screen.
- If you have trouble getting the pointer math working out, a good quick test is to just draw all pixels as being on in the text rectangle, and boot with that, and see if boxes appear on the screen at boot.
- If you're feeling advanced, instead of just skipping pixels that are 0, instead draw them in with "back_color" (the background color)

5. **Draw a vertical gradient in a color of your choice on the screen (4pts)**

   - You will put your code in `framebuffer_gradient()` in the `kernel/drivers/framebuffer/framebuffer.c` file.
   - You can use the existing `framebuffer_vline()` function to make this easier.
   - Set the color to something bright and then have a loop that goes across each Y value, drawing a line the full height of the screen. Decrement the RGB values each step. By default the screen is set to 800x600x24bit as a resolution.
   - To test things, you can use the "gradient" command at the command line. This is implemented in the shell by a syscall that calls `framebuffer_gradient()`

6. **Something Cool (1pt)**

   Do something cool with your homework. Below are just some suggestions for things you can do.

   - Add a command that draws a horizontal gradient.
   - Edit one of the font characters in `c_font.h` to look different. Mention which one you changed.
   - Add a command that will change the font being displayed. These fonts being used are old-fashioned "VGA console fonts" and are nearly impossible to google anymore. I've included two (`medieval_font.h` and `marie_font.h`) if you want to try this.
   - The default font has the IBM extended ASCII/ANSI characters. Use these to display a color ANSI art picture on the screen.
   - Add code that draws some sort of (tasteful) bitmap image to the screen.

7. **Questions (1pt)**

   (a) The framebuffer data structure is declared like this.

   ```
   struct frame_buffer_info_type fb_info  __attribute__ ((aligned(16)));
   ```

   What does the aligned attribute do, and why is it necessary?

   (b) You may have noticed that with the framebuffer console enabled, the console has become very slow. Why might that be? How could you speed it up?

8. **Submit your work**

   - Run `make submit` in your code directory and it should make a file called `hw9_submit.tar.gz`. E-mail that file to me as well as the document with the answers to the questions.