# Framebuffer Graphics

**ECE531: Advanced Operating Systems – Homework 8**

Fall 2025

**Due: Friday, 5 December 2025, 5:00pm**

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:
     `https://web.eece.maine.edu/~vweaver/classes/ece531/ece531_hw8_code.tar.gz`
   - Uncompress the code. On Linux or Mac you can just
     `tar -xzvf ece531_hw8_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 New Code**

   - 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. **Some notes**

   - The provided code does not implement character output yet. So when you hook up your HDMI it will just be printing white blocks instead of letters.
   - The printing is probably much, much slower than it was w/o graphics. This is because we use an inefficient console that redraws the whole thing after each write() system call.
   - To speed things up you can edit `framebuffer.c` and in the `framebuffer_push()` function change the `memcpy()` call to `memcpy_4byte()` which is a slightly more optimized memory copy routine (both can be found in `kernel/lib/memcpy.c`. We have also enabled L2 cache which helps a bit, hopefully that isn't a mistake.

- The `ll` utility (two lowercase Ls) might be useful when testing things, it prints some colored ASCII art to the screen.

5. **Draw a vertical gradient in a color of your choice on the screen (3pts)**

   - 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.
   - By default the screen is set to 640x480x24bit as a resolution.
   - Set the color to something bright (remember, color is a 24-bit RGB value. 0x00ffffff is white, 0x00000000 is black. 0x000000ff is bright blue).
   - 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()`
   - This should draw a solid color to the screen.
   - To make it a gradient, for each X step decrement your color to make a gradient effect.

6. **Implement font printing. (3pts)**

   - 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.
   - Draw the pixels where the font is 1 fore_color and where the font is 0 back_color. To test color support you can try running the "ll" (that's two lowercase Ls) program.

7. **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.
- See if you can write a faster memcpy routine.

8. **Questions (3pt)**

(a) On a Linux ext2 filesystem, is the filename stored in the inode? Explain why or why not.

(b) The ext2 filesystem inode contains 15 block pointers. The first 12 (0-11) point directly to disk blocks. Entry 12 points to an indirect block which contains BLOCKSIZE/4 (divided by 4 because the pointer size is 32-bits) block pointers. Entry 13 points to a double indirect block where each pointer points to an indirect block. And finally, entry 14 points to a triple indirect block. (See the diagram in Figure 1 which may or may not help).

i. Assuming a blocksize of 1kB, what is the maximum file size supported without using indirect blocks?

ii. What is the maximum file size if additionally all the blocks from the first indirect block are used?

iii. What is the maximum file size if additionally the second indirect blocks are used?

iv. What is the maximum size of a file if all possible blocks are used (including the triple indirect)?

v. For the previous answer involving the maximum size, what is the overhead involved (i.e. how much space is spent holding all of the indirect blocks)?

(c) The ext2 and fat filesystems are very different.

i. List one use case where an ext2 filesystem works better than fat.

ii. List one use case where a fat filesystem works better than ext2.

(d) The file `new_file` is shown via the `ls` command to be 41 Megabytes in size

```
rasp-pi:~% ls -lh new_file
-rw-r--r-- 1 vince weaver 41M Mar 30 21:34 new_file
```

But the command `du -h` which shows how many disk blocks a file uses only shows 16k being used.

```
rasp-pi:~% du -h new_file
16K     new_file
```

How is this possible? Why might this be a useful feature to have?

(e) Pick a filesystem supported by Linux (that isn't ext2/3/4, btrfs, or fat) and do some brief research on it. Write a few sentences describing where the filesystem originated, its strengths and weaknesses, and why there's a Linux driver for it. You can find a list of Linux filesystems under the `fs` subdirectory of a Linux source tree, which you can find online here: `http://lxr.free-electrons.com/source/fs/`.

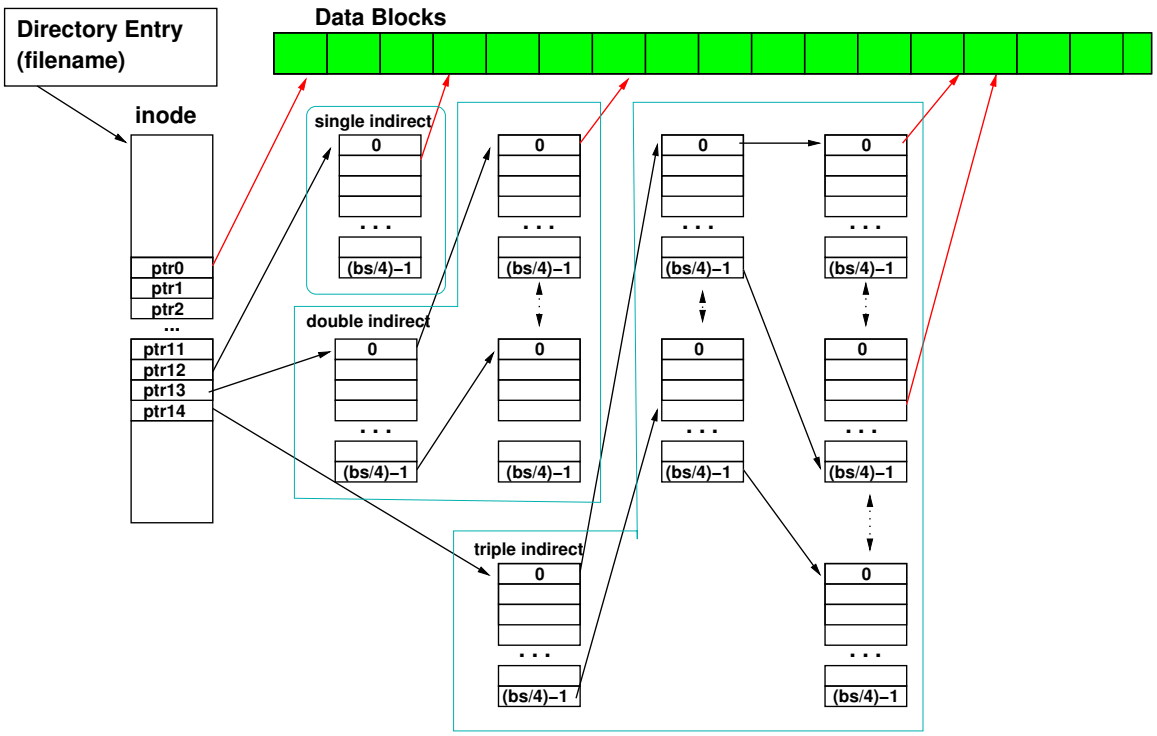


Figure 1: ext2 inode Diagram

## 9. **Submit your work**

- Run `make submit` in your code directory and it should make a file called `hw8_submit.tar.gz`. E-mail that file to me as well as the document with the answers to the questions.