Pseudo-random ramblings about programming and other geeky stuff

Sunday, 20 January 2013

Low-level Graphics on Raspberry Pi [part two]

In the part one we looked at how to get hold of the framebuffer and output some basic information about the display.

Not exactly very impressive, but a good start. Now let's actually draw something!

The framebuffer does not provide any functions/methods for drawing - instead, it just gives access to the 'raw' bytes of the buffer. Basically one could just use the standard file redirect '>' to output into the framebuffer:

```
some-bytes-from-somewhere > /dev/fb0
```

Obviously this is not very usable, but redirect the other way provides a quick-and-dirty raw screencapture that might come in handy:

```
cat /dev/fb0 > screenshot.raw
```

Proper way to access the buffer, is to memory map the file to a section of RAM using the mmap function.

Here we extend the example from part one - after opening the framebuffer device, we get also the fixed display information (to easily get the buffer size), map the file to the user accessible memory and finally draw something just setting the bytes in the buffer to some values:

```
#include <unistd.h>
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <fcntl.h>
#include <linux/fb.h>
#include <sys/mman.h>


int main(int argc, char* argv[])
{
  int fbfd = 0;
  struct fb_var_screeninfo vinfo;
  struct fb_fix_screeninfo finfo;
  long int screensize = 0;
  char *fbp = 0;

  // Open the file for reading and writing
  fbfd = open("/dev/fb0", O_RDWR);
  if (!fbfd) {
    printf("Error: cannot open framebuffer device.\n");
    return(1);
```

Blog Archive

- ► 2016 (6)
- ► 2015 (3)
- ► 2014 (9)
- ▼ 2013 (9)
  - ► April (2)
  - ► March (4)
  - ► February (1)
  - ▼ January (2)
    - Low-level Graphics on Raspberry Pi (part three)
    - Low-level Graphics on Raspberry Pi (part two)
- ► 2012 (2)

Code Repository

- Low-level Graphics on RPi

Discussion

- Low-level Graphics on RPi
- Python Programming on RPi
- Java Programming on RPi

Links

- Raspberry Pi
- Python

```
  }
  printf("The framebuffer device was opened successfully.\n");

  // Get fixed screen information
  if (ioctl(fbfd, FBIOGET_FSCREENINFO, &finfo)) {
    printf("Error reading fixed information.\n");
  }

  // Get variable screen information
  if (ioctl(fbfd, FBIOGET_VSCREENINFO, &vinfo)) {
    printf("Error reading variable information.\n");
  }
  printf("%dx%d, %d bpp\n", vinfo.xres, vinfo.yres,
         vinfo.bits_per_pixel );

  // map framebuffer to user memory
  screensize = finfo.smem_len;

  fbp = (char*)mmap(0,
                    screensize,
                    PROT_READ | PROT_WRITE,
                    MAP_SHARED,
                    fbfd, 0);

  if ((int)fbp == -1) {
    printf("Failed to mmap.\n");
  }
  else {
    // draw...
    // just fill upper half of the screen with something
    memset(fbp, 0xff, screensize/2);
    // and lower half with something else
    memset(fbp + screensize/2, 0x18, screensize/2);
  }

  // cleanup
  munmap(fbp, screensize);
  close(fbfd);
  return 0;
}
```

Save the above code to a file called **fbtest2.c** - then compile and link:

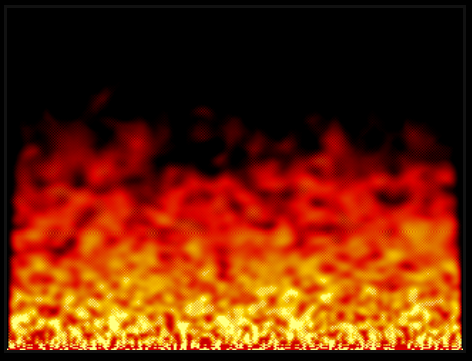
```
make fbtest2
```

Run the executable:

```
./fbtest2
```

...you should see the upper half of the display turn white and the lower blue! Hmm, except if you are in some other display mode than 16 bpp, you might get a different result...

With a bit of imaginative manipulation of what byte values to put into the buffer, one could draw more complex images - for example something like this:



...I will be getting into 'pixel plotting' in the following parts - hopefully soon...

[Continues in part three]

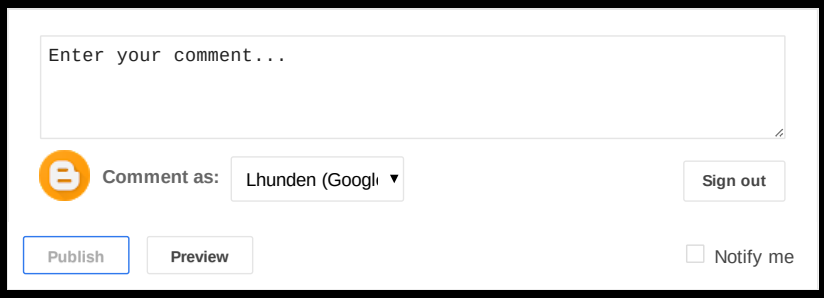
Posted by Unknown at 17:21

Labels: C, graphics, Linux, Raspberry Pi

No comments:

Post a Comment

Note: only a member of this blog may post a comment.



Newer Post Home Older Post

Subscribe to: Post Comments (Atom)