ECE 531 – Advanced Operating Systems
Lecture 27

Vince Weaver

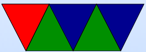
https://web.eece.maine.edu/~vweaver

vincent.weaver@maine.edu

10 November 2025

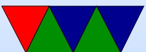
Announcements

- Homework #7 will be posted eventually
  ran into problems with the old code



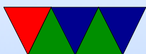
Raspberry Pi Framebuffer Interface

- You can request that the GPU firmware give you some memory to treat as a framebuffer
- You request this via the mailbox interface
- The GPU takes care of taking this framebuffer and doing whatever behind the scenes action is needed to get it to come out the HDMI port

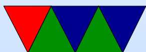


Raspberry Pi Framebuffer – Request via Mailbox

- ```
  struct frame_buffer_info_type {
          int phys_x,phys_y;       /* IN: Physical Width / Height*/
          int virt_x,virt_y;       /* IN: Virtual Width / Height */
          int pitch;               /* OUT: bytes per row */
          int depth;               /* IN: bits per pixel */
          int x,y;                 /* IN: offset to skip when copying fb */
          int pointer;             /* OUT: pointer to the framebuffer */
          int size;                /* OUT: size of the framebuffer */
  };
  ```
- Inputs
  - Specify the size you want. Typically something like 640x480 or 800x600 or even larger
    Set this in both the "phys" and "virt" x and y


  ```

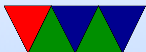
parameters. Virt lets you set a height/width bigger than the screen (so you can pan in?) We won't use that
  - Specify the bit depth you want (in bits per pixel). We're going with 24.
  - X and Y let you skip into the image? (lookup what they are for) leave them at 0 for now
- Submitting
  - Make sure the struct is aligned on 16-byte boundary
  - Need to bitwise-or the top bits of the address to the non-cacheable area of GPU RAM. On a Pi1B this is



0x4000.0000, on a Pi2/Pi3 it is 0xC000.0000

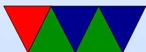
- Then do the mailbox transaction

- Outputs
  - Pitch – how many bytes between each row
  - pointer – pointer to the framebuffer you can use



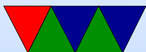
Using a Framebuffer

- How big is it?
- Why might it not just be X*Y*(bpp/8) bytes big? Alignment issues? Powers of two? Weird hardware reasons?
- Things like R/G/B order, padding bits, bits grouped together (on Apple II groups of 7 bytes), etc
- Otherwise it's just an exercise is calculating start address and then copying values
- How do you calculate colors?

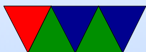


Putting a Pixel

- Depends a bit on the graphics mode you request
- For simplicity, request 800x600x24-bit
- Get back pointer, size, pitch
- Each X row has B,G,R bytes repeated for each pixel (the ordering changed at some point with a firmware release, used to be RGB)
- pitch returned by the GPU. Normally it would just be (maxy*bpp)/8, but it can vary depending on how the hardware arranges the bits.



- To get to next row increment by pitch value (bytes per row)

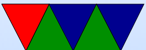


Putpixel Routine

```
void putpixel(uint8_t *fb, int x, int y, int color) {
   uint8_t b,g,r;

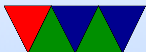
   b=(color>>16)&0xff;
   g=(color>>8)&0xff;
   r=(color)&0xff;

   fb[(x*3)+(y*pitch)]=b;
   fb[(x*3)+(y*pitch)+1]=g;
   fb[(x*3)+(y*pitch)+2]=r;
}
```



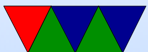
Drawing Lines

- Horizontal: just draw a bunch of pixels with same y-coord
- Vertical: draw a bunch of pixels with same x-coord
- Arbitrary: do you remember your geometry, slope, etc? Bresenham's algorithm can be used especially on systems w/o a lot of resources



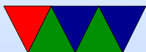
Drawing Circles

- x=sin(), y=cos()
- trig can be expensive. Can use $x^2 + y^2 = r$ too
- There's also a Bresenham circle algorithm



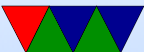
Drawing a Gradient

- Just draw a horizontal line, adjusting the color each line
- Start at 0 and for r/g/b and increment one (or multiple) components each iteration



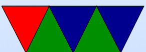
Console Display

- Can implement a text console
- Need a font to print characters
- Can parse/implement the ANSI control characters
- Scrolling: how do you scroll? Do a memory copy of framebuffer up by a line, then draw the new line



Fonts

- How do you convert an ASCII character to a pixel pattern?
- Fonts!
- Fancy fonts are vector-like, can having Turing complete language for better scaling. Variable-length. Kerning, ligatures
- We want something simpler, a fixed-width bitmap font is fine for a console

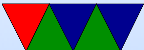


Bitmapped Font

- Each character an 8x8 (or 8x16, or similar) pattern
-

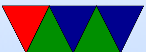
```
unsigned char smiley[8]={
    0x7e,   /*    ******    */
    0x81,   /*   *      *   */
    0xa5,   /*   * *  * *   */
    0x81,   /*   *      *   */
    0xa5,   /*   * *  * *   */
    0x99,   /*   *  **  *   */
    0x81,   /*   *      *   */
    0x7e,   /*    ******    */
};

void put_smiley(int xoff, int yoff, int color) {
   for(y=0;y<8;y++) {
      for(x=0;x<8;x++) {
         if (smiley[y]&(1<<(7-x))) {
            putpixel(fb,x+xoff,y+yoff,color);
```



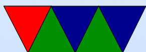
```
            }
        }
    }
}
```

- Can find source of fonts online, VGA fonts. Just a binary set of bitmapped characters indexed by ASCII code.
- Usually 8x16 though; the custom font used in the homework is a hand-made 8x8 one



Device Drivers

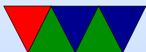
- Have you written one before?
- Writing a simple one for Linux can be fun
- This is (used to be?) a project in ECE331



Kernel Device Drivers

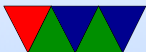
- So far we discussed talking to kernel via syscalls
- How do we talk to hardware from userspace without adding lots of custom syscalls?
- i2c, gpio, etc
- On Linux/UNIX have device drivers that you do file I/O on

  Everything is a File!



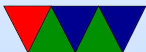
Device Type – Block Devices

- Read chunks of data with random access
- Can seek file location back and forth
- open/read/write/lseek
- Examples: disks, ramdisk, storage
- Filesystems often go on top



Device Type – Character Devices

- Read a sequence of characters incoming, write outgoing
- Can't seek around in this stream
- open/read/write/ioctl
- Examples: serial, i2c, spi, gpio



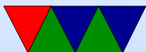
Device Nodes – Accessing Devices

- Access by opening special files (often in /dev) and doing I/O

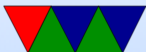
  `brw-rw---- 1 root disk 8, 1 Oct  6 10:07 /dev/s`

- Node has a major / minor number which indicate what type of device and which one (if multiple supported)
- Also that b means "block"
- Device nodes are created with "mknod" and make a special device node in filesystem

  `mknod sda b 123 45` or similar

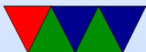


- When you open a device node, kernel looks up who owns it in table and maps this to proper device driver

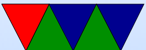


Auto-creation of Device Nodes (Linux)

- Manual creation of nodes
  - Old days if new hardware had to manually add node
  - Otherwise you distribution could just create every possible node under /dev
  - Previously: who maintained the list? Just text file in linux-kernel?
  - Also hit limits: only 16 hard drive partitions?
  - Something automatic preferable
- Automatic creation

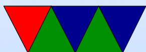


- First attempt on Linux was devfs (flamewar)
- udev/systemd now will auto-create these
- Makes life easier

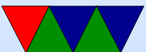


Device Workflow

- At boot, kernel sets up all devices
- The init() method for the device is called, devices "register" which device node they'll respond to
- User opens device node under /dev
- Kernel creates file descriptor entry
- This contains struct with info but also pointers for various methods like read/write/ioctl/close
- When you read/write on an opened device, the kernel used the fd to map your read call to a device_read call

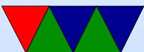


in the device driver in the kernel



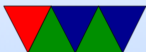
Writing Linux Devices

- This is just bare-bones
- Lots of documentation on this, see references online
- `https://sysprog21.github.io/lkmpg/`
- Complicated hardware (busses like pcie, usb, etc) lead to much more complicated devices



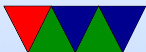
Linux Device Drivers – setup

- `init_module()` – has init code
- `cleanup_module()` – has code run at exit
- `MODULE_LICENSE("GPL");` need to specify license. Mostly to make it harder to run proprietary code in kernel (doesn't really work)
- Can use `module_init()` and `module_exit()` to use non-default names for init/cleanup



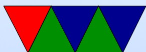
Linux Device Drivers – Loadable Modules

- Linux has idea of compiled-in drivers but also loadable modules
- Can use `lsmod`, `insmod`, `rmmod` to manage
- If compiled in the cleanup routine never runs
- If loadable module is used, the init code might be freed after running

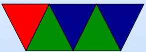


Linux Device Drivers – File Ops Structure

- Layer of indirection, list of pointers that you point to the proper function in your device driver
- device `file_operations` structure
  - llseek()
  - read()
  - write()
  - ioctl()
  - mmap()
  - open()

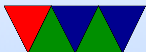


- ◦ others...
- Can leave as NULL

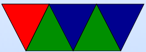


Linux Device Drivers – Other

- Routines for setting up and pointing to interrupt handler
- things like register_chardev()

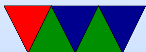


Linux VFS?

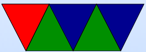


531 OS and the file indirection structure

- Ideally we'd have one too
- Currently we don't, hard-code all with syscalls

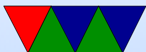


Other Device Driver Topics

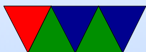


Block Device Special Concerns

- Abstraction is a linear random-access series of blocks
- Actual hardware might not map this, especially old spinning hard-disks
- Optimizing
  - prefetch (to hide slow access speeds)
  - block scheduling (what happens when multiple processes reading from same block device. SSD random access not matter, old days spinning disk not efficient to seek around)

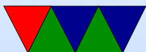


- Disk cache – disk access slow. How do we make slow things faster? Cache? Use unused RAM for this?
- Page cache?
- Wear-leveling / Bad-blocks for flash drives

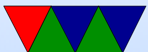


Disk performance

- Traditionally a lot of this came down to hardware.
- Spinning rust disks; head movement, cylinders/sectors. Reading consecutive faster, random access bad (millisecond bad)
  More complicated, fancy disk interfaces and embedded processors. Large caches (why can that be bad), shingled disks?
- Much of this goes away with flash disks, but still emulate old disk interface

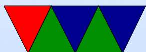


- Name lookup can also be slow.



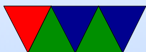
Disk Block Size

- Way too much overhead to have single byte granularity
- For a long time this was 512bytes/block
  Way too many for huge disks so disk drive companies pushing for 4k (but trying to remain backward compatible)
- Filesystems can allocate with larger blocksize.
- Large blocksize good: fewer blocks to track for each file
- Large blocksize bad: waste space on small files



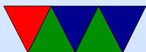
531 Storage / Block Device

- Need somewhere to store data that we can load
- block device, series of blocks
- Disk, but SD card drive complicated to write



531 Storage / RAM disk

- How about RAM disk? Treat region of RAM as if it were a disk, can read/write blocks of data
- Can load data for block device along with kernel (tacked on)
- This is common thing to do with OS, sometimes called initrd



Using a Block Device

- We can get blocks of data, but how do we find the blocks that we want...
- We will talk about this when we talk about Filesystems