ECE531: Advanced Operating Systems – Homework 7

Virtual Memory / Filesystems / Graphics

Due: Tuesday, 28 November 2023, 2:00pm

This homework is a mix of topics due to various complications getting the Pi4 going.

1. **Virtual Memory Questions**

   (a) Name one feature found in Linux/UNIX that is made possible (or easier) because of virtual memory.

   (b) What important piece of logic on a modern processor hides a lot of the overhead of page table lookups?

   (c) What is it called when a memory access happens, the CPU walks the page tables, but the virtual memory page requested is not found in the page table? Whose job is it to handle this now that the CPU has given up?

   (d) The operating system has found the page requested by the CPU, but it tries to allocate space for it in physical memory but physical memory is full. How can more room be freed up in physical memory?

2. **Filesystem Questions**

   (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)?

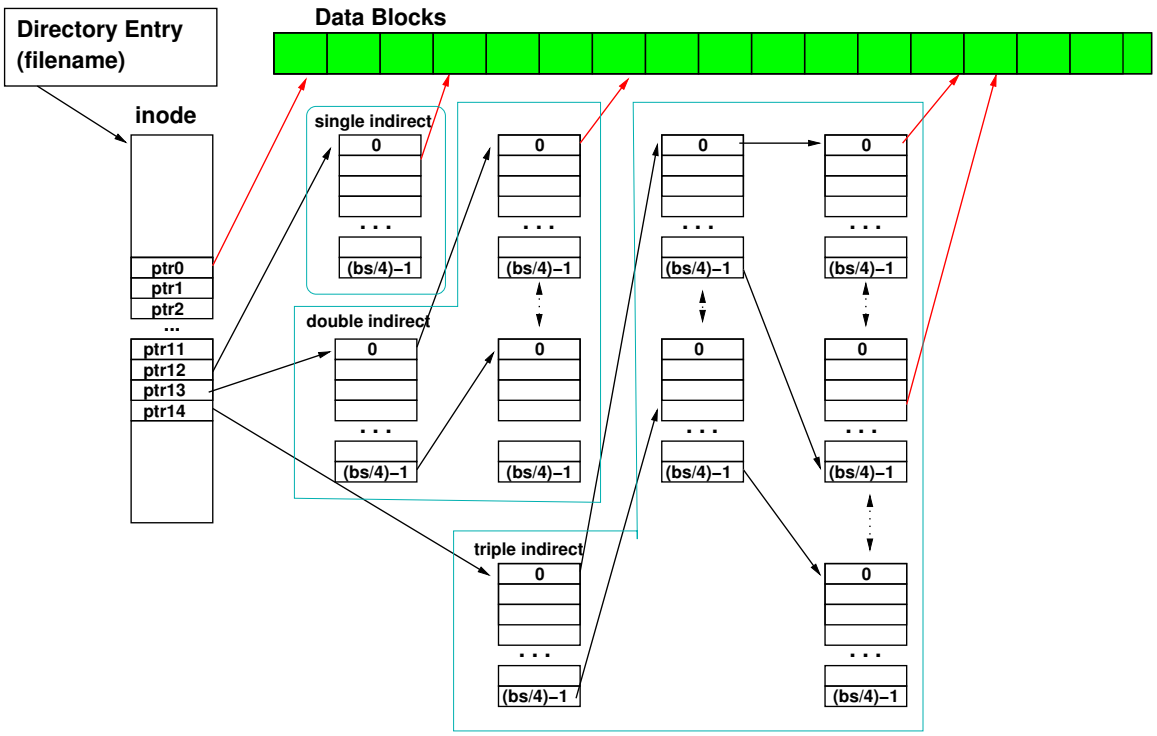

Figure 1: ext2 inode Diagram

(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/`.

3. **Device Questions**

(a) How does a character device differ from a block device?

4. **Graphics Questions**

   (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?

5. **Submit your work**

   - Please submit your answers to the above questions in some sort of document (pdf, txt, doc) via e-mail by the deadline.

6. **Code Notes**

   (a) I am including a link to a code update that might be useful as a reference for the projects. It works fine on a Pi3, including in theory writing to an HDMI monitor if you hook one up. The code will boot on a Pi4, but no graphics, and any command you run will fail. I am still working on why this happens on Pi4. Currently there is no coding assignment involving this.

   - Download the code from:
     `http://web.eece.maine.edu/~vweaver/classes/ece531/ece531_hw9_code.tar.gz`
   - Uncompress the code. On Linux or Mac you can just
     `tar -xzvf ece531_hw9_code.tar.gz`

   (b) **Code Changes for Virtual Memory**

   - The provided code implements some very simple ARMv7 virtual memory support. It sets up a large, 1:1 "sections" mapping, so the virtual and physical addresses are the same, with 1MB page granularity (4096 page table entries)
   - Look at the code in `./kernel/memory/armv7-mmu.c` for all of the details. It's a lot of low-level setting up of ARM special registers.
   - The code as provided does not provide memory protection support. Try running the provided command `cause_error kernelread`
     which will try to read operating system memory. Then try running `cause_error kernelwrite` which will try to overwrite your operating system with garbage. Note what happens.
   - Now, edit the file `./kernel/memory/memory.c` and uncomment `enable_mmu()` line that is commented out. Rebuilt, reboot. At bootup you should get some extra boot messages about the MMU.
   - Now try running `cause_error kernelwrite` again. Also try `cause_error kernelread` and `cause_error kernelexec`

   (c) **Code Changes for Filesystems**

- The filesystem we are using is called ROMFS. You can read the documentation on it here: `https://www.kernel.org/doc/Documentation/filesystems/romfs.txt`
- We have no disk support, so the romfs is loaded as a ramdisk at boot time (it is included as a binary blob during the kernel build). All block reads/writes are turned into RAM accesses with the proper offset. The code for this is simple, and can be found in: `kernel/drivers/block/ramdisk.c`
- The romfs filesystem support is in `kernel/fs/romfs/romfs.c`
  The important functions are:
  - `romfs_get_inode()` – given a filename, finds the inode for it
  - `romfs_stat()` – returns metadata from a file
  - `romfs_mount()` – mounts the filesystem
  - `romfs_statfs()` – the statfs system call, reports disk usage for tools like "df"
  - `romfs_getdents()` – returns the directory entry info

(d) **Code Changes for Graphics**

- 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.