CS 240
Computer
Systems

Week 3: File Formats and Memory

CS 240, Spring 2021 - Week 3
Wade Fagen-Ulmschneider

Beyond Characters: Files and File Types

Using binary digits, often represented as characters using an encoding like UTF-8, we can build more complex file types.

File Extensions: An Easy Identifier

The most common way to identify the contents of a file is by the **file extension**. The file extension is defined as:

Examples:

`cs240.png`	`mp1.c`	`mp1.h`	`taylor.swift.mp4`

"Plain Text" Files

We consider a file to be a "plain text" file when:

Examples of "plain text" files:

Examples of "non-plain text" files:

Deep Dive: PNG File Format

For all file types, a file specification will describe how the bytes in the file will be organized. Let's look at part of the PNG Image file specification:

Source: http://www.libpng.org/pub/png/spec/1.2/PNG-Structure.html

Section 3.1: The first eight bytes of a PNG file always contain the following (decimal) values: **`137 80 78 71 13 10 26 10`**
`Hex: 89 50 4e 47 0d 0a 1a 0a`

This signature indicates that the remainder of the file contains a single PNG image, consisting of a series of chunks beginning with an IHDR chunk and ending with an IEND chunk.

Section 3.2: Each chunk consists of four parts:

# **Length**: A 4-byte unsigned integer giving the number of bytes in the chunk's data field. The length counts only the data field, not itself, the chunk type code, or the CRC. Zero is a valid length. Although encoders and decoders should treat the length as unsigned, its value must not exceed 231 bytes.

# **Chunk Type**: A 4-byte chunk type code. For convenience in description and in examining PNG files, type codes are restricted to consist of uppercase and lowercase ASCII letters (A-Z and a-z, or 65-90 and 97-122 decimal). However, encoders and decoders must treat the codes as fixed binary values, not character strings. For example, it would not be correct to represent the type code IDAT by the EBCDIC equivalents of those letters. Additional naming conventions for chunk types are discussed in the next section.

# **Chunk Data**: The data bytes appropriate to the chunk type, if any. This field can be of zero length.

# **CRC**: A 4-byte CRC (Cyclic Redundancy Check) calculated on the preceding bytes in the chunk, including the chunk type code and chunk data fields, but not including the length field. The CRC is always present, even for chunks containing no data. See CRC algorithm.

```
$> hexdump -C cs240.png
 89 50 4e 47 0d 0a 1a 0a  00 00 00 0d 49 48 44 52  |.PNG........IHDR|
 00 00 00 fa 00 00 00 fa  08 06 00 00 00 88 ec 5a  |...............Z|
 3d 00 00 00 01 73 52 47  42 00 ae ce 1c e9 00 00  |=....sRGB.......|
 00 04 67 41 4d 41 00 00  b1 8f 0b fc 61 05 00 00  |..gAMA......a...|
 00 09 70 48 59 73 00 00  0e c3 00 00 0e c3 01 c7  |..pHYs..........|
 6f a8 64 00 00 11 86 49  44 41 54 78 5e ed dd 0f  |o.d....IDATx^...|
 90 24 65 79 c7 f1 5f cf  de 72 41 40 81 bb dd 99  |.$ey.._..rA@....|
 13 25 5e 44 09 67 3c b8  9d 39 44 94 0a a5 89 41  |.%^D.g<..9D....A|
 40 23 51 01 41 31 18 2d  4d 42 99 3f 65 12 4d 8c  |@#Q.A1.-MB.?e.M.|
 45 48 4c aa 24 89 49 15  e6 af c1 0a a8 09 26 5a  |EHL.$.I.......&Z|
 [...]
```

Memory Hierarchy:
The third foundation of a computer system is the memory -- the storage of data to be processed by our CPU. There are many different types of common memory in a system:

1. [Processor Registers]:

2. [Processor Cache]:

3. [RAM]:

4. [Solid State / Flash Memory/Storage]:

5. [Hard Disk Storage]:

6. [High-Density / Offline / Tape Storage]:

Sample Program #1:

`memory/col-row.c`

```
  for (unsigned int c = 0; c < SIZE; c++) {
    for (unsigned int r = 0; r < SIZE; r++) {
      array[(r * SIZE) + c] = (r * SIZE) + c;
    }
  }
```

What is the memory access pattern?

By locality of reference principle, will this program have good cache performance?

Sample Program #2:

`memory/row-col.c`

```
  for (unsigned int r = 0; r < SIZE; r++) {
    for (unsigned int c = 0; c < SIZE; c++) {
      array[(r * SIZE) + c] = (r * SIZE) + c;
    }
  }
```

What is different about Program #2 from Program #1?

By locality of reference principle, will this program have good cache performance?

Running Times: **`col-row.c`** (Program #1):

`row-col.c` (Program #2):

Caching Strategies: Keeping Data Close

In working with memory in any computer system, we want to access it as quickly as possible. However, space is extremely limited in the fastest memory, so we need strategies on what data to keep close.

General Purpose Memory:

- CPU Registers: Stores one word, only **8 available** on x64.
- CPU Cache: Stores a collection of 4 KB "pages" from RAM.
    - Intel Core i9-10900KF has 256 KB /CPU + 20 MB
    - Total Pages: ________ / CPU + _________
- RAM: "Dream Computer" has 64 GB (standard: 8 GB+)

Key Idea: <u>Locality of Reference</u>

System Memory: Limited, Shared, and Simple

In hardware, your system has a fixed amount of RAM:

1. Sequentially Addressed:

2. Shared:

3. We refer to this memory as:

To help us to begin to organize this RAM, we divide the RAM up into chunks called __________.

- On most systems, a page is ______ KB.
- Linux: `getconf PAGESIZE`

Naive Solution: Segmentation

Suppose we directly assign physical memory pages to programs running simultaneously. For simplicity, we'll assume **1 MB pages**:

Allocation Sequence:
1. P1(a): 3 MB
2. P3: 5 MB
3. P1(b): 2 MB
4. P3 exits, memory free'd.
5. P2(c): 4 MB
6. P2(d): 5 MB
7. P1(a) increase to 5 MB

16 MB RAM available:

- Can we meet all of the allocation requests?

- Is there enough memory available on the system?

- How can we solve this?

Virtual Memory:

Modern systems provide an abstraction between the _________ and ____________________:

1. A ____________________ translates a ____________ into a physical address.

2. Not Shared:

Let’s repeat the example, with virtual page table:

P1 Page Table:	RAM	P2 Page Table:	P3 Page Table:

(Use the same allocation sequence as seen ⇐ on the right.)

With a virtual memory system:

- Can we meet all of the allocation requests?
- Are we limited to just RAM?

Advantages of a Virtual Memory System:

1. Perception of Continuous Memory:
2. Storage on external (or even remote) storage:

memory/pointerAddr.c

```
printf("  Start of `array`: %p\n", array);
printf("    End of `array`: %p\n", &(array[(SIZE * SIZE) -1]));
```