Text Files & Binary Files

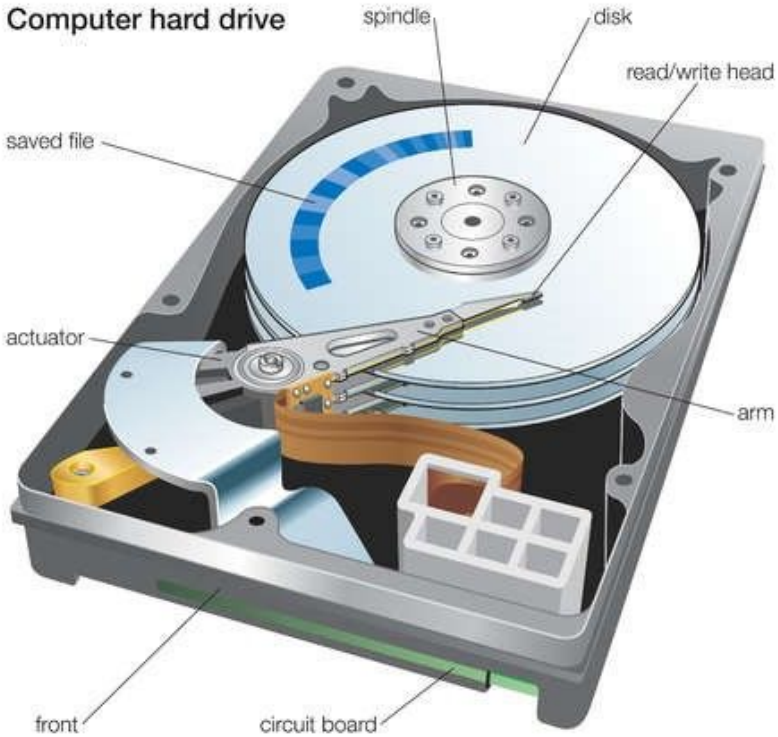
Jason Miller

Persistent Storage

Magnetic tape.

Optical disk (CD).

Solid state (no moving parts).



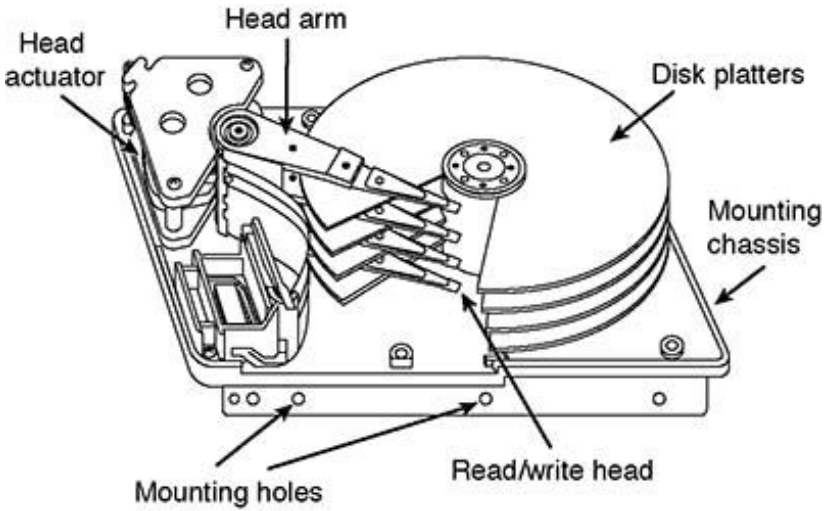
Spinning magnetic disk.

Storage speed vs cost

Register	Most expensive, not persistent, onboard CPU. Typical amount is 128 bytes. Speed in billionths of sec.
Cache	Expensive, not persistent, onboard CPU. Typical amount is 256 KB. Comes in levels L1, L2, L3.
RAM	Affordable, not persistent, separate memory card. Typical amount is 8 GB. Speed in millionths of sec.
SSD	Solid state “disk”, persistent, separate card. Typical amount is 256 GB. Faster than spinning disk.
Disk	Cheap! Spinning disk, slow persists for years. Typical amount is 1 TB. Speed in thousandths of sec.
Tape	Almost free! Persists for decades. Speed in hours. Typically for overnight backups of 1 PB.

Using spinning disk

Drawing from
https://www.informit.com/articles/article.aspx?p=2425742



Disks are divided virtually into sectors and tracks. Data is divided into blocks to fit one sector of one track (typically 4 KB). Blocks are transferred in bursts whenever a block spins by a head. And the whole process is so slow that the CPU handles it asynchronously with interrupts, akin to saying, “Here is what I need, let me know when you find it.”

File organization on disk

- This is a job for the operating system (O/S)
  - The O/S moves fixed size blocks between disk and RAM.
  - Small files take up one block (with wasted space).
  - Large files span many blocks organized as a **linked list**.
- Database management systems (DBMS)
  - Manage files on their own. May appear to the O/S as one large file.
  - Extra files called indexes provide faster access to large files.
  - The **B-tree** is a widely used data structure for such an index.
    - https://en.wikipedia.org/wiki/B-tree
- Hard disk organization
  - The disk controller stripes files across multiple disks for parallel read & write
  - Visualization
    - http://www.mathcs.emory.edu/~cheung/Courses/377/Syllabus/1-files/intro-disk.html

Work-around for slow delivery: buffer

- Devices provide access to whole chunks
  - No RAM chip lets you access one bit or even one byte. Typical delivery is “word” of 8 bytes.
  - No disk lets you access one byte. Typical delivery is “block” of 4 KB.
- Systems use buffers to improve performance
  - The CPU onboard cache buffers data on its way to and from RAM.
  - The disk controller onboard cache buffers data on its way to and from the disk.
  - The O/S usually runs its own cache in RAM and even on disk.
- Applications also use buffers
  - Despite all the system buffering, application programmers need to implement their own.
  - For best performance, match buffer size to hardware capability and application need.
  - Many applications process text files one line at a time.
  - Many applications read binary files one block at a time.
  - Applications decide when to write: by “closing” a file or just “flushing” the buffers.

Side issue: I/O errors

- Faults happen
  - File not found
  - Permission lacking
  - File corrupt
  - End of file
  - Disk full
  - Device disconnected
- Application programmer responsibilities
  - Enclose I/O in a try/catch block
  - Allow program to continue sensibly
  - Report errors according to policy

```
// Java example

try {
        // open filename
        // write lines
        // close file
        return true;
} catch (Exception e) {
        // inform the user
        // record event in a system log
        return false;
}
```

Side issue: file access rights

- This is a job for the operating system (O/S)
  - Multi-user systems protect each user from the others
  - Even single-user systems try to protect system files from the user
  - Database management systems (DBMS) implement their own internal rules
- Access rights
  - **Read** access: user or process may read the file
  - **Write** access: user or process may change or delete the file.
    - Overwrite vs append: many languages provide two ways to open a file for writing.
  - **Execute**: user or process may run the program.
- Ownership
  - Unix recognizes three levels of ownership: owner, group, world. For example, rwxr-xr-- means the owner has all rights, group members can read and execute, everyone else can read.
  - Windows maintains access control lists (**ACL**s) of who can & can't do what.

Side issue: access modes

- **Streaming** access
  - The bytes arrive in order.
  - No ability to “back up” (some utilities offer one-byte push-back).
  - No ability to “skip ahead” (except by reading and ignoring that many bytes).
  - Streaming programs are versatile: they can take data from files or web APIs.
  - Streaming algorithms form a special class called **on-line algorithms**.
- **Random** access
  - This does not mean you get random data or data in random order!
  - You can read bytes from any offset in the file. You choose the starting offset.
  - You can write bytes to any offset in the file. You choose the starting offset.
  - **Web API**s typically do not provide random access.
  - Random access programs are restricted: they may not be able to use web APIs.

Side issue: record format

- Data can be organized strictly or flexibly
- Strict, **fixed-size** fields and records are easy to process in bulk
  - Examples: 4-byte integers, 10-digit phone numbers, max 40-letter last name
  - Text file example: records in a csv file all have the same fields
  - Advantage: you can read, skip, or buffer any number of records exactly
  - Disadvantage: large fields waste space, small fields induce overflow conditions
- Flexible, **variable-size** fields and records can save space
  - Example: UTF-8 writes strings as 2-byte-length followed by that many ASCII characters
  - Text file example: records in json files can have different fields
  - Advantage: saves space if records come in a variety of lengths
  - Disadvantage: every data element must be processed individually

Critical issue: text vs binary format

- **Text** file formats
  - The file contains printable characters organized into lines
  - Readable by many programs: browsers, text editors, word processors
  - Many widely used formats: txt, csv, html, xml, json, md, yml
- **Binary** file formats
  - The file contains an array of bytes, not all of them printable
  - Impossible to use without some program that understands the format
  - Windows executables such as Microsoft Word (word.exe)
  - Documents created by Microsoft Word (myfile.docx)
  - Media files: video (mp4), sound (mp3), images (jpg, png)
  - Compressed files (zip, gz, bz, rar)

Text vs binary files: example

- Java **source code is text**
  - MyCode.java contains human-readable, printable text
  - MyCode.java should conform to the Java language specification
  - MyCode.java can be displayed by Visual Studio, TextEdit, Notepad, Word, etc.
  - The main program that reads Java source is the Java compiler (javac)
- Java **byte code is binary**
  - MyCode.class is an array of bytes, many of them non-printable
  - MyCode.class conforms to the Java class file specification
  - The main program that reads bytecode is the Java Virtual Machine (see JVM spec)
  - The first 8 bytes of a class file are specified this way:
    - Bytes 1-4: 11001010111111101011101010111110 in binary
      - That is 3405691582 in decimal or 0xCAFEBABE in hexadecimal
    - Bytes 5-8: compiler's major & minor version numbers as 2 unsigned 2-byte ints
      - What we call Java SE 17 is encoded here as major version 61

Text vs binary files: language support

- C
  - Functions for text files: open(), getc(), putc(), gets(), puts()
  - Functions for binary files: fopen(), fseek(offset), fread(#bytes), fwrite(data)
- Java
  - Classes for text files: File, FileReader, FileWriter, BufferedReader, BufferedWriter
  - Classes for binary files: File, RandomAccessFile
- Python
  - Functions for text files: open(filename, ‘r’), readline(), readlines()
  - Functions for binary files: open (filename, ‘rb’), read(#bytes)

Text or binary files: close them!

- Open files consume O/S resources
  - An open file is a data structure in RAM
    - Filename, ownership, rights, position for next access
    - Pointers to portions of the file now in RAM or cache
    - List of changed blocks in RAM waiting to be saved to disk
  - Most operating systems limit the total number of open files
- All languages support closing files
  - C: add a close() for every open, add an fclose() for every fopen()
  - Java:
    - Programmers can call close() but there's a better way!
    - When an object like FileReader falls out of scope, Java closes the associated file
  - Python:
    - Programmers can pair open() and close() functions but there's a better way!
    - Using a "with open() …" block, python closes the file at the end of the block

Text file organization

- Character encoding
  - **UTF-8** text files - one byte per char using ASCII encoding
  - UTF-16 text files - two bytes per char, Unicode Transfer Format, Windows and Java files
  - UTF-32 text files - four bytes per char, used by APIs supporting international symbols
- End of Line (**EOL**) encoding
  - ASCII 10 = line feed (**LF**) = ‘\n’.
  - ASCII 13 = carriage return (**CR**) = new line (NL) = ‘\r’
  - Unix files - lines end in LF. If viewed on Windows, these files seem corrupt
  - Windows files - lines end in NL+LF. If viewed on Unix, these files say ^M on every line
  - Java and Python libraries interact seamlessly with Unix & Windows files
- End of File (**EOF**) encoding
  - Operating systems and languages all provide some way to recognize this condition.

Text file organization: JSON example

JSON is one text file format.

JSON defines the data organization:

- Objects and arrays
  - { object }
  - [array]
  - Array of 2 objects: [{one},{two}]
  - Line breaks are optional
- Key:value pairs
  - Associate 1 key with 1 value
  - User-defined keys
  - Values can be words, objects, arrays

```
[
 {     “First” : “Joe” ,
       “Last” : “Smith”
  } ,
 {     “First” : “Jane”,
       “Last” : Miller”
 }
]
```

Text file organization: JSON example

```
$ python
>>> import json
>>> pb = [{'First' : 'Jason', 'Last' : 'Miller', 'Phone' : '304-876-5070'}]
>>> pb.append ({'First':'Kimberlee','Last':'Turlington','Phone':'039 298-72-30' } )
>>>
>>> with open("demo.json", "w") as json_file:
...     json.dump(pb, json_file, indent=4)
...     json_file.write("\n")
...
>>> quit()
```

```
[
    {
        "First": "Jason",
        "Last": "Miller",
        "Phone": "304-876-5070"
    },
    {
        "First": "Kimberlee",
        "Last": "Turlington",
        "Phone": "039 298-72-30"
    }
]
```

Binary file organization

- Standard formats
  - Formats for executable programs: Windows exe, Java class
  - Formats for compressed text: zip, gz, bz, 7z
  - Formats for images: jpeg or jpg, png
  - A database format called Hierarchical Data Format 5: hdf5 or h5
  - Formats for specific applications: Adobe pdf, Microsoft Office docx and xlsx
  - Formats defined by Python libraries: pickle, numpy, pandas
- Define your own
  - Many companies or applications define their own binary data format
  - Create specification e.g. eight bytes of text followed by fifty 4-byte integers
  - Create a writer program and a reader program that implement the specification
  - Create a demo program that writes and reads some data

Binary file organization: pickle example

```
$ python
>>> import pickle
>>> phonebook = [{'First' : 'Jason', 'Last' : 'Miller', 'Phone' : '304-876-5070'}]
>>> with open('filename.pickle', 'wb') as handle:
>>>    pickle.dump(phonebook, handle)
>>> with open('filename.pickle', 'rb') as handle:
>>>    readback = pickle.load(handle)
>>> print(phonebook == readback)
True
>>> quit()
```

Contents of filename.pickle seen with hexdump program ‘oc -h’

```
0000000      0480    3f95    0000    0000    0000    5d00    7d94    2894
0000020      058c    6946    7372    9474    058c    614a    6f73    946e
0000040      048c    614c    7473    8c94    4d06    6c69    656c    9472
0000060      058c    6850    6e6f    9465    0c8c    3033    2d34    3738
0000100      2d36    3035    3037    7594    2e61
```

Inspect binary files with a hexdump utility

od, or ‘octal dump’, is a dumper for unix. The **-h** means output hexadecimal. The left-most numbers are file offsets.

The first four bytes contain the Java magic number 0xCAFEBABE. Since Intel chips are “little endian,” they appear FE, CA, BE, BA.

```
$ od -h FileParser.class | head
0000000      feca    beba    0000    3700    4800    0009    000f    0a30
0000020      1100    3100    0008    0932    0f00    3300    0009    000f
0000040      0734    3500    0007    0a36    0700    3700    000a    0006
0000060      0938    0f00    3900    000a    000f    093a    0f00    3b00
0000100      0007    0a3c    0600    3d00    0007    0a3e    2700    3f00
0000120      0007    0140    0500    4f43    4d4d    0141    1200    6a4c
0000140      7661    2f61    616c    676e    532f    7274    6e69    3b67
0000160      0001    430d    6e6f    7473    6e61    5674    6c61    6575
0000200      0001    6608    6c69    6e65    6d61    0165    0700    7573
0000220      6363    7365    0173    0100    015a    0600    6572    6461
```

Text-vs-binary: summary

- Text files are easy to use
  - View them and print them with any text editor
  - Individual lines can be long or short.
- Binary files are harder to use
  - You need some viewer utility to inspect them
  - Writer and Reader programs must agree on the exact layout, byte-by-byte.
- Binary files can save space and time
  - Example of storing the number one million
    - In a text file, it requires 7 bytes ASCII or 14 bytes unicode: “1000000”
    - In a binary file, all integers are the same length, usually 4 bytes
  - Fixed-length numbers mean programs can do fast lookup by position