Lab 1 - Filesystem Design

Data Storage and Access

- The stack, heap and other segments of program data live in memory (RAM)
  - fast
  - byte-addressable: can quickly access any byte of data by address, but not individual bits by address
  - not persistent - cannot store data between power-offs
- The filesystem lives on disk (eg. hard drives)
  - slower
  - persistent - stores data between power-offs
  - sector-addressable: cannot read/write just one byte of data - can only read/write "sectors" of data

A hard disk is sector-addressable: cannot read/write just one byte of data - can only read/write "sectors" of data. (we will work with a sector size of 512; but size is determined by the physical drive).

Sectors and Blocks

A filesystem generally defines its own unit of data, a "block," that it reads/writes at a time.

- "Sector" = hard disk storage unit
- "Block" = filesystem storage unit (1 or more sectors) - software abstraction

Let's imagine that the hard disk creators provide software to let us interface with the disk. Only two functions to work with a disk:

```
void readSector(size_t sectorNumber, void *data);
void writeSector(size_t sectorNumber, const void *data);
```

Storing Data on Disk

Two types of data we will be working with:

1. file payload data - contents of files (e.g. text in documents, pixels in images)
2. file metadata - information about files (e.g. name, size)

Inodes

Every file/directory has an inode containing information about it and what blocks store its data.

The **inode table** at the start of the disk stores one **inode** per file.
An inode is a structure containing information about a file such as its size and which blocks elsewhere on disk store its contents.

```
struct inode {
 uint16_t i_mode; // bit vector of file type and permissions
 uint8_t i_nlink; // number of references to file
 uint8_t i_uid; // owner
 uint8_t i_gid; // group of owner
 uint8_t i_size0; // most significant byte of size
 uint16_t i_size1; // lower two bytes of size (size is encoded in a three-byte number)
 uint16_t i_addr[8]; // device addresses constituting file
 uint16_t i_atime[2]; // access time
 uint16_t i_mtime[2]; // modify time
};
```

Note: inodes live on disk. But we can read them into memory where we can represent them as structs.

inode notes:

- If the file size is small, an inode stores up to 8 direct block numbers.
- If the file size is large, an inode stores 7 singly-indirect and 1 doubly-indirect block number.
- A directory is "treated as a file" - its payload data stores the name and inumber of each of its entries in 16 byte chunks.
- To find files, we can hop between directory inodes until we reach the inode for our file. We can start at inumber 1 for the root directory.

Design Principles: Modularity & Layering

Modularity: subdivision of a larger system into a collection of smaller subsystems, which themselves may be further subdivided into even smaller sub-subsystems.

Layering: the organization of several modules that interact in some hierarchical manner, where each layer typically only opens its interface to the module above it.

Note: These ideas aren't specific to filesystems! Eg. networking systems also rely on layering.

Unix builts layers on top of the low-level readSector and writeSector to implement a higher-level filesystem:

- sectors -> blocks
- blocks -> files
- files -> inodes
- inodes -> file names
- file names -> path names
- path names -> absolute path names