EC 440 – Introduction to Operating Systems
Project 5 – Discussion

Manuel Egele

Department of Electrical &
Computer Engineering

Boston University

Simple Filesystem

Capabilities:

- Create
- Write
- Read
- Delete

… files

"Constraints" from the Description

- One 32Mb underlying "virtual disk" (i.e., a file)
  - Consisting of 8192 4K blocks
  - At least 4096 blocks (i.e., 16Mb) must be available for file storage (i.e., you can use up to 50% of the disk capacity for meta-data)
- You can only access *individual* and *entire* blocks through the interface (i.e., block_read, block_write)

“Constraints” from the Description (2)

- Max 64 files at any given time
- Exactly 1 directory (that contains all files)
- Max 15 character filenames

How To Organize File Contents?

s.t. files can be

- created
- arbitrary in size
- grow/shrink
- be deleted

Let's store file contents in a series of *not necessarily* consecutive blocks on disk!

Why list of blocks?

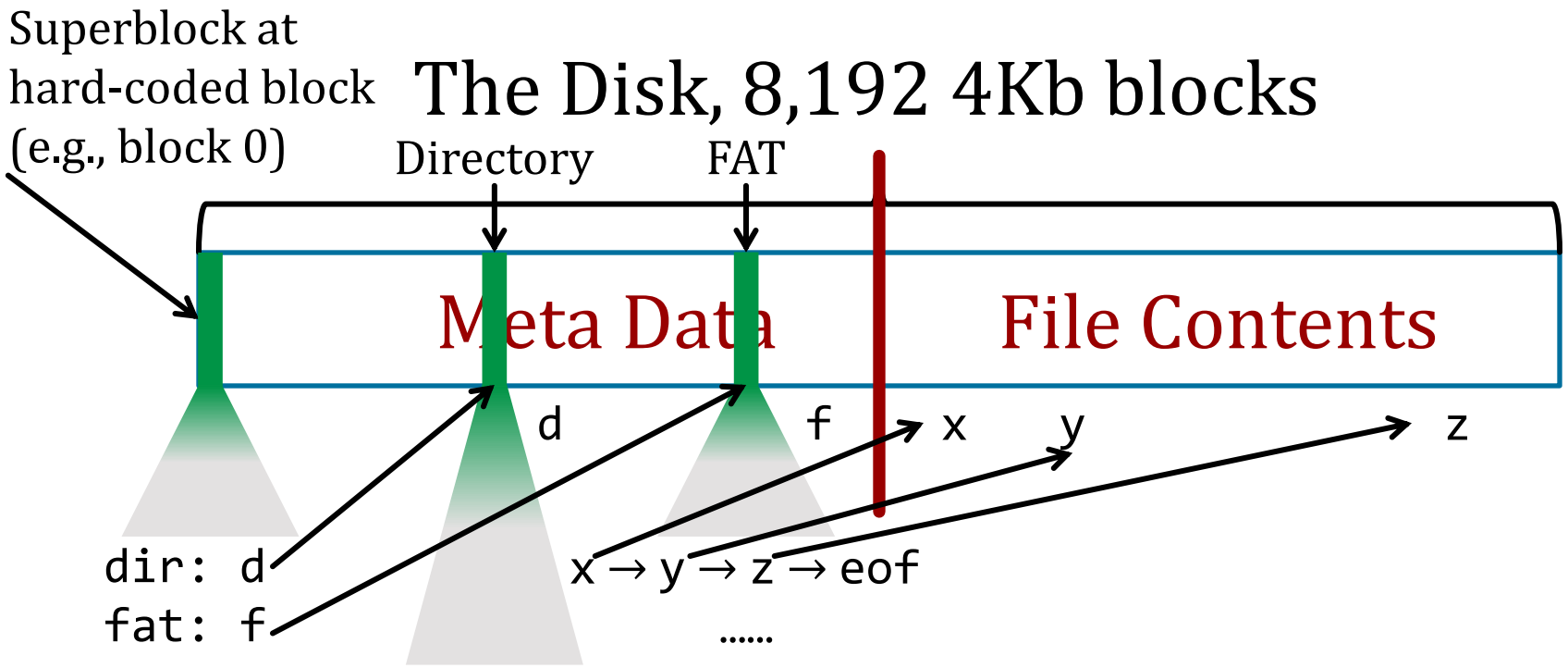
- Flexible, easy to grow shrink

Why not necessarily consecutive?

- Fragmentation might happen (e.g., middle of three files might shrink → block becomes free)

Cool! But ***HOW?!***

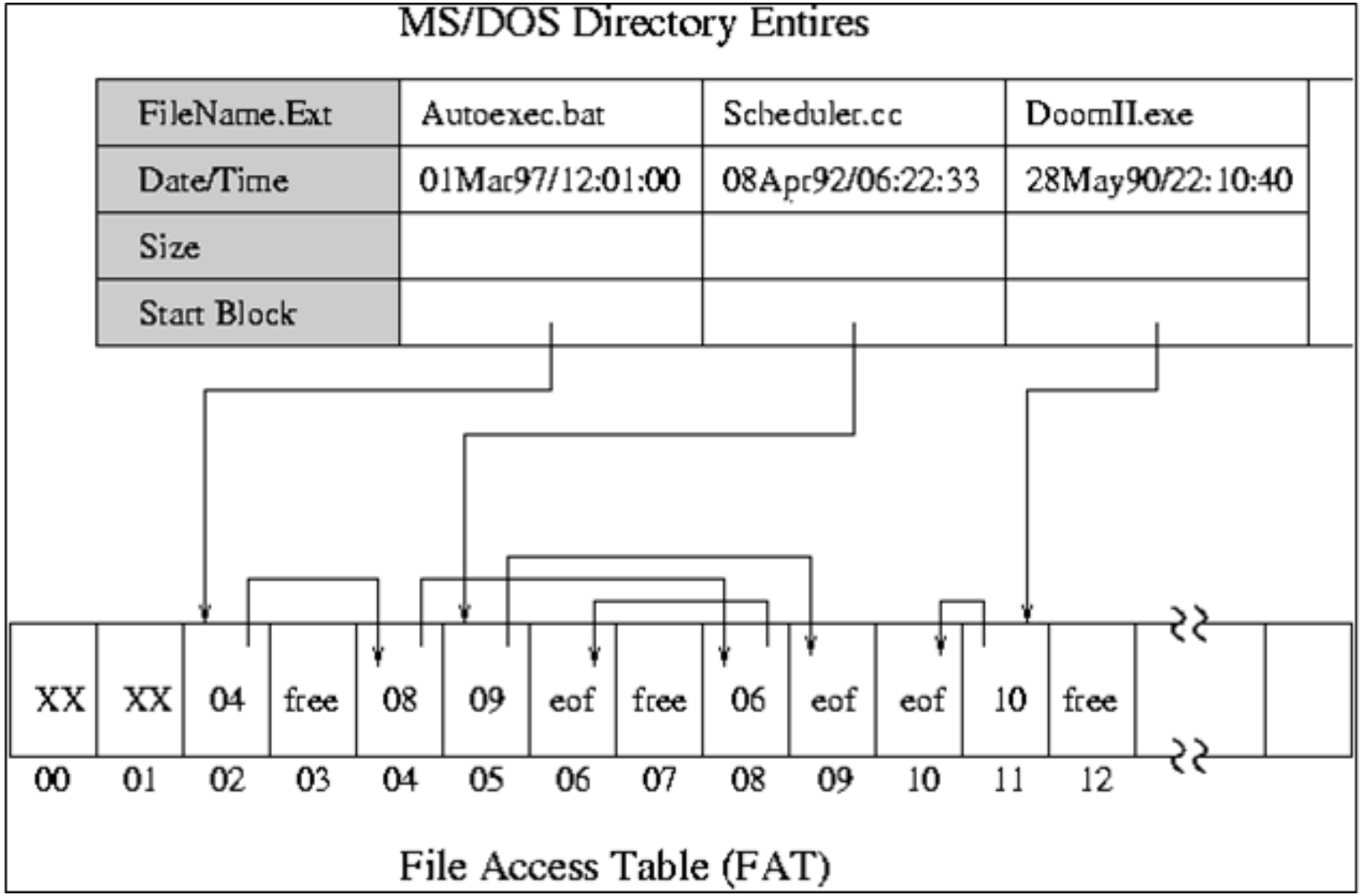
Mental Picture of the Virtual Disk




```
name: f1, start: x, length: 10,000
name: f2, start: a, length: 8888
……
```

Make a file (f1) of size 3 blocks (e.g., 10,000 bytes)

Directory Entries (aka Files) & Content



Want Some Data Structures?

Superblock

- Refers to a number of blocks for directory information (dir_idx, dir_len)
- Refers to a number of blocks that contains the FAT (fat_idx, fat_len)
- Reference to first block of file data (data_idx)

Directory

- Array of dir_entry-s

dir_entry

- Filename
- Filesize
- First Block

FAT

- Array of block_idx'es (or eof, or free)

Superblock

```
struct super_block {
   int fat_idx;  // First block of the FAT
   int fat_len;  // Length of FAT in blocks
   int dir_idx;  // First block of directory
   int dir_len;  // Length of directory in blocks
   int data_idx; // First block of file-data
}
```

Directory Entry (File Metadata)

```
struct dir_entry {
   int used;     // Is this file-”slot” in use
   char name [MAX_F_NAME + 1]; // DOH!
   int size;     // file size
   int head;     // first data block of file
   int ref_cnt;
   // how many open file descriptors are there?
   // ref_cnt > 0 -> cannot delete file
}
```

File Descriptor

```
struct file_descriptor {
   int used;     // fd in use
   int file;     // the first block of the file
                 // (f) to which fd refers too
   int offset;   // position of fd within f
}
```

Some Globals You'll Want

```
struct super_block fs;
struct file_descriptor fildes[MAX_FILDES]; // 32
int *FAT; // Will be populated with the FAT data
struct dir_entry *DIR;   // Will be populated with
                         //the directory data
```

An Aside:
The project is restricted (i.e., simple) enough that you can calculate the size of FAT[] and DIR[] in advance.

Deliverables

- `make` must produce an `fs.o`
- Do not modify `disk.h`
  - If you need additional header files, make new ones (e.g., `mydisk.h`)
- Do not re-define symbols defined in `disk.h`/`disk.c` in your own code!
- Do not include the source of the functions from `disk.c` in your source code

Extra Credit Opportunity

- Minimum requirement of storage is 16MB
- We'll try storing more data in your FS
- For every x MB > 16MB (x >= 1) that your FS can store, you get floor(ld(x)) + 1 points extra credit (capped at 5 points max)

16 + 1 MB -> ld(1) + 1 = 1 point

16 + 2 MB -> ld(2) + 1 = 2 points

16 + 4 MB -> ld(4) + 1 = 3 points

16 + 8 MB -> ld(5) + 1 = 4 points

16 + 16 MB -> ld(16) + 1 = 5 points

Questions?