Recitation 9

PennFAT!

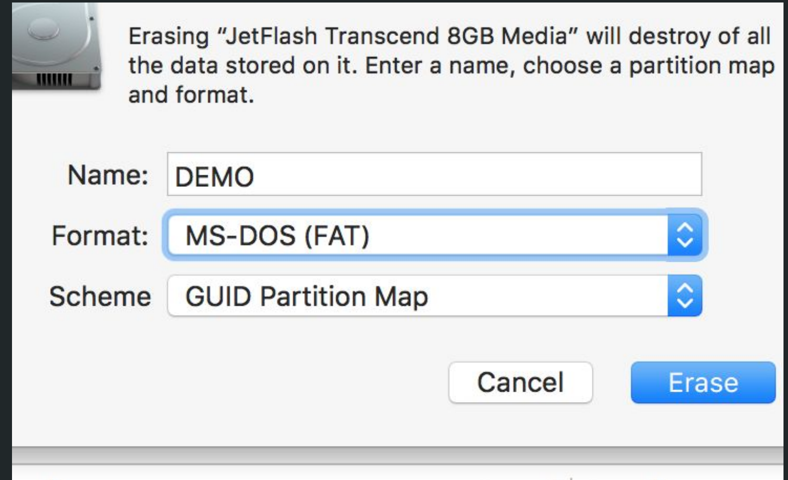


Table of Contents

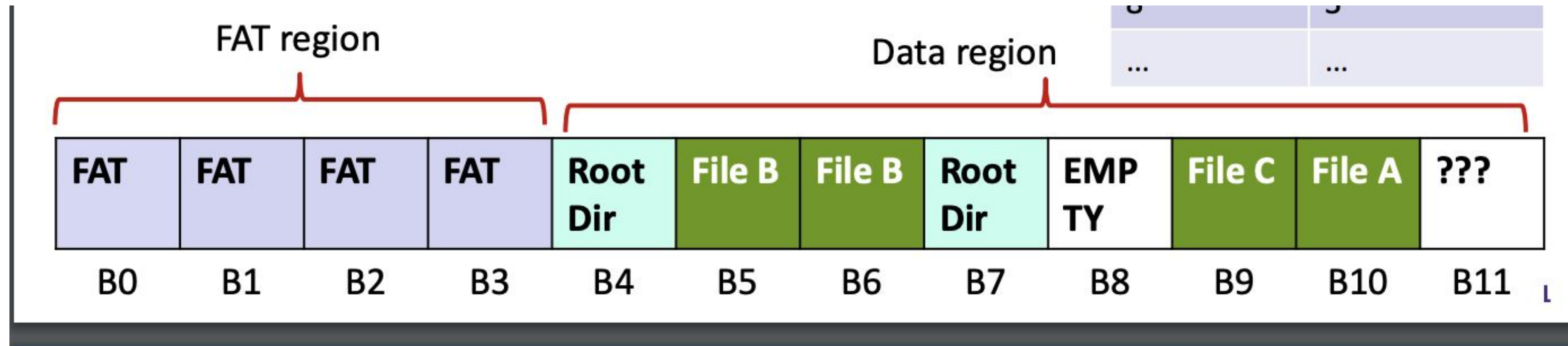
1. Introduction
2. Standalone PennFAT
3. What's After?
4. Something Cool

Disclaimer: These slides are meant for those who have a basic understanding of PennFAT. Please review PennOS lecture slides and come back.

Intro

FAT system splits to two parts:

FAT table and Data blocks



Index	Link
0	0x2004 <— MSB=0x20 (32 blocks in FAT), LSB=0x04 (4K-byte block size)
1	0xFFFF <— Block 1 is the only block in the root directory file
2	5 <— File A starts with Block 2 followed by Block 5
3	4 <— File B starts with Block 3 followed by Block 4
4	0xFFFF <— last block of File B
5	6 <— File A continues to Block 6
6	0xFFFF <— last block of File A
…	…

FAT

Each entry is 2 byte.

First entry give info : # of FAT entries(MSB) and block size(LSB).

Then, all entries are block informations: index is block number, value is next block number.

Second FAT entry must be **ROOT DIRECTORY.**

Which means, FAT[1] is root directory, so first data block must be root directory.

Next entries(FAT[1]……FAT[N])are all file block numbers.

Data block

Root Director and other files.

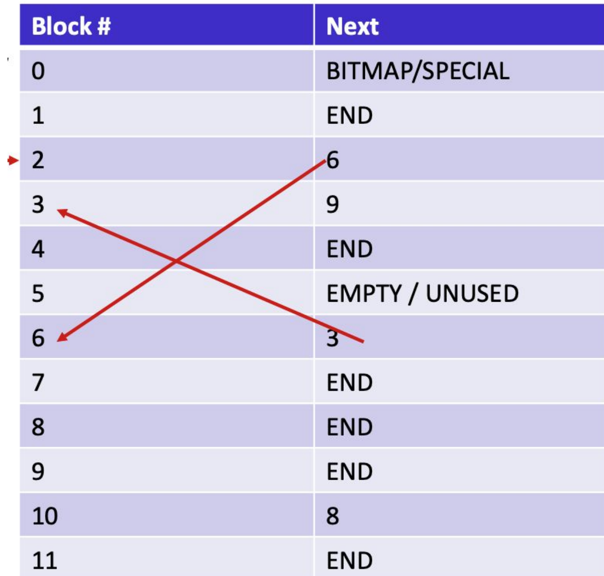
Root directory stores info of other files.

Metadata(64 bytes)

```
char name[32];
uint32_t size;
uint16_t firstBlock;
uint8_t type;
uint8_t perm;
time_t mtime;
// The remaining 16 bytes are reserved
```

With metadata, we will know first block number of the file, and we can get next block number of the file by indexing FAT table.

FAT[current]=Next.



Block #	Next
0	BITMAP/SPECIAL
1	END
2	6
3	9
4	END
5	EMPTY / UNUSED
6	3
7	END
8	END
9	END
10	8
11	END

PennFAT thinks itself as a **hard disk,** but actually a **binary file.**

Milestone 1 - Standalone PennFAT

```
# ./pennfat

pennfat> mkfs minfs 1 0
```

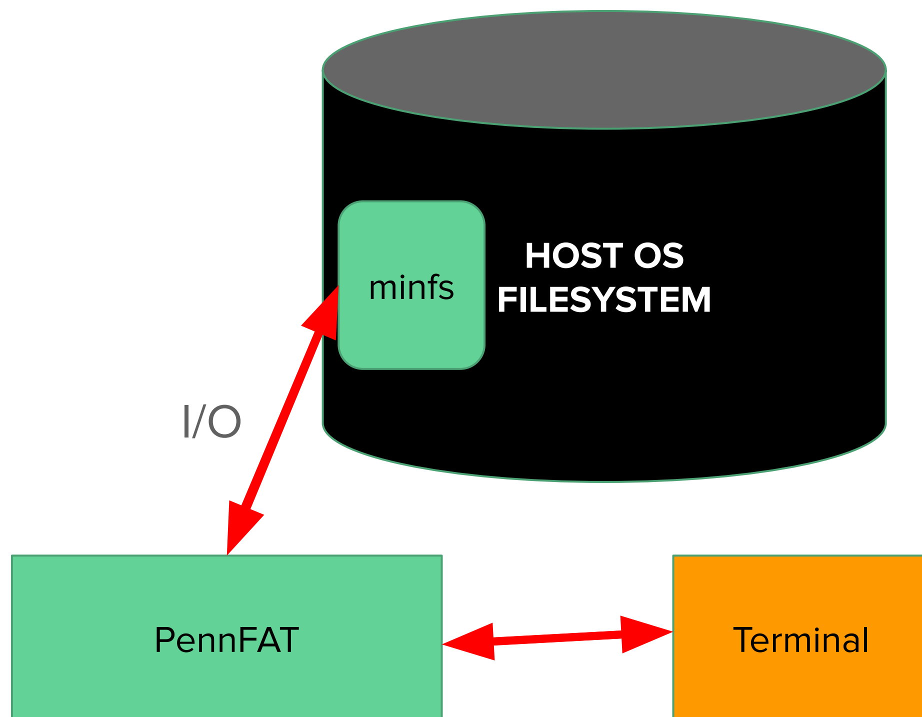
MAKE A FILE SYSTEM!

```
pennfat> mount minfs
```

MOUNT IT!

```
pennfat> touch f1 f2 f3

pennfat> cat -w f1
```



mkfs

- Do not overthink it!

```
TRUNCATE(2)                    Linux Programmer's Manual                    TRUNCATE(2)

NAME
       truncate, ftruncate - truncate a file to a specified length

SYNOPSIS
       #include <unistd.h>
       #include <sys/types.h>

       int truncate(const char *path, off_t length);
       int ftruncate(int fd, off_t length);

   Feature Test Macro Requirements for glibc (see feature_test_macros(7)):

       truncate():
           _XOPEN_SOURCE >= 500
               || /* Since glibc 2.12: */ _POSIX_C_SOURCE >= 200809L
               || /* Glibc versions <= 2.19: */ _BSD_SOURCE

       ftruncate():
           _XOPEN_SOURCE >= 500
               || /* Since glibc 2.3.5: */ _POSIX_C_SOURCE >= 200112L
               || /* Glibc versions <= 2.19: */ _BSD_SOURCE

DESCRIPTION
       The  truncate() and ftruncate() functions cause the regular file named by path or referenced by fd to be trun-
       cated to a size of precisely length bytes.
```

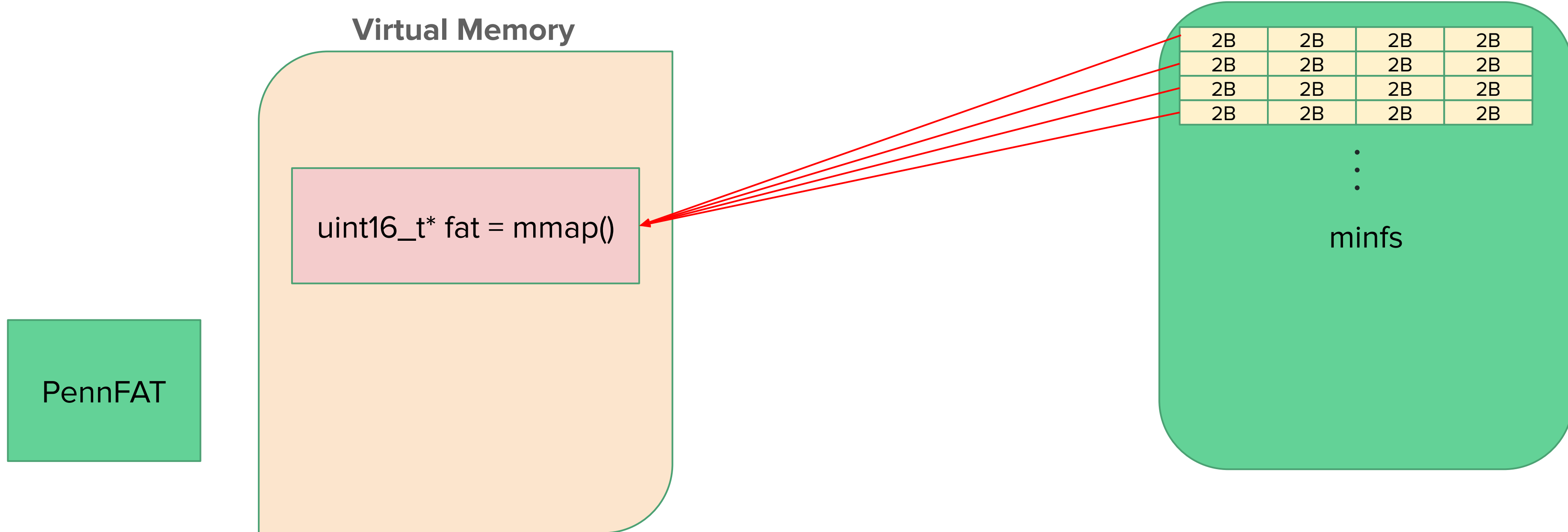
Quick mkfs exercise

```
pennfat> mkfs pikachu 16 2
```

1. Name of Filesystem? pikachu
2. How many blocks in FAT? 16
3. How many entries in FAT? 16*1024/2=8192
4. How many blocks in DATA? 8192-1=8191
5. How big is pikachu in bytes? FAT + DATA = 8192*2 + 8191*1024 = 8403968

mount

- mmap(2) - creates a new **mapping in the virtual address** space of the calling process.



More Clarifications

- touch FILE ...
  - Creates the file **ONLY**. Does not allocate any memory for it as it has no data written into it.
  - ... means multiple files can be created at once
- mv SOURCE DEST
  - Renames SOURCE to DEST **ONLY.**
  - Nothing else. Really.
- cat FILE ... [-w/a OUTPUT_FILE]
  - Read contents of FILE(s) and overwrite/append to OUTPUT_FILE
- cp -h
  - Your HOST OS is files in your **docker container**
  - Everything else are files in **your file system** (pikachu)
- chmod
  - Is included too!

Some More Clarifications...

- name[0]
  - This is the INTEGER 0 (0x00) not ASCII 0 (0x30)
  - What is 1, what is 2?
- file type
  - What is 0: Unknown, 4: Symbolic Link?
- default permissions
  - Follow UNIX! Read&Write is appropriate here
- Do we mmap FAT only or the entire Filesystem?
  - Up to you. Both ways are valid
- What if ...?
  - Up to you!

TL;DR

1. Specifications should be followed. (Read the write-up carefully!)
2. When in doubt, follow UNIX behaviors
3. Implementation details are **100% up to you!**
    a. If you think it is appropriate, go ahead!

THIS IS YOUR MILESTONE!

What's After?

- PennOS and PennFAT Interaction
- f_functions
  - These are your own **system calls!**
  - These provide the connection between PennOS Shell and your File System
- You may use functionalities you implemented in standalone PennFAT to implement f_functions
- You MUST use f_functions to run ANY user-level functions like cat, echo, touch redirections, etc.

Any Questions?