CSE312 - Operating Systems - HW#4

Fatih Kaan Salgır - 171044009

Design Explanation

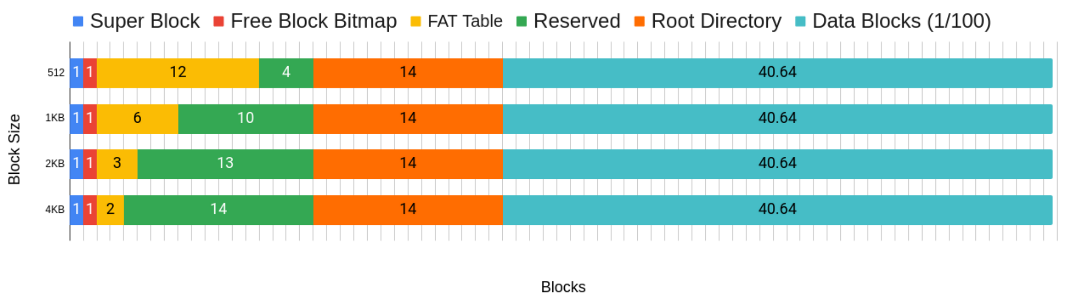


Figure 1: Disk block alignments according to different block sizes.

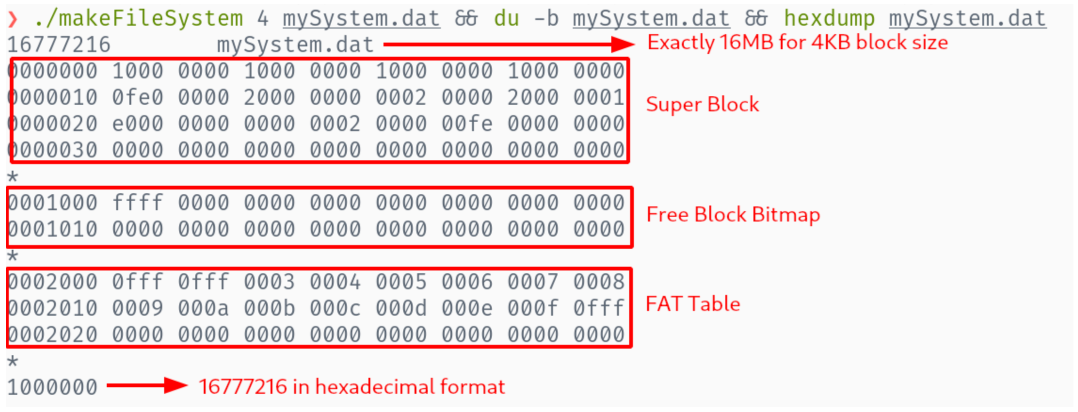
Since there are lots of data blocks, data block area in the figure represents only 1% of the whole data blocks in the disk.

`filesystem.h` has the functions, constans and variables that is used in both `makeFileSystem.c` and `fileSystemOper.c`. Global variables inside `filesystem.h`;

```
extern FILE *fp;
extern struct super_block super_blk;
extern uint32_t free_bitmap[BITMAP_WORD_SIZE];
extern uint12 fat_table[NO_BLOCKS];
```

Creating an empty file system

To test `makeFileSystem`, disk file in examined with `hexdump` command in linux. First part is the contents of the super block. The second is Free Block Bitmap is the first blocks of the bitmap is marked as full, because root directory is marked as full in bitmap. And linked one after another in FAT Table.



```
❯ ./makeFileSystem 4 mySystem.dat && du -b mySystem.dat && hexdump mySystem.dat
16777216        mySystem.dat
0000000 1000 0000 1000 0000 1000 0000 1000 0000
0000010 0fe0 0000 2000 0000 0002 0000 2000 0001
0000020 e000 0000 0000 0002 0000 00fe 0000 0000
0000030 0000 0000 0000 0000 0000 0000 0000 0000
*
0001000 ffff 0000 0000 0000 0000 0000 0000 0000
0001010 0000 0000 0000 0000 0000 0000 0000 0000
*
0002000 0fff 0fff 0003 0004 0005 0006 0007 0008
0002010 0009 000a 000b 000c 000d 000e 000f 0fff
0002020 0000 0000 0000 0000 0000 0000 0000 0000
*
1000000
```

Exactly 16MB for 4KB block size
Super Block
Free Block Bitmap
FAT Table
16777216 in hexadecimal format

Figure 2: Testing `makeFileSystem` with 4KB block size

Super Block

Super block holds information about;

- block size,
- number of blocks,
- where the free bitmap, fat table root directory and data blocks starts and their sizes
- number of free blocks

```c
// filesystem.h
struct super_block {
  uint32_t block_size;
  uint32_t block_count;
  uint32_t free_bitmap_start;
  uint32_t free_block_size;
  uint32_t no_free_blocks;
  uint32_t fat_table_start;
  uint32_t fat_table_block_size;
  uint32_t root_start;
  uint32_t root_size;
  uint32_t data_start;
  uint32_t data_size;
};
```

Free Blocks

There are about 2^{12} data blocks (some of the blocks are reserved). In bitmap representation every data block is mapped to a bit. Therefore there must be 2^{12} bits in bitmap.

$$2^{12} \text{ bits} = \frac{2^{12}}{2^3} \text{ bytes} = 512 \text{ bytes}$$

The bitmap fits inside a block, because the minimum block size is 512 bytes. During implementation of the filesytem bit maniplulation is needed. Bitmap impelemented as `uint32_t` array with size of 128 which is 2^{12} bits.

`get_bit()`, `set_bit()` and `clear_bit()` functions are implemented to abstract away the implementation details.

FAT Table

There are 2^{12} entries and every entry is 12 bits. Therefore the table size;

$$\frac{2^{12} \times 12}{2^3} = 6 \times 2^{10} \text{ Bytes} = 6 \text{ KB}$$

So the number of blocks needed for FAT table depends on the block size. If the block size is 512 Bytes, then 12 blocks needed. On the other hand if the block size is 4 KB, then 1.5 blocks will be enough. To align properly numbers completed to integers. Final table is;

Table 1: Number of blocks needed for FAT table, according to different block size

Block Size	FAT-12	No blocks needed
0.5 KB	2MB	12
1 KB	4MB	6
2 KB	8MB	3
4 KB	16MB	2

When the file is long enough to exceed the capacity of a single block, a new available block will be found with help of bitmap. Current entry of the FAT Table linked to this new block. And the new block will get the value -1 (0xFFF).

Same thing holds when a directory run out of directory entries (Number of directory entry of a block can hold varies depending on the block size).

```c
// filesystem.h
struct uint12 {
  uint16_t x : 12;
};
typedef struct uint12 uint12;
extern uint12 fat_table[NO_BLOCKS];
```

Directory Entry

32 bytes used for each entry. Address is the block number of the first block. Attributes shows if the entry is file or directory. If the entry is a file then size shows how many bytes the file size. However if the entry is a directory, the size shows how many files/folders inside the directory.

Size of the directory can be reperesented with 3 bytes, since it doesn't exceed maximum file size in the file system. In order to store the size in 3 bytes, array with size 3 and `uint8_t` type is used. `pack3b()` and `unpack3b()` functions are used to fit 32 bit integer into 24 bits.

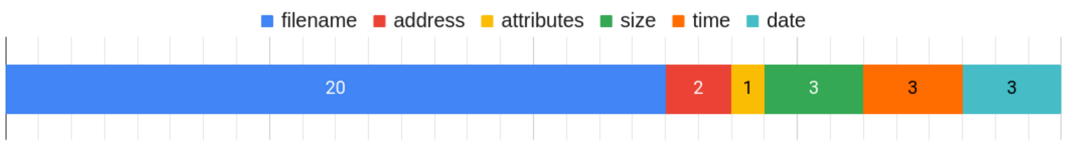


Figure 3: Directory entries - number of bytes used

```
struct dir_entry {
  char name[20];            // 20 bytes
  uint16_t address;         // 2 bytes FAT-12 (used only least significant 12 bits)
  uint8_t attr;             // 1 byte
  uint8_t size[3];          // 3 bytes : max 16 MB size
  uint8_t time[3];          // 3 bytes : HMS
  uint8_t date[3];          // 3 bytes : DMY (Y: 1900+)
} __attribute__((packed)); // required for struct to be exactly 32 bytes

typedef struct dir_entry dir_entry;
```

Running & Test Results

File system manipulation functions are implemented in `fileSystemOper.c`.

Table 2: Function names for corresponding operations

Operation	Function	Notes
dir	`fs_dir()`	- prints error if the target is not a directory
mkdir	`fs_mkdir()`	- prints error if the prior path doesn't exist - prints error if the directory already exist
rmdir	`fs_rmdir()`	- prints error if the target is not a directory - prints error if the directory has files/directories inside
dumpe2fs	`dumpe2fs()`	- prints superblock, free bitmap, occupied blocks
write	`fs_write()`	- if file exists in the file system, overwrites - if the linux file doesn't exist, prints error
read	`fs_read()`	- if the linuxfile exits, overwrites
del	`fs_del()`	- if the target is a directory, it will print error

Test cases in the assignment pdf:

- Long file content is used to demonstrate multiple number of blocks used for a single file.
- Middle part of the `dumpe2fs` is trunctaed for readibility.

```
> ./makeFileSystem 0.5 mySystem.data
> ./fileSystemOper mySystem.data mkdir '\usr'
> ./fileSystemOper mySystem.data mkdir '\usr\ysa'
> ./fileSystemOper mySystem.data mkdir '\bin\ysa'
ERROR: Path is incorrect, couldn't find: bin
> echo 'Lorem ipsum dolor sit amet, consetetur sadipscing elitr, sed diam nonumy eirmod tempor
↪  invidunt ut labore et dolore magna aliquyam erat, sed diam voluptua. At vero eos et accusam
↪  et justo duo dolores et ea rebum. Stet clita kasd gubergren, no sea takimata sanctus est
↪  Lorem ipsum dolor sit amet.

Lorem ipsum dolor sit amet, consetetur sadipscing elitr, sed diam nonumy eirmod tempor invidunt
↪  ut labore et dolore magna aliquyam erat, sed diam voluptua. At vero eos et accusam et justo
↪  duo dolores et ea rebum. Stet clita kasd gubergren, no sea takimata sanctus est Lorem ipsum
↪  dolor sit amet.' > linuxFile.data
> ./fileSystemOper mySystem.data write '\usr\file2' linuxFile.data
> ./fileSystemOper mySystem.data write '\file3' linuxFile.data
> ./fileSystemOper mySystem.data dir '\'
26-05-2021  19:18  <DIR>         usr
26-05-2021  19:18           594  file3
         1 File(s)        594 bytes
         1 Dir(s)    2076672 bytes free
> ./fileSystemOper mySystem.data del '\usr\ysa\file1'
> ./fileSystemOper mySystem.data dump2fs
==== SUPERBLOCK ====
Block Size:            512
Block Count:           4096
Free Bitmap Start:     512
Free Blocks Size:      4096
Numer of Free Blocks:  4057
FAT Table Start:       4608
FAT Table Block Size:  12
Root Start:            12800
Root Size:             7168
Data Start:            19968
Data Size:             2080768
====== BITMAP ======
    0 : 1111111111111111 1101111100000000 0000000000000000 0000000000000000
   64 : 0000000000000000 0000000000000000 0000000000000000 0000000000000000
  128 : 0000000000000000 0000000000000000 0000000000000000 0000000000000000
  192 : 0000000000000000 0000000000000000 0000000000000000 0000000000000000
...(truncated)
 3968 : 0000000000000000 0000000000000000 0000000000000000 0000000000000000
 4032 : 0000000000000000 0000000000000000 0000000000000000 0000000000000000
====== FILENAME : OCCUPIED BLOCKS ======
usr            : 16
ysa            : 17
file2          : 20 21
file3          : 22 23
Total number of directories: 2
Total number of files: 2
> ./fileSystemOper mySystem.data read '\usr\file2' linuxFile2.data
> cmp linuxFile2.data linuxFile.data
```