

# Основы построения файловых систем



## Сегодня мы рассмотрим устройство ФС FAT16

Эта ФС использовалась для дискет и небольших жёстких дисков. Примитивная, но простая.

Немного терминологии:

- FAT – File Allocation Table,
- Sector – минимальный блок данных, который диск может прочесть или записать (512 байт для наших примеров),
- Cluster – несколько подряд идущих секторов; в FAT является минимальной единицей места, выделяемого под файл.

## Два способа записать целое число в память или на диск

## Два способа записать целое число в память или на диск

В начале идут старшие байты (**Big-endian**)

u32 x = 0x1A2B3C4D;

На диске:

1A 2B 3C 4D | . . . . . | ..  
 нумерация байтов на диске


Используется в:

- PowerPC
- Itanium

# Два способа записать целое число в память или на диск

В начале идут старшие байты (**Big-endian**)

u32 x = 0x1A2B3C4D;


На диске:  
1A 2B 3C 4D | . . . . . | ..  
 нумерация байтов на диске

Используется в:

- PowerPC
- Itanium

В начале идут младшие байты (**little-endian**)

u32 x = 0x1A2B3C4D;

На диске:  
4D 3C 2B 1A | . . . . . | ..  
 нумерация байтов на диске

Используется в:


- x86

# Два способа записать целое число в память или на диск

В начале идут старшие байты (**Big-endian**)

u32 x = 0x1A2B3C4D;

На диске:

1A 2B 3C 4D | . . . . . | ..  
 нумерация байтов на диске


Используется в:

- PowerPC
- Itanium

В начале идут младшие байты (**little-endian**)

u32 x = 0x1A2B3C4D;

На диске:

4D 3C 2B 1A | . . . . . | ..  
 нумерация байтов на диске

Используется в:

- x86

Примечание: PowerPC, Itanium, ARM, MIPS на самом деле bi-endian, т.е. умеют работать как с little-endian, так и big-endian данными.

## Как обмениваться данными между little-endian и big-endian системами?

При сохранении преобразовать данные из host byte order в некоторый фиксированный:

```
dmap_ext_t ext = {
    .slice_id = it->last_slice_id, .wr_seq = UINT64_MAX, .item_id = item_id,
    .ext = { .offs = offs < max_ext_len ? 0 : (offs - max_ext_len), .len = 0 }
};
struct dmap_ext_ondisk dsk;
dmap_ext2ondisk(&dsk, &ext);
.....

void dmap_ext2ondisk(struct dmap_ext_ondisk *dsk, const dmap_ext_t *ext)
{
    dsk->wr_seq = cpu_to_be64(ext->wr_seq);
    dsk->slice_id = cpu_to_be32(ext->slice_id);
    dsk->item_id = cpu_to_be64(ext->item_id);
    dsk->ext_offs = cpu_to_be64(ext->ext.offs);

    /* pack extent len and deleted bit into 3 bytes */
    u32 len = ext->ext.len;
    dsk->ext_len[0] = (len >> 16) & 0xFF;
    dsk->ext_len[1] = (len >> 8) & 0xFF;
    dsk->ext_len[2] = len & 0xFF;
}
```

При чтении данных проделать обратное преобразование.

## Как обмениваться данными между little-endian и big-endian системами?

Определение struct dmap\_ext\_ondisk

```
struct dmap_ext_ondisk {  
    be64      item_id;  
    be64      ext_offs;  
    u8        ext_len[3];  
    be32      slice_id;  
    be64      wr_seq;  
} __attribute__((packed));
```



# Как обмениваться данными между little-endian и big-endian системами?

Определение struct dmap\_ext\_ondisk

```
struct dmap_ext_ondisk {  
    be64      item_id;  
    be64      ext_offs;  
    u8        ext_len[3];  
    be32      slice_id;  
    be64      wr_seq;  
} __attribute__((packed));
```

Более простой способ

```
struct dmap_ext_ondisk {  
    long long  item_id;  
    long long  ext_offs;  
    char       ext_len[3];  
    int        slice_id;  
    long long  wr_seq;  
}
```

# Как обмениваться данными между little-endian и big-endian системами?

Определение struct dmap_ext_ondisk	Более простой способ
<pre>struct dmap_ext_ondisk {     be64      item_id;     be64      ext_offs;     u8        ext_len[3];     be32      slice_id;     be64      wr_seq; } __attribute__((packed));</pre>	<pre>struct dmap_ext_ondisk {     long long  item_id;     long long  ext_offs;     char       ext_len[3];     int        slice_id;     long long  wr_seq; }</pre>

## Как структуры будут выглядеть в памяти на x86\_64?

8 байт	item_id
8 байт	ext_offs
3 байта	ext_len
8 байт	wr_seq
4 байта	slice_id

# Как обмениваться данными между little-endian и big-endian системами?

Определение struct dmap\_ext\_ondiskБолее простой способ

```
struct dmap_ext_ondisk {
    be64      item_id;
    be64      ext_offs;
    u8        ext_len[3];
    be32      slice_id;
    be64      wr_seq;
} __attribute__((packed));
```

```
struct dmap_ext_ondisk {
    long long item_id;
    long long ext_offs;
    char      ext_len[3];
    int       slice_id;
    long long wr_seq;
}
```

Как структуры будут выглядеть в памяти на x86\_64?

8 байт	item_id	8 байт	item_id
8 байт	ext_offs	8 байт	ext_offs
3 байта	ext_len	3 байта	ext_len
8 байт	wr_seq	5 байт	padding
4 байта	slice_id	8 байт	wr_seq
		4 байта	slice_id
		4 байта	padding

# Как обмениваться данными между little-endian и big-endian системами?

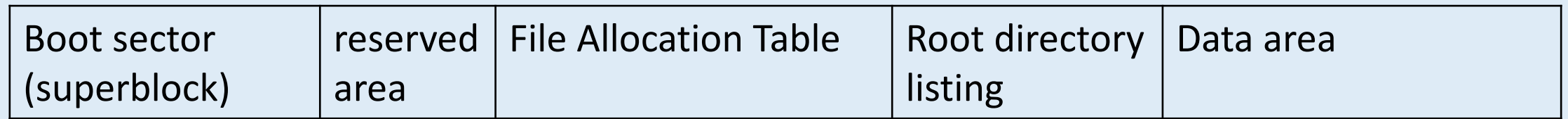
Определение struct dmap\_ext\_ondiskБолее простой способ


```
struct dmap_ext_ondisk {
    be64      item_id;
    be64      ext_offs;
    u8        ext_len[3];
    be32      slice_id;
    be64      wr_seq;
} __attribute__((packed));
```

```
struct dmap_ext_ondisk {
    long long item_id;
    long long ext_offs;
    char      ext_len[3];
    int       slice_id;
    long long wr_seq;
}
```

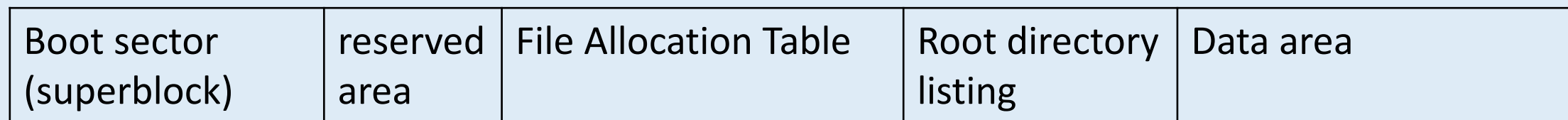
Как структуры будут выглядеть в памяти на x86_64?				А как на x86_32?	
8 байт	item_id	8 байт	item_id	4 байта	item_id
8 байт	ext_offs	8 байт	ext_offs	4 байта	ext_offs
3 байта	ext_len	3 байта	ext_len	3 байта	ext_len
8 байт	wr_seq	5 байт	padding	1 байт	padding
4 байта	slice_id	8 байт	wr_seq	4 байта	wr_seq
		4 байта	slice_id	4 байта	slice_id
		4 байта	padding		

## Общий вид диска с FAT16



 от младших адресов к старшим

## Общий вид диска с FAT16

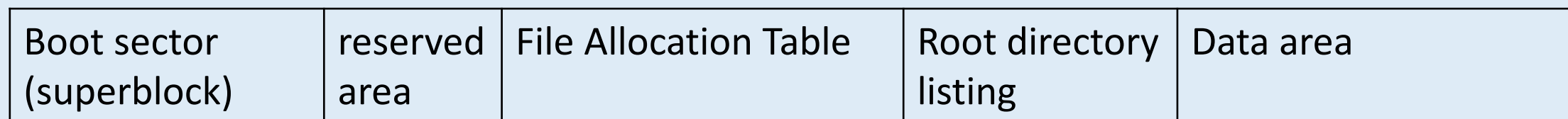


 от младших адресов к старшим

Superblock хранит данные об ФС в целом:

- Размер ФС,
- Размер кластера,
- Положение root directory listing,
- ...

## Общий вид диска с FAT16



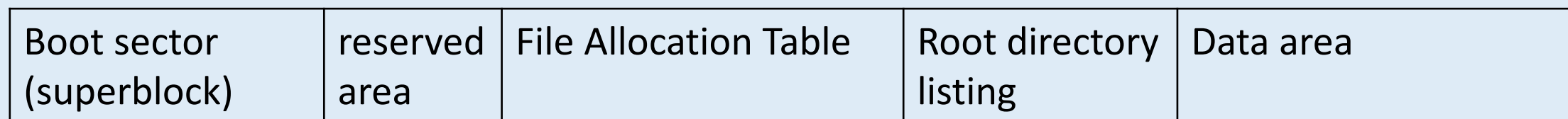
 от младших адресов к старшим

Superblock хранит данные об ФС в целом:

- Размер ФС,
- Размер кластера,
- Положение root directory listing,
- ...

File Allocation Table представляет собой множество односвязных списков кластеров; каждый список описывает один файл.

## Общий вид диска с FAT16



 от младших адресов к старшим

Superblock хранит данные об ФС в целом:

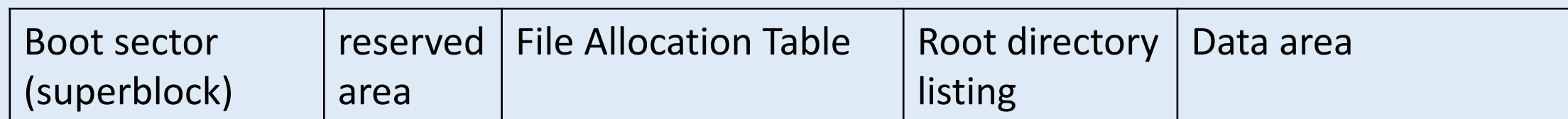
- Размер ФС,
- Размер кластера,
- Положение root directory listing,
- ...

File Allocation Table представляет собой множество односвязных списков кластеров; каждый список описывает один файл.

Root directory listing содержит список элементов в корневом каталоге (он выделяется особо, поскольку в ранних версиях FAT некорневых каталогов не было).



## Общий вид диска с FAT16



 от младших адресов к старшим

Superblock хранит данные об ФС в целом:

- Размер ФС,
- Размер кластера,
- Положение root directory listing,
- ...

File Allocation Table представляет собой множество односвязных списков кластеров; каждый список описывает один файл.

Root directory listing содержит список элементов в корневом каталоге (он выделяется особо, поскольку в ранних версиях FAT некорневых каталогов не было).

Data area состоит из кластеров, в которых записано содержимое файлов; порядок, в котором кластеры соответствуют файлам, задаёт File Allocation Table.

## FAT16 boot sector (/usr/include/linux/msdos\_fs.h)

```
struct fat_boot_sector {
    __u8    ignored[3];    /* Boot strap short or near jump */
    __u8    system_id[8];  /* Name - can be used to special case
                           partition manager volumes */
    __u8    sector_size[2]; /* bytes per logical sector */
    __u8    sec_per_clus;  /* sectors/cluster */
    __le16  reserved;     /* reserved sectors */
    __u8    fats;          /* number of FATs */
    __u8    dir_entries[2]; /* root directory entries */
    __u8    sectors[2];    /* number of sectors */
    __u8    media;         /* media code */
    __le16  fat_length;    /* sectors/FAT */
    __le16  secs_track;    /* sectors per track */
    __le16  heads;         /* number of heads */
    __le32  hidden;        /* hidden sectors (unused) */
    __le32  total_sect;    /* number of sectors (if sectors == 0) */

    struct {
        /* Extended BPB Fields for FAT16 */
        __u8    drive_number; /* Physical drive number */
        __u8    state;        /* undocumented, but used
                               for mount state. */
        __u8    signature; /* extended boot signature */
        __u8    vol_id[4];  /* volume ID */
        __u8    vol_label[11]; /* volume label */
        __u8    fs_type[8]; /* file system type */
        /* other fields are not added here */
    } fat16;
};
```

FAT16 boot sector (отсутпление)

00000000	eb 3c 90 6d 6b 66 73 2e	66 61 74 00 02 04 01 00	.<.mkfs.fat.....
00000010	02 00 02 00 20 f8 06 00	20 00 40 00 00 00 00 00	.... .@.....
00000020	00 00 00 00 80 00 29 20	55 62 09 4e 4f 20 4e 41	.....) Ub.NO NA
00000030	4d 45 20 20 20 20 46 41	54 31 32 20 20 20 0e 1f	ME FAT12 ..
00000040	be 5b 7c ac 22 c0 74 0b	56 b4 0e bb 07 00 cd 10	.[".t.V.....
00000050	5e eb f0 32 e4 cd 16 cd	19 eb fe 54 68 69 73 20	^..2.....This
00000060	69 73 20 6e 6f 74 20 61	20 62 6f 6f 74 61 62 6c	is not a bootabl
00000070	65 20 64 69 73 6b 2e 20	20 50 6c 65 61 73 65 20	e disk. Please
00000080	69 6e 73 65 72 74 20 61	20 62 6f 6f 74 61 62 6c	insert a bootabl
00000090	65 20 66 6c 6f 70 70 79	20 61 6e 64 0d 0a 70 72	e floppy and..pr
000000a0	65 73 73 20 61 6e 79 20	6b 65 79 20 74 6f 20 74	ess any key to t
000000b0	72 79 20 61 67 61 69 6e	20 2e 2e 2e 20 0d 0a 00	ry again ... ..
000000c0	00 00 00 00 00 00 00 00	00 00 00 00 00 00 00 00	.....
*			
000001f0	00 00 00 00 00 00 00 00	00 00 00 00 00 00 55 aa	.....U.
00000200	f8 ff ff 00 00 00 00 00	00 00 00 00 00 00 00 00	.....

# FAT16 boot sector (отсутпление)

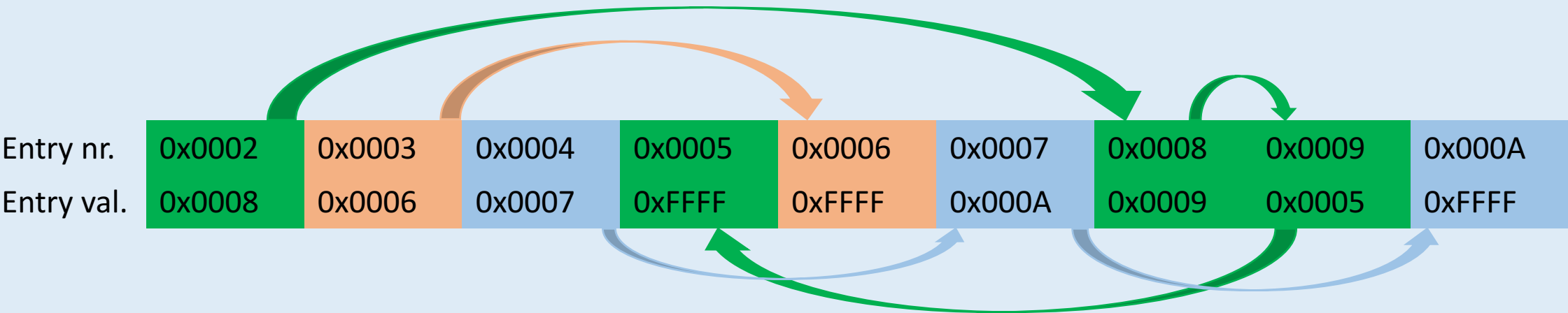
00000000	eb 3c 90 6d 6b 66 73 2e	66 61 74 00 02 04 01 00	.<.mkfs.fat.....
00000010	02 00 02 00 20 f8 06 00	20 00 40 00 00 00 00 00	.... .@.....
00000020	00 00 00 00 80 00 29 20	55 62 09 4e 4f 20 4e 41	.....) Ub.NO NA
00000030	4d 45 20 20 20 20 46 41	54 31 32 20 20 20 0e 1f	ME FAT12 ..
00000040	be 5b 7c ac 22 c0 74 0b	56 b4 0e bb 07 00 cd 10	.[".t.V.....
00000050	5e eb f0 32 e4 cd 16 cd	19 eb fe 54 68 69 73 20	^..2.....This
00000060	69 73 20 6e 6f 74 20 61	20 62 6f 6f 74 61 62 6c	is not a bootabl
00000070	65 20 64 69 73 6b 2e 20	20 50 6c 65 61 73 65 20	e disk. Please
00000080	69 6e 73 65 72 74 20 61	20 62 6f 6f 74 61 62 6c	insert a bootabl
00000090	65 20 66 6c 6f 70 70 79	20 61 6e 64 0d 0a 70 72	e floppy and..pr
000000a0	65 73 73 20 61 6e 79 20	6b 65 79 20 74 6f 20 74	ess any key to t
000000b0	72 79 20 61 67 61 69 6e	20 2e 2e 2e 20 0d 0a 00	ry again ... ..
000000c0	00 00 00 00 00 00 00 00	00 00 00 00 00 00 00 00	.....
*			
000001f0	00 00 00 00 00 00 00 00	00 00 00 00 00 00 55 aa	.....U.
00000200	f8 ff ff 00 00 00 00 00	00 00 00 00 00 00 00 00	.....

В первых трёх байтах стоит  
jmp 0x3e  
nop

Первый jmp прыгает через суперблок в код, который напечатает “this is not a bootable disk blah-blah-blah”.  
Почему так – узнаем позже, когда будем говорить про загрузку компьютера, MBR и GPT.

# File Allocation Table: массив из 16-битных чисел-номеров секторов

Если мы знаем номер кластера, принадлежащего файлу, то FAT позволяет определить номер следующего кластера:



Тут представлены три файла:

- 1. Состоит из секторов 2, 8, 9, 5 (в таком порядке),
- 2. Состоит из секторов 3 и 6,
- 3. Состоит из секторов 4, 7, и A.

## FAT16 root directory listing

Содержимое корневого каталога представляется в виде массива 32-байтных записей следующего формата:

```
struct msdos_dir_entry {  
    __u8    name[MSDOS_NAME]; /* name and extension */  
    __u8    attr;              /* attribute bits */  
    __u8    lcase;            /* Case for base and extension */  
    __u8    ctime_cs;         /* Creation time, centiseconds (0-199) */  
    __le16   ctime;           /* Creation time */  
    __le16   cdate;           /* Creation date */  
    __le16   adate;           /* Last access date */  
    __le16   starthi;         /* High 16 bits of cluster in FAT32 */  
    __le16   time,date,start; /* time, date and first cluster */  
    __le32   size;            /* file size (in bytes) */  
}
```

## FAT16 root directory listing

Содержимое корневого каталога представляется в виде массива 32-байтных записей следующего формата:

```
struct msdos_dir_entry {
    __u8    name[MSDOS_NAME]; /* name and extension */
    __u8    attr;              /* attribute bits */
    __u8    lcase;             /* Case for base and extension */
    __u8    ctime_cs;          /* Creation time, centiseconds (0-199) */
    __le16  ctime;             /* Creation time */
    __le16  cdate;             /* Creation date */
    __le16  adate;             /* Last access date */
    __le16  starthi;           /* High 16 bits of cluster in FAT32 */
    __le16  time,date,start; /* time, date and first cluster */
    __le32  size;              /* file size (in bytes) */
}
```

Поле **name** содержит имя, компоненты которого дополнены пробелами: “prog . . . . c . .” вместо “prog.c”.

## FAT16 root directory listing

Содержимое корневого каталога представляется в виде массива 32-байтных записей следующего формата:

```
struct msdos_dir_entry {
    __u8    name[MSDOS_NAME]; /* name and extension */
    __u8    attr;              /* attribute bits */
    __u8    lcase;             /* Case for base and extension */
    __u8    ctime_cs;          /* Creation time, centiseconds (0-199) */
    __le16  ctime;             /* Creation time */
    __le16  cdate;             /* Creation date */
    __le16  adate;             /* Last access date */
    __le16  starthi;           /* High 16 bits of cluster in FAT32 */
    __le16  time,date,start; /* time, date and first cluster */
    __le32  size;              /* file size (in bytes) */
}
```

Поле **name** содержит имя, компоненты которого дополнены пробелами: “prog . . . . c . .” вместо “prog.c”.

У удалённых файлов первая буква имени заменяется на 0xE5.



## FAT16 root directory listing

Содержимое корневого каталога представляется в виде массива 32-байтных записей следующего формата:

```
struct msdos_dir_entry {
    __u8    name[MSDOS_NAME]; /* name and extension */
    __u8    attr;              /* attribute bits */
    __u8    lcase;             /* Case for base and extension */
    __u8    ctime_cs;          /* Creation time, centiseconds (0-199) */
    __le16  ctime;             /* Creation time */
    __le16  cdate;             /* Creation date */
    __le16  adate;             /* Last access date */
    __le16  starthi;           /* High 16 bits of cluster in FAT32 */
    __le16  time,date,start; /* time, date and first cluster */
    __le32  size;              /* file size (in bytes) */
}
```

Поле **name** содержит имя, компоненты которого дополнены пробелами: “prog . . . . c . .” вместо “prog.c”.

У удалённых файлов первая буква имени заменяется на 0xE5.

Если имя начинается на 0x00, то это признак конца каталога.

## FAT16 root directory listing

Содержимое корневого каталога представляется в виде массива 32-байтных записей следующего формата:

```
struct msdos_dir_entry {
    __u8    name[MSDOS_NAME]; /* name and extension */
    __u8    attr;              /* attribute bits */
    __u8    lcase;             /* Case for base and extension */
    __u8    ctime_cs;          /* Creation time, centiseconds (0-199) */
    __le16  ctime;             /* Creation time */
    __le16  cdate;             /* Creation date */
    __le16  adate;             /* Last access date */
    __le16  starthi;           /* High 16 bits of cluster in FAT32 */
    __le16  time,date,start;    /* time, date and first cluster */
    __le32  size;              /* file size (in bytes) */
}
```

Поле **name** содержит имя, компоненты которого дополнены пробелами: “prog . . . . с . .” вместо “prog.c”.

У удалённых файлов первая буква имени заменяется на 0xE5.

Если имя начинается на 0x00, то это признак конца каталога.

Атрибуты **attr**: read only (bit 0), hidden (bit 1), system, volume label, subdirectory, archive; биты 6 и 7 не используются.

## Соберём всё вместе: как прочесть файл с FAT16

1. Прочесть root directory listing, отыскать файл с заданным именем,
2. Запомнить  $i := \text{dir\_entry} \rightarrow \text{start}$  – номер первого кластера в файле,
3. Прочесть кластер  $i$ ,
4. В FAT прочесть  $i$ -й элемент – это будет следующий кластер файла,
5. Повторять #3 и #4, пока не прочтём 0xFFFF из FAT.

## Расширения FAT16

Подкаталоги: хранятся как обычные файлы; содержат, как и корневой каталог, массив из `struct msdos_dir_entry`.

## Расширения FAT16

Подкаталоги: хранятся как обычные файлы; содержат, как и корневой каталог, массив из `struct msdos_dir_entry`.

В FAT32 и корневой каталог хранится как файл – это позволяет не ограничивать его в размере.

## Расширения FAT16

Подкаталоги: хранятся как обычные файлы; содержат, как и корневой каталог, массив из `struct msdos_dir_entry`.

В **FAT32** и корневой каталог хранится как файл – это позволяет не ограничивать его в размере.

**VFAT**: длинные имена у файлов

Вместо одного `struct msdos_dir_entry` в каталоге хранится много таких записей, в каждой хранится часть имени.

У всех записей, кроме последней, `entry->attr` содержит `0xF` (невозможное значение), в последней хранится короткое имя в формате 8.3.

## Расширения FAT16

Подкаталоги: хранятся как обычные файлы; содержат, как и корневой каталог, массив из `struct msdos_dir_entry`.

В **FAT32** и корневой каталог хранится как файл – это позволяет не ограничивать его в размере.

**VFAT**: длинные имена у файлов

Вместо одного `struct msdos_dir_entry` в каталоге хранится много таких записей, в каждой хранится часть имени. У всех записей, кроме последней, `entry->attr` содержит `0xF` (невозможное значение), в последней хранится короткое имя в формате 8.3.

Больше деталей можно почитать тут:

- <https://www.win.tue.nl/~aeb/linux/fs/fat/fat-1.html>
- <http://www.tavi.co.uk/phobos/fat.html>
- <http://lxr.free-electrons.com/source/fs/fat/>

## Домашнее задание

На разделе FAT16 расположен файл длиной 1024 кластера, кластеры которого идут подряд. Один кластер имеет размер 1024 байта.

Сколько времени потребуется (для типичного HDD), чтобы прочесть этот файл в следующих случаях:

1. Чтение выполняется по алгоритму из лекции (прочли кластер, посмотрели номер следующего, прочли его, etc.),
2. Содержимое FAT зачитывается в память целиком, формируются большие запросы на чтение данных, эти запросы исполняются.



## Домашнее задание

На разделе FAT16 расположен файл длиной 1024 кластера, кластеры которого идут подряд. Один кластер имеет размер 1024 байта.

Сколько времени потребуется (для типичного HDD), чтобы прочесть этот файл в следующих случаях:

1. Чтение выполняется по алгоритму из лекции (прочли кластер, посмотрели номер следующего, прочли его, etc.),
2. Содержимое FAT зачитывается в память целиком, формируются большие запросы на чтение данных, эти запросы исполняются.

Разберитесь с `mkfs.vfat`, создайте образ диска с FAT16. Примонтируйте этот образ и создайте в нём несколько файлов.

Теперь напишите программу, которая

- Распечатывает список файлов в корневом каталоге,
- (\*) Распечатывает список файлов и напротив каждого пишет атрибуты и время создания/изменения,
- (\*) Читает файл, сохранённый в образе FAT16, и печатает его в `stdin`.

(\*) Напишите программу, которая умеет показать список элементов в подкаталоге.

(\*) Поддержите FAT32 в программе, которая печатает список элементов каталога.

(\*\*\*) Напишите программу, которая с помощью FUSE монтирует ФС, содержимое которой берётся из файла с образом FAT16.