

Основы построения файловых систем



Сегодня мы рассмотрим устройство ФС ext2

Немного терминологии:

- Block – несколько подряд идущих секторов; в ext2 является минимальной единицей места, выделяемого под файл.
- Inode (index node) – структура, описывающая, как расположен на диске один файл.

Index nodes (src/linux/fs/ext2/ext2.h)

В ext2 информация о свойствах файла и его расположении на диске локализована в самом файле:

```
struct ext2_inode {  
    __le16 i_mode;      /* File mode */  
    __le16 i_uid;       /* Low 16 bits of Owner Uid */  
    __le32 i_size;      /* Size in bytes */  
    __le32 i_atime;     /* Access time */  
    __le32 i_ctime;     /* Creation time */  
    __le32 i_mtime;     /* Modification time */  
    __le32 i_dtime;     /* Deletion Time */  
    __le16 i_gid;       /* Low 16 bits of Group Id */  
    __le16 i_links_count; /* Links count */  
};
```

Index nodes (src/linux/fs/ext2/ext2.h)

В ext2 информация о свойствах файла и его расположении на диске локализована в самом файле:

```
__le32  i_blocks;    /* Blocks count */
__le32  i_flags;     /* File flags */
__le32  osd1;
__le32  i_block[EXT2_N_BLOCKS]; /* Pointers to blocks */
__le32  i_generation; /* File version (for NFS) */
__le32  i_file_acl;  /* File ACL */
__le32  i_dir_acl;   /* Directory ACL */
__le32  i_faddr;     /* Fragment address */
__le8  osd2[12];
};
```

Index nodes (src/linux/fs/ext2/ext2.h)

В `ext2_inode->i_block[]` хранится список блоков, которые составляют файл.

Index nodes (src/linux/fs/ext2/ext2.h)

В `ext2_inode->i_block[]` хранится список блоков, которые составляют файл. Но в этом массиве 15 элементов. Как быть с файлами, которые длиннее 15 блоков?

Index nodes (src/linux/fs/ext2/ext2.h)

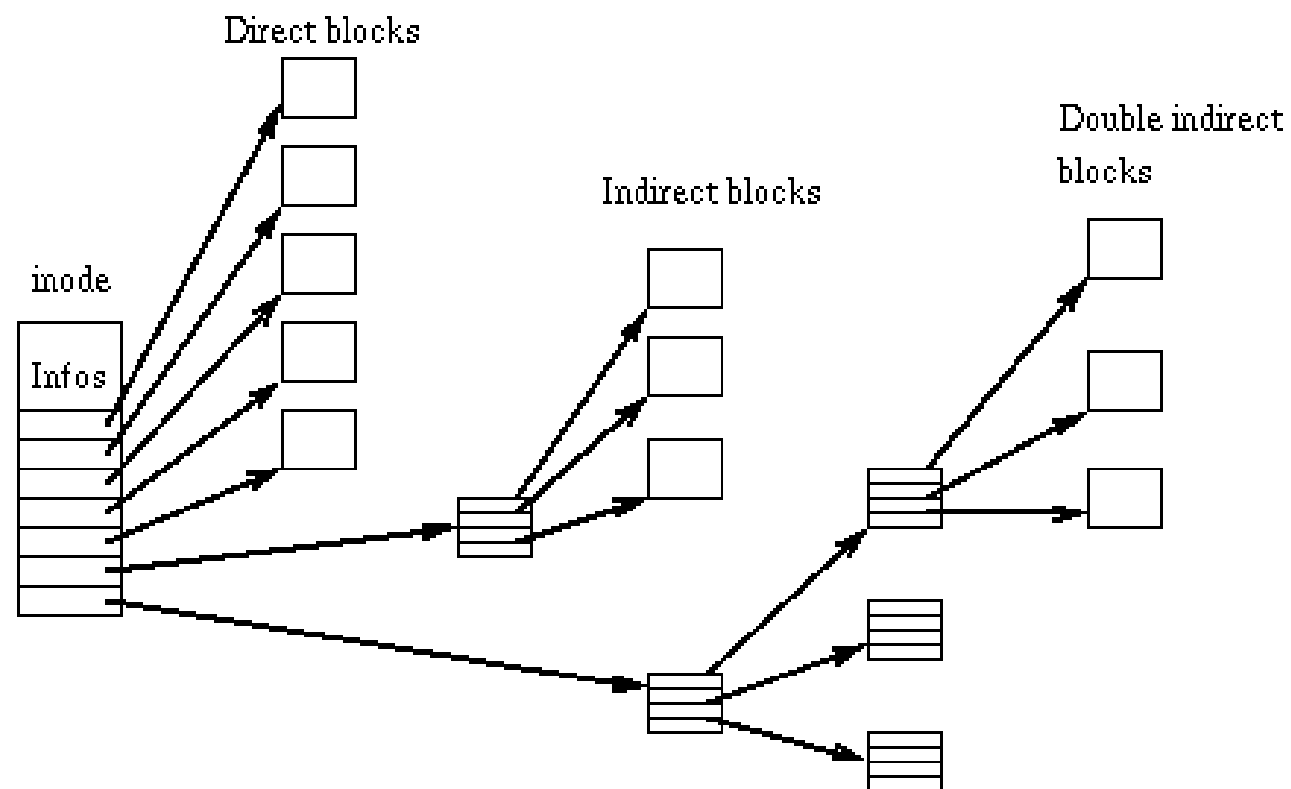
В `ext2_inode->i_block[]` хранится список блоков, которые составляют файл. Но в этом массиве 15 элементов. Как быть с файлами, которые длиннее 15 блоков?

Последние три элемента в `i_block[]` косвенные, т.е. указывают на блоки, которые сами являются списками блоков.

Index nodes (src/linux/fs/ext2/ext2.h)

В `ext2_inode->i_block[]` хранится список блоков, которые составляют файл. Но в этом массиве 15 элементов. Как быть с файлами, которые длиннее 15 блоков?

Последние три элемента в `i_block[]` косвенные, т.е. указывают на блоки, которые сами являются списками блоков. Они имеют уровни косвенности 1, 2 и 3, соответственно.



Каталоги в ext2

Каталог хранится в файле специального типа (у которого младший байт `i_mode` равен `EXT2_FT_DIR`).

Содержимое файла представляет собой последовательность записей переменной длины:

- В заголовке записи стоит

```
struct ext2_dir_entry_2 {  
    __le32  inode;           /* Inode number */  
    __le16  rec_len;         /* Directory entry length */  
    __u8    name_len;        /* Name length */  
    __u8    file_type;       /* File type */  
    char    name[];          /* File name, up to EXT2_NAME_LEN */  
};
```

- После `struct ext2_dir_entry_2` следует имя файла.

Каталоги в ext2

Каталог хранится в файле специального типа (у которого младший байт `i_mode` равен `EXT2_FT_DIR`).

Содержимое файла представляет собой последовательность записей переменной длины:

- В заголовке записи стоит

```
struct ext2_dir_entry_2 {  
    __le32  inode;           /* Inode number */  
    __le16  rec_len;         /* Directory entry length */  
    __u8    name_len;        /* Name length */  
    __u8    file_type;       /* File type */  
    char    name[];          /* File name, up to EXT2_NAME_LEN */  
};
```

- После `struct ext2_dir_entry_2` следует имя файла.

Примечание: запись об элементе каталога никогда не пересекает границы блока; `rec_len` у последней записи подбирается так, чтобы она заканчивалась точно на границе.

Ещё примечание: если поле `inode` равно нулю, то считается, что список в текущем блоке закончился.

Каталоги в ext2

Каталог хранится в файле специального типа (у которого младший байт `i_mode` равен `EXT2_FT_DIR`).

Содержимое файла представляет собой последовательность записей переменной длины:

- В заголовке записи стоит

```
struct ext2_dir_entry_2 {  
    __le32  inode;           /* Inode number */  
    __le16  rec_len;         /* Directory entry length */  
    __u8    name_len;        /* Name length */  
    __u8    file_type;       /* File type */  
    char    name[];          /* File name, up to EXT2_NAME_LEN */  
};
```

- После `struct ext2_dir_entry_2` следует имя файла.

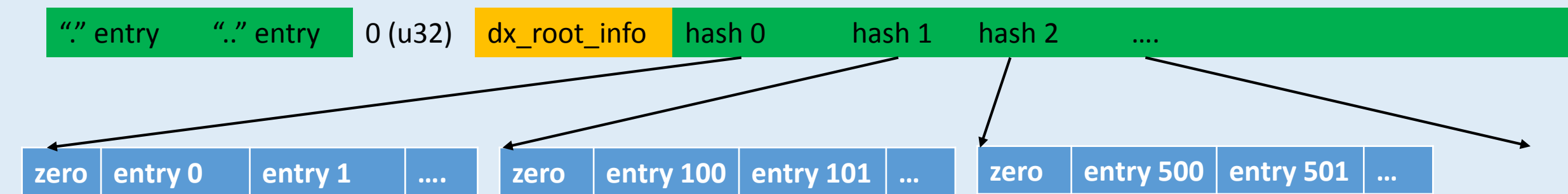
Примечание: запись об элементе каталога никогда не пересекает границы блока; `rec_len` у последней записи подбирается так, чтобы она заканчивалась точно на границе.

Ещё примечание: если поле `inode` равно нулю, то считается, что список в текущем блоке закончился.

Как быть с удалением элементов каталога?

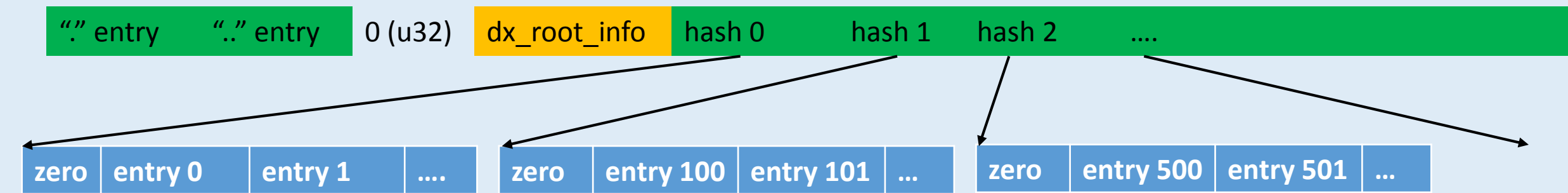
Каталоги в ext3 (hash tree directories)

Для больших каталогов используется следующее представление:



Каталоги в ext3 (hash tree directories)

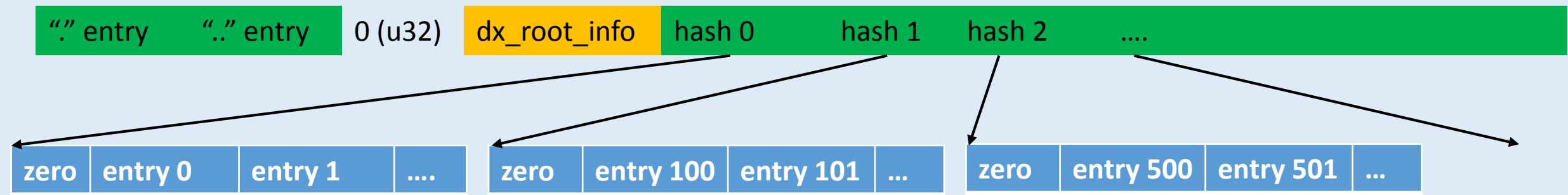
Для больших каталогов используется следующее представление:



- Если много имён имеют совпадающий хеш и их список не умещается в один блок, то в карте “хеш → номер блока” ставится флаг «список продолжается в следующем блоке».
- Разные хеши могут ссылаться на один блок.

Каталоги в ext3 (hash tree directories)

Для больших каталогов используется следующее представление:

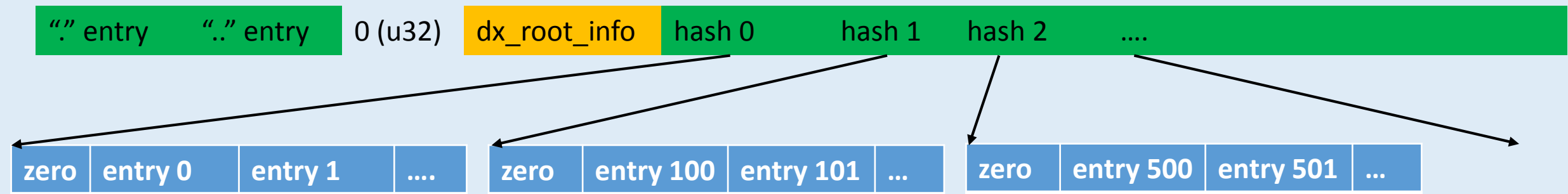


- Если много имён имеют совпадающий хеш и их список не умещается в один блок, то в карте “хеш → номер блока” ставится флаг «список продолжается в следующем блоке».
- Разные хеши могут ссылаться на один блок.

Изображённые выше блоки на диске располагаются подряд (и составляют один файл).

Каталоги в ext3 (hash tree directories)

Для больших каталогов используется следующее представление:



- Если много имён имеют совпадающий хеш и их список не умещается в один блок, то в карте “хеш → номер блока” ставится флаг «список продолжается в следующем блоке».
- Разные хеши могут ссылаться на один блок.

Изображённые выше блоки на диске располагаются подряд (и составляют один файл).

Нулевые записи в блоках нижнего уровня и нулевое 4-байтовое значение в корневом блоке поставлены затем, чтобы алгоритм линейного поиска из ext2 увидел правильный список элементов (вспоминаем, что элемент с нулевым полем `inode` – это признак «в этом блоке больше нет записей»).

Слежение за свободным и занятым местом

ФС разделена на группы блоков; учёт занятого места ведётся в пределах группы. Информация об одной группе умещается в памяти.

```
struct ext2_group_desc
{
    __le32  bg_block_bitmap;          /* Blocks bitmap block */
    __le32  bg_inode_bitmap;         /* Inodes bitmap block */
    __le32  bg_inode_table;          /* Inodes table block */
    __le16  bg_free_blocks_count;    /* Free blocks count */
    __le16  bg_free_inodes_count;    /* Free inodes count */
    __le16  bg_used_dirs_count;      /* Directories count */
    __le16  bg_pad;
    __le32  bg_reserved[3];
};
```

- Число блоков и инод в каждой группе одинаковое и задаётся при создании ФС
- Имеющиеся в одной группе иноды расположены непрерывным блоком
- Блоки, на которые ссылается инода, ФС старается выделять из группы, которой та принадлежит.

Устройство ext2 в целом

Пустой блок длиной 1Кб	SB	Block group 0					Block group 1				
		hdr	block bitmap	inode bitmap	inode table	data	hdr	block bitmap	inode bitmap	inode table	data	

Суперблок ext2 (src/linux/fs/ext2/ext2.h, struct ext2_super_block)

```
struct ext2_super_block {  
    __le32  s_inodes_count;      /* Inodes count */  
    __le32  s_blocks_count;     /* Blocks count */  
    __le32  s_r_blocks_count;   /* Reserved blocks count */  
    __le32  s_free_blocks_count; /* Free blocks count */  
    __le32  s_free_inodes_count; /* Free inodes count */  
    __le32  s_first_data_block; /* First Data Block */  
    __le32  s_log_block_size;   /* Block size */  
    __le32  s_log_frag_size;    /* Fragment size */  
    __le32  s_blocks_per_group; /* # Blocks per group */  
    __le32  s_frags_per_group;  /* # Fragments per group */  
    __le32  s_inodes_per_group; /* # Inodes per group */  
};
```

Суперблок ext2 (src/linux/fs/ext2/ext2.h, struct ext2_super_block)

```
__le32  s_mtime;          /* Mount time */
__le32  s_wtime;          /* Write time */
__le16  s_mnt_count;       /* Mount count */
__le16  s_max_mnt_count;   /* Maximal mount count */
__le16  s_magic;           /* Magic signature */
__le16  s_state;           /* File system state */
__le16  s_errors;         /* Behaviour when detecting errors */
__le16  s_minor_rev_level; /* minor revision level */
__le32  s_lastcheck;       /* time of last check */
__le32  s_checkinterval;   /* max. time between checks */
__le32  s_creator_os;      /* OS */
__le32  s_rev_level;       /* Revision level */
```

Суперблок ext2 (src/linux/fs/ext2/ext2.h, struct ext2_super_block)

```
__le32  s_feature_compat;    /* compatible feature set */
__le32  s_feature_incompat;   /* incompatible feature set */
__le32  s_feature_ro_compat;  /* readonly-compatible feature set */
__u8    s_uuid[16];          /* 128-bit uuid for volume */
char    s_volume_name[16];    /* volume name */
char    s_last_mounted[64];   /* directory where last mounted */
__le32  s_algorithm_usage_bitmap; /* For compression */
/*
 * Performance hints.  Directory preallocation should only
 * happen if the EXT2_COMPAT_PREALLOC flag is on.
 */
__u8    s_prealloc_blocks;    /* Nr of blocks to try to preallocate*/
__u8    s_prealloc_dir_blocks; /* Nr to preallocate for dirs */
__u16   s_padding1;
```

Compat, ro-compat, incompat features

Compat features:

- EXT4_FEATURE_COMPAT_DIR_PREALLOC
- EXT4_FEATURE_COMPAT_HAS_JOURNAL
- EXT4_FEATURE_COMPAT_EXT_ATTR
- EXT4_FEATURE_COMPAT_RESIZE_INODE
- EXT4_FEATURE_COMPAT_DIR_INDEX

Compat, ro-compat, incompat features

Compat features:

- EXT4_FEATURE_COMPAT_DIR_PREALLOC
- EXT4_FEATURE_COMPAT_HAS_JOURNAL
- EXT4_FEATURE_COMPAT_EXT_ATTR
- EXT4_FEATURE_COMPAT_RESIZE_INODE
- EXT4_FEATURE_COMPAT_DIR_INDEX

Ro-compat features:

- EXT4_FEATURE_RO_COMPAT_SPARSE_SUPER
- EXT4_FEATURE_RO_COMPAT_BTREE_DIR
- EXT4_FEATURE_RO_COMPAT_HUGE_FILE
- EXT4_FEATURE_RO_COMPAT_QUOTA

Compat, ro-compat, incompat features

Compat features:

- EXT4_FEATURE_COMPAT_DIR_PREALLOC
- EXT4_FEATURE_COMPAT_HAS_JOURNAL
- EXT4_FEATURE_COMPAT_EXT_ATTR
- EXT4_FEATURE_COMPAT_RESIZE_INODE
- EXT4_FEATURE_COMPAT_DIR_INDEX

Ro-compat features:

- EXT4_FEATURE_RO_COMPAT_SPARSE_SUPER
- EXT4_FEATURE_RO_COMPAT_BTREE_DIR
- EXT4_FEATURE_RO_COMPAT_HUGE_FILE
- EXT4_FEATURE_RO_COMPAT_QUOTA

Incompat features:

- EXT4_FEATURE_INCOMPAT_COMPRESSION
- EXT4_FEATURE_INCOMPAT_JOURNAL_DEV
- EXT4_FEATURE_INCOMPAT_EXTENTS
- EXT4_FEATURE_INCOMPAT_INLINE_DATA
- EXT4_FEATURE_INCOMPAT_ENCRYPT

Пример (почти) compat feature: 32-битные UID и GID владельца

Напоминание: хвост struct ext2_inode выглядит так:

```
__le32 i_block[EXT2_N_BLOCKS]; /* Pointers to blocks */
__le32 i_generation; /* File version (for NFS) */
__le32 i_file_acl; /* File ACL */
__le32 i_dir_acl; /* Directory ACL */
__le32 i_faddr; /* Fragment address */
__le8 osd2[12];
};
```

Операционные системы, которые не используют поле osd2, должны сохранять его без изменений.

Пример (почти) compat feature: 32-битные UID и GID владельца

Для linux хвост struct ext2_inode выглядит так:

```
__le32  i_block[EXT2_N_BLOCKS]; /* Pointers to blocks */
__le32  i_generation; /* File version (for NFS) */
__le32  i_file_acl; /* File ACL */
__le32  i_dir_acl; /* Directory ACL */
__le32  i_faddr; /* Fragment address */
union {
    struct {
        __u8  l_i_frag; /* Fragment number */
        __u8  l_i_fsize; /* Fragment size */
        __u16  i_pad1;
        __le16 l_i_uid_high; /* these 2 fields */
        __le16 l_i_gid_high; /* were reserved2[0] */
        __u32  l_i_reserved2;
    } linux2;
    } osd2; /* OS dependent 2 */
};
```

Дополнительное чтение

- <http://www.nongnu.org/ext2-doc>
- https://ext4.wiki.kernel.org/index.php/Ext4_Disk_Layout
- <http://wiki.osdev.org/Ext2>
- <https://lwn.net/Articles/322823/>

Домашнее задание

1. Почему htree directories (EXT4_FEATURE_COMPAT_DIR_INDEX) – это compat feature?
2. Разобраться с mkfs.ext2, создать образ ext2, и написать программу, которая
 - Перечислит элементы в любом каталоге по номеру его inode,
 - (*) Перечислит элементы в любом каталоге, заданном путём,
 - Прочтёт файл, заданный номером его inode,
 - (*) Прочтёт файл, заданный путём.
3. (***) Реализовать модуль для FUSE, который примонтирует образ ext2 в режиме только для чтения.

