ext2 File System Walkthrough

- The main thing to remember about file systems is that they are ultimately data structures — knowing a file system is a matter of knowing how to interpret the sequence of bytes/blocks that it writes onto a disk
- Implementing a file system is a matter of designing a scheme for how the bytes/blocks in a volume can be organized into files, directories, and other constructs, then implementing this scheme in code
- To drive these points home, we'll walk through a raw image of a particular file system: Linux's *ext2*

Preliminaries and Tools

- The information in this walkthrough was produced through a 2-megabyte disk image, on which an "empty" *ext2* file system was installed
- The disk image was then mounted and modified:
  - Two files, *hello.txt* and *goodbye.txt* were placed in the root directory
  - A subdirectory, *mydir*, was also placed at the root
  - Two links were placed in *mydir*: a symbolic link to *hello.txt*, and a hard link to *goodbye.txt*
- Two utilities help in the walkthrough: *dumpe2fs* displays the superblock in a more readable form, and *hexdump* displays the raw bytes on the disk image

```
Filesystem volume name:   <none>
Last mounted on:          <not available>
Filesystem UUID:          ab1e9464-f4a2-480d-81c0-e7f05df99988
Filesystem magic number:  0xEF53
Filesystem revision #:    1 (dynamic)
Filesystem features:      has_journal filetype sparse_super
Default mount options:    (none)
Filesystem state:         clean
Errors behavior:          Continue
Filesystem OS type:       Linux
Inode count:              256
Block count:              2048
Reserved block count:     102
Free blocks:              965
Free inodes:              241
First block:              1
Block size:               1024
Fragment size:            1024
Blocks per group:         8192
Fragments per group:      8192
Inodes per group:         256
Inode blocks per group:   32
Filesystem created:       Tue Apr  4 17:28:08 2006
Last mount time:          Tue Apr 11 14:38:24 2006
Last write time:          Tue Apr 11 14:39:05 2006
Mount count:              7
Maximum mount count:      23
Last checked:             Tue Apr  4 17:28:08 2006
Check interval:           15552000 (6 months)
Next check after:         Sun Oct  1 17:28:08 2006
Reserved blocks uid:      0 (user root)
Reserved blocks gid:      0 (group root)
First inode:              11
Inode size:               128
Journal inode:            8
Default directory hash:   tea
Directory Hash Seed:      8d91c7c3-5116-4b71-b7ac-255d6907d8a1
Journal backup:           inode blocks

Group 0: (Blocks 1-2047)
  Primary superblock at 1, Group descriptors at 2-2
  Block bitmap at 3 (+2), Inode bitmap at 4 (+3)
  Inode table at 5-36 (+4)
  965 free blocks, 241 free inodes, 3 directories
  Free blocks: 1082-1088, 1090-2047
  Free inodes: 16-256
```

Superblock: *dumpe2fs* helps us get our bearings, but in the end it's just a helper — in *ext2*, the superblock C structure maps directly onto what's written to the disk

- 2048 blocks * 1024 bytes per block = 2 megabytes ("mebibytes" by today's latest terminology) — take note, 1024 is 400 hex
- 128 bytes per inode: 80 hex
- Groups form an intermediate structure within an *ext2* volume; the superblock and group descriptors are copied within each group in case of corruption

```
00000000  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00  |................|
*
00000400  00 01 00 00 00 08 00 00  66 00 00 00 c5 03 00 00  |........f.......|
00000410  f1 00 00 00 01 00 00 00  00 00 00 00 00 00 00 00  |................|
00000420  00 20 00 00 00 20 00 00  00 01 00 00 d0 21 3c 44  |. ... .......!<D|
00000430  f9 21 3c 44 07 00 17 00  53 ef 01 00 01 00 00 00  |.!<D....S.......|
00000440  18 0f 33 44 00 4e ed 00  00 00 00 00 01 00 00 00  |..3D.N..........|
00000450  00 00 00 00 0b 00 00 00  80 00 00 00 04 00 00 00  |................|
00000460  02 00 00 00 01 00 00 00  ab 1e 94 64 f4 a2 48 0d  |...........d..H.|
00000470  81 c0 e7 f0 5d f9 99 88  00 00 00 00 00 00 00 00  |....]...........|
00000480  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00  |................|
*
000004e0  08 00 00 00 00 00 00 00  00 00 00 00 8d 91 c7 c3  |................|
000004f0  51 16 4b 71 b7 ac 25 5d  69 07 d8 a1 02 01 00 00  |Q.Kq..%]i.......|
00000500  00 00 00 00 00 00 00 00  18 0f 33 44 32 00 00 00  |..........3D2...|
00000510  33 00 00 00 34 00 00 00  35 00 00 00 36 00 00 00  |3...4...5...6...|
00000520  37 00 00 00 38 00 00 00  39 00 00 00 3a 00 00 00  |7...8...9...:...|
00000530  3b 00 00 00 3c 00 00 00  3d 00 00 00 3e 00 00 00  |;...<...=...>...|
00000540  3f 01 00 00 00 00 00 00  00 00 00 00 00 00 10 00  |?...............|
00000550  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00  |................|
*
```

- Note how "reading" the superblock is a matter of following its corresponding C structure from the *ext2* source code
- Other file systems might not be quite so direct, requiring additional decoding
- For conciseness, *hexdump* skips sequences of 00 bytes, and marks them with a "*"
- hexdump can also place ASCII on the right; these settings are activated with the *-C* ("canonical") switch

```
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 */
    __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 */
    __le16  s_def_resuid;       /* Default uid for reserved blocks */
    __le16  s_def_resgid;       /* Default gid for reserved blocks */
    __le32  s_first_ino;        /* First non-reserved inode */
    __le16   s_inode_size;      /* size of inode structure */
    __le16  s_block_group_nr;   /* block group # of this superblock */
    __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 */
    __u8    s_prealloc_blocks;  /* Nr of blocks to try to preallocate*/
    __u8    s_prealloc_dir_blocks;  /* Nr to preallocate for dirs */
    __u16   s_padding1;
    __u8    s_journal_uuid[16]; /* uuid of journal superblock */
    __u32   s_journal_inum;     /* inode number of journal file */
    __u32   s_journal_dev;      /* device number of journal file */
    __u32   s_last_orphan;      /* start of list of inodes to delete */
    __u32   s_hash_seed[4];     /* HTREE hash seed */
    __u8    s_def_hash_version; /* Default hash version to use */
    __u8    s_reserved_char_pad;
    __u16   s_reserved_word_pad;
    __le32  s_default_mount_opts;
    __le32  s_first_meta_bg;    /* First metablock block group */
    __u32   s_reserved[190];    /* Padding to the end of the block */
};
```

Group descriptor: Again, as long as you have the C structure, it's fairly easy to read in its raw form

```
00000800  03 00 00 00 04 00 00 00  05 00 00 00 c5 03 f1 00  |................|
00000810  03 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00  |................|
00000820  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00  |................|
```

```c
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];
};
```

- At this point, you can start "doing the math"
- Since the inode table starts at block 5, and blocks are 1024 bytes long, then you can expect to see the inodes at the linear hex location 5 * 400 = 1400
- Data blocks 3 and 4 (locations c000 and 1000, respectively) are straightforward bit fields indicating what's available (if the bit is set, then the corresponding entity has been allocated for use)

```
00000c00  ff ff ff ff ff ff ff ff  ff ff ff ff ff ff ff ff  |................|
*
00000c80  ff ff ff ff ff ff ff 01  01 00 00 00 00 00 00 00  |................|
00000c90  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00  |................|
*
00000cf0  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 80  |................|
00000d00  ff ff ff ff ff ff ff ff  ff ff ff ff ff ff ff ff  |................|
*
```

```
00001000  ff 7f 00 00 00 00 00 00  00 00 00 00 00 00 00 00  |................|
00001010  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00  |................|
00001020  ff ff ff ff ff ff ff ff  ff ff ff ff ff ff ff ff  |................|
*
```

```
00001400  00 00 00 00 00 00 00 00  18 0f 33 44 18 0f 33 44  |..........3D..3D|
00001410  18 0f 33 44 00 00 00 00  00 00 00 00 00 00 00 00  |..3D............|
00001420  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00  |................|
*
00001480  ed 41 f4 03 00 04 00 00  ef 21 3c 44 9d 21 3c 44  |.A.......!<D.!<D|
00001490  9d 21 3c 44 00 00 00 00  f4 03 04 00 02 00 00 00  |.!<D............|
000014a0  00 00 00 00 00 00 00 00  25 00 00 00 00 00 00 00  |........%.......|
000014b0  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00  |................|
*
00001780  80 81 00 00 00 00 10 00  00 00 00 00 18 0f 33 44  |..............3D|
00001790  18 0f 33 44 00 00 00 00  00 00 01 00 0c 08 00 00  |..3D............|
000017a0  00 00 00 00 00 00 00 00  32 00 00 00 33 00 00 00  |........2...3...|
000017b0  34 00 00 00 35 00 00 00  36 00 00 00 37 00 00 00  |4...5...6...7...|
000017c0  38 00 00 00 39 00 00 00  3a 00 00 00 3b 00 00 00  |8...9...:...;...|
000017d0  3c 00 00 00 3d 00 00 00  3e 00 00 00 3f 01 00 00  |<...=...>...?...|
000017e0  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00  |................|
*
00001900  c0 41 00 00 00 30 00 00  ec 21 3c 44 18 0f 33 44  |.A...0...!<D..3D|
00001910  18 0f 33 44 00 00 00 00  00 00 02 00 18 00 00 00  |..3D............|
00001920  00 00 00 00 00 00 00 00  26 00 00 00 27 00 00 00  |........&.......|
00001930  28 00 00 00 29 00 00 00  2a 00 00 00 2b 00 00 00  |(...)...*...+...|
00001940  2c 00 00 00 2d 00 00 00  2e 00 00 00 2f 00 00 00  |,...-......./...|
00001950  30 00 00 00 31 00 00 00  00 00 00 00 00 00 00 00  |0...1...........|
00001960  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00  |................|
*
00001980  b0 81 f4 03 0d 00 00 00  6e 0f 33 44 71 0f 33 44  |........n.3Dq.3D|
00001990  71 0f 33 44 00 00 00 00  f4 03 01 00 02 00 00 00  |q.3D............|
000019a0  00 00 00 00 00 00 00 00  38 04 00 00 00 00 00 00  |........8.......|
000019b0  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00  |................|
*
000019e0  00 00 00 00 34 3e 1d 2c  00 00 00 00 00 00 00 00  |....4>.,........|
000019f0  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00  |................|
00001a00  f8 41 f4 03 00 04 00 00  d7 21 3c 44 83 21 3c 44  |.A.......!<D.!<D|
00001a10  83 21 3c 44 00 00 00 00  f4 03 02 00 02 00 00 00  |.!<D............|
00001a20  00 00 00 00 00 00 00 00  39 04 00 00 00 00 00 00  |........9.......|
00001a30  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00  |................|
*
00001a60  00 00 00 00 0b a7 c8 39  00 00 00 00 00 00 00 00  |.......9........|
00001a70  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00  |................|
00001a80  ff a1 f4 03 0c 00 00 00  d7 21 3c 44 83 21 3c 44  |.........!<D.!<D|
00001a90  83 21 3c 44 00 00 00 00  f4 03 01 00 00 00 00 00  |.!<D............|
00001aa0  00 00 00 00 00 00 00 00  2e 2e 2f 68 65 6c 6c 6f  |........../hello|
00001ab0  2e 74 78 74 00 00 00 00  00 00 00 00 00 00 00 00  |.txt............|
00001ac0  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00  |................|
*
00001ae0  00 00 00 00 0c a7 c8 39  00 00 00 00 00 00 00 00  |.......9........|
00001af0  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00  |................|
00001b00  b0 81 f4 03 0d 00 00 00  28 48 33 44 9d 21 3c 44  |........(H3D.!<D|
00001b10  28 48 33 44 00 00 00 00  f4 03 02 00 02 00 00 00  |(H3D............|
00001b20  00 00 00 00 00 00 00 00  41 04 00 00 00 00 00 00  |........A.......|
00001b30  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00  |................|
*
00001b60  00 00 00 00 e6 32 42 c9  00 00 00 00 00 00 00 00  |.....2B.........|
00001b70  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00  |................|
```

Inodes: Based on the information in the superblock and group descriptor, we expect the inodes to show up at hex location 1400

- And indeed, they're there; at 128 bytes per inode, it's easy to jump from one inode to another — 128 is 80 hex, so we'll find inodes at 1400, 1480, 1500, 1580, etc.
- The first few inodes are reserved for system use, as indicated in the source code:

```c
/*
 * Special inode numbers
 */
#define EXT2_BAD_INO         1  /* Bad blocks inode */
#define EXT2_ROOT_INO        2  /* Root inode */
#define EXT2_BOOT_LOADER_INO     5  /* Boot loader inode */
#define EXT2_UNDEL_DIR_INO   6  /* Undelete directory inode */

/* First non-reserved inode for old ext2 filesystems */
#define EXT2_GOOD_OLD_FIRST_INO 11
```

*Inode 11 (b hex) = 1400 + (80 * (b – 1)) = 1900*

- To go on with reading the volume, we focus on inode 2, which is the root directory's inode; since inode 1 is in 1400, we expect inode 2 in 1480

Inode Structure: As you've probably guessed by now, an *ext2* inode is mapped directly from its C structure

- Let's start with the inode for the root directory — the key information here, for getting to the rest of the volume, is to locate its first data block; in this case, it is also the only data block, which is 25 (hex, of course)

```
00001480  ed 41 f4 03 00 04 00 00  ef 21 3c 44 9d 21 3c 44  |.A......!<D.!<D|
00001490  9d 21 3c 44 00 00 00 00  f4 03 04 00 02 00 00 00  |.!<D............|
000014a0  00 00 00 00 00 00 00 00  25 00 00 00 00 00 00 00  |........%.......|
000014b0  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00  |................|
*
```

- The rest of the fields should be easy to parse out now; for instance, the root directory's *mode* is *41 ed* or binary *0100 0001 1110 1101*
- The 9 low-order bits correspond to the traditional Unix permissions, *rwxrwxrwx*; this directories permissions are thus *rwxr-xr-x*
- The *0100* on the high end indicates that this inode represents a directory (S_IFDIR in *stat.h*)

```c
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 */
    __le32  i_blocks;   /* Blocks count */
    __le32  i_flags;    /* File flags */
    union {
        struct {
            __le32  l_i_reserved1;
        } linux1;
        struct {
            __le32  h_i_translator;
        } hurd1;
        struct {
            __le32  m_i_reserved1;
        } masix1;
    } osd1;             /* OS dependent 1 */
    __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;
        struct {
            __u8    h_i_frag;   /* Fragment number */
            __u8    h_i_fsize;  /* Fragment size */
            __le16  h_i_mode_high;
            __le16  h_i_uid_high;
            __le16  h_i_gid_high;
            __le32  h_i_author;
        } hurd2;
        struct {
            __u8    m_i_frag;   /* Fragment number */
            __u8    m_i_fsize;  /* Fragment size */
            __u16   m_pad1;
            __u32   m_i_reserved2[2];
        } masix2;
    } osd2;             /* OS dependent 2 */
};
```

Directories: A directory's data block is an array of directory entries; here's the one for the root directory, located at data block 25 or offset 9400

- As should be obvious at this point, we use the directory entry's C structure in the source code to read the directory:

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

So the file called *hello.txt* is in the twelfth inode, and its directory entry is 20 bytes long

```
00009400  02 00 00 00 0c 00 01 02  2e 00 00 00 02 00 00 00  |................|
00009410  0c 00 02 02 2e 2e 00 00  0b 00 00 00 14 00 0a 02  |................|
00009420  6c 6f 73 74 2b 66 6f 75  6e 64 00 00 0c 00 00 00  |lost+found......|
00009430  14 00 09 01 68 65 6c 6c  6f 2e 74 78 74 00 00 00  |....hello.txt...|
00009440  0f 00 00 00 14 00 0b 01  67 6f 6f 64 62 79 65 2e  |........goodbye.|
00009450  74 78 74 00 0d 00 00 00  ac 03 05 02 6d 79 64 69  |txt.........mydi|
00009460  72 62 79 65 2e 74 78 74  2e 73 77 70 00 00 00 00  |rbye.txt.swp....|
00009470  94 03 0c 01 67 6f 6f 64  62 79 65 2e 74 78 74 7e  |....goodbye.txt~|
00009480  2e 73 77 78 00 00 00 00  00 00 00 00 00 00 00 00  |.swx............|
00009490  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00  |................|
*
```

- Note how the current directory ("**.**") and parent directory ("**..**") are stored as explicit directory entries too; since this is the root directory, it make sense that both **.** and **..** refer to the same inode

The *mydir* directory entry is immediately followed by what appears to be garbage — you're seeing the remnants of prior directory entries that have since been deleted or overwritten (see how the filename is 5 bytes long, and how the directory entry itself is *3ac* bytes long — i.e., the remainder of the data block!)

Files etc.: At last, we get to some actual files — the text files are easy to locate, and additional directories are read in the same way as the root directory

- *hello.txt* is in inode *c*, which translates to offset 1980; the inode then says that the file's first data block is in overall data block 0438

```
00001980  b0 81 f4 03 0d 00 00 00  6e 0f 33 44 71 0f 33 44  |........n.3Dq.3D|
00001990  71 0f 33 44 00 00 00 00  f4 03 01 00 02 00 00 00  |q.3D............|
000019a0  00 00 00 00 00 00 00 00  38 04 00 00 00 00 00 00  |........8.......|
000019b0  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00  |................|
*
```

- *mydir* is in inode *d* (right after *hello.txt*'s inode), and the file type of 02 indicates that it is a directory; its first data block is 0439

```
00001a00  f8 41 f4 03 00 04 00 00  d7 21 3c 44 83 21 3c 44  |.A.......!<D.!<D|
00001a10  83 21 3c 44 00 00 00 00  f4 03 02 00 02 00 00 00  |.!<D............|
00001a20  00 00 00 00 00 00 00 00  39 04 00 00 00 00 00 00  |........9.......|
00001a30  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00  |................|
*
```

- Thus, the file's data can be found at offset 400 * 0438 = 10e000, and indeed the bytes are there!
- Data block 0439 is at offset 10e400, which should now be somewhat recognizable as an array of directory entries (note how **.** and **..** now refer to different inodes)

```
0010e000  48 65 6c 6c 6f 20 77 6f  72 6c 64 21 0a 00 00 00  |Hello world!....|
0010e010  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00  |................|
*
0010e400  0d 00 00 00 0c 00 01 02  2e 00 00 00 02 00 00 00  |................|
0010e410  0c 00 02 02 2e 2e 00 00  0f 00 00 00 14 00 0b 01  |................|
0010e420  67 6f 6f 64 62 79 65 2e  74 78 74 00 0e 00 00 00  |goodbye.txt.....|
0010e430  d4 03 09 07 68 65 6c 6c  6f 2e 74 78 74 00 00 00  |....hello.txt...|
0010e440  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00  |................|
*
```

Special Files: If we follow the directory entries in *mydir*, we'll notice a few more variations in how files are handled

```
0010e400  0d 00 00 00 0c 00 01 02  2e 00 00 00 02 00 00 00  |................|
0010e410  0c 00 02 02 2e 2e 00 00  0f 00 00 00 14 00 0b 01  |................|
0010e420  67 6f 6f 64 62 79 65 2e  74 78 74 00 0e 00 00 00  |goodbye.txt.....|
0010e430  d4 03 09 07 68 65 6c 6c  6f 2e 74 78 74 00 00 00  |....hello.txt...|
0010e440  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00  |................|
*
```

- *goodbye.txt* is indicated to be in inode *f*, just like *goodbye.txt* in the root directory — this is how *hard links* are implemented: they just directory entries that refer to the same inode!

```
0010e400  0d 00 00 00 0c 00 01 02  2e 00 00 00 02 00 00 00  |................|
0010e410  0c 00 02 02 2e 2e 00 00  0f 00 00 00 14 00 0b 01  |................|
0010e420  67 6f 6f 64 62 79 65 2e  74 78 74 00 0e 00 00 00  |goodbye.txt.....|
0010e430  d4 03 09 07 68 65 6c 6c  6f 2e 74 78 74 00 00 00  |....hello.txt...|
0010e440  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00  |................|
*
```

```
/*
 * Ext2 directory file types.  Only the
 * low 3 bits are used.  The other bits
 * are reserved for now.
 */
enum {
    EXT2_FT_UNKNOWN,  /* 00 */
    EXT2_FT_REG_FILE, /* 01 */
    EXT2_FT_DIR,      /* 02 */
    EXT2_FT_CHRDEV,   /* 03 */
    EXT2_FT_BLKDEV,   /* 04 */
    EXT2_FT_FIFO,     /* 05 */
    EXT2_FT_SOCK,     /* 06 */
    EXT2_FT_SYMLINK,  /* 07 */
    EXT2_FT_MAX       /* 08 */
};
```

- *hello.txt*'s directory entry lists its file type as 07, which, according to the source code, means a symbolic link
- When we look at inode e, we see that the symbolic link's path is stored in the inode itself, where the data blocks would normally be:

```
00001a80  ff a1 f4 03 0c 00 00 00  d7 21 3c 44 83 21 3c 44  |.........!<D.!<D|
00001a90  83 21 3c 44 00 00 00 00  f4 03 01 00 00 00 00 00  |.!<D............|
00001aa0  00 00 00 00 00 00 00 00  2e 2e 2f 68 65 6c 6c 6f  |........../hello|
00001ab0  2e 74 78 74 00 00 00 00  00 00 00 00 00 00 00 00  |.txt............|
00001ac0  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00  |................|
*
```