Lecture 6. File System

1. Goal

Storing files efficiently in secondary memory system such as a disk. "Efficiently" means two things:

- space efficiency: no waste of space in disk
- time efficiency: fast accessing (file name => physical location)

2. Several example file systems

assumption:

disk = collection of blocks (1 block=1K byte)
file is stored by the unit of blocks.

scenario:

write f1 (2 blk)
write f2 (3 blk)
delete f1
write f3 (3 blk)

1) consecutive allocation fs

Blocks are allocated consecutively. Space efficiency low; access speed fast.

2) linked list allocation fs

Blocks are allocated non-consecutively. Need links to connect blocks belonging to the same file. Space efficiency high; access speed low.

3) FAT fs

Use another meta-block to contain all link information. Space efficiency high; access speed medium fast.

4) Inode fs

Use Inode table to contain information for files. Space efficiency high; access speed fast.

3. EXT2 fs: default fs in Linux 2.4

1 block = 4K bytes in default (1K bytes for a small disk)
disk in ext2 fs = boot loader, block group 0, block group 1, ...
block group x = S, G, D, I, Inode table, files

S: super block (1 blk) – global information about this fs
G: group descriptors (n blk) – info for each descriptor
D: data block bitmap(1 blk) – info about block usage
I : inode bitmap (1 blk) – info about inode usage
Inode table: inode table (n blk) – inode table
files: (n blk) – files

1) files

regular file: text, program, graphic file, ...
directory file: a file with information about member files belonging to this directory such as inode number, file name, file type.

```
ext2_dir_entry_2{
    __u32 inode; // 4 byte for inode number
    __u16 rec_len; // 2 byte for this record length
    __u8  name_len; // 1 byte for name length
    __u8  file_type; // 1 byte for file type
    char  name[EXT2_NAME_LEN]; // file name
}
```

symbolic link file: a file containing symbolic link information
device file, socket, pipe : special file that corresponds some device, socket, pipe, etc

2) inode table

inode table: a table of inodes
inode :

- every file has an inode
- root directory is inode 2 (inode number starts from 1)
- one inode is 128 byte (or determined by superblock->m_inode_size)

- an inode contains all information about the corresponding file:
  -- block location of this file, file type, protection mode, file size,
     creation time, etc.
  -- example: inode 2 = (file type=2, block location=50, file size=24, ...)
     => root directory is “directory”, located at block 50, size is 24 bytes,...

- data structure

```
struct ext2_inode {
    __u16   i_mode;  // 0-1. file type, access mode.
    __u16   i_uid   ;  // 2-3. owner identifier
    __u32   i_size  ;  // 4-7. file size in bytes
    .................
    __u16   i_links_count;  // hard links counter
    __u32   i_blocks;  // file size in blocks
    __u32   i_block[EXT2_N_BLOCKS]; // block location array
}
```

i_block[0] to i_block[11] tells the location of corresponding block
i_block[12] is an indirect address:
  if i_block[12]=100, block 100 has the location of next 1024 blocks
i_block[13] is a double indirect address:
  if i_block[13]=200, block 200 has the location of 1024 indirect addr block
i_block[14] is a triple indirect address

3) inode bit map
The use/free information of each inode. With 4K-byte block, 1 block can show the usage info of 4*1024*8 = 32768 inodes.
4) data block bit map
The usage info of 32768 blocks.
5) group descriptor
The location of inode bit map,data block bit map, inode table, etc.
6) super block
Overall information about this file system such as total number of inodes, total number of blocks, block size, etc.

data structures:

```
typedef struct  // super block
{
u32 m_inodes_count;  // 0-3
u32 m_blocks_count;  // 4-7
u32 m_r_blocks_count; // 8-B
u32 m_free_blocks_count; // C-F
u32 m_free_inodes_count;  // 10-13
u32 m_first_data_block;  // 14-17
                // block location of superblock.
u32 m_log_block_size;    // 18-1B. block size=1024*pow(2, m_log_block_size)
u32 m_log_frag_size;     // 1C-1F
u32 m_blocks_per_group;  // 20-23
u32 m_frrags_per_group; // 24-27
u32 m_inodes_per_group;  // 28-2B
```

```
u32 m_mtime;  // 2C-2F
u32 m_wtime;  // 30-33
u16 m_mnt_count; // 34-35
u16 m_max_mnt_count;  // 36-37
u16 m_magic;  // 38-39
u16 m_state;  // 3A-3B
u16 m_errors;  // 3C-3D
u16 m_minor_rev_level;  // 3E-3F
u32 m_lastcheck;  // 40-43
u32 m_checkinterval; // 44-47
u32 m_creator_os;  // 48-4b
u32 m_rev_level;  // 4c-4f
u16 m_def_resuid;  // 50-51
u16 m_def_resgid;  // 52-53
u32 m_first_ino;  // 54-57
u16 m_inode_size;  //58-59
u16 m_block_group_nr;  //5a-5b
u32 m_feature_compat;  //5c-5f
u32 m_feature_incompat;  //60-63
u32 m_feature_ro_compat;  //64-67
u08 m_uuid[16]; //68-77
char m_volume_name[16]; //78-87
char m_last_mounted[64]; //88-c7
u32 m_algorithm_usage_bitmap; //c8-cb
u08 m_prealloc_blocks; //cc
u08 m_prealloc_dir_blocks; //cd
u16 m_padding; // ce-cf
u08 m_journal_uuid[16]; // d0-df
u32 m_journal_inum; // e0-e3
u32 m_journal_dev; // e4-e7
u32 m_last_orphan; // e8-eb
u32 m_hash_seed[4]; // ec-fb
} SuperBlock;

typedef struct // group descriptor
{
```

```
u32 m_block_bitmap;  // block location of DBM
u32 m_inode_bitmap;  // block location of IBM
u32 m_inode_table;   // block location of inode table
u16 m_free_blocks_count;
u16 m_free_inodes_count;
u16 m_used_dir_count;
u16 m_padding;
u32 m_reserved[3];
} GroupDescriptor;

typedef struct  // inode
{
u16 m_mode; // 0-1
u16 m_uid;  // 2-3
u32 m_size; // 4-7
u32 m_atime; // 8-B
u32 m_ctime; // C-F
u32 m_mtime;  // 10-13
u32 m_dtime; // 14-17
u16 m_gid; // 18-19
u16 m_links_count; // 1A-1B
u32 m_blocks; // 1C-1F. shows num of data blocks for this file in units of 512 bytes
u32 m_flags; // 20-23
u32 m_reserved1; // 24-27
u32 m_block[15];   // block location of this file
u32 m_generation;
u32 m_file_acl;
u32 m_dir_acl;
u32 m_faddr;
u32 m_reserved2[3];
} Inode;

typedef struct // directory
{
u32 m_inode;
u16 m_rec_len;
```

u08 m_name_len;

u08 m_file_type;

char m_name[255];

} DirectoryEntry;

4. Finding a file's block location in ext2
Find /d1/f1
=> get the location of Inode Table from Group Descriptor (block 5)
=> read block 5
=> get inode 2 (this is the inode for "/")
=> get the block shown in i_block[0] of inode 2 (block 32)
=> find the inode of d1 in this directory block (12)
=> get inode 12 from block 5
=> get the block shown in i_block[0] of inode 12 (block 54)
=> find the inode of f1 in this directory block (23)
=> get inode 23 from block 5
=> the block location of /d1/f1 is written in inode 23

5. Creating an ext2 fs in a floppy disk
$ mkfs –t ext2 /dev/fd0
or
$ mke2fs /dev/fd0
creates an ext2 file system in a floppy disk.
After creation, we have
block 1 : super block
block 2 : group descriptor
block 3 : data block bitmap
block 4 : inode bitmap
block 5-49 : inode table
block 50 : root directory
super block: blk size, total inode num, total block num, etc are written
group descriptor: the locations of dbm, ibm, inode table are written
data block bitmap: bit 0-50 are set to 1; the rest set to 0
inode bitmap : bit 0, 1, 2 are set to 1. inode 0, 1 are not used. inode 2 is for /
inode table : inode 2 contains
file type: 2 (directory)
i_block[0]=50
.....
root directory : Currently no file exists. But two files shown in this directory
file name = ".", inode num = 2 : represents root itself
file name="..", inode num=2 : parent of root is itself

If we make a file, "/f1", and write "korea" in it
=> get inode table from block 5
=> read inode 2 and find its location (block 50)
=> create "/f1"
- write file name "f1" in an empty directory entry
- get a free inode from ibm (inode 3) and write the number in this entry
- get a free data block from dbm (block 51)
- write "korea" in block 51
- write 51 in i_block[0] of inode 3

6. (homework) Read the disk and analyze the contents of the meta blocks.
1) Make a virtual floppy disk
dd bs=1024 count=1440 if=/dev/zero of=myfd

will make a virtual floppy disk of size 1.44MB with name “myfd”.
“dd” is a command to write data into disk. “bs” is block-size. bs=1024 means 1 block is 1024 byte. “count” is the number of blocks to write. count=1440 means write 1440 blocks, which is 1440*1024=1440KB=1.44 MB. “if” is input file to read data from. /dev/zero is a special file that gives out zeros when being read. “of” is the output file.
2) Format
mkfs -t ext2 myfd
will format myfd with ext2 file system.
3) Mount
mkdir temp
We need an empty directory to mount myfd.
mount -o loop myfd temp
will mount myfd to temp directory. Since myfd is not a physical disk but a file that contains a disk image, we need to use –o loop option.
4) Make some files
cd temp
echo korea > f1
will make f1 in temp.
5) Read
cd ..
umount temp
We need “umount temp” to write the change in the virtual disk, myfd.
xxd -g1 myfd > x
“-g1” option will display each byte of myfd separately.
vi x

6) Read superblock. Superblock starts at offset 1024(400h). Find the superblock and confirm the magic number(0xEF53), block size, and the first block number
7) Read the group descriptor and confirm the block location of IBM, DBM, and inode table.
8) Read the IBM and DBM, and confirm the inode numbers and block numbers in use. Draw the layout of myfd disk that shows the block location of all meta blocks: super block, group descriptor, IBM, DBM, and inode table.
9) Read the inode table and find the block location of the root directory file. What is the byte size and block size of this file? Who is the owner of this file?
10) Read the root directory file. How many member files it has? What are the inode numbers and file names of them? Go to the inode table and find the block location of each member file.
11) Read the member file and confirm the contents.
12) You can see all files including hidden ones with "ls -a". Confirm you can see all files you found in the file system with this command.
ls -a
13) You can see the inode number of a file with "ls -i". Confirm the inode numbers of all files.
ls -ai
14) Make another file in your virtual disk. Confirm the changes in the file system: IBM, DBM, Inode table, and root directory file. Now delete this file (with "rm" command). What happens to the file system? How can you recover this file?

14-1) Make a new directory (d7) in the root directory with "mkdir" command. Show the disk block content of the root directory file and find out the inode number of d7.

14-2) Show the inode content of d7. What is the block location of d7? Show the block content of d7. What files do you have in d7?

14-3) Run "mv f1 d7/f2" and show the changes in the root directory file, d7 file, and inode table.

15) Examine the file system in the hard disk (/dev/sda3) and find file names in the root directory.

dd bs=1024 count=8000 if=/dev/sda3 of=myhd
xxd –g1 myhd > x

vi x

16) Write a program that opens a disk device formatted with EXT2 and reads and displays the super block, group descriptor, ibm, dbm, and inode table. Also display the file names in the root directory file, their inode numbers, and their block locations. Use open(), lseek(), read(), etc.

```
struct superblock{
   int  total_inode_num;
   int  total_block_num;
   .........
};
int x; char buf[1024]; struct superblock *sb;
x=open("myfd", O_RDONLY, 00777); //open a virtual disk
lseek(x, 1024, SEEK_SET); // move the file pointer to offset 1024 where the
                          // superblock starts
read(x, buf, 1024); // read the superblock into buf
sb=(struct superblock *)buf; // interpret the data in buf as "struct superblock"
printf("total inode num:%x, total_block_num:%x, ...",
     sb->total_inode_num, sb->total_block_num, ....);
```