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 block group

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 descriptors

The location of inode bit map,data block bit map, inode table, etc. for each block group.

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\*)*

*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 (this "block" is not the file system block, it is just the unit of data transfer in "dd" command). “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. If temp is already there, skip this step.

# 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. This command will connect myfd disk to /temp directory and you can access "myfd" disk via /temp.

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 myfd disk.

# xxd -g1 myfd > x

“-g1” option will display each byte of myfd separately.

# vi x

(if it doesn't start at line 0, type ":1" to go to line 1)

6) Read superblock. Superblock starts at offset 1024(400h). Find the superblock and read the total number of inodes in this disk (m\_inodes\_count), total number of blocks in this disk (m\_blocks\_count), the magic number (m\_magic), block size (1024\*), and the first data block number (m\_first\_data\_block). Every multi-byte number is stored in little endian, and you have to reverse the byte order to compute the actual value.

7) Read the group descriptor and find the block number of DBM, IBM, and inode table. The group descriptor is located in the next block after the superblock. If the superblock is at block 0, the group descriptor will start at block 1, if the superblock is at block 1, the group descriptor start at block 2, etc. The address of block x will be x\*block\_size.

8) Read the DBM, IBM, and find 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 files 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 (of myfd) 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, ....);