File system

Case studies: Linux (Ext2) and Windows (NTFS)

Lecture 3

~ Fall, 2007 ~

Contents

- **Linux File System (Ext2)**
- **Windows NT (New Technology) File System (NTFS)**

Linux File System

General considerations

- Virtual File System (VFS)
- Ext2
  - Second Extended Filesystem
  - The native FS of Linux
- The first version of Linux were based on the Minix file system
- Ext2 was introduces in 1994
- Comply with the POSIX interface
- Ext4 – newest version (October 10, 2006)

Linux File System

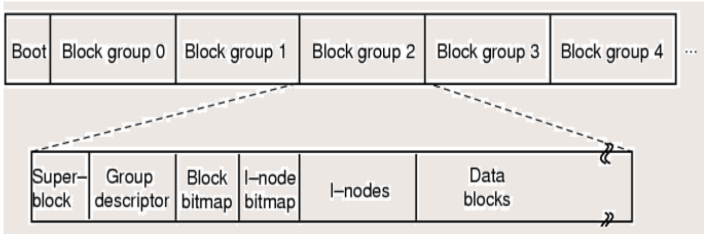
Characteristics

- optional block size at creation of an Ext2 file system (from 1KB to 4KB)
- good allocation strategy
- support for immutable and for append-only files
- a good implementation of file-updating strategy – minimize the impact of crashes
- support for automatic consistency checks on the file system status at boot time *(/sbin/e2fsck*)

Linux File System

Disk's structure

- Each partition is split into block groups
- Pre-allocates disk data blocks to regular files at adjacent positions in the same block group – reduces file fragmentation



Linux File System

Superblock structure

- total number of inodes
- filesystem size in blocks
- free blocks counter
- free inodes counter
- block size
- number of blocks per group
- number of inodes per group
- time of last mount operation
- time of last write operation
- mount operations counter
- number of mount operation before check
- magic signature
- size of on-disk inode structure
- block group number of this superblock
- number of blocks to pre-allocate

Linux File System

Group descriptors and bitmap

- block number of block bitmap
- block number of inode bitmap
- block number of first inode table block
- number of free blocks in the group
- number of free inodes in the group
- number of free directories in the group

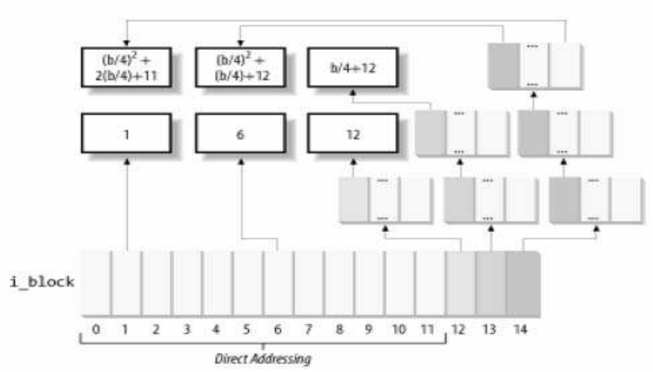
Linux File System

Inode structure

- All inodes have the same size = 128 bytes → a 1024 block contains 8 inodes
- Each inode contains
  - file type and access rights
  - owner identifier
  - file length in bytes (32 bits) => 4GB limit (actually 2GB)
  - time of last file access
  - time that inode last changed
  - group identifier
  - hard links counter
  - number of data blocks of the file
  - pointers to data blocks (BAT)

Linux File System

BAT structure



Linux File System

File management system calls

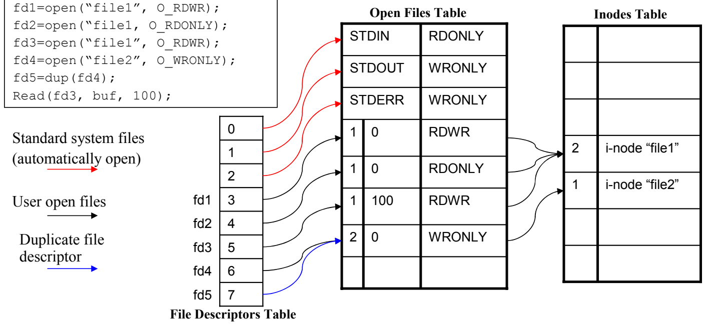
- fd=creat(name, acces_right)
  - access_rights: *0644 (rw-r—r--)*
- fd=open(name, mode)
  - mode: O_RDWR, O_RDONLY, O_APPEND etc.
- n=read(fd, buffer, nbytes)
- n=write(fd, buffer, nbytes)
- pos=lseek(fd, offset, whence)
- close(fd)
- dup, dupd2
- link
- stat, fstat

Linux File System

System data structures for open files

```
int fd1, fd2, fd3, fd4, fd5;
char buf[100];

fd1=open("file1", O_RDWR);
fd2=open("file1, O_RDONLY);
fd3=open("file1", O_RDWR);
fd4=open("file2", O_WRONLY);
fd5=dup(fd4);
Read(fd3, buf, 100);
```



Standard system files (automatically open)

User open files

Duplicate file descriptor

File Descriptors Table

	0
	1
	2
fd1	3
fd2	4
fd3	5
fd4	6
fd5	7

Open Files Table

STDIN		RDONLY
STDOUT		WRONLY
STDERR		WRONLY
1	0	RDWR
1	0	RDONLY
1	100	RDWR
2	0	WRONLY

Inodes Table

2	i-node "file1"
1	i-node "file2"

Linux File System

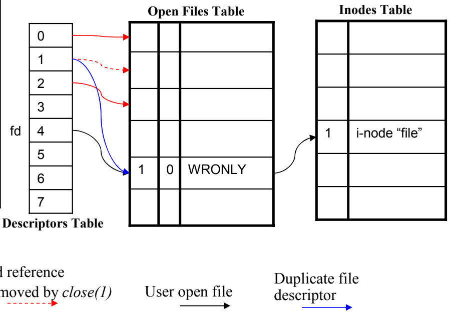
Output redirection of an application

```
int fd;

write(1, "msg", 3);
          // write on STDOUT
          // (on the screen)

// output redirection
fd=open("file", O_WRONLY);
close(1);
dup(fd); //dup2(fd, 1);

write(1, "msg", 3);
          // write into file
```



File Descriptors Table

	0
	1
	2
	3
fd	4
	5
	6
	7

Open Files Table

1	0	WRONLY

Inodes Table

1	i-node "file"

Standard files (automatically open)

Old reference Removed by *close(1)*

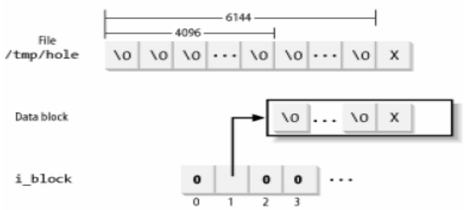
User open file

Duplicate file descriptor

Linux File System

File holes

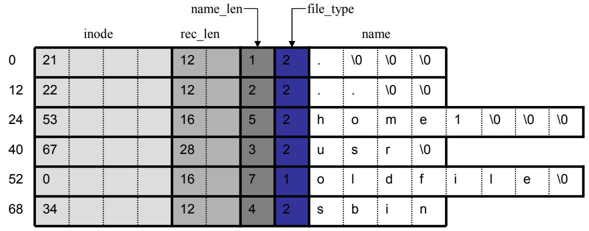
- Holes
  - portion of a regular file that contains null characters
  - not stored in any data block on disk
- Example
  - `echo –n “X” | dd of=/tmp/hole bs=1024 seek=6`



Linux File System

Directories’ structure

- Directory entry has a variable length
- Directory entry length acts as a pointer to the next valid directory entry
- *oldfile* was deleted, so the previous directory entry seems to be larger in order to point the next valid directory entry *sbin*



	inode	rec_len	name_len	file_type	name							
0	21	12	1	2	.	\0	\0	\0				
12	22	12	2	2	.	.	\0	\0				
24	53	16	5	2	h	o	m	e	1	\0	\0	\0
40	67	28	3	2	u	s	r	\0				
52	0	16	7	1	o	l	d	f	i	l	e	\0
68	34	12	4	2	s	b	i	n				

Windows 2000’s File System

Supported FSs

- Supports several FSs: FAT16, FAT32, NTFS (NT File System), CD-ROM’s FS
- FAT16
  - 16 bits → disk partitions of up to 2 GB
- FAT32
  - 32 bits → disk partitions of up to 2 TB
- NTFS
  - 64 bits → disk partitions of up to:
    - in theory: 2^{64}-1clusters (theoretically)
    - real (Windows XP): 2^{32}-1clusters → 16TB volumes for 4KB clusters

Windows 2000’s File System

Main Features

- Quickly perform file operations on very large-capacity storage units
- Build-in security and data compression system
- Transaction-processing model based on special logs → reliability and automatically recoverability

Windows 2000’s File System

Characteristics and Concepts

- File names’ length – up to 255
- Path names’ length limited to 32,767
- Supports Unicode characters
- Case sensitive
    - Win32 API does not fully support case-sensitivity!
- Hierarchical structure – tree of files and directories
    - Paths of files: *absolute* and *relative*
    - ‘\’ component separator
- A file is a collection of attributes of the form (*name*, *stream of bytes*)
- Attribute
    - Name specification: *file_name:attr_name*
    - Examples of attributes: file name, file ID, data
    - Maximum stream length = 2^{64} bytes

Windows 2000’s File System

File System API Calls – for files

- CreateFile: create or open a file; return a handle
- DeleteFile
- CloseHandle
- ReadFile
- WriteFile
- SetFilePointer
- GetFileAttributes
- LockFile: lock a region of the file
- UnlockFile: unlock a previously locked region

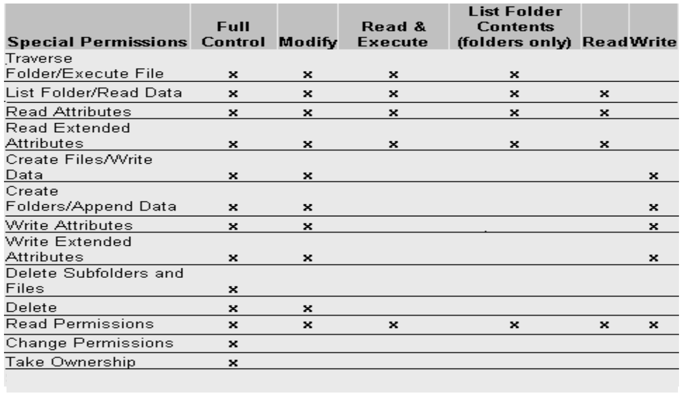
Windows 2000’s File System

File System API Calls – for directories

- CreateDirectory
- RemoveDirectory
- FindFirstFile
    - initialize to start reading the directory entries
- FindNextFile
    - read the next directory entry
- MoveFile
- SetCurrentDirectory

Windows 2000’s File System

Files and directories access rights



Special Permissions	Full Control	Modify	Read & Execute	List Folder Contents (folders only)	Read	Write
Traverse Folder/Execute File	×	×	×	×		
List Folder/Read Data	×	×	×	×	×	
Read Attributes	×	×	×	×	×	
Read Extended Attributes	×	×	×	×	×	
Create Files/Write Data	×	×				×
Create Folders/Append Data	×	×				×
Write Attributes	×	×				×
Write Extended Attributes	×	×				×
Delete Subfolders and Files	×					
Delete	×	×				
Read Permissions	×	×	×	×	×	×
Change Permissions	×					
Take Ownership	×					

Windows 2000’s File System

Volume structure (1)

- The basic NTFS disk unit is a volume
- Volume generally corresponds to a logical disk partition
- The fundamental unit of allocation on the hard disk is a cluster (block)
- Each volume is a linear sequence of fixed-sized blocks (clusters)
- Block size: 512 bytes – 64KB, depending on the volume size

Windows 2000’s File System

Volume structure (2)

Partition size	Sectors per cluster	Cluster size
512 MB or less	1	512 bytes
513 MB - 1024 MB (1GB)	2	1K
1025 MB - 2048 MB (2GB)	4	2K
2049 MB - 4096 MB (4GB)	8	4K
4097 MB - 8192 MB (8GB)	16	8K
8193 MB – 16,384 MB (16GB)	32	16K
16,385 MB - 32,768 MB (32GB)	64	32K
> 32, 768 MB	128	64K

Default block size depending on the volume size

Windows 2000’s File System

Volume structure (3)

- NTFS compression cannot be used when the cluster size is greater than 4KB
- 4KB is the most used
  - good compromise between large and small blocks
- Each block is referred to by its offset or address (a 64 bits number) → 2^{64} clusters
- Supposing a cluster size of 1K that means a 2^{64} * 1K = 16 million TB hard disk size

Windows 2000’s File System

Volume structure (4)

- The first information on an NTFS volume is the Partition Boot Sector (PBS)
- PBS starts at sector 0 and can be up to 16 sectors
  - **BIOS Parameter Block (BPB) and Extended BPB**
  - **Code** that is the OS loader (NTLDR)
- A duplicate of the Partition Boot Sector
  - at the end of the volume (Windows NT version 4.0)
  - in the logical center of the volume (Windows NT version 3.51 and earlier)

Windows 2000’s File System

Partition Boot Sector

Byte Offset	Field Length	Field Name
0x00	3 bytes	Jump Instruction
0x03	LONGLONG	OEM ID
0x0B	25 bytes	BPB
0x24	48 bytes	Extended BPB
0x54	426 bytes	Bootstrap Code
0x01FE	WORD	End of Sector Marker

No. 25

Windows 2000’s File System

BPB and Extended BPB

Byte Offset	Field Length	Sample Value	Field Name
0x0B	WORD	0x0002	Bytes Per Sector
0x0D	BYTE	0x08	Sectors Per Cluster
0x0E	WORD	0x0000	Reserved Sectors
0x10	3 BYTES	0x000000	*always 0*
0x15	BYTE	0xF8	Media Descriptor
0x16	WORD	0x0000	*always 0*
0x18	WORD	0x3F00	Sectors Per Track
0x1A	WORD	0xFF00	Number Of Heads
0x1C	DWORD	0x3F000000	Hidden Sectors
0x28	LONGLONG	0x4AF57F0000000000	Total Sectors
0x30	LONGLONG	0x0400000000000000	Logical Cluster Number for the file $MFT
0x38	LONGLONG	0x54FF070000000000	Logical Cluster Number for the file $MFTMirr
0x40	DWORD	0xF6000000	Clusters Per File Record Segment
0x44	DWORD	0x01000000	Clusters Per Index Block
0x48	LONGLONG	0x14A51B74C91B741C	Volume Serial Number
0x50	DWORD	0x00000000	Checksum

No. 26

Windows 2000’s File System

NTFS General Structure

- Everything on the volume is a file and everything in a file is an attribute
- Every sector on an NTFS volume that is allocated belongs to some file, even the system metadata
- The main file on every volume is called MFT (Master File Table)

No. 27

Windows 2000’s File System

Master File Table (MFT)

- Organized as a linear sequence of 1KB records
- A record for each file or directory
    - file name, time stamps, addresses of blocks
- Contains information about all the files and folders on the NTFS volume
- MFT is itself a file → it must not be in a fixed place on the HDD
- The first 16 records are reserved for metadata files
- The address of the first block of MFT is stored in the boot block at installation

No. 28

Windows 2000’s File System

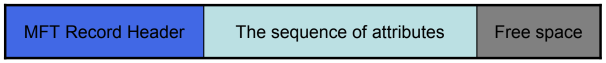
System Metadata Files

File	MFT Record	Purpose
$MFT	0	A list of all contents of the NTFS volume.
$MFTMirr	1	Mirror of part of MFT
$LogFile	2	Log file use to recover from crashes.
$Volume	3	Volume file: name, volume dirty flag, NTFS version etc
$AttrDef	4	Attribute definitions file: attribute names, numbers, and descriptions
$.	5	Root directory
$Bitmap	6	A bitmap for keeping track of used and free blocks.
$Boot	7	Bootstrap loader, if the volume is bootable.
$BadClus	8	Bad cluster file: the list of all bad clusters on the volume.
$Secure	9	Security descriptors for all files.
$UpCase	10	Case conversion table.
$Extend	11	Extensions: $Quota, $ObjId, $Reparse, $UsnJrnl
	12-15	Reserved for future use.
User file1	16	
…….		

Windows 2000’s File System

MFT File Record

- A MFT Record
  - Header
  - A sequence of (*attr_header*, *attr_value*) pairs


MFT Record Header | The sequence of attributes | Free space

- Each file has at least one MFT record
  - Small files and small directories need one record
  - Large files and small directories need more records
    - » the first = base record
    - » the others = extended records

Windows 2000’s File System

The header of MFT File Record

- Magic number
- Sequence number: incremented each time the record is reused for a new file
- Count of references to the file
- Flags: 00 – free, 01 – used, 02 – directory
- Number of bytes used in the record
- The identifier of the base record
  - 0 – for base records
  - (an index or sequence number) – for extended records
- A pointer to the first attribute in the record
- A pointer to the first free byte in the record

Windows 2000’s File System

The attribute types – NTFS

No	Attribute type	Description
1	$STANDARD_INFORMATION	Include information such as owner, timestamps, flag bits, link count etc.
2	$ATTRIBUTE_LIST	Location of extension MFT records, if attributes don’t fit in MFT record.
3	$FILE_NAME	Repeatable attribute for short (MS-DOS) or long (max 255) Unicode name
4	$SECURITY_DESCRIPTOR	Obsolete. Security information is now in $Extend$Secure
5	$OBJECT_ID	64-bit file identifier unique on this volume
6	$REPARSE_POINT	Used for mounting and symbolic links
7	$VOLUME_NAME	Name of this volume (used only in $Volume)
8	$VOLUME_INFORMATION	Volume version (used only in $Volume)
9	$INDEX_ROOT	Used for directories
10	$INDEX_ALLOCATION	Used for very large directories
11	$BITMAP	Used for very large directories
12	$LOGGED_UTILITY_STREAM	Controls logging to $LogFile
13	$DATA	Stream data; may be repeatable

Windows 2000's File System

The attributes of MFT record

- A file = a sequence of attributes
- An attribute = header + value (stream)
- Resident attribute – its value fits in MFT record
  - Its value fits in the MFT record beside its header
  - Attributes that are always resident
    - $FILE_NAME, $STANDARD_INFORMATION, and $SECURITY
  - Immediate files (few hundred size)
    - $DATA attribute is resident
- Nonresident attributes – its value doesn't fit
    - Are allocated one or more disk clusters elsewhere on the disk
- Some attributes may be repeated, but all attributes must appear in a fixed order in the MFT record

Windows 2000's File System

The attributes' header – resident

Offset	Length	Description
0	4	Type
4	4	Length
8	1	Non-resident flag
9	1	Name length
A	2	Offset to the stream
C	2	Compressed flag
E	2	Identifier
10	4	Length of the stream
14	2	Offset to the stream
16	2	Indexed flag

SIZE = 24 bytes

Windows 2000's File System

The attributes' header – nonresident

Offset	Length	Description
0	4	Type
4	4	Length
8	1	Non-resident flag
9	1	Name length
A	2	Offset to the stream
C	2	Compressed flag
E	2	Identifier
10	8	Starting VCN
18	8	Last VCN
20	2	Offset to the run list
22	2	Number of compression engine.
28	8	Allocated size of the stream
30	8	Real size of the stream
38	8	Initialized data size for the stream

SIZE = 64 bytes

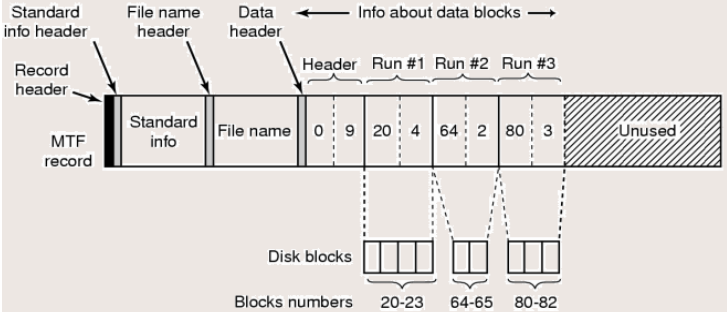
Windows 2000's File System

The attributes' value (stream)

- For resident attribute
    - The value follows the attribute header in MFT record
- For non-resident attribute
    - Large size streams (example: large files)
    - Need for extra clusters allocation – the stream
    - Need for extra data mapping VCN onto LCN – in header
      - VCN (Virtual Cluster Number) = a relative cluster offset within the attribute's data
      - LCN (Logical Cluster Number) = the location on the disk where the data resides
    - Mapping information is a sequence of records based on *runs* of consecutives blocks

Windows 2000’s File System

The nonresident attributes’ stream (1)



MFT entry of a small file

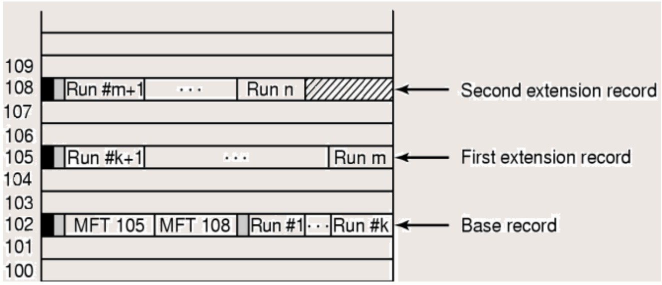
Windows 2000’s File System

The nonresident attributes’ stream (2)

- Record = header + sequence of (start LCN, count) pairs
- Header
  - VCN of the first block within the file
  - VCN of the first uncovered block
- Files without holes - written in order from beginning to end
  - need one record
- Files with holes – not continuously written from beginning to end
  - need more records
  - For example: if only blocks 0-49 and 60-79 are defined → two records with (0,50) and (60, 80) headers

Windows 2000’s File System

The nonresident attributes’ stream (3)



For large or highly fragmented files an $ATTRIBUTE_LIST is used for extended records

Windows 2000’s File System

Special features of NTFS5 (1)

- Attribute indexing
  - A generalization of the method used for directories
  - NTFS5 uses general indexing to manage security descriptors, quota information, reparse points, and file object identifiers
- Reparse points
  - associate data and code with a file or directory
  - used to implement *mount points*, NTFS *junctions*, and *Hierarchical Storage Management* (HSM)
- Quota tracking

Windows 2000’s File System

Special features of NTFS5 (2)

- Distributed link tracking
  - DLT automatically updates *shell links* (*shortcuts*) to point at moved link sources
  - link source's original and final locations must both be on NTFS5 volumes in the same domain
  - based on unique IDs associated to files
- Sparse files
  - unused portions can be indicated as being empty → release disk space
- Alternate data streams
  - a way to embed files within other files
  - every file contains an embedded file that has no name - *default* or *unnamed data stream*
  - Example: Summary information
  - `echo hello > file.txt:alternatestream`
  - `more < file.txt:alternatestream`

Bibliography

[Tann01]
Andrew Tannenbaum, *“Modern Operating Systems”*, second edition, Prentice Hall, 2001, pg. 830 – 842, pg. 732 – 744

[R98]
Mark Russinovich, *‘‘Inside NTFS’’*, January 1998, www.winnetmag.com

[R00-1]
Mark Russinovich, *‘‘Inside Win2K NTFS, Part 1’’*, November 2000, www.winnetmag.com

[R00-2]
Mark Russinovich, *‘‘Inside Win2K NTFS, Part 2’’*, Winter 2000, www.winnetmag.com

[BC01]
D. Bovet, M. Cesati, “Understanding Linux Kernel”, O’Reilly, 2001, pg. 495 – 523

[WWW]
www.ntfs.com