Slide 1

Week 12

COMP3231 Operating Systems

2005 S2

File Systems, Part 2:

➜ Case Study I: UNIX/Linux

➜ Case Study II: Windows NTFS

Slide 2

Traditional Unix File Management

➜ We will focus on two types of files:

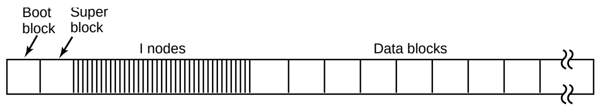
- Regular files
- Dictionaries

➜ And mostly ignore the others:

- Device files
- Symbolic links
- Pipes, sockets, etc

Slide 3

UNIX disk partition



➜ Block 0 not used by UNIX, often boot code

➜ Block 1: superblock contains information about layout of file system:

- number of i-nodes
- number of disk blocks
- start of free list

➜ i-nodes

➜ data blocks

➜ directories consist of 16-byte entries containing file name (max 14 chars) and i-node number

Slide 4

Unix I-Nodes

➜ Each file is represented by an i-node

➜ i-node contains meta-data of file

- attributes
- part of the block index table of the file

➜ each i-node has a unique number

- system oriented name
- try `ls -i`

➜ directories map file names to i-node numbers

- maps human oriented to machine oriented identifier
- hard links: mapping of many to one

Slide 5

DIRECTORIES

To open file in current dir:

- ➜ system reads through dir entries and compares names
- ➜ if found, extracts i-node number
- ➜ puts i-node in i-node table (kernel data structure)

What is the difference between

- ➜ `ls .`
- ➜ `ls /home/keller/work/projects/polymer/c`

if `/home/keller/work/projects/polymer/c` is current working directory?

Slide 6

mode
uid
gid
atime
ctime
mtime
size
block count
ref count
10 direct blocks
single indirect
double indirect
triple indirect

i-node contents:

- ➜ mode
  - type: regular file or directory?
  - access mode; `rwxrwxrwx`
- ➜ uid
  - user id
- ➜ gid
  - group id

Slide 7

mode
uid
gid
atime
ctime
mtime
size
block count
ref count
10 direct blocks
single indirect
double indirect
triple indirect

i-node contents:

- ➜ atime
  - time of last access
- ➜ ctime
  - time of creation
- ➜ mtime
  - time of last modification

Slide 8

mode
uid
gid
atime
ctime
mtime
size
block count
ref count
10 direct blocks
single indirect
double indirect
triple indirect

i-node contents:

- ➜ size
  - size of the file in bytes
- ➜ block count
  - number of blocks used by file
  - is not file size / block size
  - file can be sparsely populated:

    `write (f, "hello"); lseek (f, 10000000); write (f, "bye");`
  - only requires 2 blocks for data, but size of file is 10000003 bytes

Slide 9

How do we store files with more than 10 blocks?

- ➜ add more direct entries?
  - ✘ many unused entries for average sized files

Single indirection:

- ➜ entry points to a block on disk with contains block numbers

Slide 10

Single Indirection

- ➜ blocks referenced through the single indirect block require two disk accesses to be read:
  - one to read the index block
  - one to read the actual data block
- ➜ what is the max. file size now?
  - assume 1kbyte block size
  - 4byte block numbers
  - 10 * 1kbyte + (1/4) * 1 kbyte = 266 kbyte
- ➜ for most files (<266K), at most two disk accesses required to read any block

Slide 11

Double and Triple Indirection

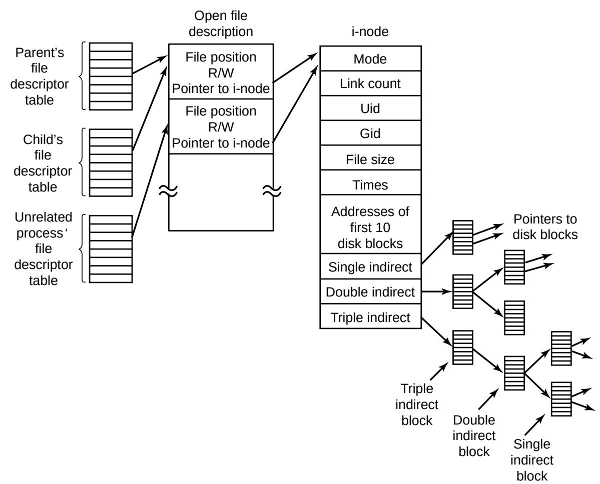
Double Indirect Block:

- ➜ a block on disk containing block numbers for single indirect blocks
- ➜ ie, a block containing block numbers of blocks which contain block numbers

Triple Indirect Block:

- ➜ a block on disk containing block numbers for double indirect blocks
- ➜ ie, a block containing block numbers of blocks containing block numbers of blocks which contain block numbers

Slide 12



Slide 13

File Size

What is the max file size?

- ➜ again, assume 1k blocks, 4byte block numbers
- ➜ direct blocks: 10
- ➜ single indirect: 256
- ➜ double indirect: 256 * 256 = 65536
- ➜ triple indirect: 256 * 256 * 256 = 1677716

Max. file size: 16GB

Slide 14

Access Patterns

Read one byte:

- ➜ best: 1 read access via direct block
- ➜ worst: 4 read accesses, via triple indirect block

Write one byte:

- ➜ best: 1 write access via direct block (in case there is no previous content)
- ➜ worst: 4 read accesses, via triple indirect block, 1 write (previous content)

Slide 15

Access Patterns

What happens if a (triple indirectly referenced) block is not allocated yet?

- ➜ no indirection block is allocated yet:
  - 4 writes: 3 indirect blocks, 1 data block
- ➜ only single indirect block is allocated:
  - 1 read, 4 writes: read single indirect, write single indirect, write double indirect, write triple indirect, write data
- ➜ single and double indirect block are allocated:
  - 2 read, 3 writes: read single indirect, read double indirect, write double indirect, write triple indirect, write data
- ➜ single, double, and triple indirect blocks are allocated:
  - 3 read, 2 writes: read single indirect, read double indirect, read triple indirect, write triple indirect, write data

Slide 16

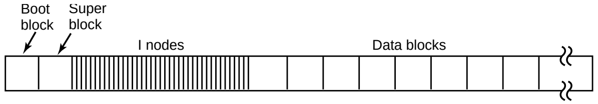
I-node Summary

- ➜ contain on disk data associated with a file
- ➜ provide efficient random and sequential access
- ➜ good support of small files
- ➜ large files require progressivly more disk accesses for random access
- ➜ sequential access for large files still efficient

Problems with s5fs

Slide 17

Let us have another look at the disk layout:



- ➜ i-nodes at start of disk, data blocks at the end
  - poor locality, we must read i-node before data block
- ➜ only single super block
  - entire file system is lost if superblock is corrupted
- ➜ block allocation
  - no support for consecutive block allocation
- ➜ i-node allocation
  - random
  - listing a directory results in random i-node access patterns

Berkeley Fast Filesystem (FFS)

Slide 18

- ➜ successor of s5fs
- ➜ Linux file system very similar
- ➜ we discuss Linux fs

Linux Ext2 File System

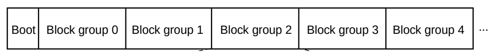
Slide 19

- ➜ Second extended file system
  - evolved from Minix filesystem (via "extended file system")
- ➜ features:
  - supports different block sizes: 1024, 2046, 4096
  - block size configured at FS creation
  - blocks groups to increase locality
  - symbolic links < 60 characters are stored within i-node
- ➜ problems:
  - unclean unmount (`e2fsck`)
  - ext3fs keeps journal of meta-data updates
  - journal contains update logs
  - compatible with ext2fs

Layout of Ext2fs Partition

Slide 20

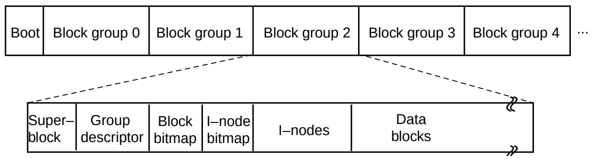
- ➜ disk divided into one or more partitions
- ➜ partition:
  - reserved boot block
  - collection of block groups
  - all block groups have same size and structure



Slide 21

Layout of Block Group

- ➜ replication of superblock on each group
- ➜ group descriptors
- ➜ bitmaps identify i-nodes/blocks
- ➜ all block groups have same number of data blocks
- ➜ advantages:
  - replications simplifies recovery
  - proximity of i-node tables and data blocks



Slide 22

Superblocks

Contain:

- ➜ size of file system
- ➜ overall free i-node, block counters
- ➜ data indicating if filesystem check is needed:
  - cleanly unmounted?
  - inconsistent?
  - number of mounts since last check
  - time expired since last check

Is replicated to add to recoverability

Slide 23

Group Descriptors

- ➜ location of bitmaps
- ➜ counter for free blocks in group
- ➜ counter for i-nodes in group
- ➜ number of directories in group

Slide 24

Performance Considerations

Ext2 optimisations:

- ➜ read ahead for directories (directory searching)
- ➜ block groups cluster related i-nodes and data blocks
- ➜ pre-allocation of blocks to write (up to 8 blocks)
  - 8 bit in tables
  - better contiguity

FFS optimisations:

- ➜ files within a directory in the same group

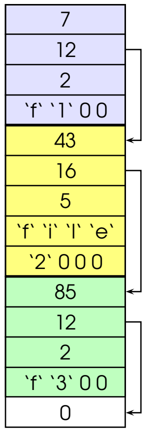
Slide 25

Ext2fs Directories

- ➜ i-node describe file layout on disk
- ➜ i-node are an internal structure
- ➜ user deals with name of files, not i-node number
- ➜ directories are file of special format, managed by kernel
- ➜ directories translate names to i-nodes numbers
- ➜ directory entries have variable length
- ➜ entries can be deleted in place
  - set i-node number to 0
  - add to length of previous entry

Slide 26

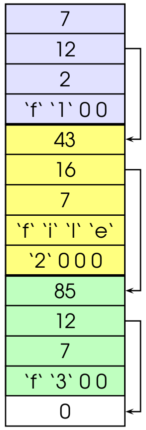
Ext2fs Directories



- ➜ "f1" = i-node 7
- ➜ "file2" = i-node 43
- ➜ "f3" = i-node 85

Slide 27

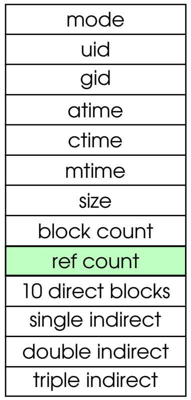
Ext2fs Directories



- ➜ i-node can have more than one name!
- ➜ called a hard link
- ➜ i-node 7 has three names
  - "f1" = i-node 7
  - "file2" = i-node 7
  - "f3" = i-node 7

Slide 28

I-node Contents



- ➜ possibly many names for same i-node
- ➜ when we delete file identified by name we always remove directory entry
- ➜ how can system decide when to delete underlying i-node?
- ➜ keeps a reference count in i-node
  - adding directory entry increments counter
  - removing entry decrements counter
  - if counter is zero, delete i-node

Slide 29

Kernel Data Structures and Interfaces

- ➜ We know how files and directories are stored on disk
- ➜ We know the UNIX system call interface
- ➜ What is inbetween?

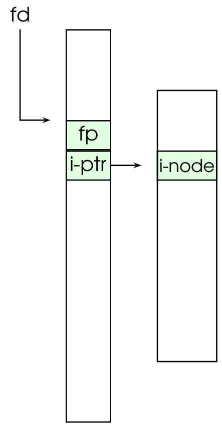
Slide 30

We need to keep track of

- ➜ File descriptors
  - each open file has a file descriptor
  - file operations use them to specify which file to operate on
- ➜ File pointer
  - where in the file is the next read performed?
- ➜ Mode
  - how was the file opened?

Slide 31

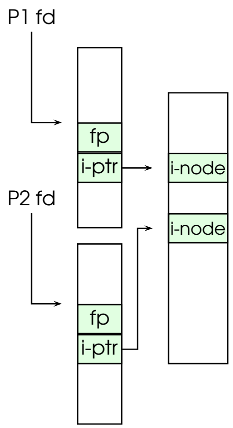
An Option?



- ➜ use the i-node numbers as file descriptors, add file pointer to i-node
- ➜ what happens if two processes open the same file?
- ➜ we need two separate file descriptors and file pointers?

Slide 32

Per Process File Descriptor Array



Idea:

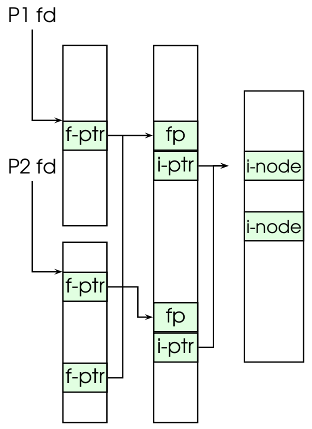
- ➜ each process has its own open file array
- ➜ contains fp, i-ptr
- ➜ fd 1 can be any i-node, different for each process

Issues:

- ➜ `fork`, `dup2` define that child shares file pointer with parent
- ➜ with per-process table, we can only have independent file pointers

Slide 33

Per Process fd table, Global open file table



- ➜ per-process fd array contains pointer to open file table entry
- ➜ open file table array contains entries with a fp and pointer to i-node
- ➜ supports
  - sharing of file pointers
  - independent file pointers
- ➜ Example:
  - all three fds refer to same file
  - two share a fp
  - one has independent fp
- ➜ used by Linux, most UNIXes

Slide 34

Supporting Multiple File Systems

- ➜ older OS supported only a single file system
- ➜ `open`, `close` etc had system specific implementations
- ➜ open file table pointed to in-memory representation of i-node
- ➜ i-node format specific to file system
- ➜ modern OSs need to support many different file systems
  - ISO9660 (CDROM)
  - MSDOS (floppy)
  - ext2fs

Slide 35

Supporting Multiple File Systems

Alternative:

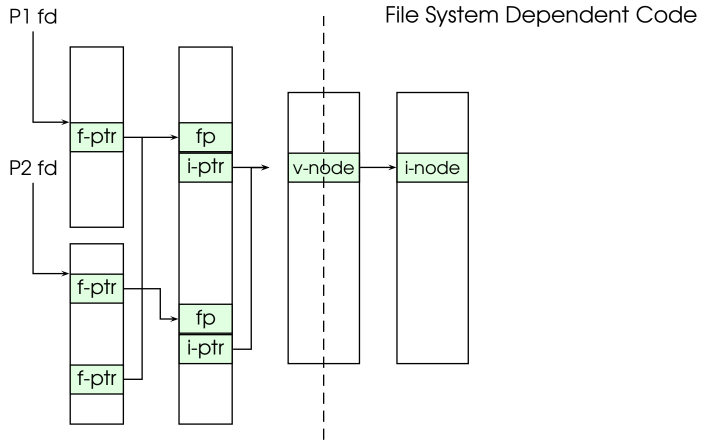
- ➜ change file system code to understand different file sysyem types
  - leads to code bloat, complex, hard to extend
- ➜ add abstraction layer to separate file system independent code
  - allows different fs to be "plugged-in"
  - large part of infrastructure is independent of specific file system

Slide 36

Virtual File System

- ➜ Provides uniform interface to many file systems
- ➜ Transparent handling of network file systems
- ➜ File-based interface to arbitrary device drivers (`/dev`)
- ➜ File-based interface to kernel data structures (`/proc`)
- ➜ Provides indirection layer for system calls
  - file operation table set up at file open time
  - points to actual handling of code for particular type
  - further file operations redirected to those functions

Slide 37



Slide 38

VFS INTERFACE

Two major data types:

➜ VFS

- represents all file system types
- contains pointers to functions to manipulate each file system as a whole (mount etc)

➜ v-node

- represents file in the underlying file system
- points to real node
- contains pointer to functions to manipulate files/i-nodes (open, read, etc)

Slide 39

BUFFER

Temportary storage used when transferring data between two different entities:

➜ especially useful when entities working at different rates, or

➜ unit of transfer incompatible

➜ e.g., application program and disk

Slide 40

BUFFERING DISK BLOCKS

➜ allow application to work with arbitrary sized regions of a file

- application can still optimise for certain block size

➜ writes can return immediately after copying to kernel buffer

- avoid waiting until write to disk is complete
- write is scheduled in the background

➜ can implement read-ahead by preloading next block on disk into kernel buffer

Slide 41

Cache

Fast storage used to temporarily hold data to speed up repeated access

- ➜ caching on access:
  - before loading from disk, check if in cache first
  - reduce number of disk accesses
  - can optimise for repeated access for single or several processes

Buffering and caching are related:

- ➜ data is read into buffer, extra cache copy would be wasteful
- ➜ after use, block should be put into cache
- ➜ future access may hit cached copy
- ➜ cache utilises unused kernel memory, may have to shrink

Slide 42

UNIX Buffer Cache

UNIX usually uses hashed buffer cache

- ➜ on read
  - hash device & block number
  - check if match in buffer cache (hash table)
- ➜ what happens if buffer is full?
  - choose entry to replace (FIFO, Clock, LRU,. . .)
  - disk accesses less frequent, take longer: different trade offs (LRU possible)
  - do we want LRU??
  - what is the difference between paged data in RAM, and file data in RAM?

Slide 43

File System Consistency

- ➜ File data is expected to survive crashes, power failure
- ➜ Strict LRU may keep data in cache for too long
- ➜ prioritise write back of disk blocks if they are important to consistency:
  - directory blocks
  - i-node blocks
- ➜ UNIX flush daemon (flushd) flushes modified blocks every 30 secs
- ➜ alternative: write-through cache
  - write modified blocks immediately
  - generates much more disk traffic
  - still used for some devices

Slide 44

The Network File System (NFS)

Sun Microsystem's Network file system joins file systems on separate computers to logical whole

- ➜ NFS Architecture
- ➜ NFS Protocol
- ➜ NFS Implementation

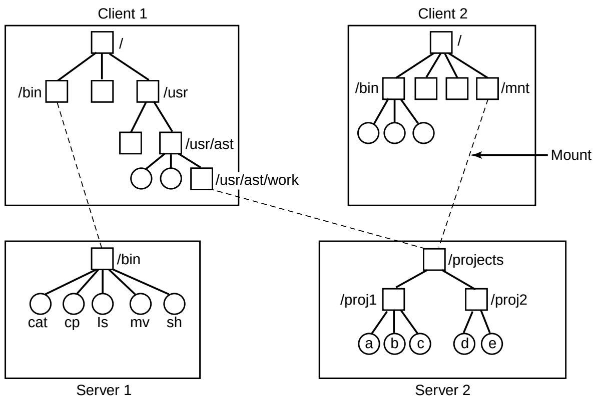
Slide 45

NFS Architecture

- ➜ concept of client and server machines (can be both at the same time)
- ➜ on the same LAN or connected through wide area network
- ➜ server:
  - exports directory trees for access by remote clients
- ➜ clients:
  - import directory trees by mounting them
  - becomes part of its own directory hierachy
  - mount point local to client

Slide 46

NFS Architecture



Slide 47

NFS Protocol

- ➜ NFS supports heterogeneous systems
- ➜ may run different operating systems

Mounting Protocol:

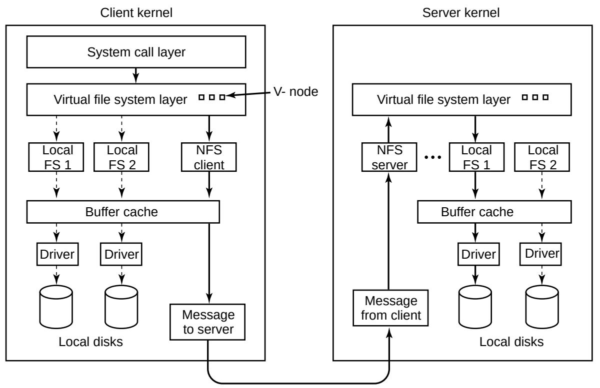
- ➜ client send path name to server
- ➜ requests permission to mount the directory
- ➜ does not specify mount point
- ➜ if pathname legal and exported, server returns file handle
- ➜ file handle contains
  - file system type
  - disk
  - i-node
  - security information
- ➜ static mounting or auto mounting supported
- ➜ most UNIX file system calls are supported
- ➜ not open and close

Slide 48

- server does not keep track of open files
- stateless
- how does locking work?

Slide 49

NFS Implementation



Slide 50

Windows File System

Several file systems supported

➜ FAT-16

- old MS-DOS file system
- 16-bit addresses
- disk partition limited to max of 2GB

➜ FAT-32

- 32-bit addresses
- disk partition limited to max of 2TB

➜ NTFS

- developed for NT
- 64-bit addresses

➜ Read-only file systems for CD-ROMs, DVDs

We'll have a closer look at NTFS

Slide 51

NTFS

➜ file is not just a linear sequence of bytes

➜ file consists of multiple attributes

➜ each attribute represented as stream of bytes:

- name of file
- 64-bit object id
- one or more data streams

Use of multiple data streams:

➜ Macintosh compatibility

➜ pack related data in same file

- full size and thumbnail version of picture
- previous and current version of a document

Slide 52

File System APIs

➜ similar to UNIX

➜ more parameters

➜ different security model

Win32 API function	UNIX	Description
CreateFile	open	Create a file or open an existing file; return a handle
DeleteFile	unlink	Destroy an existing file
CloseHandle	close	Close a file
ReadFile	read	Read data from a file
WriteFile	write	Write data to a file
SetFilePointer	lseek	Set the file pointer to a specific place in the file
GetFileAttributes	stat	Return the file properties
LockFile	fcntl	Lock a region of the file to provide mutual exclusion
UnlockFile	fcntl	Unlock a previously locked region of the file

Slide 53

Parameters to `CreateFile` function:

① pointer to file name
② flags to specify if file can be read/written/both
③ flags to specify if multiple processes can open the file
④ pointer to security descriptor
⑤ flags to specify what to do if file exists/does not exist
⑥ attributes
⑦ handle to file whose attributes should be cloned

Slide 54

```
/* Open files for input and output. */
inhandle = CreateFile("data", GENERIC_READ, 0, NULL, OPEN_EXISTING, 0, NULL);
outhandle = CreateFile("newf", GENERIC_WRITE, 0, NULL, CREATE_ALWAYS,
    FILE_ATTRIBUTE_NORMAL, NULL);

/* Copy the file. */
do {
    s = ReadFile(inhandle, buffer, BUF_SIZE, &count, NULL);
    if (s && count > 0) WriteFile(outhandle, buffer, count, &ocnt, NULL);
} while (s > 0 && count > 0);

/* Close the files. */
CloseHandle(inhandle);
CloseHandle(outhandle);
```


Slide 55

Win32 API function	UNIX	Description
CreateDirectory	mkdir	Create a new directory
RemoveDirectory	rmdir	Remove an empty directory
FindFirstFile	opendir	Initialize to start reading the entries in a directory
FindNextFile	readdir	Read the next directory entry
MoveFile	rename	Move a file from one directory to another
SetCurrentDirectory	chdir	Change the current working directory

Slide 56

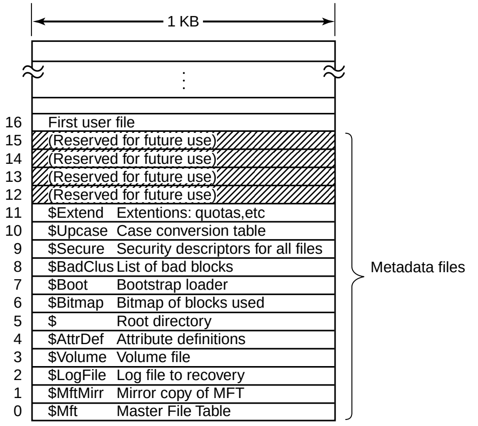
IMPLEMENTATION OF NTFS

➜ Each NTFS partition (volume) contains
  - files
  - directories
  - bitmaps
  - other admin data structures

➜ Each partition is a linear sequence of blocks (clusters)
  - block size fixed for each cluster
  - 512 bytes to 64 KB, usually 4KB

➜ Master File Table (MFT):
  - sequence of 1KB records
  - each entry describes one file or directory
  - large files may require more than one MFT record (list of)
  - bitmap used to keep track of free MFT records
  - regular file, can be placed anywhere on disk
  - can grow to have up to 2^{48} records

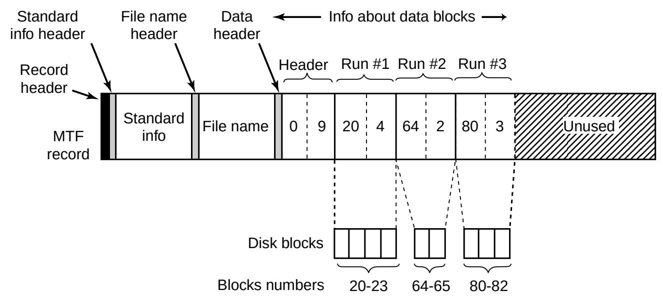
Slide 57



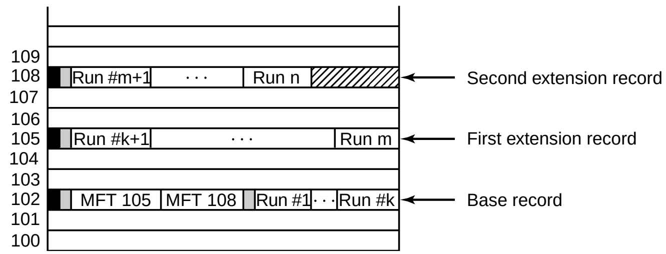
Slide 58

Attribute	Description
Standard information	Flag bits, timestamps, etc.
File name	File name in Unicode; may be repeated for MS-DOS name
Security descriptor	Obsolete. Security information is now in $Extend$Secure
Attribute list	Location of additional MFT records, if needed
Object ID	64-bit file identifier unique to this volume
Reparse point	Used for mounting and symbolic links
Volume name	Name of this volume (used only in $Volume)
Volume information	Volume version (used only in $Volume)
Index root	Used for directories
Index allocation	Used for very large directories
Bitmap	Used for very large directories
Logged utility stream	Controls logging to $LogFile
Data	Stream data; may be repeated

Slide 59



Slide 60



NTFS Directories

Slide 61

Small Directories:

- ➜ collection of directory entries
- ➜ each describes a file or directory
- ➜ each entry consists of
  - index of MFT entry
  - length of file name
  - other flags and fields
- ➜ looking up a file potentially involves examining all the file names in the directory

Large Directories:

- ➜ use B-trees to for alphabetical lookpu
- ➜ easy to insert new entries at the right place

Slide 62

File Name Lookup: