CS 537
Lecture 17
NTFS internals

Michael Swift

NTFS Recoverability

PC disk I/O in the old days: Speed was most important

NTFS changes this view – Reliability counts most:

- I/O operations that alter NTFS structure are implemented as atomic transactions
  - Change directory structure,
  - extend files, allocate space for new files
- Transactions are either completed or rolled back
- NTFS uses redundant storage for vital FS information
  - Contrasts with FAT / HPFS on-disk structures, which have single sectors containing critical file system data
  - Read error in these sectors -> volume lost

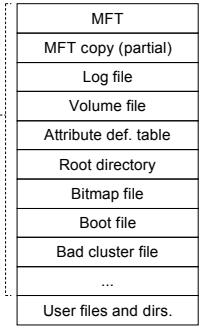
NTFS On-Disk Structure

- Volumes correspond to logical partitions on disk
- Fault tolerant volumes may span multiple disks
  - Same block stored on separate disks for redundancy
- Volume consists of series of files + unallocated space
  - FAT volume: some areas specially formatted for file system
  - NTFS volume: all data are stored as ordinary files
    - e.g. inodes stored in files
- NTFS refers internally to clusters
  - Cluster factor: #sectors/cluster; varies with volume size; (integral number of physical sectors; always a power of 2)
- Logical Cluster Numbers (LCNs):
  - refer to physical location == block number in Unix
  - LCNs are contiguous enumeration of all clusters on a volume

Master File Table

All data stored on a volume is contained in a file

- MFT: Heart of NTFS volume structure
  - Implemented as array of file records
  - One row for each file on the volume (including one row for MFT itself)
  - Metadata files store file system structure information (hidden files; $MFT; $Volume...)
  - More than one MFT record for highly fragmented files
  - Nfi.exe Utility from OEM Support Tools allows to dump MFT content (see support.microsoft.com/support/kb/articles/Q253/0/66.asp)



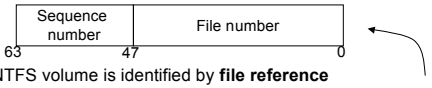
NTFS metadata

- **NTFS writes to log file ($LogFile)**
  - Record all commands that change volume structure
- **Root directory:**
  - When NTFS tries to open a file, it starts search in the root directory
  - Once the file is found, NTFS stores the file's MFT file reference
  - Subsequent read/write ops. may access file's MFT record directly
- **Bitmap file ($Bitmap):**
  - stores allocation state volume; each bit represents one cluster
- **Boot file ($Boot):**
  - Stores bootstrap code
  - Has to be located at special disk address
  - Represented as file by NTFS -> file ops. possible (!) (no editing)

NTFS metadata (contd.)

- Bad-cluster file ($BadClus)
  - Records bad spots on the disk
- Volume file ($Volume)
  - Contains: volume name, NTFS version
  - Bit, which indicates whether volume is corrupted
- Attribute Definition Table ($AttrDef)
  - Defines attribute types supported on the volume
  - Indicates whether they can be indexed, recovered, etc.

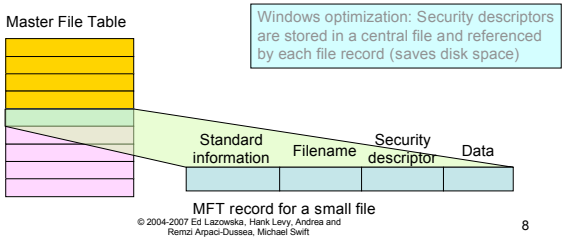
File Records & File Reference Numbers



- File on NTFS volume is identified by **file reference**
  - File number == index in MFT
  - Sequence number – used by NTFS for consistency checking; incremented each time a reference is re-used
  - Like inode numbers, but need not be allocated; just assigned
- File Records: File is collection of attribute/value pairs
  - Unnamed data attribute
  - Other attributes: filename, time stamp, security descriptor,...
  - Each file attribute is stored as separate stream of bytes within a file

File Records (contd.)

- NTFS doesn't read/write files:
  - It reads/writes attribute streams
  - Operations: create, delete, read (byte range), write (byte range)
  - Read/write normally operate on unnamed data attribute
- Record in MFT contains set of attributes
  - attributes are named
  - can be stored in MFT or externally



Standard Attributes for NTFS Files

Attribute	Description
Standard information	File attributes: read-only, archive, etc; time stamps; creation/modification time; hard link count
Filename	Name in Unicode characters; multiple filename attributes possible; short names for access by MS-DOS and 16-bin Win applications
Security descriptor	Specifies who owns the file and who can access it
data	Contents of the file; a file has one default unnamed data attribute; directory has no default data attrib.
Index root, index	Three attributes used to implement filename allocation, bitmap index for large directories (dirs. only)
Attribute list	List of attributes that make up the file and first reference of the MFT record in which the attribute is located (for files which require multiple MFT file records)

Attributes (contd.)

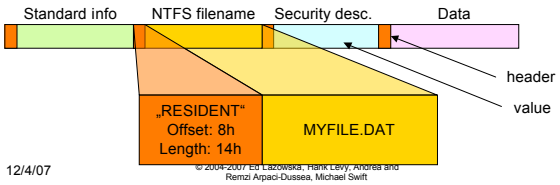
- Each attribute in a file record has a name and a value
- NTFS identifies attributes:
  - Uppercase name starting with $: $FILENAME, $DATA
- Attribute‘s value: Byte stream
  - The filename for $FILENAME
  - The data bytes for $DATA
- Attribute names correspond to numeric typecodes
- File attributes in an MFT record are ordered by typecodes
  - Some attribute types may appear more than once (e.g. Filename)

Filenames

- Name parsing, including wildcards, is handed by NTFS
  - In Linux, shell expands wildcards
- Directory traversal is handled by NTFS
  - The kernel hands it a full pathname
  - In Linux, an FS only expands one name at a time

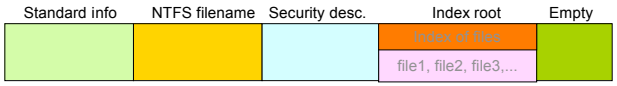
Resident & Nonresident Attributes

- Small files:
  - All attributes and values fit into MFT
  - Attribute with value in MFT is called „resident“
  - All attributes start with header (always resident)
  - Header contains offset to attr. value and length of value
  - Data is contained within data attribute
    - Very efficient for small files - no extra seeks/reads



Attributes (contd.)

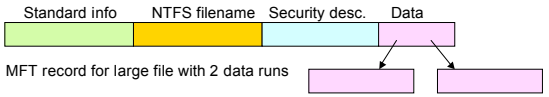
- Small directory:
  - index root attribute contains index of file references for files and subdirectories



MFT file record for a small directory

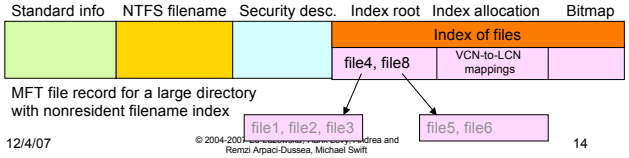
- If file attribute does not fit into MFT:
  - NTFS allocates separate cluster (run, extent) to store the values
  - NTFS allocates additional runs if an attribute's value later grows
  - Those attributes are called „non-resident"
  - Header of non-resident attribute contains location info

Large files & directories



MFT record for large file with 2 data runs

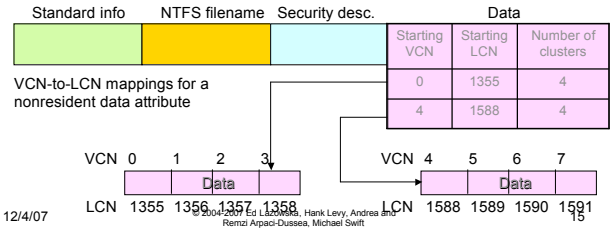
- Only attributes that can grow can be non-resident
- Filename & standard info are always resident
- Index of files for directories forms B+ tree



MFT file record for a large directory with nonresident filename index

Large files (contd.)

- NTFS keeps track of runs by means of VCN (Virtual Cluster Numbers)
  - Logical Cluster Numbers represent an entire volume
  - Virtual Cluster Numbers represent clusters belonging to one file
  - Attribute lists may extend over multiple runs (not only data)
- The run of clusters is called an extent
  - NTFS allocates new extents as necessary
  - When there is no more space left in the MFT entry, then another MFT entry is allocated. This design is effectively a list of extents, rather than the Unix tree of blocks.



Differences from Unix

- File names are stored in directory and MFT for files
  - Makes directory listings faster
- Extent-based vs. block based allocation
  - Provides better on-disk locality
- Extended support for attributes/streams
  - Files in NTFS need not be a single stream of bytes
  - But nobody uses this …

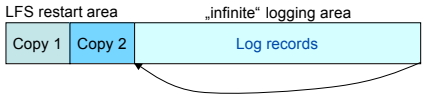
NTFS Recovery Support

- Transaction-based logging scheme
  - Avoids need to scan disk for errors
- Recovery is limited to file system data
  - User data may become corrupted
  - Why?
  - What could you do about it?
- Design options for file I/O & caching:
  - **Careful write**: VAX/VMS fs, other proprietary OS fs
  - **Lazy write**: most UNIX fs, OS/2 HPFS

Recoverable File System
(Journaling File System)

- Safety of careful write fs / performance of lazy write fs
- Log file of metadata updates
  - Optimization over lazy write: distance between cache flushes increased
  - No extra disk I/O to update fs data structures necessary: all changes to fs structure are recorded in log file which can be written in a single operation
- Log provided by "LFS" - log file service
  - A transaction contains a list of metadata updates needed for an operation
  - All or none must execute to leave disk uncorrupted

Log File Regions



- NTFS calls LFS to read/write restart area
  - Restart area: location of logging area to be used for recovery
    - 2 copies for reliability
  - Logging area: circularly reused
  - LFS uses logical sequence numbers (LSNs) to identify log records
- NTFS never reads/writes transactions to log file directly; only LFS does
- During recovery:
  - NTFS reads log forward; recorded transactions are redone
  - NTFS reads log backward; undo all incompletely logged transactions

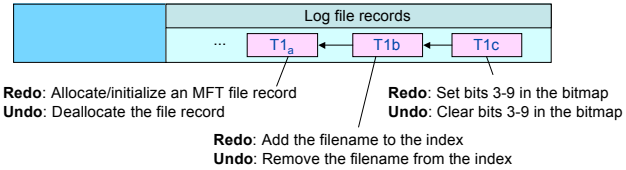
Operation of the LFS/NTFS

1. NTFS calls LFS to record in (cached) log file any transactions that will modify volume structure
2. NTFS modifies the volume (also in the cache)
3. LFS flushes log file to disk
4. NTFS flushes volume changes

-> Transactions of unsuccessful modifications can be retrieved from log file and un-/redone
Recovery begins automatically the first time a volume is used after system is rebooted.

Log Record Types

- Update records (series of ...)
  - Most common; each record contains:
  - **Redo information**: how to reapply on subop. of a committed trans.
  - **Undo information**: how to reverse a partially logged sub operation
- Last record commits the transaction (not shown here)



Recovery: redo committed/undo incompletely logged transact.

Log Records (contd.)

- Physical vs. logical description of redo/undo actions:
  - Delete file „a.dat“ vs. Delete byte range on disk
  - NTFS writes update records with physical descriptions
- NTFS writes update records (usually several) for:
  - Creating a file
  - Deleting a file
  - Extending a file
  - Truncating a file
  - Setting file information
  - Renaming a file
  - Changing security applied to a file
- Redo/undo ops. must be idempotent (might be applied twice)

Checkpoint Records

- NTFS periodically writes a checkpoint record
  - Describes, what processing would be necessary to recover a volume if a crash would occur immediately
  - How far back in the log file must NTFS go to begin recovery
  - LSN of checkpoint record is stored in restart area