CSC 539: Operating Systems Structure and Design

Spring 2006

File-system management
- files and directories
- directory organization
- access methods: sequential, direct, indexed
- file system and directory implementation
- allocation schemes: contiguous, linked, indexed
- performance and efficiency
- case study: Windows XP

File concept

the logical unit within a file system is the file
- logical files are mapped into physical entities by the OS
- in user's view, file is the smallest unit that can be saved to disk

attributes that a file might possess:
- *name* : provides handle for reference
  - DOS (8 chars + 3 char extension), Windows (unlimited? length)
  - UNIX (spaces tricky, no extension needed)

- *type* : indicates how the file should be treated
  - DOS/Windows rely on extension, can map extensions to programs
  - Mac OS X associates creator attribute with each file
  - UNIX uses "magic number", first few bytes of file specify file type

- *protection* : permissions, access control information
  - UNIX utilizes permission string: `chmod 644 foo.txt` → `-rw-r--r–`
    owner & group: `chown, chgrp`
  - Windows utilizes file properties/attributes: `NoAccess, List, Read, Read&Add, …`

- *location* & *size*
- *accounting information*

File operations

a file is an ADT, so we manipulate it through a set of operations
- *create* : find space on disk and make entry in directory
- *write* : write to file, requires positioning within the file
- *read* : read from file, involves positioning within the file
- *delete* : delete directory entry, reclaim disk space
- *reposition* : move read/write position

the OS must maintain information about all open files
- *file pointer* : the current position of the read/write pointer in the file
- *disk location* : the location of the file on the disk
- *file open count* : keep track of number of processes currently accessing the file

such a table of information allows the OS to enforce policies such as only one process can write to a file at a given time
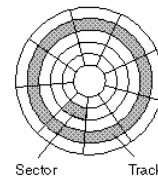
File structure

files can be stored physically as
- bytes
- lines
- records

whatever entity is stored, OS must map into a disk sector
- because on a physical disk, sectors are smallest writeable unit

Sector   Track

access methods:
- *sequential* : information in the file is accessed from first to last
  readNext, writeNext, reset

- *direct* : possible to reposition read/write pointer to any position
  such files are generally made up of fixed-length records
  readRecord N, writeRecord N, positionAt N, reset

- *indexed* : built on top of direct access, but accesses records in file using a key
  each record has a key associated with it, an index of keys is stored with the file
  readRecord KEY, writeRecord KEY, positionAt KEY, reset

Directory structure

to manage a large number of files, structure is essential

disks are broken into one or more *partitions*
- each partition is logically mounted separately
- each partition can have its own file system method (FAT, NTFS, …)

within each partition, a *device directory* keeps information about stored files
- may be viewed as a symbol table that maps file names to directory entries

directory operations include
- searching for a file
- creating a file
- deleting a file
- renaming a file
- traversing the file system (e.g., for backups)

5

Directory organization

single-level directory
- all files stored in same directory
- simple and easy to implement (used in IBM VM/CMS)
- not well suited to multiple users

two-level directory
- can assign separate directories to each user/account
- no further structure within the directories
- not used anymore – why limit to 2 levels?

most directory structures are hierarchical
- one top-level directory (root)
- within directory are file & other directories
- can define the location of a file/directory by its path from the root, traverse up and down
- note: if hard links (shared access) or soft links (aliasing) are allowed, then cycles are possible
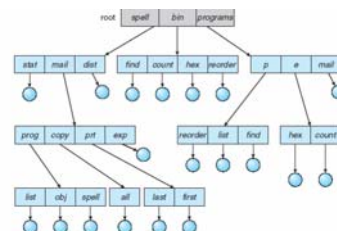
File system mounting

a file system must be mounted before it can be accessed by the OS
- the file system is attached to a directory (e.g., /home) & verified

- advantage: can have separate file systems for different types of files
    e.g., user accounts, system libraries, documentation, executables, …
- can even have different operating systems mounted from different partitions
    e.g., Linux/Windows dual boot

UNIX: mounts are explicit – configuration file lists devices & mount points, other mounts can be executed manually

Mac: when OS finds a disk (hard drives at boot, floppy when inserted), searches for file system and mounts at root level (& adds icon to desktop)

Windows: maintains a 2-level directory structure – each device and partition is assigned a drive letter

7

File sharing

with multiple users, file sharing is a necessity
- only need one copy of executables, documentation, …
- allow members of a team to access/update project files

common approach: links/shortcuts
- UNIX has hard links and soft/symbolic links
    - if delete a file, all hard links are deleted as well
    - if delete symbolic link, garbage link remains
- Windows shortcuts are equivalent to symbolic links
    - using NTFS, can create hard links (`fsutil hardlink create`)
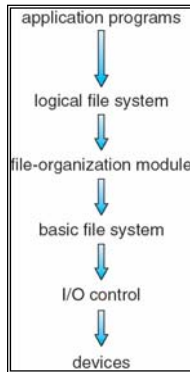
consistency semantics
- how does the OS handle simultaneous access to files?

- UNIX semantics: writes to an open file are immediately seen by all readers
    there is a mode where users share a pointer to a file, synchronized
- Session semantics: writes to an open file are seen only by users who later open it

8

4

File system implementation

file systems are implemented primarily on disks

- although other media are used, most of the ideas about implementing file systems were developed with disks in mind

- disks read and write *sectors* (usually 512 bytes)
- logical I/O transfers between CPU and disk involve *blocks/clusters* of sectors

```
application programs
        ↓
logical file system
        ↓
file-organization module
        ↓
basic file system
        ↓
I/O control
        ↓
    devices
```

file system is generally composed of many levels

- *logical file system* : manages directories, manipulates files as ADT's

- *file organization module* : maps logical block addresses to physical blocks, manages free space

- *basic file system* : issues generic commands to read/write physical blocks

- *I/O control* : consists of device drivers and interrupt handlers to transfer info between RAM and disk

9

File system structures

### the file system stores long-term information in secondary memory

- *boot control block:* info needed by the system to boot the OS from that volume
  known as *partition boot sector* in NTFS
- *volume control block:* contains # of blocks, block size, free-block pointers, …
  known as *master file table* in NTFS
- *directory structure:* represents the file/directory hierarchy
  known as *master file table* in NTFS
- *file-control block (FCB):* one per file, contains file name, size, owner, permissions, …

dynamic information is stored in main memory

- *mount table:* contains info about each mounted volume
- *directory-structure cache:* contains info on recently accessed directories
- *system-wide open file table:* FCB for each open file
- *per-process open file table :* with pointer to entry in system-wide table

10

5

Directory implementation: contiguous allocation
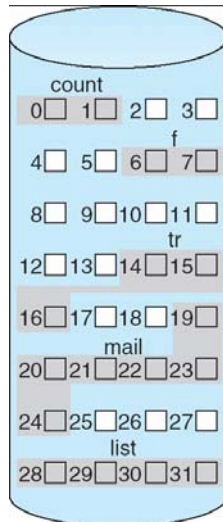
directory		
file	start	length
count	0	2
tr	14	3
mail	19	6
list	28	4
f	6	2

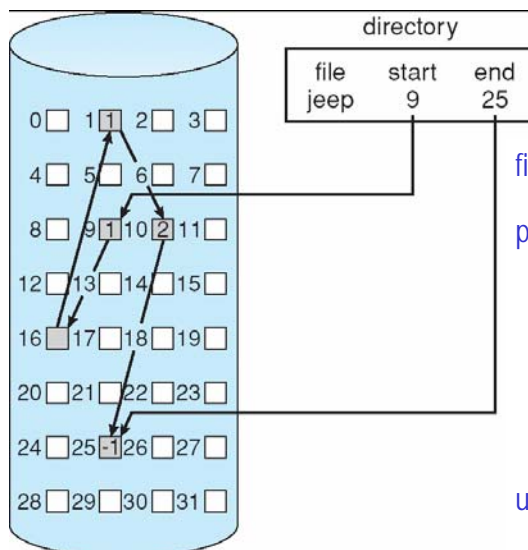each file occupies a contiguous set of blocks on the disk

problems:
- managing free space is complex, external fragmentation is possible
- total space needed for file is often unknown at creation
- over-allocation leads to internal fragmentation

used by the Veritas File System

11

Directory implementation: linked allocation

| directory		
file	start	end
jeep	9	25

file is a linked list of disk blocks

problems:
- only supports sequential access
- extra space is needed for pointers (can compromise by allocating blocks in contiguous clusters)
- a single bad sector can corrupt an entire file

used in MS-DOS & Windows
- File-Allocation Table (FAT) is stored in first block of the partition

12

Directory implementation: indexed allocation
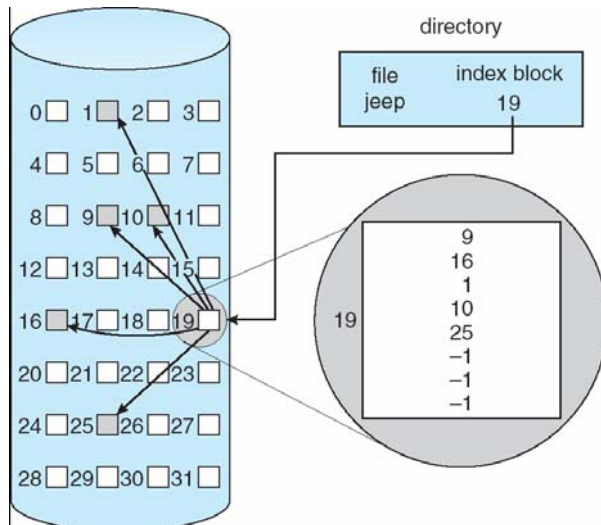
for each file, an index block stores pointers to all the blocks

problems:
- index block requires space
- large file may need several index blocks
- growing files may need index blocks dynamically extended

used in Unix File System
- uses multiple levels of indirection

13

Performance

performance depends greatly on how the system is used

contiguous allocation
- only one access required since can calculate where any particular block is

linked allocation
- since sequential, accessing Nth block requires N accesses

indexed allocation
- if index is in memory, 2 accesses required (but index might be very large)

some systems optimize by using different schemes for different file types
- direct access files use contiguous; sequential access files use linked
- small files use contiguous; large files use indexed

14

Free space management

the OS needs to keep track of free blocks on the disk
- bit vector
  use a bit string (Nth block is free/allocated, Nth bit is 1/0)

  e.g., blocks 2, 3, 4, 5, 8, 9, 10, 11, 12, 13, 17, 18, 25, 26 and 27 are free
  ```
  0011110011111100011000000011100000
  ```

  simple, but bit vector can be very large & needs to be in memory

- linked list
  use a linked list of free blocks
  slow and inefficient since it can't be easily cached

- grouping
  use a free block as an index to a group of free blocks

15

Efficiency and performance

efficient use of disk space is dependent on design choices, actual data
e.g., consider the choice of pointer size in directories, most use 16-bit or 32-bit pointers
→ file size limited to 2^{16} (64K) or 2^{32} (4G) blocks
64-bit pointers can be used to handle larger files, but take up more space

caching can improve performance
- cache an entire track, then read sectors from RAM
- cache requires effort to maintain consistency

hidden inefficiencies
- maintaining "last access time" requires that every access to the file also requires an update to the directory, requiring an extra disk access

16

8

Case study: Windows XP file system

on readable/writable disks, XP supports FAT and NTFS
- File Allocation Table (FAT) is simpler, backward compatible
- NTFS supports large drives (> 32 GB)
    security features: encryption, file/directory protections, transaction logging, …
    advanced features: quotas, compression, mounted drives, …

operating system and file system compatibility

Operating System	FAT16	FAT32	NTFS
Windows XP	•	•	•
Windows 2000	•	•	•
Windows NT 4.0[1]	•		•
Windows 95 OSR2, Windows 98, and Windows Me	•	•	
Windows 95 (prior to OSR2)	•		
MS-DOS	•		

Windows XP: clustering

a *cluster* is the smallest amount of disk space that can be allocated to a file
    smaller cluster size → more efficient use of disk (less fragmentation)
                more storage for directory (more, longer addresses)

administrator can specify cluster size when volume/partition is formatted
    • otherwise, default cluster sizes are used

Volume Size	FAT16 Cluster Size	FAT32 Cluster Size	NTFS Cluster Size
7 MB–16 MB	2 KB	Not supported	512 bytes
17 MB–32 MB	512 bytes	Not supported	512 bytes
33 MB–64 MB	1 KB	512 bytes	512 bytes
65 MB–128 MB	2 KB	1 KB	512 bytes
129 MB–256 MB	4 KB	2 KB	512 bytes
257 MB–512 MB	8 KB	4 KB	512 bytes
513 MB–1,024 MB	16 KB	4 KB	1 KB
1,025 MB–2 GB	32 KB	4 KB	2 KB
2 GB–4 GB	64 KB	4 KB	4 KB
4 GB–8 GB	Not supported	4 KB	4 KB
8 GB–16 GB	Not supported	8 KB	4 KB
16 GB–32 GB	Not supported	16 KB	4 KB
32 GB–2 terabytes	Not supported	Not supported[1]	4 KB

Windows XP: size limits

FAT16

Description	Limit
Maximum file size	4 GB minus 1 byte (2^{32} bytes minus 1 byte)
Maximum volume size	4 GB
Files per volume	Approximately 65,536 (2^{16} files)
Maximum number of files and folders within the root folder	512 (Long file names can reduce the number of available files and folders in the root folder.)

FAT32

Description	Limit
Maximum file size	4 GB minus 1 byte (2^{32} bytes minus 1 byte)
Maximum volume size	32 GB (implementation)
Files per volume	4,177,920
Maximum number of files and subfolders within a single folder	65,534 (The use of long file names can significantly reduce the number of available files and subfolders within a folder.)

NTFS

Description	Limit
Maximum file size	Theory: 16 exabytes minus 1 KB (2^{64} bytes minus 1 KB) Implementation: 16 terabytes minus 64 KB (2^{44} bytes minus 64 KB)
Maximum volume size	Theory: 2^{64} clusters minus 1 cluster Implementation: 256 terabytes minus 64 KB (2^{32} clusters minus 1 cluster)
Files per volume	4,294,967,295 (2^{32} minus 1 file)

19

Windows XP: FAT

File Allocation Table (FAT) file systems
- locates the file allocation table near the beginning of the volume
- the location of the FAT is specified in the boot sector (BIOS Parameter Block)
- actually, 2 copies of the FAT are stored for redundancy

- the FAT number refers to the number of bits per table entry
  - FAT12 → 2^{12} = 4M different clusters can be addressed (used for floppy disks)
  - FAT16 → 2^{16} = 64M different clusters can be addressed (MS-DOS compatible)
  - FAT32 → 2^{28} = 256G different clusters (4 bits are reserved)

| Boot Sector	Reserved Sector	FAT 1	FAT 2 (Duplicate)	Root Folder	Other Folders and All Files

organization of a FAT partition
  each FAT entry identifies a cluster as BAD, UNUSED, USED-IN-FILE, END-OF-FILE

20

10

Windows XP: FAT (cont.)

FAT utilizes a linked allocation scheme
- each cluster contains a pointer to the next cluster in the file
- a pointer with value (0xFFFF) marks the last cluster in the file

- FAT stores the file attributes (name, date, type, …) & pointer to first cluster on disk
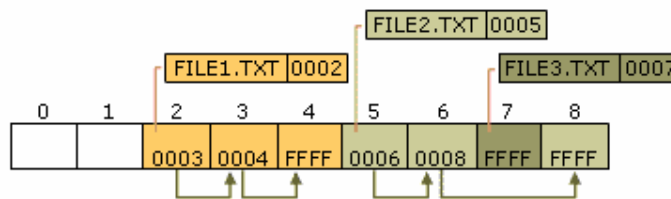
Windows XP: NTFS

NTFS is the preferred file system for Windows XP
- can set permissions to allow users/groups to share files, control types of actions
- can encrypt a file/folder using private and public key encryption
- can compress a file/folder, any Windows app will automatically expand as needed
- can enforce quotas on disk usage
- can mount new drives; create hard links to shared files/folders

- recovery features: each file operation broken down into atomic transaction
  maintains a transaction log – updates disk after each transaction
  if failure occurs during a transaction, info is sufficient to complete or rollback
  if a bad sector is found when writing, will automatically map to a different sector

NTFS utilizes
Master File Table (MFT): a relational database that stores file records & attributes
metadata files: are contained within the MFT to store relevant info and attributes

Windows XP: metadata files in the MFT

System File	File Name	MFT Record	Purpose of the File
Master file table	$Mft	0	Contains one base file record for each file and folder on an NTFS volume. If the allocation information for a file or folder is too large to fit within a single record, other file records are allocated as well.
Master file table mirror	$MftMirr	1	Guarantees access to the MFT in case of a single-sector failure. It is a duplicate image of the first four records of the MFT.
Log file	$LogFile	2	Contains a list of transaction steps used for NTFS recoverability. The log file is used by Windows XP Professional to restore consistency to NTFS after a system failure. The size of the log file depends on the size of the volume, but you can increase the size of the log file by using the Chkdsk command. For more information about the log file, see "NTFS Recoverability" earlier in this chapter. For more information about Chkdsk, see "Troubleshooting Disks and File Systems" in this book.
Volume	$Volume	3	Contains information about the volume, such as the volume label and the volume version.
Attribute definitions	$AttrDef	4	Lists attribute names, numbers, and descriptions.
Root file name index	.	5	The root folder.
Cluster bitmap	$Bitmap	6	Represents the volume by showing free and unused clusters.
Boot sector	$Boot	7	Includes the BPB used to mount the volume and additional bootstrap loader code used if the volume is bootable.
Bad cluster file	$BadClus	8	Contains bad clusters for a volume.
Security file	$Secure	9	Contains unique security descriptors for all files within a volume.
Upcase table	$Upcase	10	Converts lowercase characters to matching Unicode uppercase characters.
NTFS extension file	$Extend	11	Used for various optional extensions such as quotas, reparse point data, and object identifiers.
		12–15	Reserved for future use.

23

Windows XP: NTFS (cont.)

similar to FAT, the location of the MFT and backup MFT are stored in the boot sector

the MFT contains a record for each file
- record contains links to cluster groupings for that file
- if too large & fragmented, utilizes indexing (record points to external index block)

for large folders, contents are organized into a B-tree (balanced tree structure) to optimize searches

24