

10. File Systems

- 10.1 Basic Functions of File Management
- 10.2 Hierarchical Model of a File System
- 10.4 File Directories
 - Hierarchical Directory Organizations
 - Operations on Directories
 - Implementation of File Directories
- 10.5 Basic File System
 - File Descriptors
 - Opening and Closing of Files
- 10.6 Physical Organization Methods
 - Contiguous Organization
 - Linked Organization
 - Indexed Organization
 - Management of Free Storage Space
- 10.7 Distributed File Systems
 - Directory Structures and Sharing
 - Semantics of File Sharing
- 10.8 Implementing DFS

ICS 143

1

Basic functions of FS

- present logical (abstract) view of files and directories
 - hide complexity of hardware devices
- facilitate efficient use of storage devices
 - optimize access, e.g., to disk
- support sharing
 - provide protection

ICS 143

2

Hierarchical model of FS

Figure 10-1

- abstract user interface
 - present convenient view
- directory management
 - map logical name to unique Id, find descriptor
- basic file system
 - open/close files
- physical organization methods
 - map file data to disk blocks

ICS 143

3

User view of files

- file name and type
 - valid name
 - number of characters
 - lower vs upper case
 - extension
 - tied to type of file
 - used by applications
 - file type recorded in header
 - cannot be changed (even when extension changes)
 - basic types: text, object, load file; directory
 - application-specific types, e.g., .doc, .ps, .html

ICS 143

4

User view of files

- logical file organization
 - fixed or variable-size records
 - addressed
 - implicitly (sequential access to next record)
 - explicitly by position (record#) or key
- Figure 10.2
- memory mapped files
 - map file contents $0:(n-1)$ to $va:(va+n-1)$
 - read virtual memory instead of $read(i,buf,n)$

ICS 143

5

Operations on files

- create/delete
 - create/delete file descriptor
 - modify directory
- open/close
 - modify open file table
- read/write (sequential or direct)
 - modify file descriptor
 - transfer data between disk and memory
- seek/rewind
 - modify open file table

ICS 143

6

File directories

- tree-structured

Figure 10-3

- simple search, insert, delete operations
- sharing is asymmetric (only one parent)

ICS 143

7

File directories

- DAG-structured

Figure 10-5

- sharing is symmetric, but
- what are semantics of delete:
 - any parent can remove file
 - only last parent can remove it: need reference count
- must prevent cycles

Figure 10-6

- search is difficult (infinite loops)
- deletion needs garbage collection (reference count not enough)

ICS 143

8

File directories

- symbolic links

- compromise to allow sharing but avoid cycles

Figure 10-7

- for read/write access: symbolic link is the same as actual link
- for deletion: only symbolic link is deleted

ICS 143

9

File directories

- file naming: path names
 - concatenated local names with delimiter (. or / or \)
 - absolute path name: start with root (/)
 - relative path name: start with current dir (.)
 - notation to move upward (..)

ICS 143

10

Operations on file directories

- create/delete
- list
 - sorting, wild cards, recursion, info displayed
- change directory
 - path name, home directory (default)
- move
- rename
- change protection
- create/delete link (symbolic)
- find/search routines

ICS 143

11

Implementation of directories

- what information to keep in each entry
 - only symbolic name and pointer to descriptor
 - needs an extra disk access to descriptor
 - all descriptive information
 - directory can become very large
- how to organize entries within directory
 - fixed-size array of slots or a linked list
 - easy insertion/deletion
 - search is sequential
 - hash table
 - B-tree

ICS 143

12

Basic file system

- open/close files
 - retrieve and set up descriptive information for efficient access
- file descriptor (i-node in Unix)
 - owner id
 - file type
 - protection information
 - mapping to physical disk blocks
 - time of creation, last use, last modification
 - reference counter

ICS 143

13

Basic file system

- open file table (OFT)
- open command:
 - verify access rights
 - allocate OFT entry
 - allocate read/write buffers
 - fill in OFT entry
 - initialization (e.g., current position)
 - information from descriptor (e.g. file length, disk location)
 - pointers to allocated buffers
 - return OFT index

ICS 143

14

Basic file system

- close command:
 - flush modified buffers to disk
 - release buffers
 - update file descriptor
 - file length, disk location, usage information
 - free OFT entry

ICS 143

15

Basic file system

- Example: Unix

Figure 10-11

- unbuffered access

```
fd = open(name, rw, ...)  
stat = read(fd, mem, n)  
stat = write(fd, mem, n)
```

- buffered access

```
fp = fopen(name, rwa)  
c = read(fp)
```

ICS 143

16

Physical organization methods

- contiguous organization

Figure 10-12a

- simple implementation
- fast sequential access (minimal arm movement)
- insert/delete is difficult
- how much space to allocate initially
- external fragmentation

ICS 143

17

Physical organization methods

- linked organization

Figure 10-12b

- simple insert/delete, no external fragmentation
- sequential access less efficient (seek, latency)
- direct access not possible
- poor reliability (when chain breaks)

- variation 1: keep pointers segregated

Figure 10-12c

- may be cached

ICS 143

18

Physical organization methods

- variation 2: link sequences of adjacent blocks, rather than individual blocks
Figure 10-12d
- indexed organization
 - index table: sequential list of records
 - simplest implementation: keep index list in descriptor
Figure 10-12e
 - insert/delete is easy
 - sequential and direct access is efficient
 - drawback: file size limited by number of index entries

ICS 143

19

Physical organization methods

- variations of indexing
 - multi-level index hierarchy
 - primary index points to secondary indices
 - problem: number of disk accesses increases with depth of hierarchy
 - incremental indexing
 - fixed number of entries at top-level index
 - when insufficient, allocate additional index levels
 - Example: Unix -- 3-level expansion

Figure 10-13

ICS 143

20

Free storage space management

- similar to main memory management
- linked list organization
 - linking individual blocks - inefficient:
 - no block clustering to minimize seek operations
 - groups of blocks are allocated/released one at a time
 - linking groups of consecutive blocks
- bit map organization
 - analogous to main memory

ICS 143

21

Distributed file systems

- directory structures differentiated by:
 - global/local naming:
 - single global structure or different for each user
 - location transparency:
 - does the path name reveal anything about machine or server
 - location independence
 - when a file moves between machines, does its path name change

ICS 143

22

Global directory structure

- combine local directory structures under a new common root

Figure 10-14(a,b)

- problem: using “/” for new root invalidates existing local names
- solution (Unix United):
 - use “/” for local root
 - use “.” to move to new root
 - Example: reach u1 from u2:
../S1/usr/u1 or ../S1/usr/u1
 - names are *not* location transparent

ICS 143

23

Local directory structures

- mounting
 - subtree on one machine is mounted over a directory on another machine
 - contents of original directory invisible during mount
 - structure changes dynamically
 - each user has own view of FS

Figure 10-14c

ICS 143

24

Shared directory substructure

- each machine has local file system
- one subtree is shared by all machines

Figure 10-14d

ICS 143

25

Semantics of file sharing

- Unix semantics
 - all updates are immediately visible
 - generates a lot of network traffic
- session semantics
 - updates visible when file closes
 - simultaneous updates are unpredictable (lost)
- transaction semantics
 - updates visible at end of transaction
- immutable-files semantics
 - updates create a new version of file

ICS 143

26

Implementing DFS

- basic architecture
 - client/server
- Figure 10-15
- virtual file system:
 - if file is local, access local file system
 - if file is remote, communicate with remote server

ICS 143

27

Implementing DFS

- caching reduces
 - network delay
 - disk access delay
- server caching - simple
 - no disk access on subsequent access
 - no cache coherence problems
 - but network delay still exists
- client caching - more complicated
 - when to update file on server?
 - when/how to inform other processes?

ICS 143

28

Implementing DFS

- client caching
 - write-through
 - allows Unix semantics but overhead is significant
 - delayed writing
 - requires weaker semantics
 - server propagate changes to other caches
 - violates client/server relationship
 - clients need to check periodically
 - requires weaker semantics

ICS 143

29

Implementing DFS

- stateless versus stateful server
- stateful: maintain state of open files
 - Figure 10-16a
- client passes commands and data between user process and server
- problem when server crashes:
 - state of open files is lost
 - client must restore state when server recovers

ICS 143

30

Implementing DFS

- stateless server: client maintains state of open files

Figure 10-16b

- commands are idempotent (can be repeated)
- when server crashes:
 - client waits until server recovers
 - client reissues read/write commands

ICS 143

31

Implementing DFS

- file replication improves
 - availability
 - multiple replicas available
 - reliability
 - multiple replicas help in recovery
 - performance
 - multiple copies remove bottlenecks and reduce network latency
 - scalability
 - multiple copies reduce bottlenecks

ICS 143

32

Implementing DFS

- problem: file replicas must be consistent
- replication protocols
 - read any/write all
 - problem: what if a server is temporarily unavailable
 - quorum-based read/write
 - read quorum r , write quorum w
 - $r+w > N$
 - any read will see at least one current replica

Figure 10-17

ICS 143

33
