CSE 421/521 - Operating Systems
Fall 2011

LECTURE - XVIII

FILE SYSTEMS

Tevfik Koşar

University at Buffalo
November 3rd, 2011

File Systems

- Provides organized and efficient access to data on secondary storage:

1. Organizing data into files and directories and supporting primitives to manipulate them (create, delete, read, write etc)
2. Improve I/O efficiency between disk and memory (perform I/O in units of blocks rather than bytes)
3. Ensure confidentiality and integrity of data

- Contains file structure via a File Control Block (FCB)
  - Ownership, permissions, location..

A Typical File Control Block

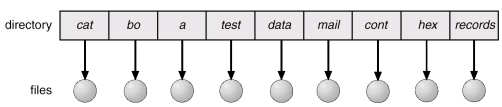
file permissions
file dates (create, access, write)
file owner, group, ACL
file size
file data blocks or pointers to file data blocks

Directories

- Directories are special files that keep track of other files
  - ✓ the collection of files is systematically organized
  - ✓ first, disks are split into partitions that create logical volumes (can be thought of as "virtual disks")
  - ✓ second, each partition contains information about the files within
  - ✓ this information is kept in entries in a **device directory** (or volume table of contents)
  - ✓ the directory is a symbol table that translates file names into their entries in the directory
    - it has a logical structure
    - it has an implementation structure (linked list, table, etc.)

Directories

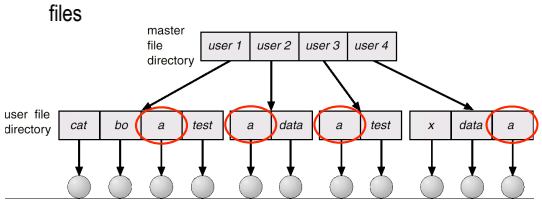
- Single-level directory structure
  - ✓ simplest form of logical organization: one global or **root** directory containing all the files
  - ✓ problems
    - global namespace: unpractical in multiuser systems
    - no systematic organization, no groups or logical categories of files that belong together



Single-level directory

Directories

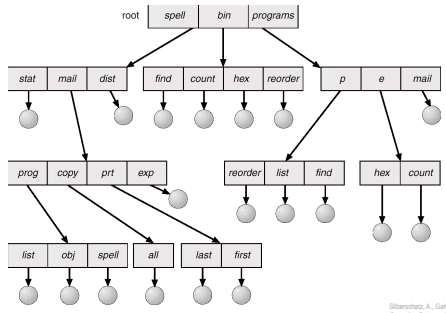
- Two-level directory structure
  - ✓ in multiuser systems, the next step is to give each user their own private directory
  - ✓ avoids filename confusion
  - ✓ however, still no grouping: not satisfactory for users with many files



Two-level directory

Directories

- Tree-structured directory structure

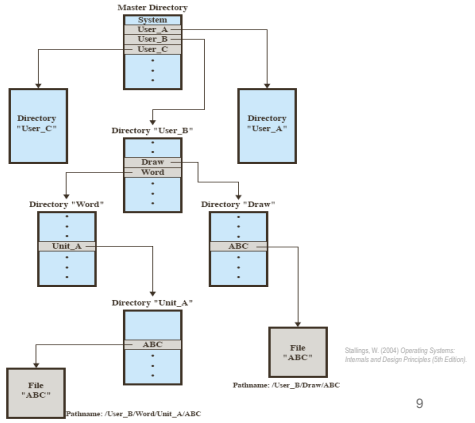


Tree-structured directory

Directories

- Tree-structured directory structure
  - ✓ natural extension of the two-level scheme
  - ✓ provides a general hierarchy, in which files can be grouped in natural ways
  - ✓ good match with human cognitive organization: tendency to categorize objects in embedded sets and subsets
  - ✓ navigation through the tree relies on **pathnames**
    - absolute pathnames start from the root, example: /jsmith/academic/teaching/cs446/assignment4/grades
    - relative pathnames start at from a current **working directory**, example: assignment4/grades
    - the current and parent directory are referred to as . and ..

Directory Implementation



Directory Implementation

- **Linear list** of file names with pointer to the data blocks.
  - simple to program
  - time-consuming to execute
- **Hash Table** - linear list with hash data structure.
  - decreases directory search time
  - **collisions** - situations where two file names hash to the same location
  - fixed size

UNIX Directories

- Directory is a special file that contains list of names of files and their inode numbers
- to see contents of a directory:

```
$ls -1ia .
9535554 .
9535489 ..
9535574 .bash_history
9535555 bin
9535584 .emacs.d
9535560 grading
9535803 hw1
9535571 test
9535801 .viminfo
```


Example inode listing

```
$ ls -iaR demodir/y
865 .       193 ..      277 a       520 c     491 y

demodir/a/y:
277 .       865 ..      402 x

demodir/c:
520 .       865 ..      651 d1      247 d2

demodir/c/d1:
651 .       520 ..      402 xlink

demodir/c/d2:
247 .       520 ..      680 xcopy
```

Directories - System View

- user view vs system view of directory tree
  - representation with "dirlists (directory files)"
- The real meaning of "A file is in a directory"
  - directory has a link to the inode of the file
- The real meaning of "A directory contains a subdirectory"
  - directory has a link to the inode of the subdirectory
- The real meaning of "A directory has a parent directory"
  - ".." entry of the directory has a link to the inode of the parent directory

User View vs System View

Consider the following directory structure (user view):

Assume **mydir** (10), **a** (20), and **b** (30) are directories and **x** (40), **y** (50), and **z** (60) are files with inode numbers given in parenthesis. The inode number for mydir's parent directory is 1.

1) Please show the system representation (system view) of this directory tree.

Link Counts

- The kernel records the number of links to any file/ directory.
- The *link count* is stored in the inode.
- The *link count* is a member of *struct stat* returned by the *stat* system call.

Change Links

- What will be the resulting changes in directory tree?

```
$ cp mydir/x mydir/b

$ ln mydir/a/z mydir/b/t

$ mv mydir/x mydir/a
```


Implementing "pwd"

1. "." is 247
   chdir ..
2. 247 is called "d2"
   "." is 520
   chdir ..
3. 520 is called "c"
   "." is 865
   chdir ..
4. 865 is called "demodir"
   "." is 193
   chdir ..

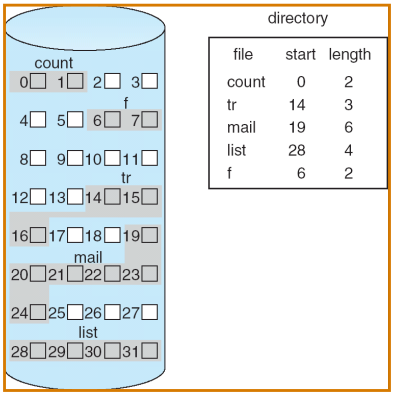
Allocation Methods

- An allocation method refers to how disk blocks are allocated for files:
- **Contiguous allocation**
- **Linked allocation**
- **Indexed allocation**

Contiguous Allocation

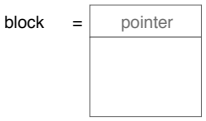
- Each file occupies a set of contiguous blocks on the disk
- + Simple – only starting location (block #) and length (number of blocks) are required
- - Wasteful of space (dynamic storage-allocation problem - fragmentation)
- - Files cannot grow

Contiguous Allocation of Disk Space



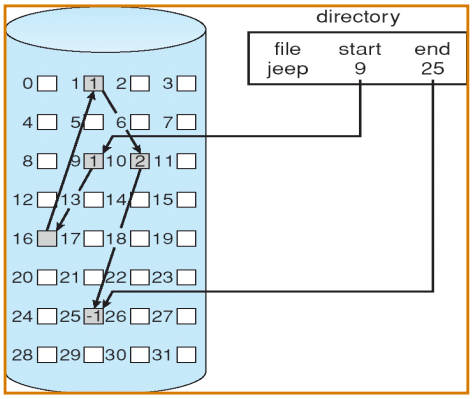
Linked Allocation

- Each file is a linked list of disk blocks: blocks may be scattered anywhere on the disk.

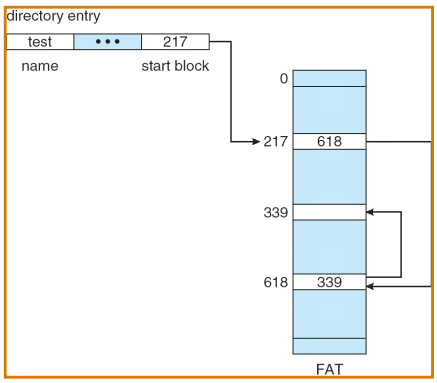


+ Simple – need only starting address
+ Free-space management system – no waste of space
+ Defragmentation not necessary
- No random access
- Extra space required for pointers
- Reliability: what if a pointer gets corrupted?

Linked Allocation

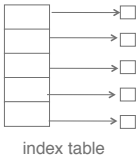


File-Allocation Table



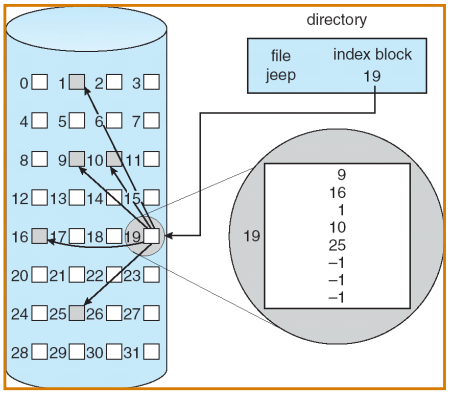
Indexed Allocation

- Brings all pointers together into the *index block*, to allow random access to file blocks.
- Logical view.



+ Supports direct access
+ Prevents external fragmentation
- High pointer overhead --> wasted space

Example of Indexed Allocation



Free Space Management

- Disk space limited
- Need to re-use the space from deleted files
- To keep track of free disk space, the system maintains a free-space list
  - Records all free disk blocks
- Implemented using
  - Bit vectors
  - Linked lists

Free-Space Management (Cont.)

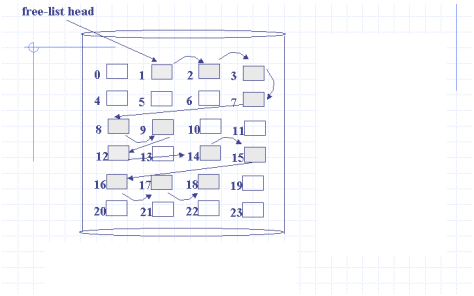
- Bit vector (*n* blocks)
  - Each block is represented by 1 bit
  - 1: free, 0: allocated

$$bit[i] = \begin{cases} 1 \Rightarrow block[i] \text{ free} \\ 0 \Rightarrow block[i] \text{ occupied} \end{cases}$$

- e.g. 000011111000100010010000

Free-Space Management (Cont.)

- Linked List Approach

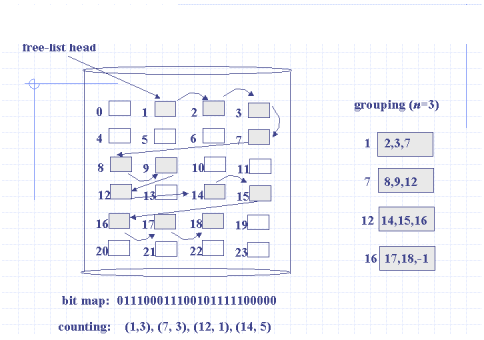


Free-Space Management (Cont.)

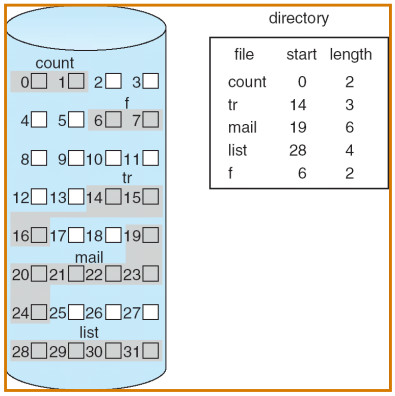
- Bit map requires extra space
  - Example:
    block size = 2^{12} bytes
    disk size = 2^{30} bytes (1 gigabyte)
    $n = 2^{30}/2^{12} = 2^{18}$ bits (or 32K bytes)
- Easy to get contiguous files
- Linked list (free list)
  - Cannot get contiguous space easily
  - requires substantial I/O
- Grouping
  - Modification of free-list
  - Store addresses of n free blocks in the first free block
- Counting
  - Rather than keeping list of n free addresses:
    - Keep the address of the first free block
    - And the number n of free contiguous blocks that follow it

Free-Space Management (Cont.)

- Linked List



Exercise



Acknowledgements

- "Operating Systems Concepts" book and supplementary material by A. Silberschatz, P. Galvin and G. Gagne
- "Operating Systems: Internals and Design Principles" book and supplementary material by W. Stallings
- "Modern Operating Systems" book and supplementary material by A. Tanenbaum
- R. Doursat and M. Yuksel from UNR