Principles of Operating Systems

CS 446/646

6. File System

René Doursat

Department of Computer Science & Engineering
University of Nevada, Reno

Spring 2006

Principles of Operating Systems

CS 446/646

0. Course Presentation
1. Introduction to Operating Systems
2. Processes
3. Memory Management
4. CPU Scheduling
5. Input/Output
6. **File System**
7. **Case Studies**

Principles of Operating Systems

CS 446/646

6. File System

a. **Overview of the File System**

b. **User Interface: Files**

c. **User Interface: Directories**

d. **File System Implementation**

Principles of Operating Systems

CS 446/646

6. File System

a. **Overview of the File System**

b. **User Interface: Files**

c. **User Interface: Directories**

d. **File System Implementation**

6.a Overview of the File System

- The need for long-term storage
  - ✓ it must be possible to store a very **large amount** of information
    - memory is too small to hold large databases of records, for example airline reservations, bank accounts, etc.
  - ✓ the information must **survive** the termination of the processes using it
    - it must also not go away if the computer crashes
  - ✓ multiple processes must be able to **access** the information **concurrently**
    - for example, a phone directory should not be only stored inside the address space of a single process
  - → *store information on disk, and group it in units called* ***files***

6.a Overview of the File System

➢ Chart of Operating System Responsibilities

§E – The O/S is responsible for providing a uniform logical view of information storage

- ✓ the O/S defines a logical unit of storage, the **file**, and groups files in a hierarchy of **directories**
- ✓ the O/S supports primitives for manipulating files and directories (create, delete, rename, read, write, etc.)
- ✓ the O/S ensures data confidentiality and integrity
- ✓ the O/S implements files on stable (nonvolatile) storage media
- ✓ the O/S keeps a mapping of the logical files onto the physical secondary storage

6.a Overview of the File System

- The file system is the most visible aspect of an O/S
  - ✓ files are managed by the O/S
  - ✓ how files are
    - structured
    - named
    - accessed
    - used
    - protected
    - implemented

  . . . are major topics in operating system design

6.a Overview of the File System

- Users' standpoint vs. designers' standpoint
  - ✓ for the O/S users
    - the most important aspect is how files **appear** to them
    - how files are named and protected
    - what operations are allowed, etc.
  - ✓ for the O/S designers
    - must decide whether to **implement** files with linked lists, tables, etc.
    - how to map file blocks to disk sectors
    - how to keep track of free storage, etc.

Principles of Operating Systems

CS 446/646

6. File System

a. Overview of the File System

b. User Interface: Files

c. User Interface: Directories

d. File System Implementation

6.b User Interface: Files

- Files are an abstraction mechanism
  - ✓ the concept of “file” is the central element of the file system
  - ✓ a file is a complete collection of data (as text or a program) treated by a computer as a unit especially for purposes of input and output
  - ✓ files provide a convenient way to store information on the disk and read it back later
  - ✓ they shield the user from the details of where the information is stored and how the disk works

6.b User Interface: Files

File naming

- Naming is the most important aspect of abstraction
  - ✓ when a process creates a file, it gives it a name; when it terminates, the file continues to exist
  - ✓ naming rules vary from system to system
    - allowed name length can go from 8 to 255 characters
    - UNIX systems distinguish between uppercase and lowercase, MS-DOS and Windows do not
    - many systems support two-part, period-separated naming: the second part is called the **extension**
    - in UNIX, the extension is a user convention; not enforced
    - Windows is extension-aware and associates files with specific applications

6.b User Interface: Files

File naming

file type	usual extension	function
executable	exe, com, bin or none	read to run machine-language program
object	obj, o	compiled, machine language, not linked
source code	c, cc, java, pas, asm, a	source code in various languages
batch	bat, sh	commands to the command interpreter
text	txt, doc	textual data, documents
word processor	wp, tex, rrf, doc	various word-processor formats
library	lib, a, so, dll, mpeg, mov, rm	libraries of routines for programmers
print or view	arc, zip, tar	ASCII or binary file in a format for printing or viewing
archive	arc, zip, tar	related files grouped into one file, sometimes compressed, for archiving or storage
multimedia	mpeg, mov, rm	binary file containing audio or A/V information

Silberschatz, A., Galvin, P. B. and Gagne. G. (2003) *Operating Systems Concepts with Java (6th Edition).*

Common file types & extensions

6.b User Interface: Files

File structure

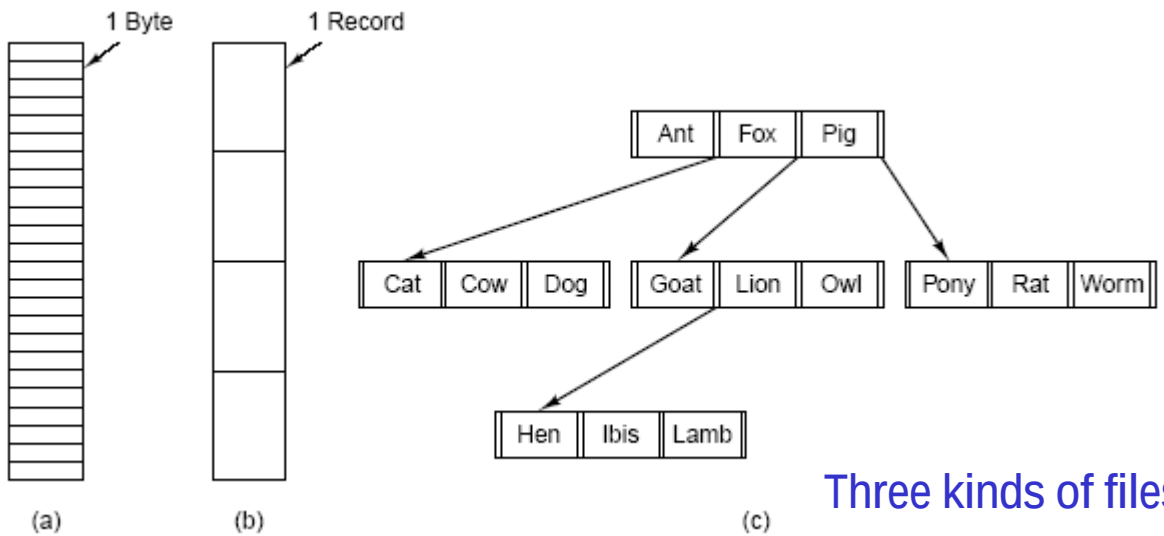
- A file can be internally structured in several ways

a) **pure byte sequence** — O/S doesn't care about the contents; all meaning imposed by user application; generic O/S (UNIX, Win)

closer to database system techniques

b) **record sequence** — fixed or variable-length records with internal structure; historical 80-column punch card systems

c) **tree** — key-accessible records; mainframes commercial data processing



Tanenbaum, A. S. (2001)
Modern Operating Systems (2nd Edition).

Three kinds of files

6.b User Interface: Files

File types

- An O/S supports different types of files

Windows

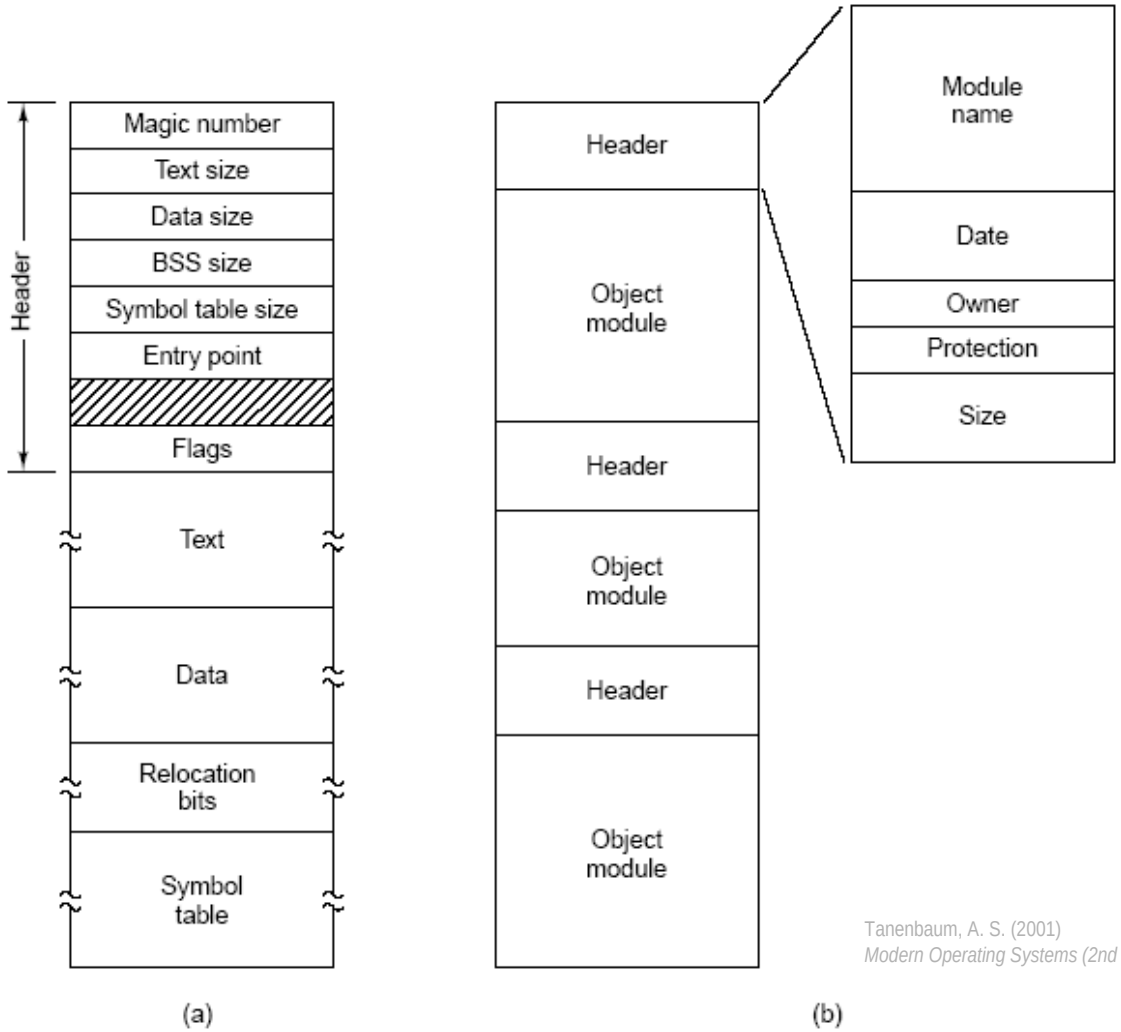
- ✓ regular files
  - the files that contain user information, ASCII or binary
- ✓ directories (directory files)
  - system files that contain information about the file system organization

UNIX

- ✓ character special files
  - used to model serial (character-mode) I/O devices: terminals, network
- ✓ block special files
  - used to model parallel (block-mode) I/O devices: disks

6.b User Interface: Files

File types



Tanenbaum, A. S. (2001)
Modern Operating Systems (2nd Edition).

(a) An executable file and (b) an archive of unlinked compiled modules

6.b User Interface: Files

File attributes

- The O/S associates management information with files
  - ✓ in addition to its name and data, a file also has **file attributes**
  - ✓ the list of attributes varies considerably from system to system, but typically:
    - file's owner and protection
    - various bit flags: hidden, read/write, etc.
    - record length, key, etc. for record-structured files
    - timestamps: created, accessed, modified, etc.
    - size values
  - ✓ just as process control blocks (PCBs), the O/S maintains file control blocks (FCBs) → *see file system implementation*

6.b User Interface: Files

File attributes

Attribute	Meaning
Protection	Who can access the file and in what way
Password	Password needed to access the file
Creator	ID of the person who created the file
Owner	Current owner
Read-only flag	0 for read/write; 1 for read only
Hidden flag	0 for normal; 1 for do not display in listings
System flag	0 for normal files; 1 for system file
Archive flag	0 for has been backed up; 1 for needs to be backed up
ASCII/binary flag	0 for ASCII file; 1 for binary file
Random access flag	0 for sequential access only; 1 for random access
Temporary flag	0 for normal; 1 for delete file on process exit
Lock flags	0 for unlocked; nonzero for locked
Record length	Number of bytes in a record
Key position	Offset of the key within each record
Key length	Number of bytes in the key field
Creation time	Date and time the file was created
Time of last access	Date and time the file was last accessed
Time of last change	Date and time the file has last changed
Current size	Number of bytes in the file
Maximum size	Number of bytes the file may grow to

Some possible file attributes

Tanenbaum, A. S. (2001)
Modern Operating Systems (2nd Edition).

6.b User Interface: Files

File operations

- Most common system calls related to files
  - ✓ create/delete
    - creates a file with no data, initializes file attributes
  - ✓ open/close
    - loads file attributes and disk addresses in memory
  - ✓ read/write, append
    - transfers data from/to a buffer starting at a current position
  - ✓ seek
    - in random access files: repositions file pointer for read/write
  - ✓ get/set attributes, rename
    - some attributes are user-settable (name, protection flags)

6.b User Interface: Files

File operations

```
#define BUF_SIZE 4096                   /* use a buffer size of 4096 bytes */
#define OUTPUT_MODE 0700                /* protection bits for output file */

int main(int argc, char *argv[])
{
    int in_fd, out_fd, rd_count, wt_count;
    char buffer[BUF_SIZE];

    if (argc != 3) exit(1);             /* syntax error if argc is not 3 */

    /* Open the input file and create the output file */
    in_fd = open(argv[1], O_RDONLY);    /* open the source file */
    if (in_fd < 0) exit(2);             /* if it cannot be opened, exit */
    out_fd = creat(argv[2], OUTPUT_MODE);   /* create the destination file */
    if (out_fd < 0) exit(3);            /* if it cannot be created, exit */

    /* Copy loop */
    while (TRUE) {
        rd_count = read(in_fd, buffer, BUF_SIZE);  /* read a block of data */
    if (rd_count <= 0) break;           /* if end of file or error, exit loop */
        wt_count = write(out_fd, buffer, rd_count);  /* write data */
        if (wt_count <= 0) exit(4);     /* wt_count <= 0 is an error */
    }

    /* Close the files */
    close(in_fd);
    close(out_fd);
    if (rd_count == 0)                  /* no error on last read */
        exit(0);
    else
        exit(5);                        /* error on last read */
}
```

Tanenbaum, A. S. (2001)
Modern Operating Systems (2nd Edition).

Principles of Operating Systems

CS 446/646

6. File System

a. Overview of the File System

b. User Interface: Files

c. User Interface: Directories

d. File System Implementation

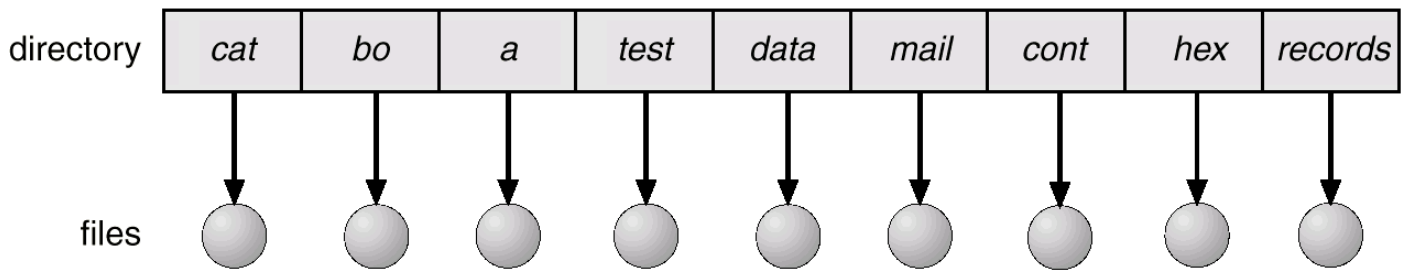
6.c User Interface: Directories

- 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.)

6.c User Interface: 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



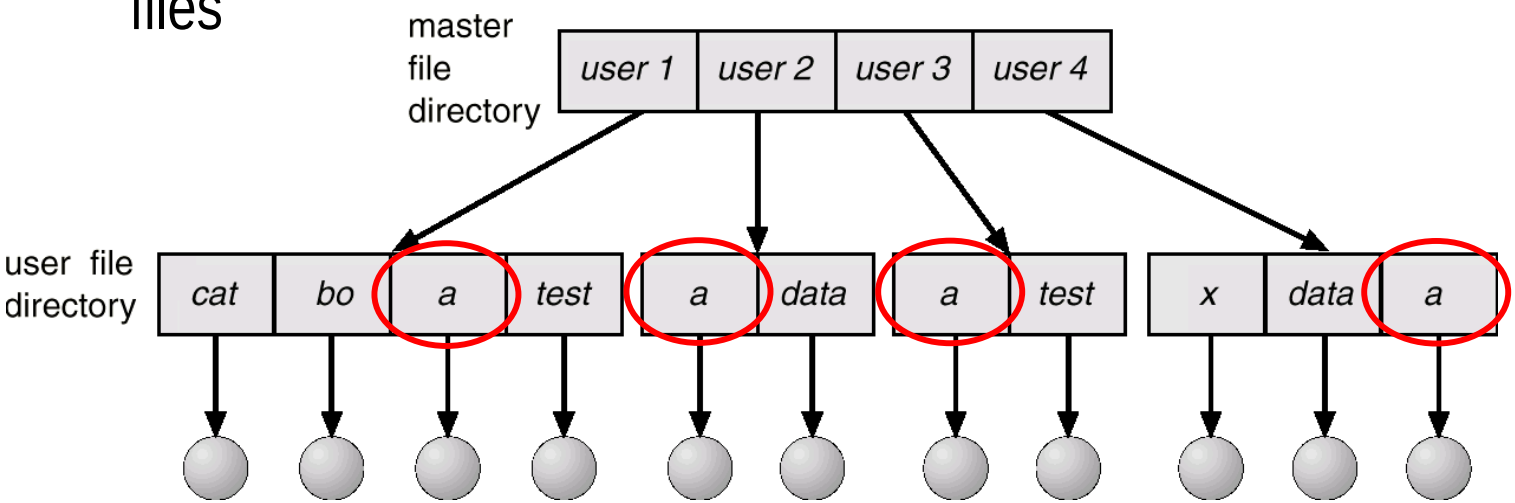
Single-level directory

Silberschatz, A., Galvin, P. B. and Gagne. G. (2003) *Operating Systems Concepts with Java (6th Edition).*

6.c User Interface: 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

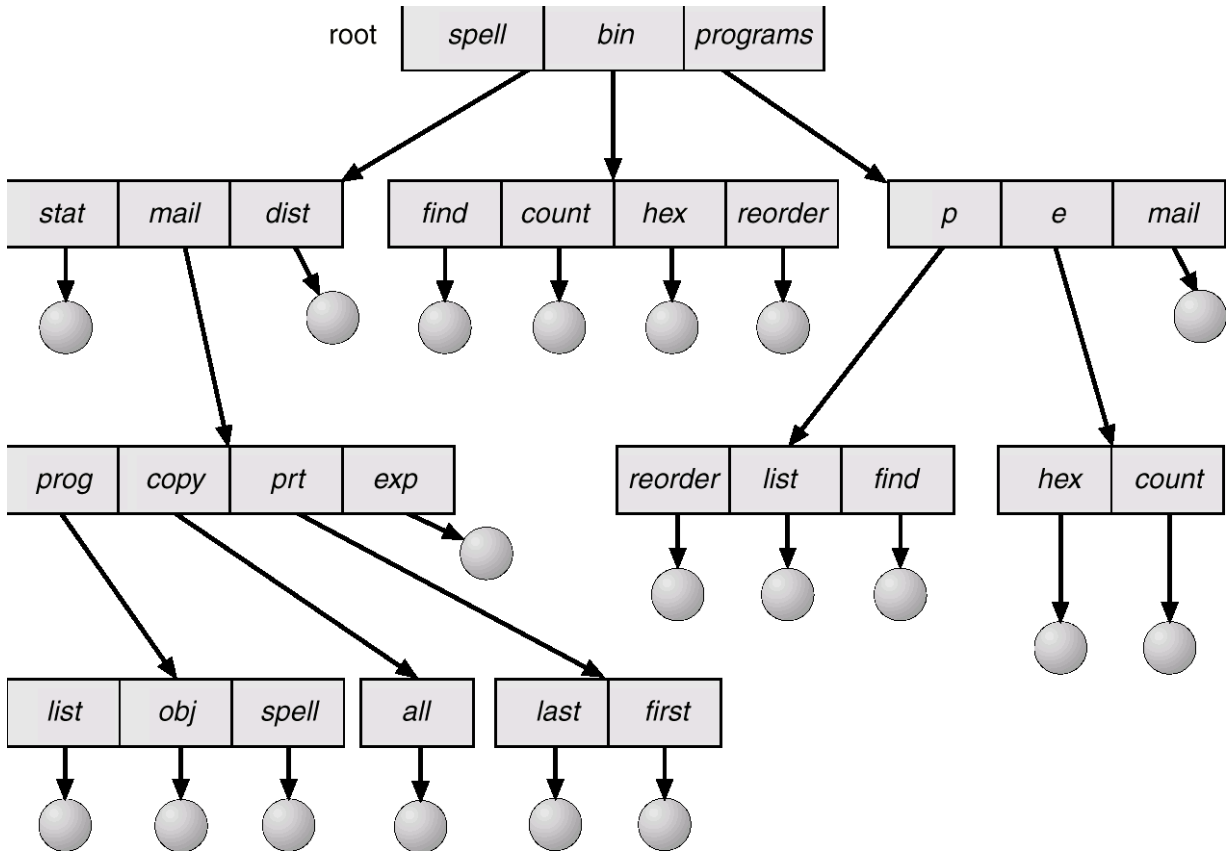


Two-level directory

Silberschatz, A., Galvin, P. B. and Gagne. G. (2003) *Operating Systems Concepts with Java (6th Edition).*

6.c User Interface: Directories

- Tree-structured directory structure



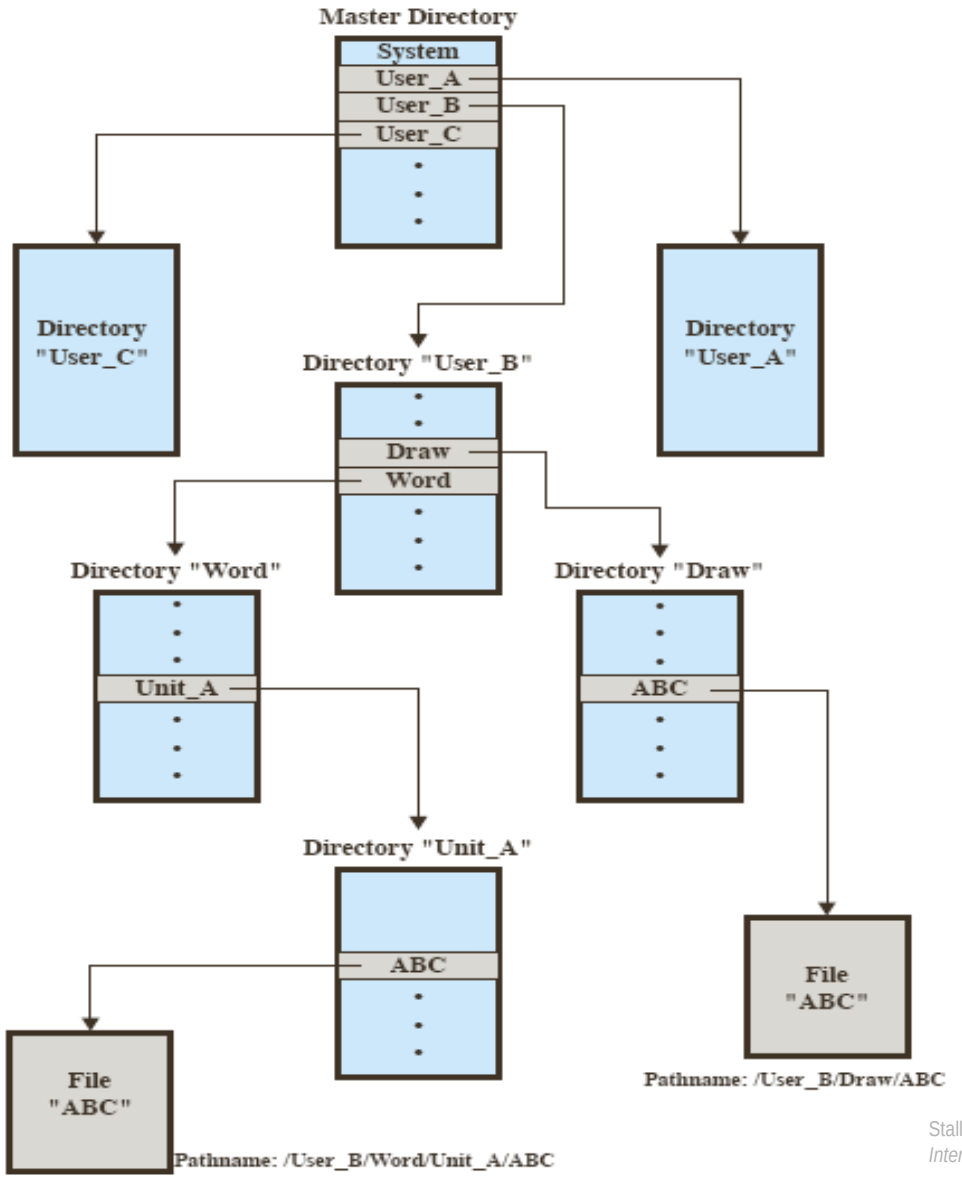
Tree-structured directory

Silberschatz, A., Galvin, P. B. and Gagne. G. (2003) *Operating Systems Concepts with Java (6th Edition).*

6.c User Interface: 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: propensity to categorize objects in embedded sets and subsets
  - ✓ navigation through the tree relies on **pathnames**
    - absolute pathnames start from the root, example: /doursat/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 ..

6.c User Interface: Directories



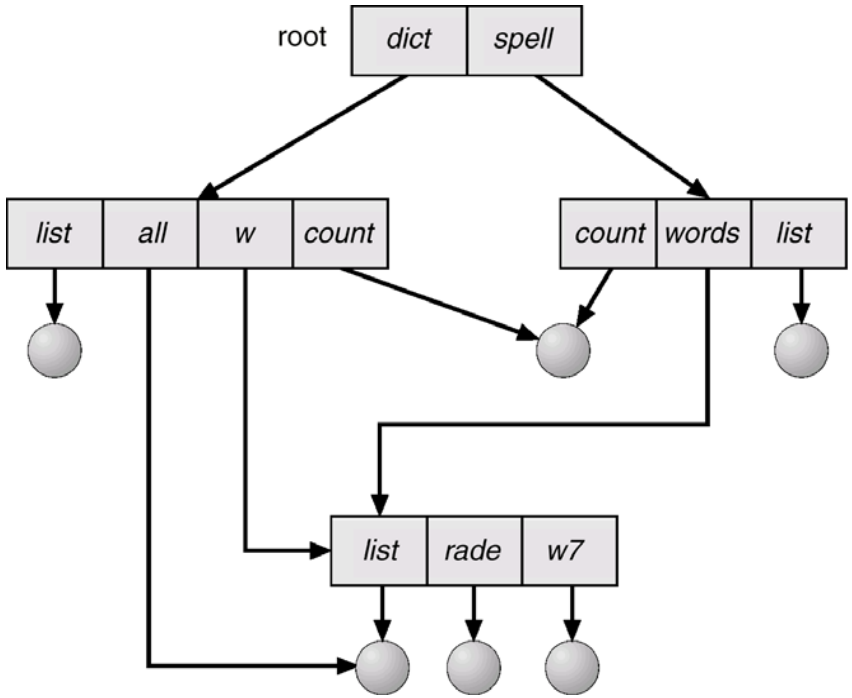
Stallings, W. (2004) *Operating Systems: Internals and Design Principles (5th Edition).*

6.c User Interface: Directories

- Common system calls related to directory operations
  - ✓ create/delete
    - creates or deletes an *empty* directory (except for . and ..)
  - ✓ opendir/closedir
    - loads directory attributes in memory
  - ✓ readdir
    - reads the entries in a directory (more abstract than **read**)
  - ✓ rename
    - renames a directory like a file
  - ✓ link/unlink
    - shares files by making them appear in more than one dir

6.c User Interface: Directories

- Acyclic-graph (shared file) directory structure
  - ✓ allows for different users to work on the same files while keeping their own view of the files (implemented with links)



Acyclic-graph directory

Silberschatz, A., Galvin, P. B. and Gagne. G. (2003) *Operating Systems Concepts with Java (6th Edition).*

Principles of Operating Systems

CS 446/646

6. File System

a. Overview of the File System

b. User Interface: Files

c. User Interface: Directories

d. File System Implementation

6.d File System Implementation

- The file system implementation relies on several "on-disk" and "in-core" structures
  - ✓ the **on-disk** structures contain persistent (static) information:
    - how to boot an O/S stored in the partition
    - number of blocks and free blocks
    - directory structure
    - individual files
  - ✓ the **in-core** (memory) structures are used for process-related file management and performance improvement via caching
    - tables of open files (system-wide and per-process)
    - recently opened directories

6.d File System Implementation

- The central elements are the File Control Blocks
  - ✓ In UNIX, a File Control Block is called an **i-node** ("index-node")
  - ✓ each file has a corresponding i-node structure, which contains information describing the file
  - ✓ on-disk i-node (file system dependent)
    - persistent accounting information: user & group ownership, time stamps, etc.
    - information to locate the disk blocks holding the file's data
  - ✓ in-core i-node (file system independent)
    - transient management information: access flags (locked, modified), processes holding it, read/write pointer, etc.

6.d File System Implementation

On-disk i-nodes

- ✓ each i-node has an absolute i-number
- ✓ each i-node has a fixed size, generally 64 bytes long

Field	Bytes	Description
Mode	2	File type, protection bits, setuid, setgid bits
Nlinks	2	Number of directory entries pointing to this i-node
Uid	2	UID of the file owner
Gid	2	GID of the file owner
Size	4	File size in bytes
Addr	39	Address of first 10 disk blocks, then 3 indirect blocks
Gen	1	Generation number (incremented every time i-node is reused)
Atime	4	Time the file was last accessed
Mtime	4	Time the file was last modified
Ctime	4	Time the i-node was last changed (except the other times)
	= 64	

Tanenbaum, A. S. (2001) *Modern Operating Systems (2nd Edition).*

- ✓ therefore, 8 to 16 i-nodes fit on a 512 to 1,024 byte disk block

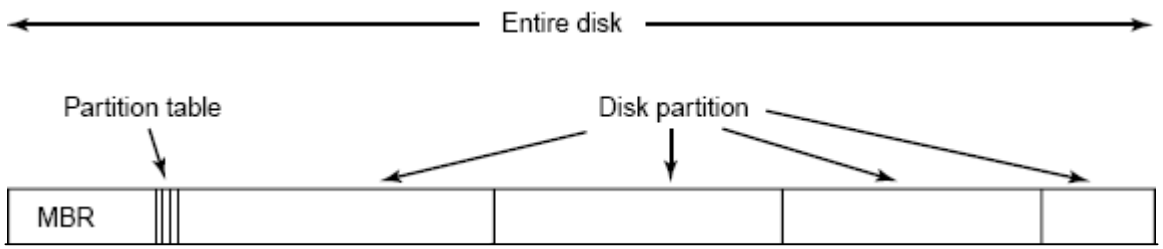
6.d File System Implementation

- In-core i-nodes
  - ✓ for every file *in use*, the on-disk i-node is loaded into memory ("core")
  - ✓ the in-core i-node contains all the information from the on-disk i-node, plus more:
    - file access status flags: locked? other process waiting? file status modified? file contents modified?
    - reference count of processes accessing the file
    - pointer to disk block where persistent i-node resides
    - current read/write file position (pointer to disk block)

6.d File System Implementation

On-disk layout

- Layout of disk partitions
  - ✓ the disk can be divided up into several partitions that each hold an independent file system
  - ✓ block (sector) 0 of the disk contains the Master Boot Record (MBR), which is read in by the BIOS to boot the computer
  - ✓ then, the MBR locates the active partition in a table, loads and executes its “boot block” in block 0



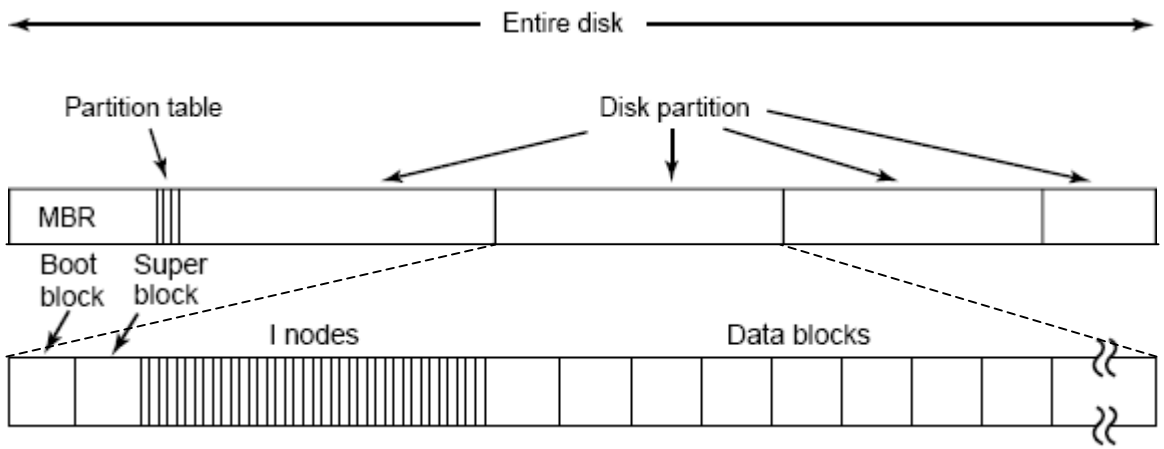
Tanenbaum, A. S. (2001)
Modern Operating Systems (2nd Edition).

6.d File System Implementation

On-disk layout

- Layout of file system inside a partition
  - ✓ within one file system, the on-disk structures include
    - block 0, the “boot block” — information to boot the O/S
    - block 1, the “superblock” — partition details
    - all the i-nodes
    - all the file and directory data, split in blocks

O/S dependent File System

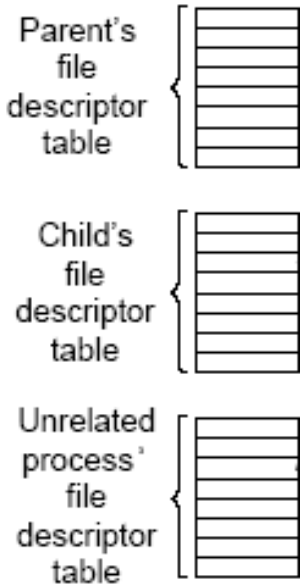


Tanenbaum, A. S. (2001) *Modern Operating Systems (2nd Edition).*

6.d File System Implementation

In-core structures

- Per-process file descriptor tables
  - ✓ within a process, each accessed file for read/write has a **file descriptor** number
    - `fd = open(name, mode);`
    - `n = read(fd, buf, size);`
  - ✓ therefore, the O/S maintains a file descriptor table for each process
  - ✓ this table associates file descriptors with pointers to i-node(-related) file structures



Tanenbaum, A. S. (2001)
Modern Operating Systems (2nd Edition).

6.d File System Implementation

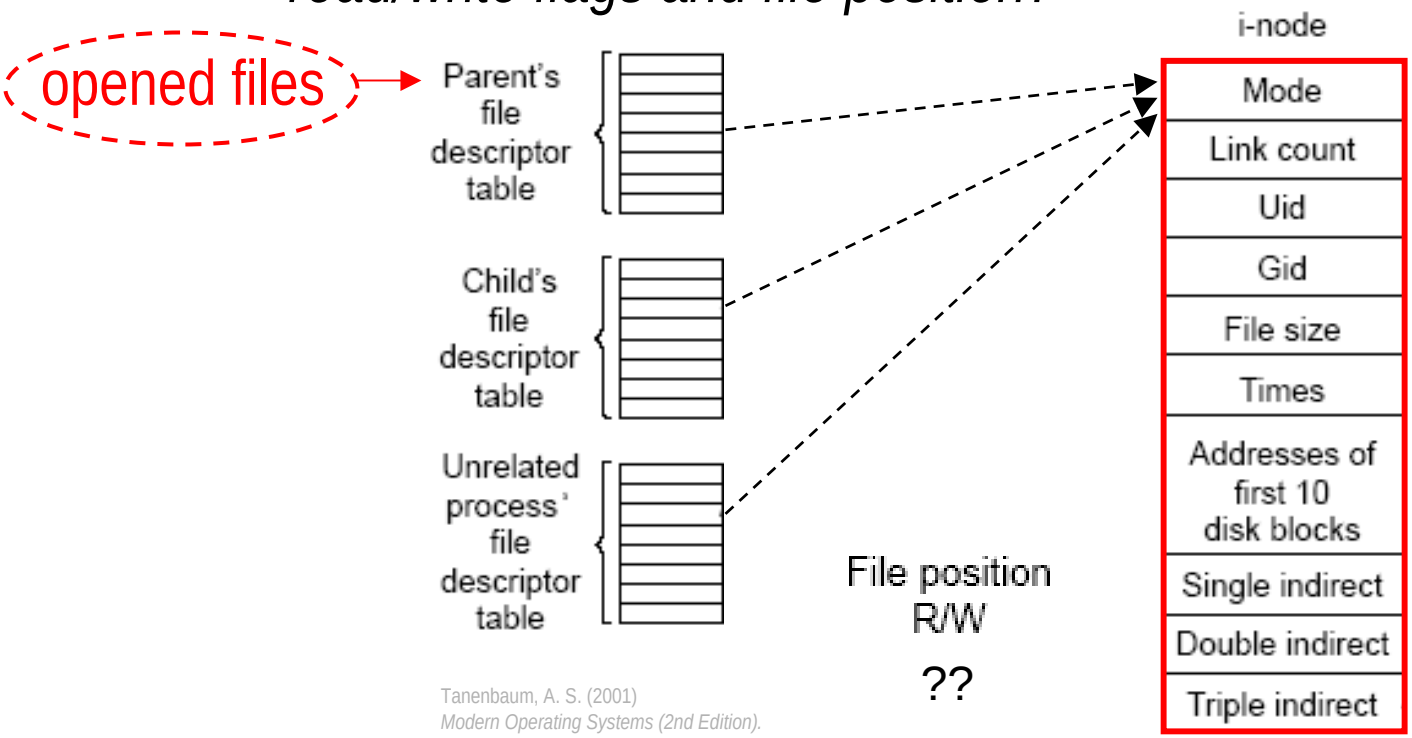
In-core structures

- In the process table, the O/S keeps one ID structure per process, the *Process Control Block* (PCB), containing:
  - ✓ process identification data
    - numeric identifiers of the process, the parent process, the user, etc.
  - ✓ CPU state information
    - user-visible, control & status registers
    - stack pointers
  - ✓ process control information
    - scheduling: state, priority, awaited event
    - used memory and I/O, opened files, etc.
    - pointer to next PCB

6.d File System Implementation

In-core structures

- Association between file descriptor and i-node
  - ✓ one possibility: direct link
  - → *problem: where is the additional in-core information about read/write flags and file position?*

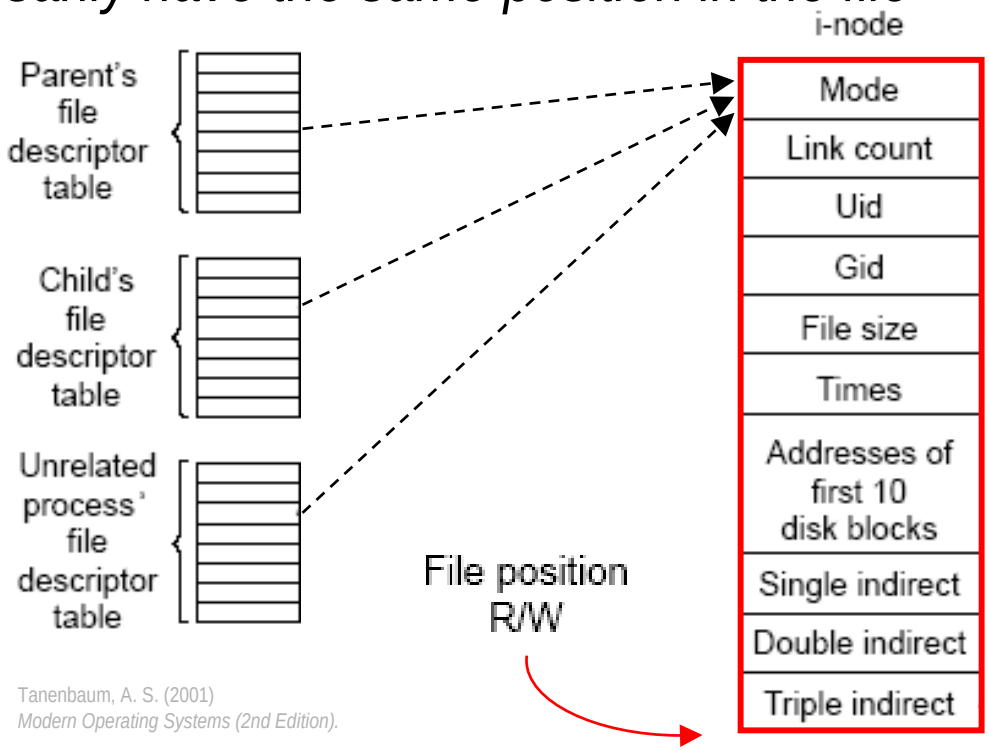


Tanenbaum, A. S. (2001)
Modern Operating Systems (2nd Edition).

6.d File System Implementation

In-core structures

- Association between file descriptor and i-node (2)
  - ✓ first attempt: put the file position in the i-node
  - → *problem: different processes accessing the same file don't necessarily have the same position in the file*

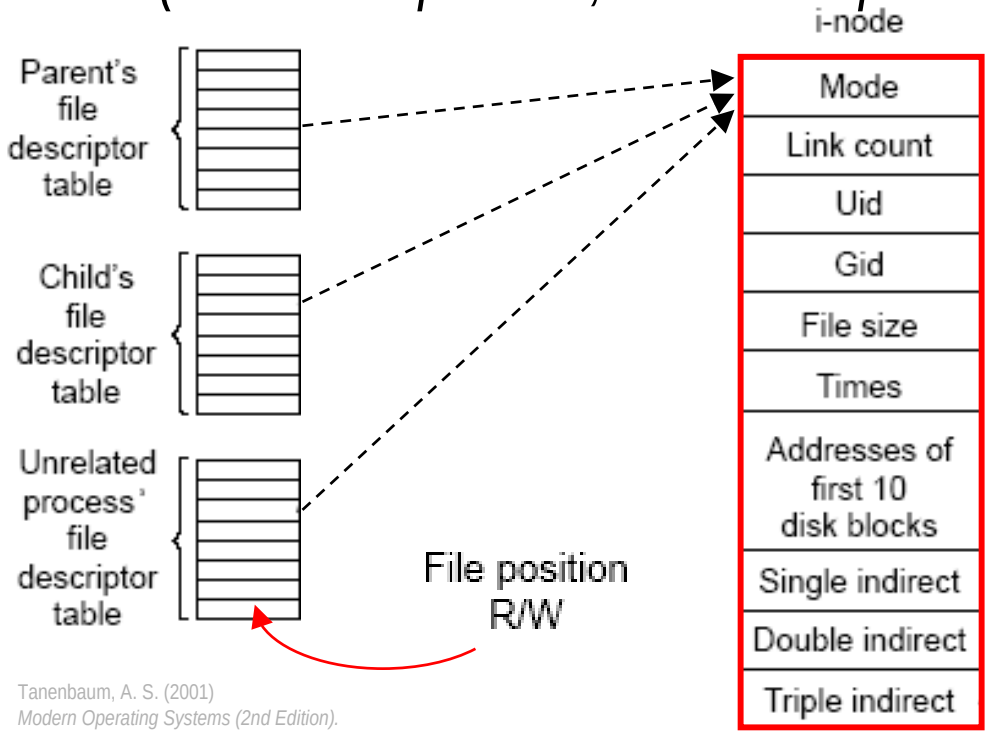


Tanenbaum, A. S. (2001)
Modern Operating Systems (2nd Edition).

6.d File System Implementation

In-core structures

- Association between file descriptor and i-node (3)
  - ✓ 2nd attempt: put it in each process descriptor table
  - → *problem: a newly forked child process must start at the parent's last position (for ex: script > file, where script = cmd1, cmd2)*

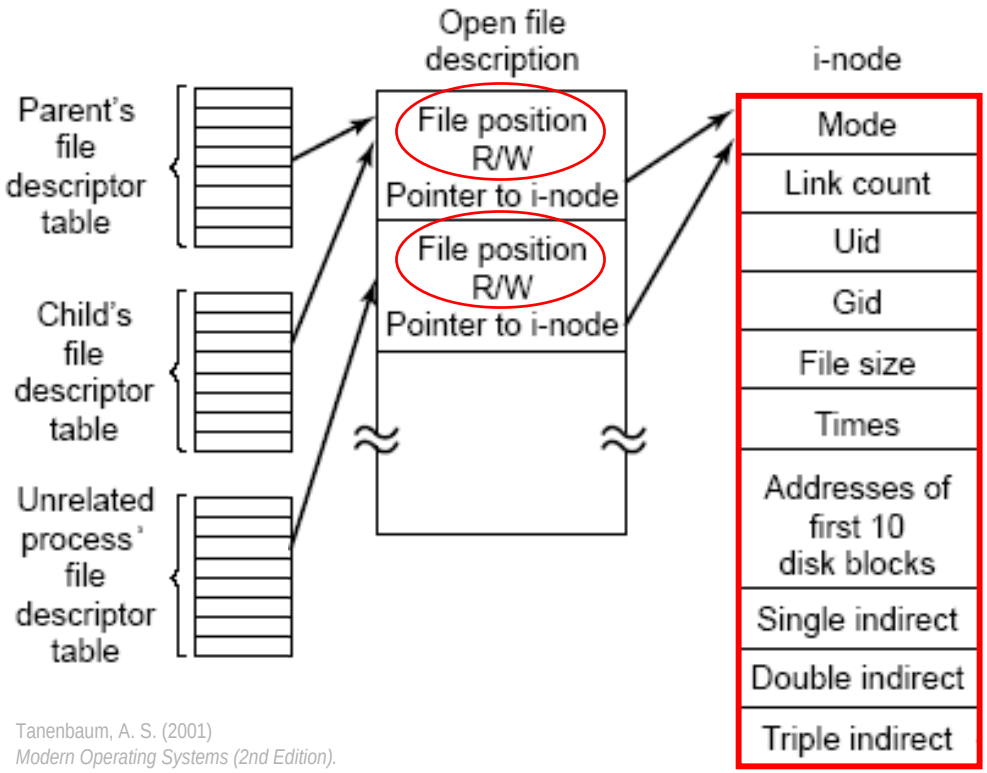


Tanenbaum, A. S. (2001)
Modern Operating Systems (2nd Edition).

6.d File System Implementation

In-core structures

- Association between file descriptor and i-node (4)
  - ✓ solution: introduce one level of indirection with a new table
  - ✓ this way, children inherit file positions but other processes don't

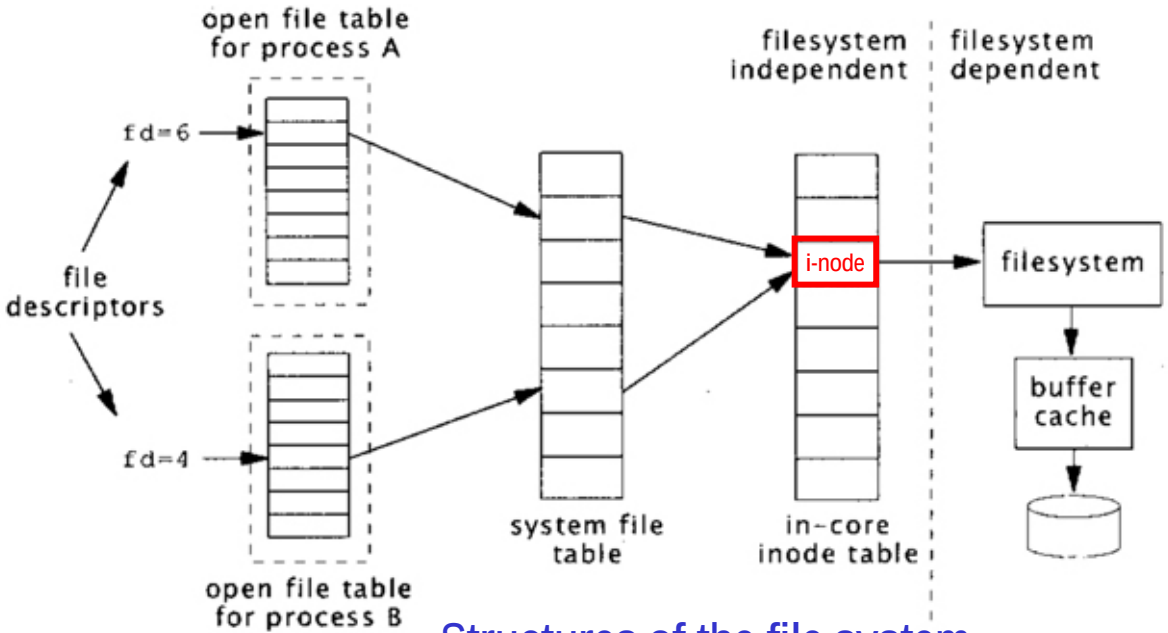


Tanenbaum, A. S. (2001)
Modern Operating Systems (2nd Edition).

6.d File System Implementation

In-core structures

- Summary of main in-core file system structures
  - ✓ per-process open file tables — list of unique FDs
  - ✓ system file table — multiple entries can reference same file
  - ✓ in-core i-node table — one entry per file



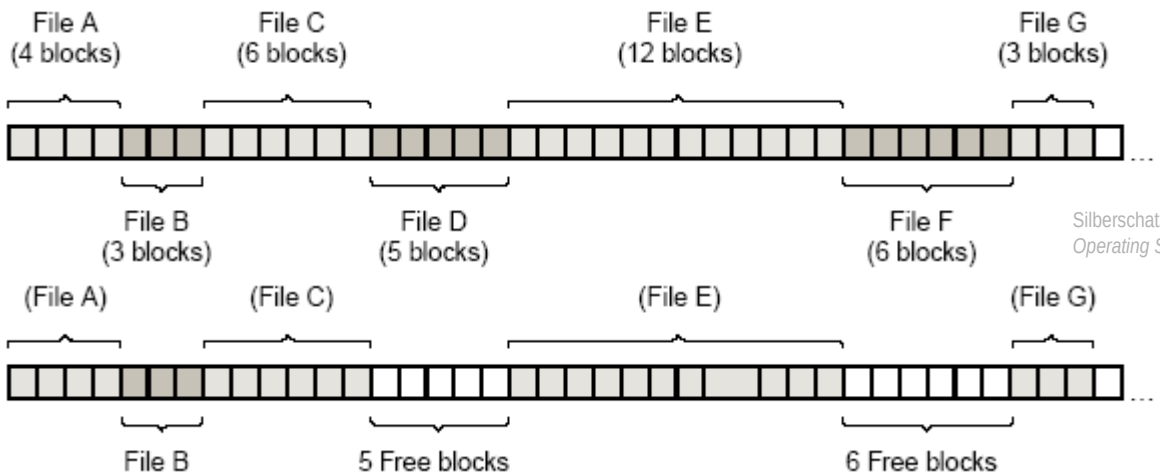
Pate, S. D. (1996)
UNIX Internals: A Practical Approach

Structures of the file system

6.d File System Implementation

File block allocation

- Contiguous allocation
  - ✓ each file is stored as a contiguous sequence of disk blocks
  - ✓ analogous to dynamic memory partitioning, except on disk
    - same advantages: simplicity + access speed (high locality)
    - but also same flaws: fragmentation + need to declare size



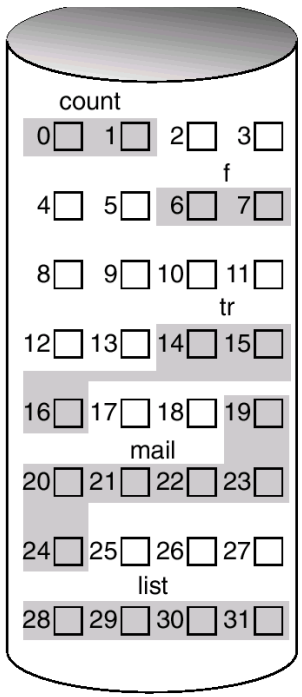
Silberschatz, A., Galvin, P. B. and Gagne. G. (2003) *Operating Systems Concepts with Java (6th Edition).*

→ *however, widely used in CD-ROMs! no fragmentation in R-only*

6.d File System Implementation

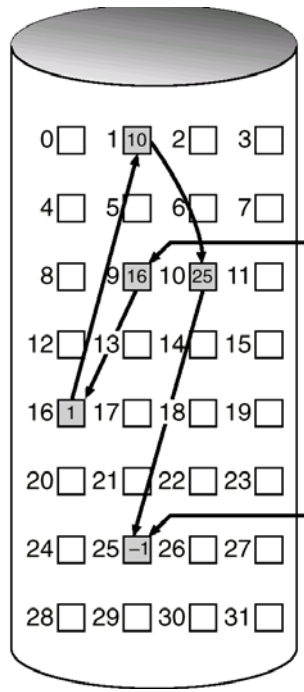
File block allocation

- Linked allocation
  - ✓ each file is scattered in blocks: same idea as memory paging!
  - ✓ one way to keep track of the blocks is to link them to each other



directory

file	start	length
count	0	2
tr	14	3
mail	19	6
list	28	4
f	6	2



directory

file	start	end
jeep	9	25

Silberschatz, A., Galvin, P. B. and Gagne. G. (2003) *Operating Systems Concepts with Java (6th Edition).*

Contiguous vs. linked allocation of disk space

6.d File System Implementation

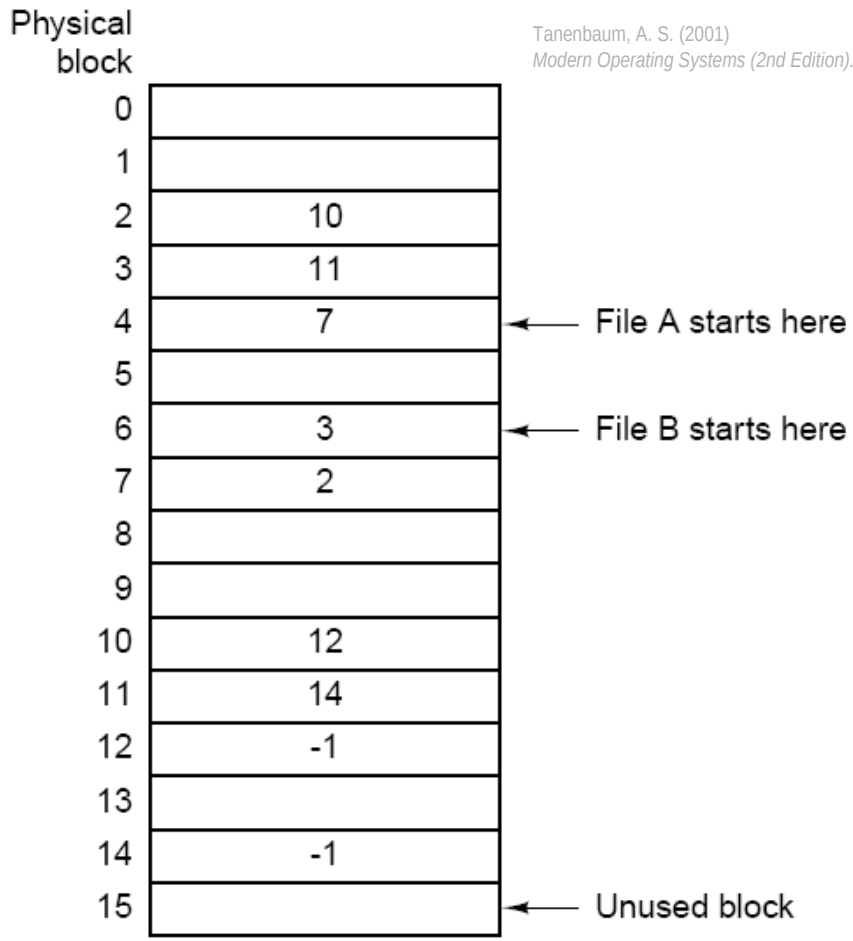
File block allocation

- Linked allocation
  - ✓ advantages: no fragmentation, file can change size by appending or removing blocks
  - ✓ main problem: access time! effective for sequential-access files, but not random-access
  - ✓ to find the i-th block, one must start at the beginning and follow all the pointers
  - ✓ other problem: slight waste of disk space, as a pointer of 4 bytes occupies ~1% of a block of 512 bytes

6.d File System Implementation

File block allocation

- File allocation table (FAT)
  - ✓ instead of scattering block pointers, gather them in one global table: the file allocation table
  - ✓ each block entry points to the next block in the chain
  - ✓ end blocks get -1, free blocks get 0
  - ✓ used in MS-DOS and OS/2
  - → *problem: size and caching of table in memory*



Tanenbaum, A. S. (2001) *Modern Operating Systems (2nd Edition).*

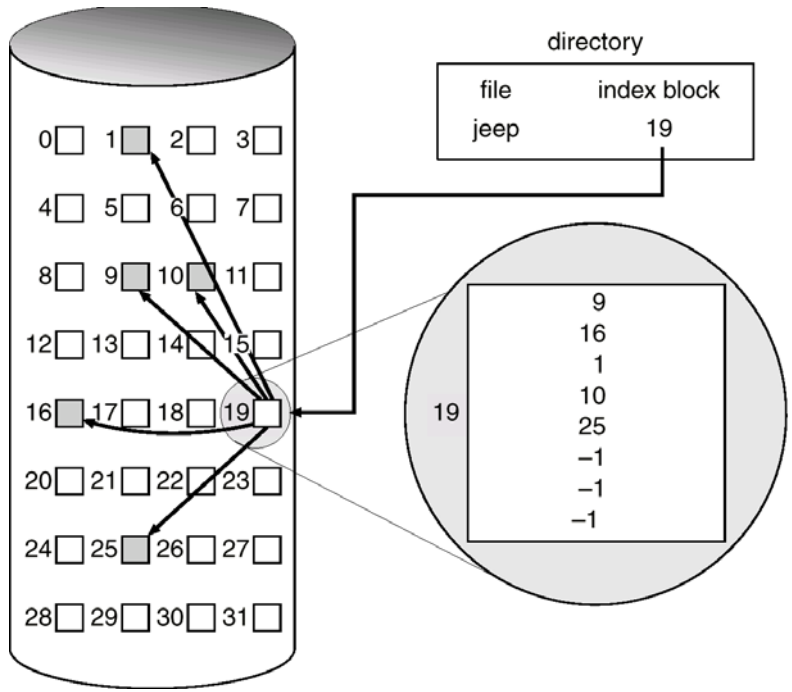
Linked list allocation using a FAT in memory

6.d File System Implementation

File block allocation

- Indexed allocation
  - ✓ a global table is too big: so we are back to distributing block pointers into blocks
  - ✓ but this time, we keep them together in one location per file: the **index block**
  - ✓ same idea as paging tables: local table of scattered pieces
  - ✓ ex: 512b block holds 128 #'s
  - → *problem: what if a file is bigger then 128 blocks?*

Silberschatz, A., Galvin, P. B. and Gagne. G. (2003) *Operating Systems Concepts with Java (6th Edition).*

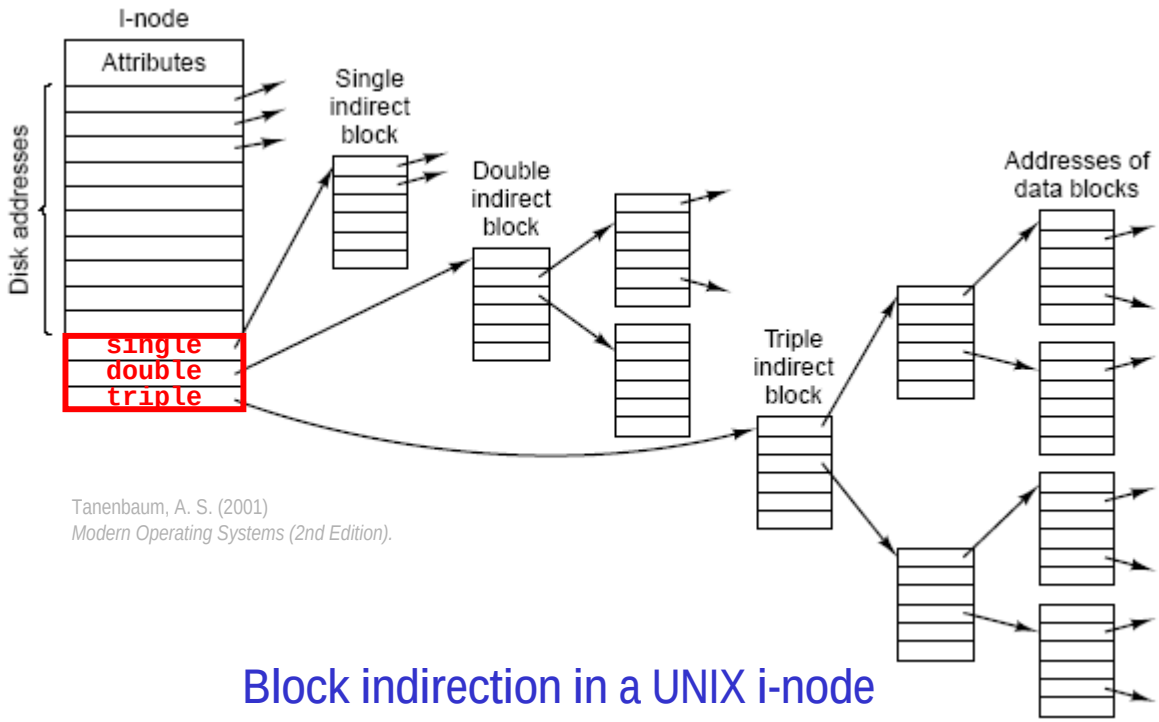


Indexed allocation of disk space

6.d File System Implementation

File block allocation

- Multilevel indexing: the i-node block table
  - ✓ keep the first 10 block pointers in the i-node structure
  - ✓ then export the next 128 into a block accessed through single indirection; and the next 16184 into a block of 128 blocks, etc.



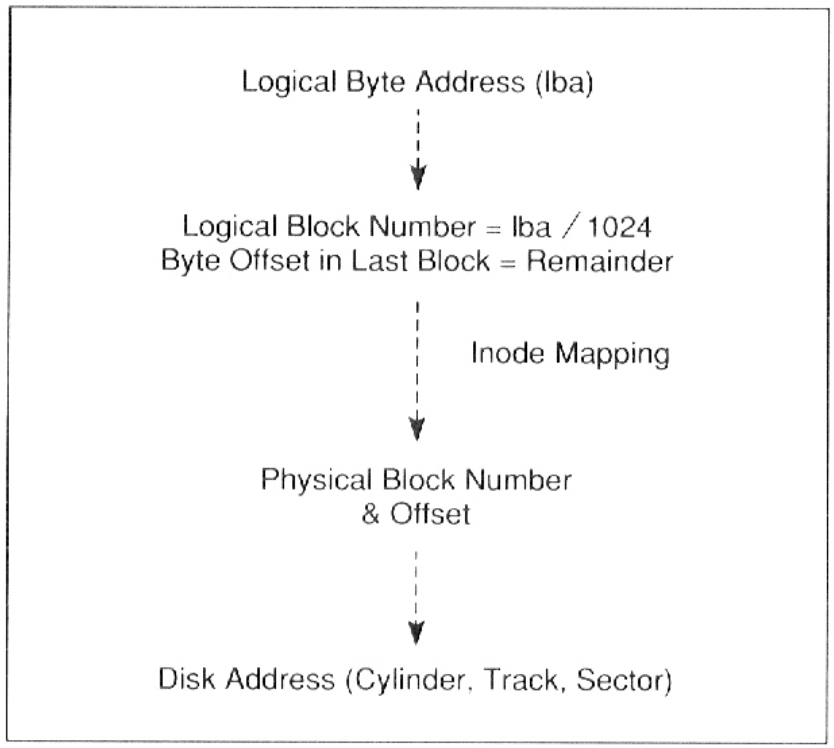
Block indirection in a UNIX i-node

6.d File System Implementation

File block allocation

- Logical to physical address translation
  - ✓ the logical byte address in the file is converted to a logical block number in the file
  - ✓ the logical block # is mapped to a physical block # + offset through the i-node tables
  - ✓ finally, the physical block # is converted to disk-specific coordinates (cylinder, track, sector)

Andleigh P. K. (1990)
UNIX System Architecture.

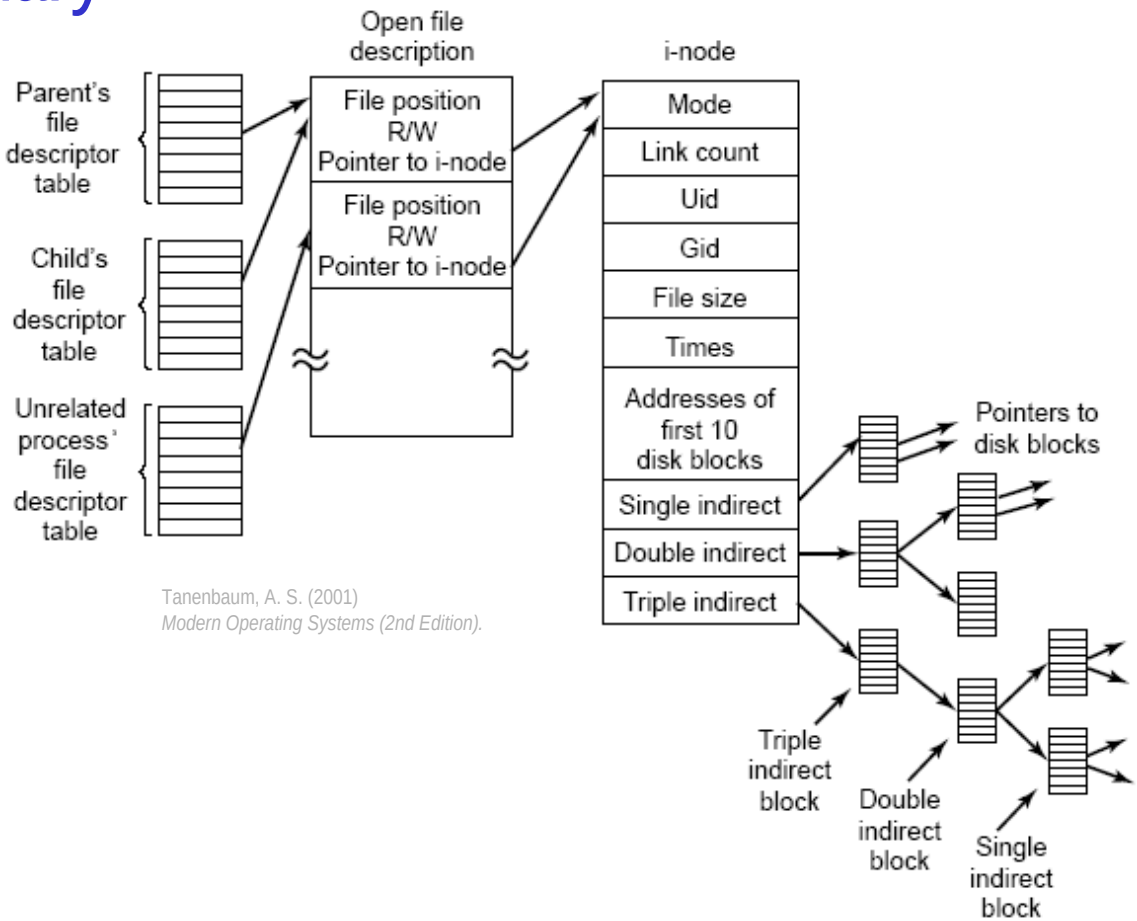


Logical to physical address translation

6.d File System Implementation

File block allocation

- Summary



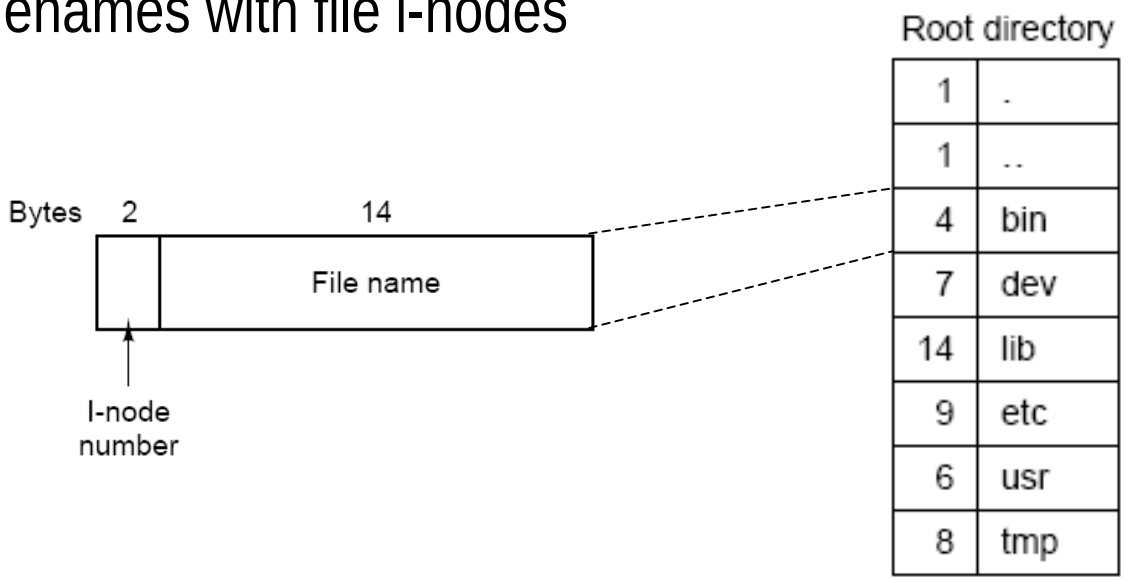
Tanenbaum, A. S. (2001)
Modern Operating Systems (2nd Edition).

Relation between file descriptor table, open (or "system") file table, and i-node table

6.d File System Implementation

Directory structure

- Structure of directory files
  - ✓ directories are special files whose contents is managed by the O/S
  - ✓ in UNIX a directory is simply a list of entries that associate filenames with file i-nodes

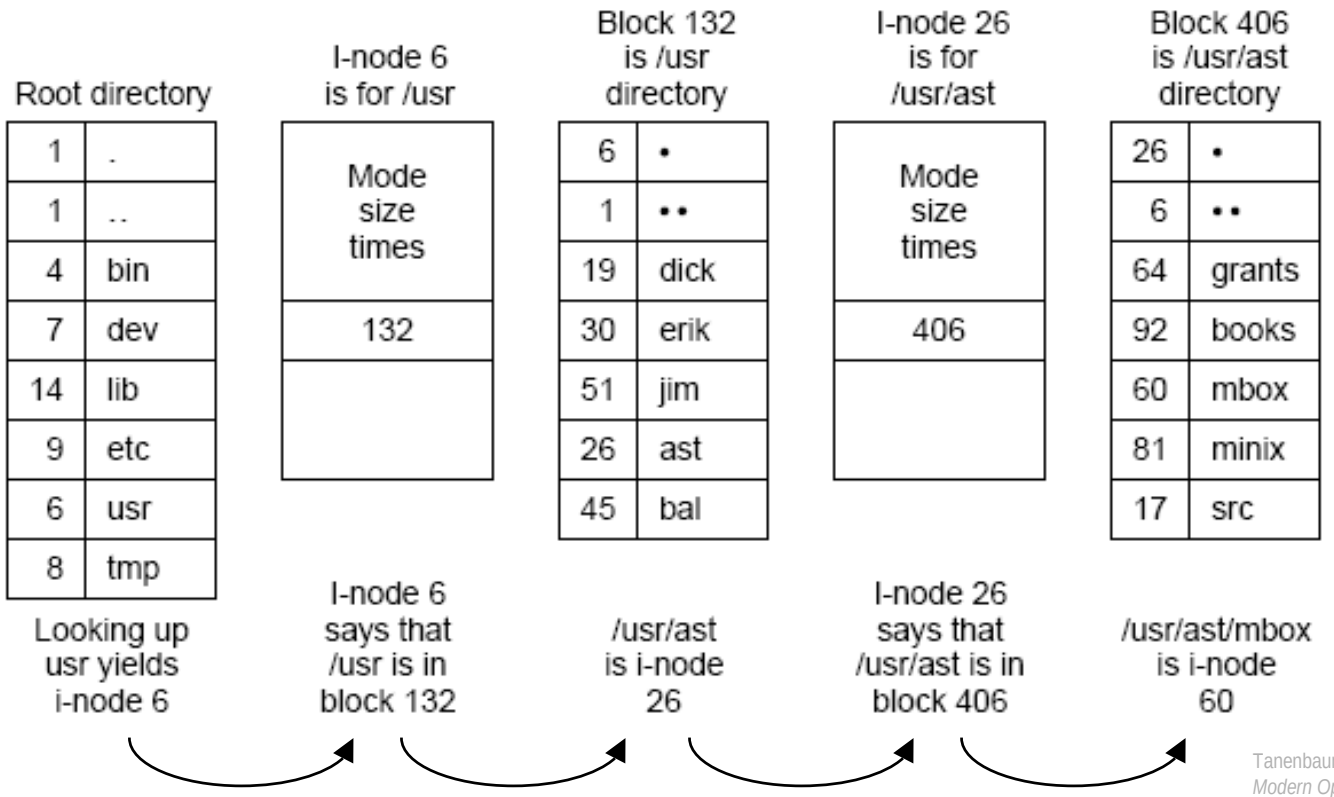


Tanenbaum, A. S. (2001)
Modern Operating Systems (2nd Edition).

Directory entry and directory contents in UNIX

6.d File System Implementation

- How the O/S searches for a requested file
  - ✓ ex: looking up **`/usr/ast/mbox`**



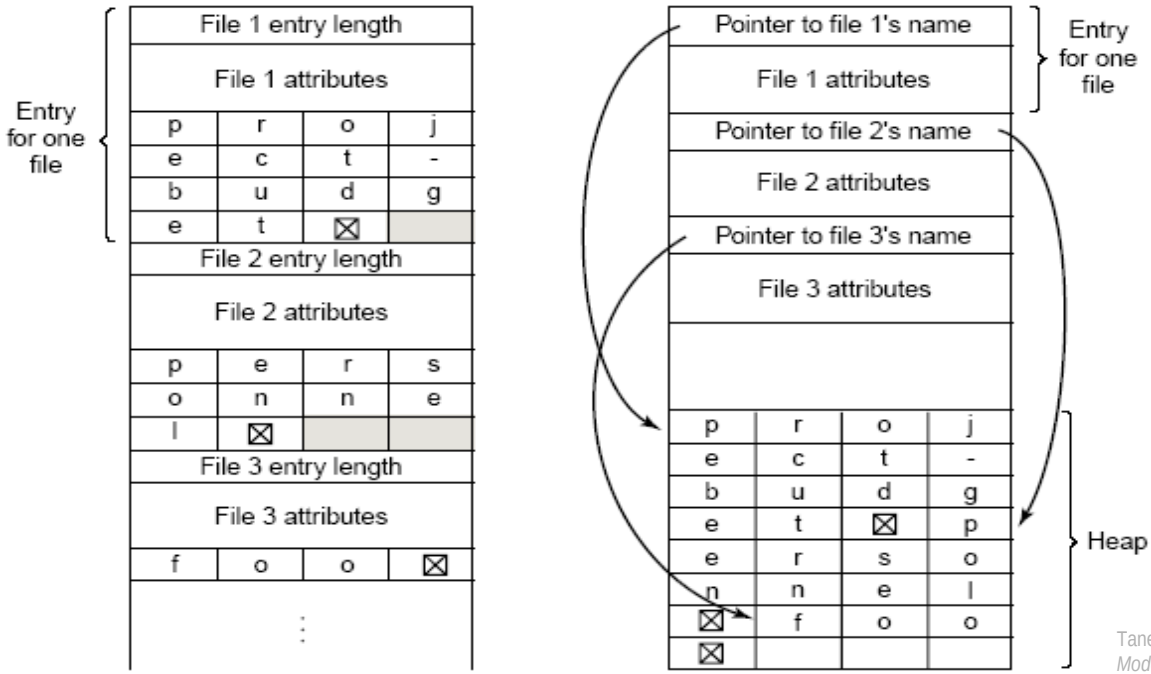
Tanenbaum, A. S. (2001)
Modern Operating Systems (2nd Edition).

The steps in looking up **`/usr/ast/mbox`**

6.d File System Implementation

➢ Handling long file names

- ✓ either put the filename in each file entry → variable-size entries
- ✓ or log all the filenames in a heap at the end of the directory

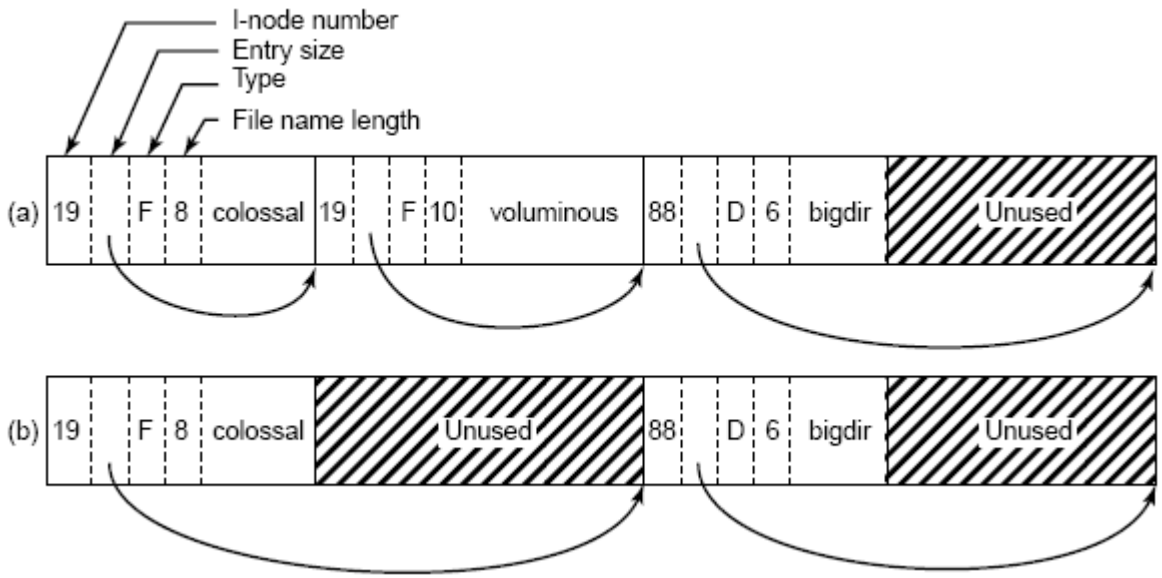


Tanenbaum, A. S. (2001) *Modern Operating Systems (2nd Edition).*

Two ways of handling long file names: (a) in-line and (b) in a heap

6.d File System Implementation

- Handling long file names (cont'd)
  - ✓ example: BSD



Tanenbaum, A. S. (2001)
Modern Operating Systems (2nd Edition).

A BSD directory with three files (a) before and (b) after one file is removed

Principles of Operating Systems

CS 446/646

6. File System

a. Overview of the File System

b. User Interface: Files

c. User Interface: Directories

d. File System Implementation

Principles of Operating Systems

CS 446/646

0. Course Presentation
1. Introduction to Operating Systems
2. Processes
3. Memory Management
4. CPU Scheduling
5. Input/Output
6. File System
7. **Case Studies**