File Management

B. Ramamurthy

Introduction

- File system is the most visible aspect of an operating system. It provides the mechanism for on-line storage of and access to programs and data. It provides the persistent storage capability to a system.
- File systems consists of a collection of **files,** a **directory structure, access methods, secondary storage management and partitions** (which separate logical and physical collection of directories.)

Topics for Discussion

- File Attributes
- File operations and structures
- File Management
- File Organization
- File Directories
- Directory Structure
- File sharing

File Attributes

- Name: Symbolic reference for identifying the file object.
- Type: Information that indicates the contents of the file.
- Location: This information is a pointer a device and to the location of the file on that device.
- Size : The current size of the file (in bytes, words, or blocks).
- Protection: Access control information (RWX)
- Time, date and user identification: This information may be kept for (1) creation, (2) last modification and (3) last use. Useful for protection, security and usage monitoring.

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

Possible file attributes

File Operations and structures

- A file is an abstract data type.
- Operations: open, close, create, destroy, copy, rename, list, read, write, update, insert item, delete item, size,...
- Open file table: Table containing information about open files. When a file operation is requested, an index into this table is used for locating the file. When a file is closed the entry is removed from the table.
- Current file pointer: Last read/write location is kept as a current-file-position pointer. Each process using the file has a unique pointer. Where is it kept?
- File open count: Number of opens done on a given file. To allow deletion from Open file table, once the count reaches 0.

File Operations

1. Create
2. Delete
3. Open
4. Close
5. Read
6. Write
7. Append
8. Seek
9. Get attributes
10. Set Attributes
11. Rename

An Example Program Using File System Calls (1/2)

```
/* File copy program. Error checking and reporting is minimal. */

#include <sys/types.h>                  /* include necessary header files */
#include <fcntl.h>
#include <stdlib.h>
#include <unistd.h>

int main(int argc, char *argv[]);       /* ANSI prototype */

#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 */
```

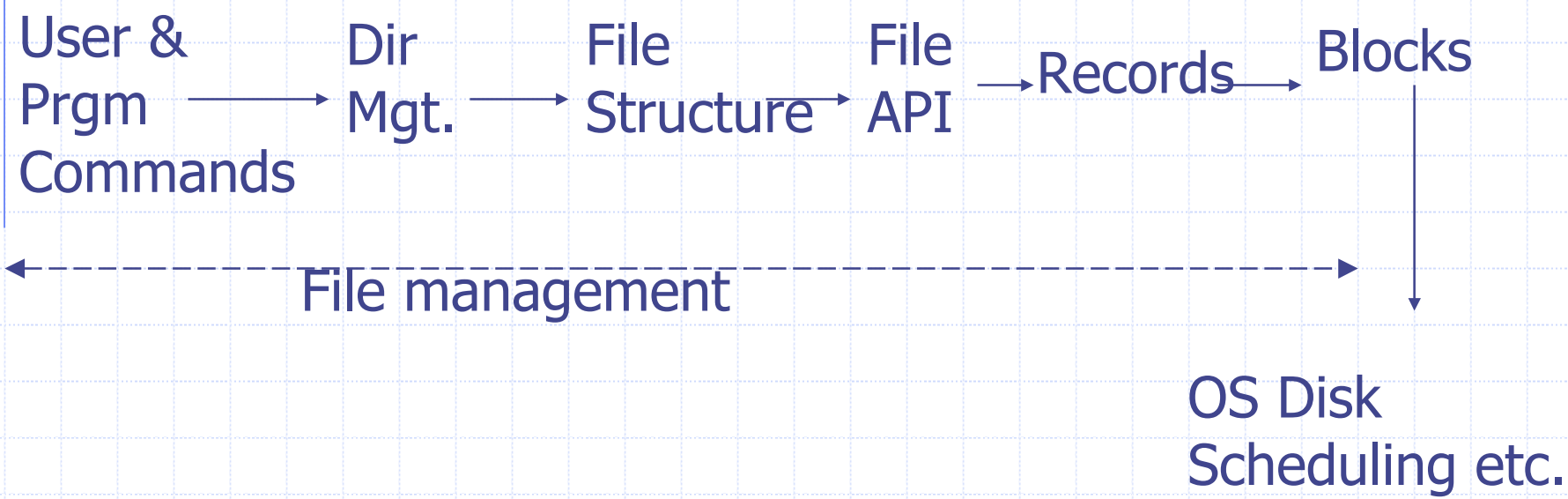
An Example Program Using File System Calls (2/2)

```
/* 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 */
}
```

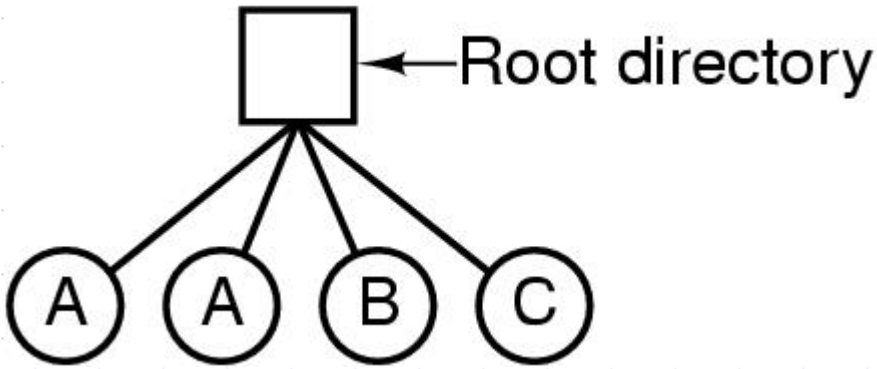
File management

- Users and application programs interact with file system by means of commands for performing operations on files.
- These commands are translated into specific file manipulation commands, after ensuring that the kind of access requested is allowed.
- User view may be that of records or few bytes, but the actual IO is done in blocks. Data conversion to block "packing" is done. Optimized where applicable.
- Now IO subsystems takes over by translating the file sub commands into IO subsystem (disk IO) commands.

Elements of File Management

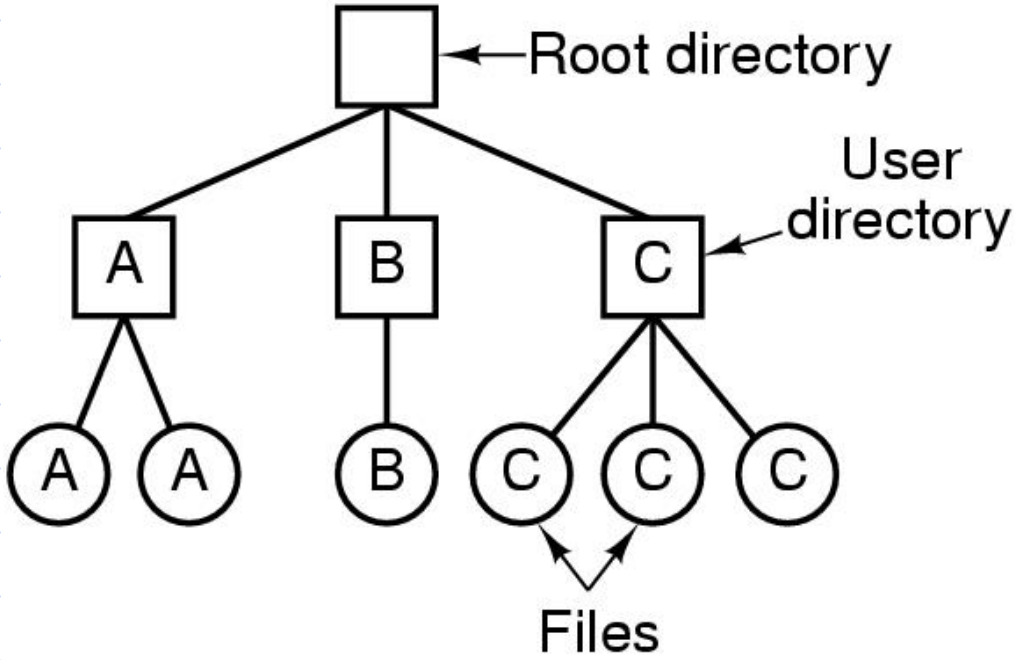


Directories
Single-Level Directory Systems



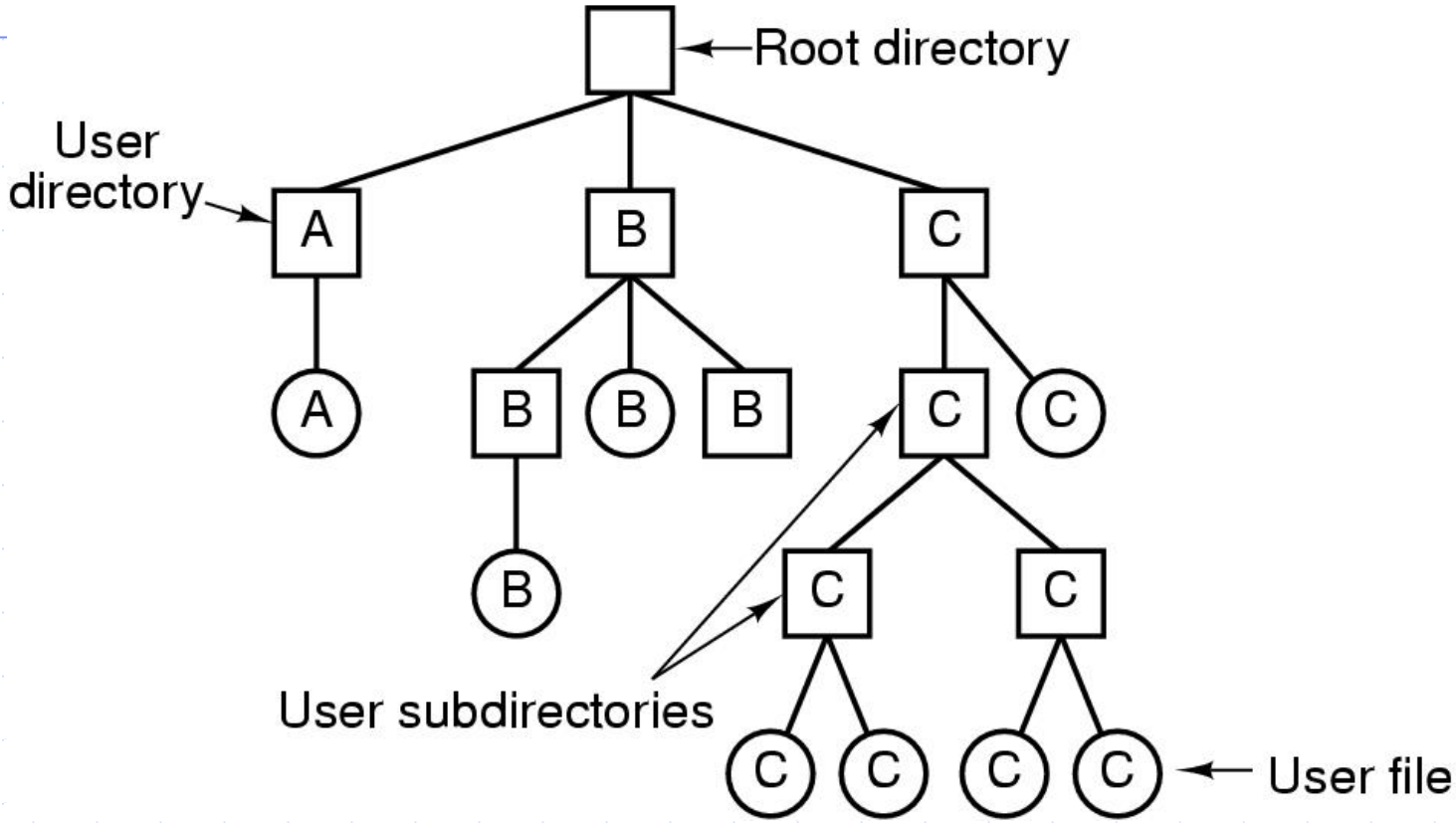
- A single level directory system
  - contains 4 files
  - owned by 3 different people, A, B, and C

Two-level Directory Systems



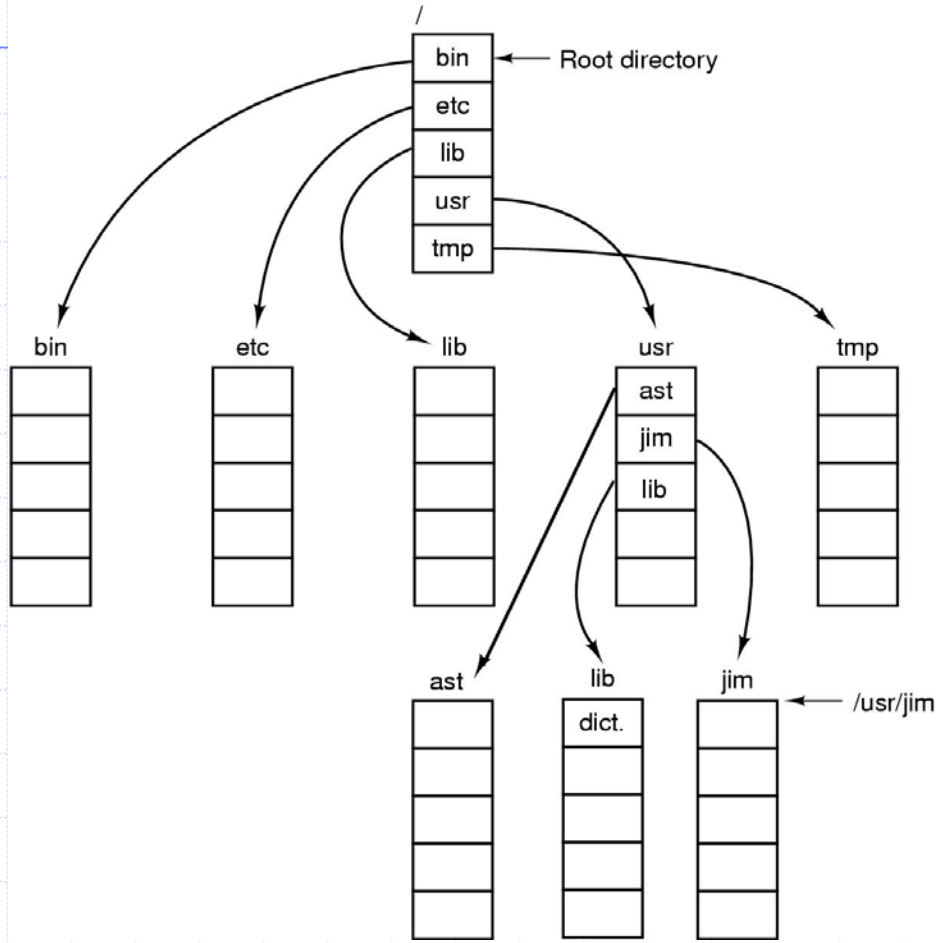
Letters indicate *owners* of the directories and files

Hierarchical Directory Systems



A hierarchical directory system

Path Names

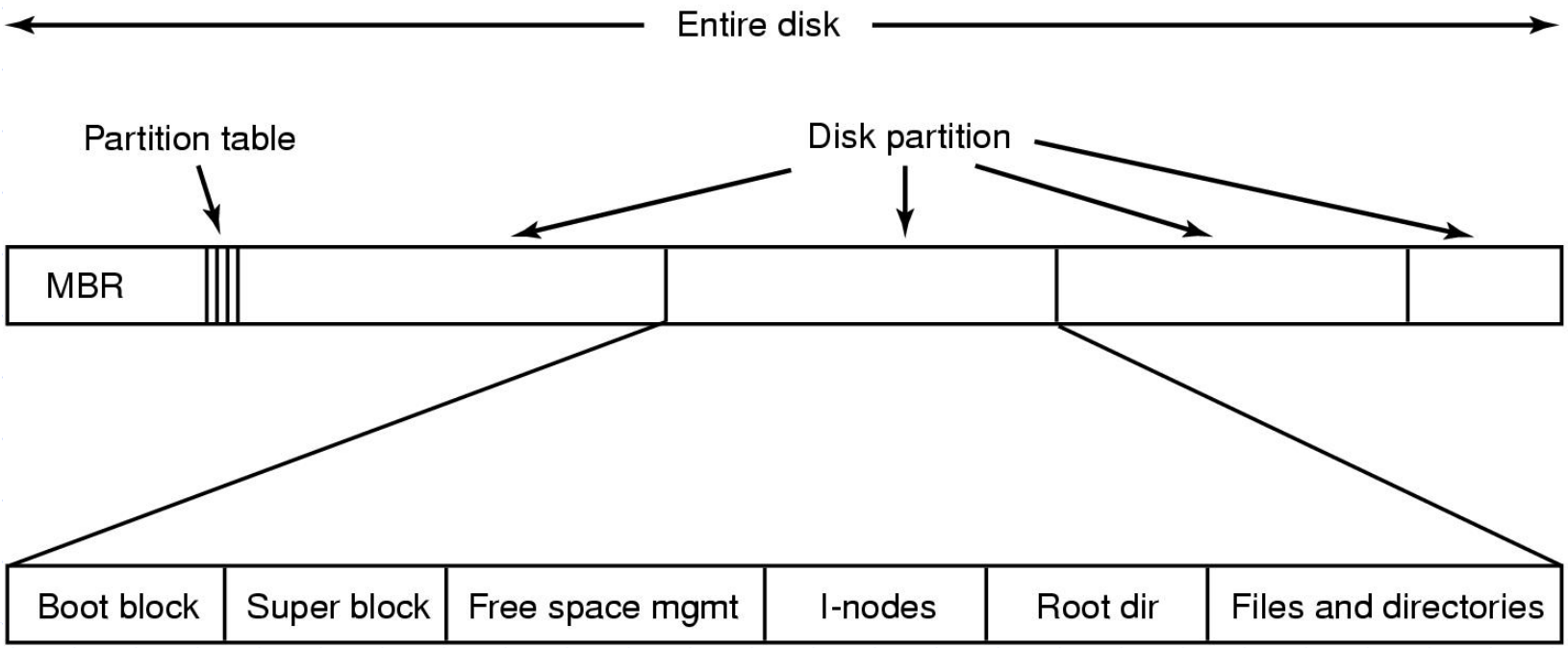


A UNIX directory tree

Directory Operations

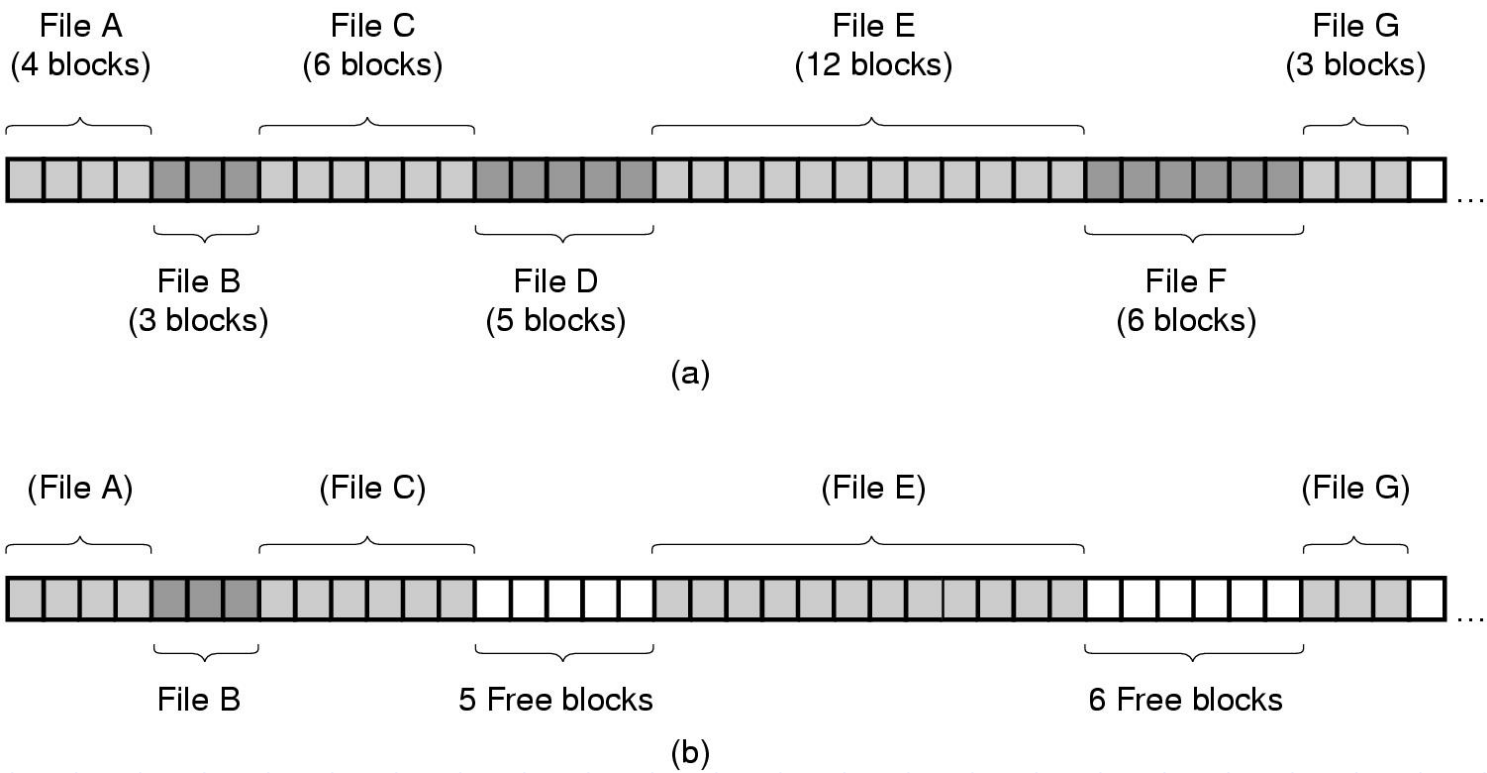
1. Create
2. Delete
3. Opendir
4. Closedir
5. Readdir
6. Rename
7. Link
8. Unlink

File System Implementation



A possible file system layout

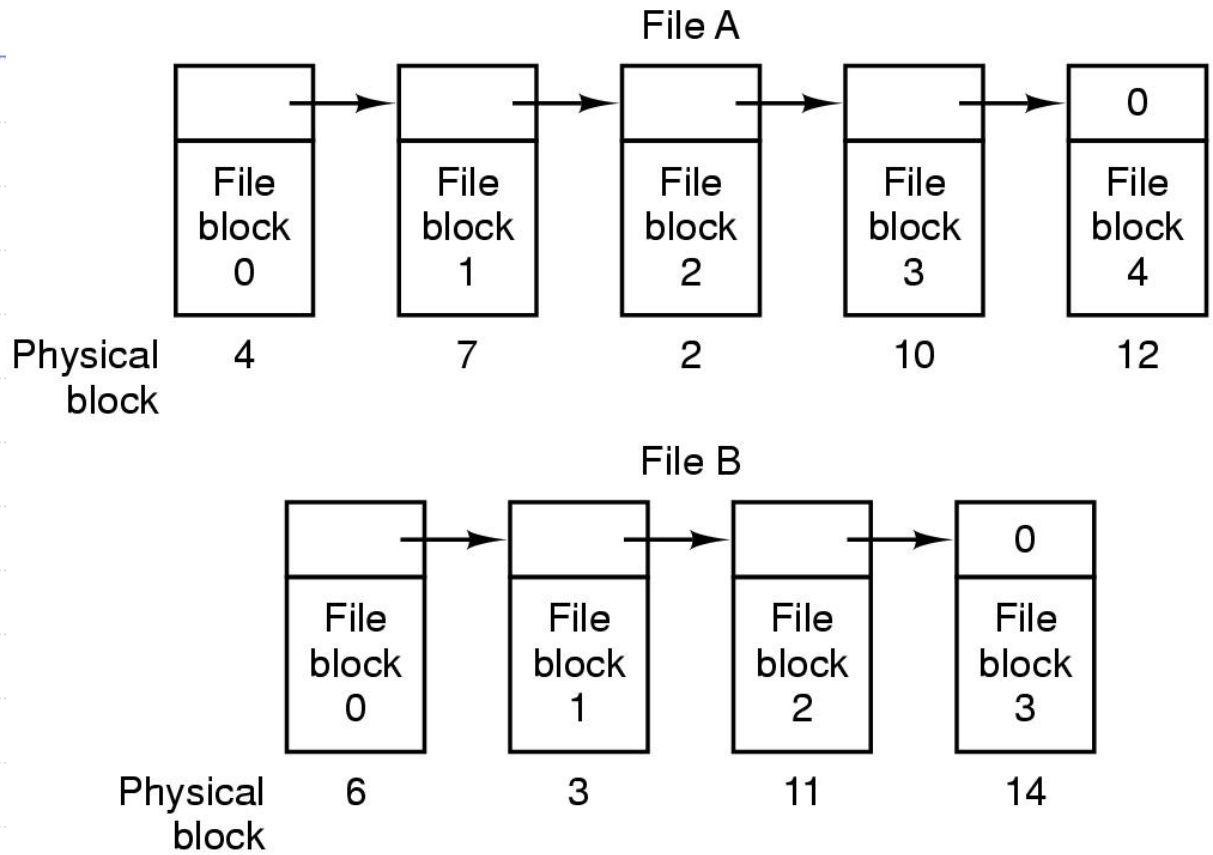
Implementing Files (1)



(a) Contiguous allocation of disk space for 7 files

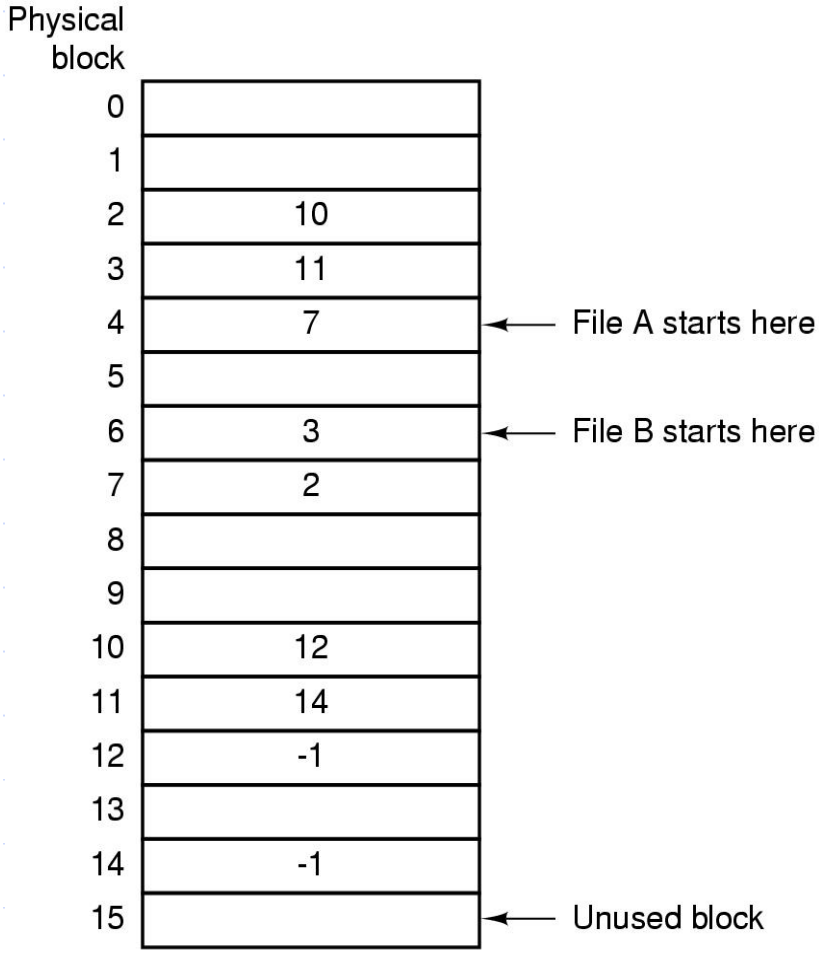
(b) State of the disk after files *D* and *E* have been removed

Implementing Files (2)



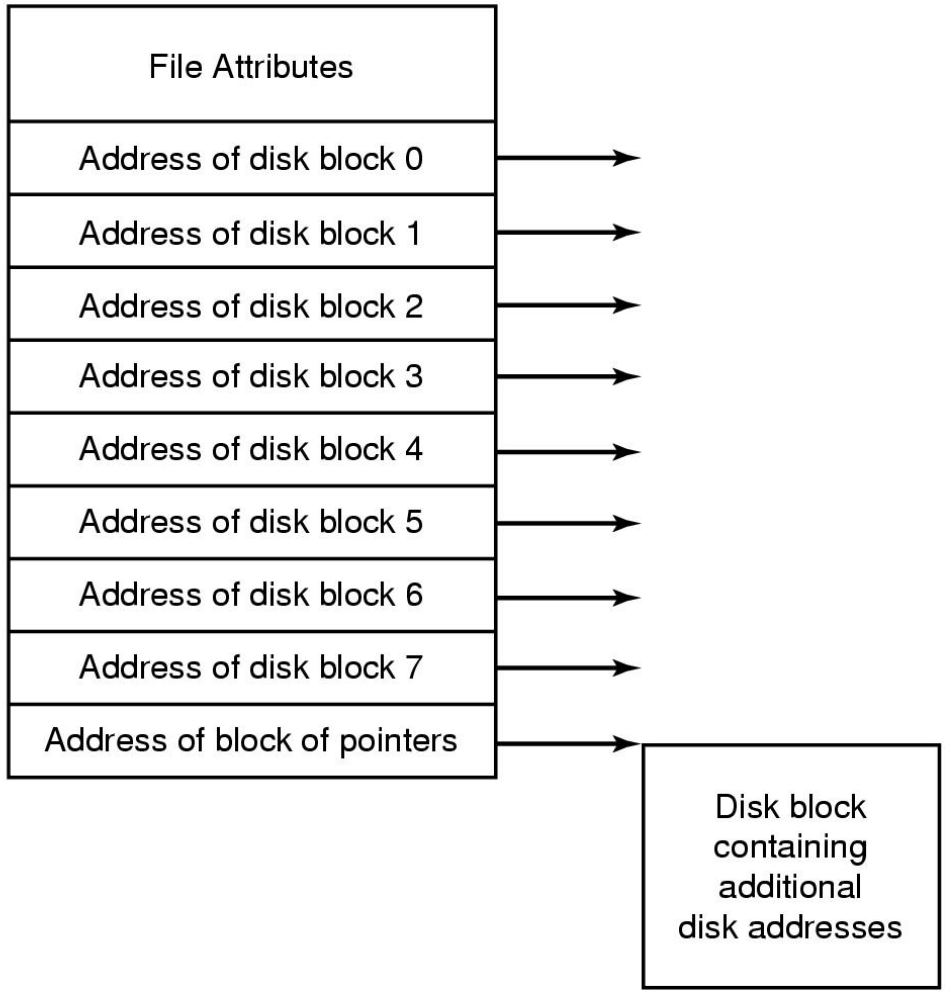
Storing a file as a linked list of disk blocks

Implementing Files (3)



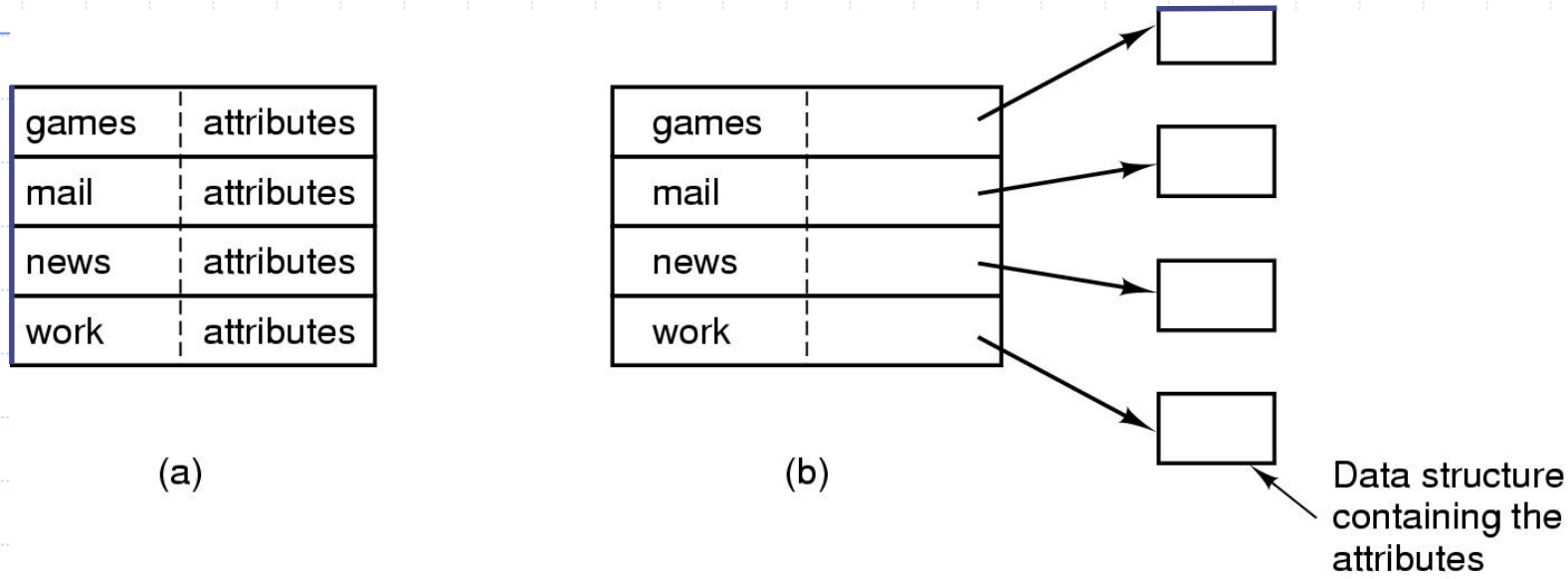
Linked list allocation using a file allocation table in RAM

Implementing Files (4)



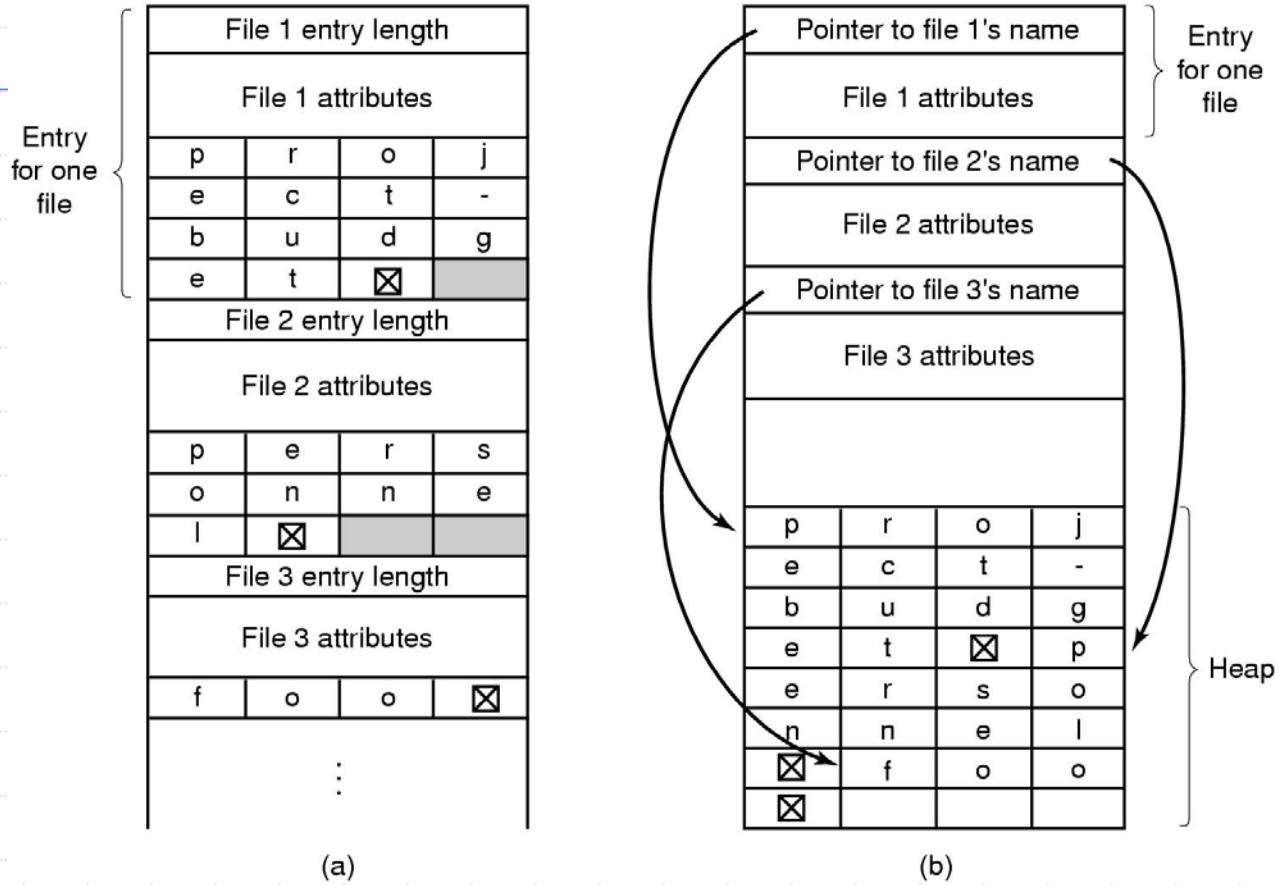
An example i-node

Implementing Directories (1)



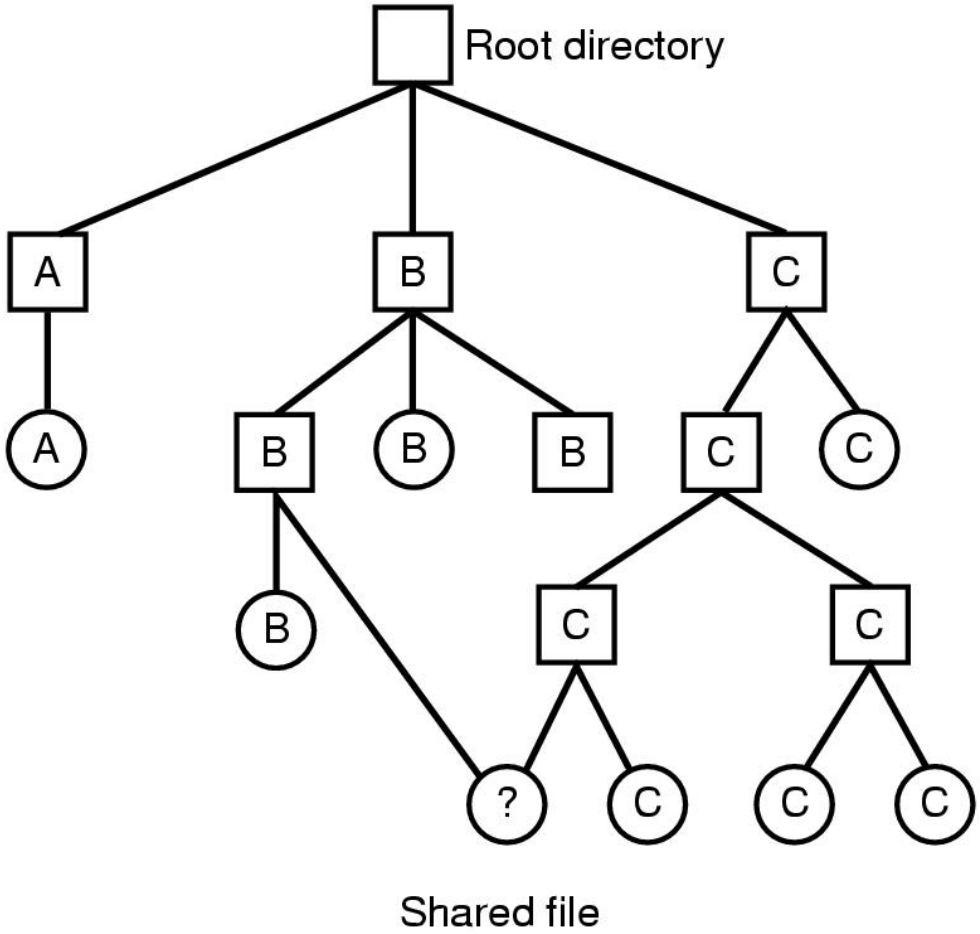
(a) A simple directory
fixed size entries
disk addresses and attributes in directory entry
(b) Directory in which each entry just refers to an i-node

Implementing Directories (2)



- Two ways of handling long file names in directory
  - (a) In-line
  - (b) In a heap

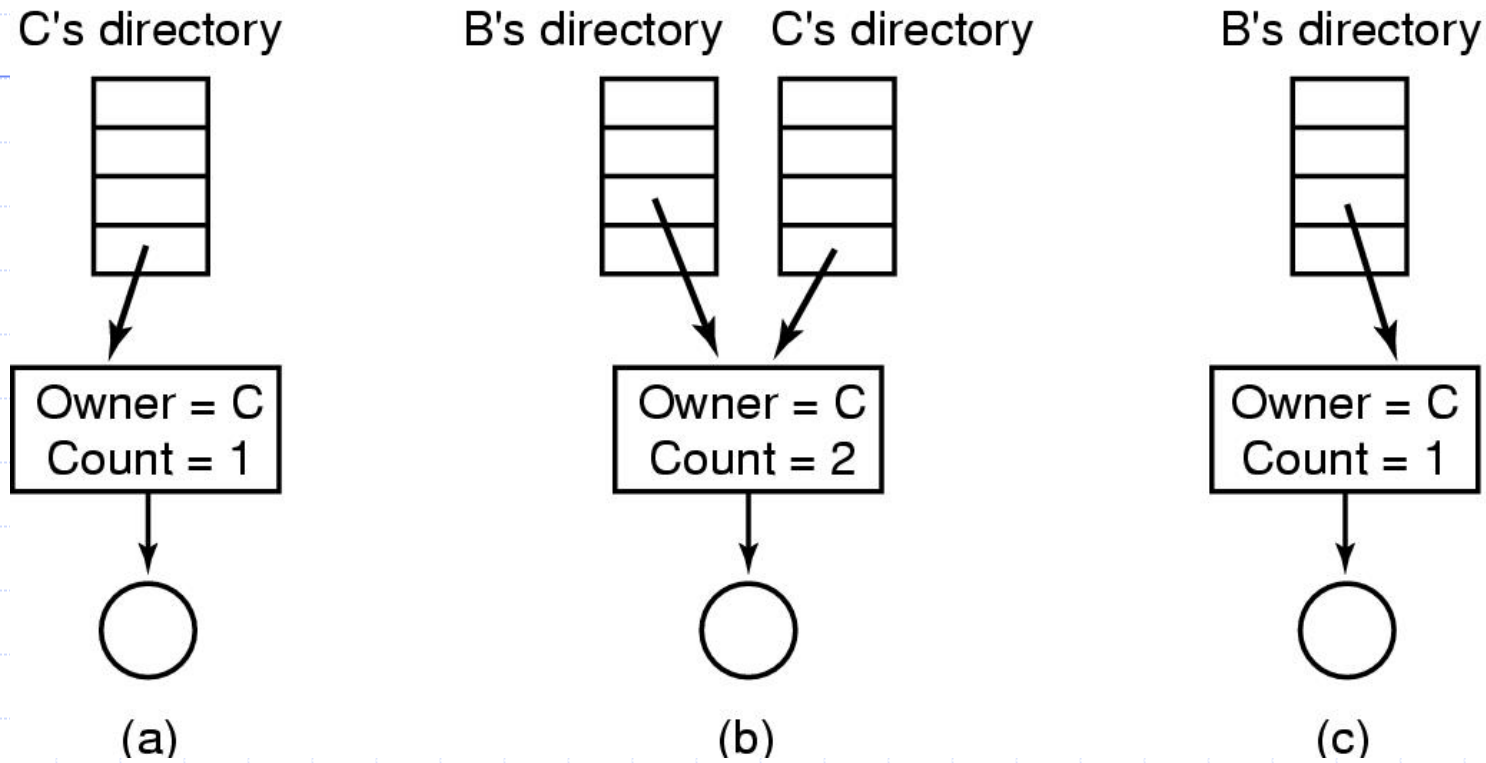
Shared Files (1)



Shared file

File system containing a shared file

Shared Files (2)

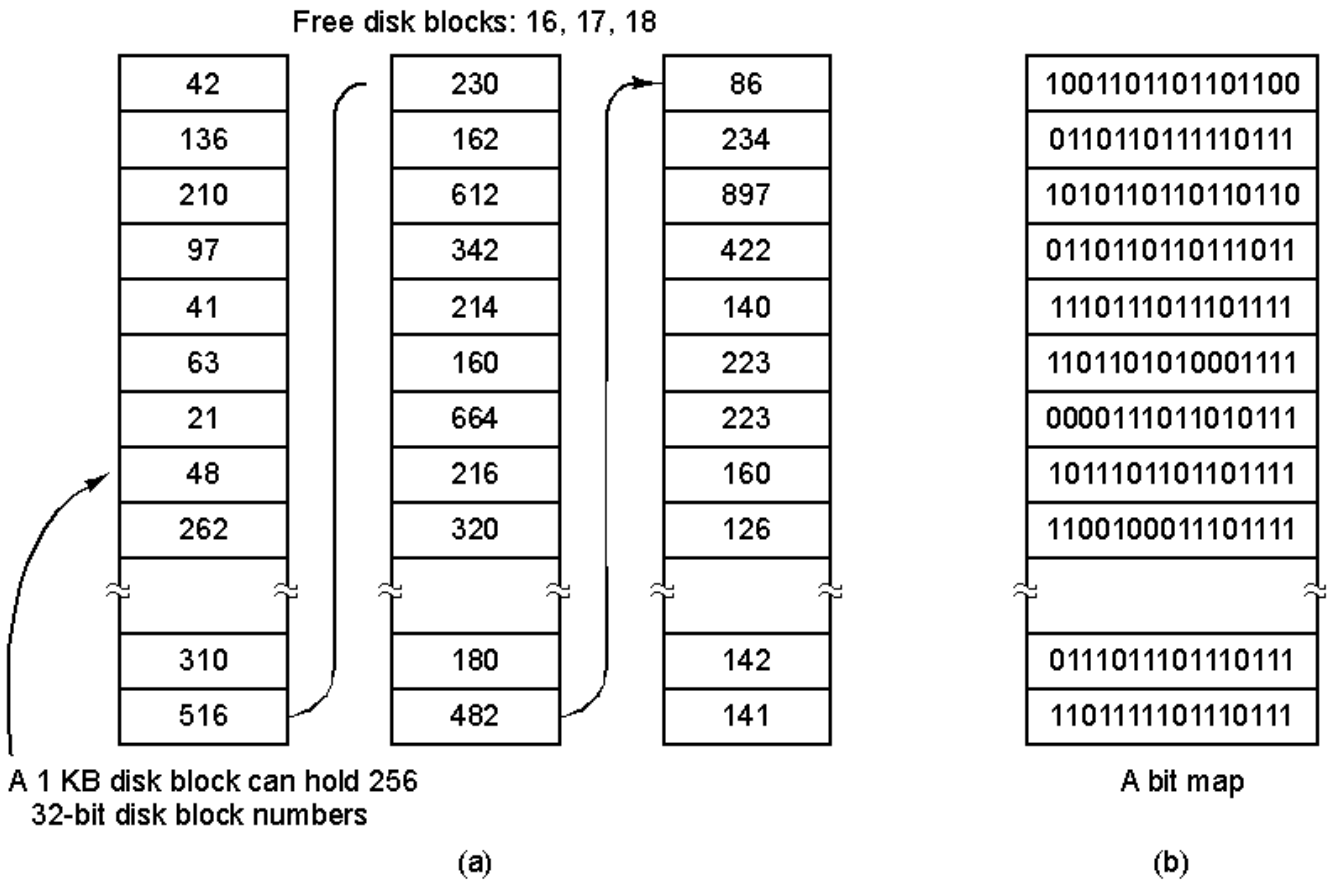


(a) Situation prior to linking

(b) After the link is created

(c)After the original owner removes the file

Disk Space Management



(a) Storing the free list on a linked list

(b) A bit map

Unix File Management

- Unix kernel views all files as streams of bytes.
- Four types of files are distinguished:
  - Ordinary : regular files with data from user, or an application.
  - Directory : Contains list of file names + pointers to associated inodes.
  - Special : Terminals and other peripherals are abstracted as files.
  - Named : named pipes.

Operations

- pathname searching : lookup
- name creation : creat, mknod, link, symlink, mkdir
- name change/deletion: rename, remove, rmdir
- attribute manipulation: access, getattr, setattr
- object interpretation: open, readir,readlink,mmap,close
- process control : advlock,ioctl,select
- object management : lock, unlock,inactive,reclaim,abortop

Inodes

- Inode (information node) is a structure that contains the key information needed for managing a file.
- Several files names may be associated with an inode.
- But each file contains exactly one file.

Information in an inode

- File mode (access and execution permissions)
- Link count (how many references)
- Owner ID
- Group ID
- File Size
- File Address : 39 bytes of address information as explained in the next slide
- Last accessed time, last modified time, late inode modification time

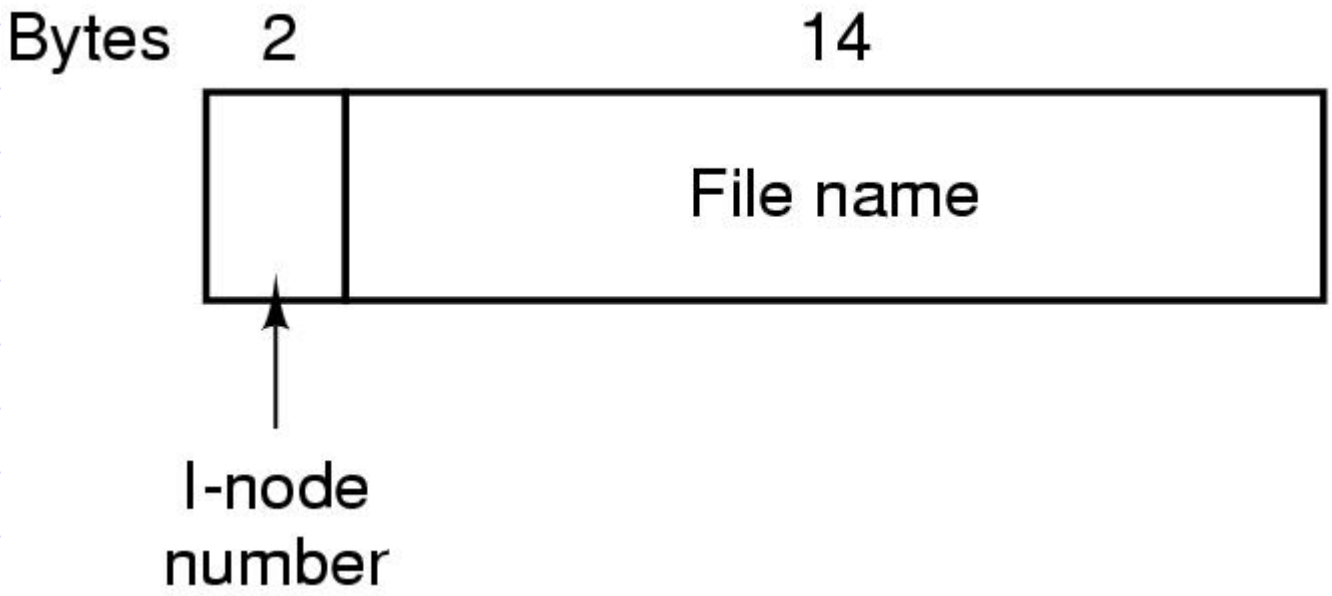
File address

- 13 3-byte addresses (39 bytes)
- Direct :10 : direct pointer 10 1K chunks of memory
- Single indirect: 1 : indirect block of 256 points each of which points to a 1K : 256 K
- Double indirect: 1: 256 X 256 : 65M
- Triple Indirect : 1: 256X 256 X 256 : 16G

Directories

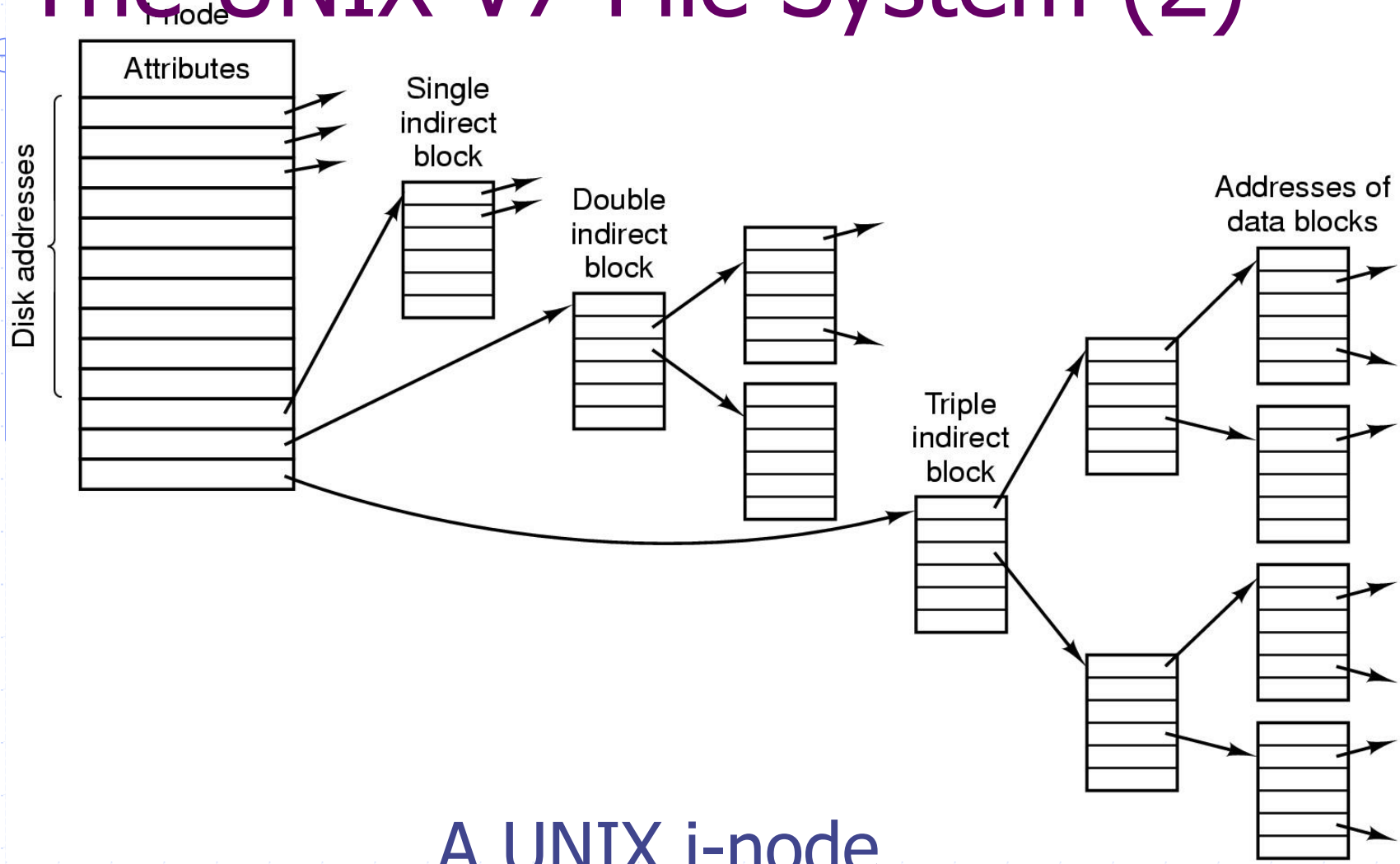
- Directories are allocated in units called chunks.
- Chunks are broken up into variable-length directory entries.
- A directory entry contains: index to inode structures, the size of entry in bytes, type of entry, length of filename, pointer to next entry.

The UNIX V7 File System (1)



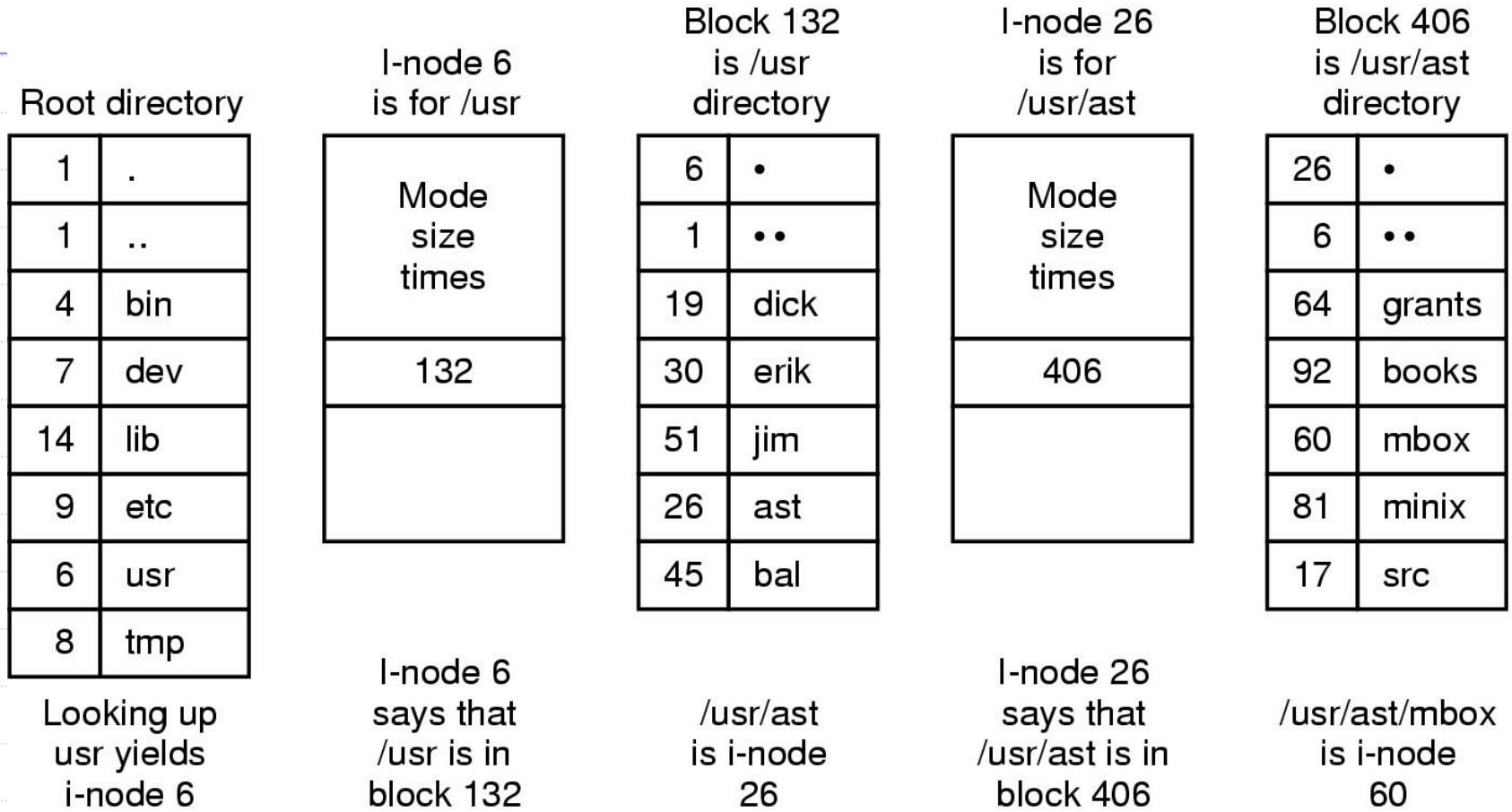
A UNIX V7 directory entry

The UNIX V7 File System (2)



A UNIX i-node

The UNIX V7 File System (3)



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

Summary

- We studied
  - The file abstraction and file API.
  - File structure, directory structure and storage allocation.
  - Unix file system case study.