操作系统原理

第十一章：文件系统

洪明坚

重庆大学软件学院

February 19, 2016

目录

Outline

File concept

File concept

- Logical contiguous secondary storage

File concept

- Logical contiguous secondary storage
  - The smallest allotment of logical continuous secondary storage from a user's perspective

File concept

- Logical contiguous secondary storage
  - The smallest allotment of logical continuous secondary storage from a user's perspective
- Types

File concept

- Logical contiguous secondary storage
  - The smallest allotment of logical continuous secondary storage from a user's perspective
- Types
  - Data

File concept

- Logical contiguous secondary storage
  - The smallest allotment of logical continuous secondary storage from a user's perspective
- Types
  - Data
    - numeric

File concept

- Logical contiguous secondary storage
  - The smallest allotment of logical continuous secondary storage from a user's perspective
- Types
  - Data
    - numeric
    - character

File concept

- Logical contiguous secondary storage
  - The smallest allotment of logical continuous secondary storage from a user's perspective
- Types
  - Data
    - numeric
    - character
    - binary

File concept

- Logical contiguous secondary storage
  - The smallest allotment of logical continuous secondary storage from a user's perspective
- Types
  - Data
    - numeric
    - character
    - binary
    - .....

File concept

- Logical contiguous secondary storage
  - The smallest allotment of logical continuous secondary storage from a user's perspective
- Types
  - Data
    - numeric
    - character
    - binary
    - .....
  - Program

File structure (1/2)

File structure (1/2)

- None - just a sequence of bytes

File structure (1/2)

- None - just a sequence of bytes
- Simple record structure

File structure (1/2)

- None - just a sequence of bytes
- Simple record structure
  - Lines

File structure (1/2)

- None - just a sequence of bytes
- Simple record structure
  - Lines
  - Fixed length

File structure (1/2)

- None - just a sequence of bytes
- Simple record structure
  - Lines
  - Fixed length
  - Variable length

File structure (1/2)

- None - just a sequence of bytes
- Simple record structure
  - Lines
  - Fixed length
  - Variable length
- Complex structures
  - Formatted document

File structure (1/2)

- None - just a sequence of bytes
- Simple record structure
  - Lines
  - Fixed length
  - Variable length
- Complex structures
  - Formatted document
  - Object file

File structure (1/2)

- None - just a sequence of bytes
- Simple record structure
  - Lines
  - Fixed length
  - Variable length
- Complex structures
  - Formatted document
  - Object file
  - Executable file

File structure (1/2)

- None - just a sequence of bytes
- Simple record structure
  - Lines
  - Fixed length
  - Variable length
- Complex structures
  - Formatted document
  - Object file
  - Executable file
- Who decides?

File structure (1/2)

- None - just a sequence of bytes
- Simple record structure
  - Lines
  - Fixed length
  - Variable length
- Complex structures
  - Formatted document
  - Object file
  - Executable file
- Who decides?
  - Operating system

File structure (1/2)

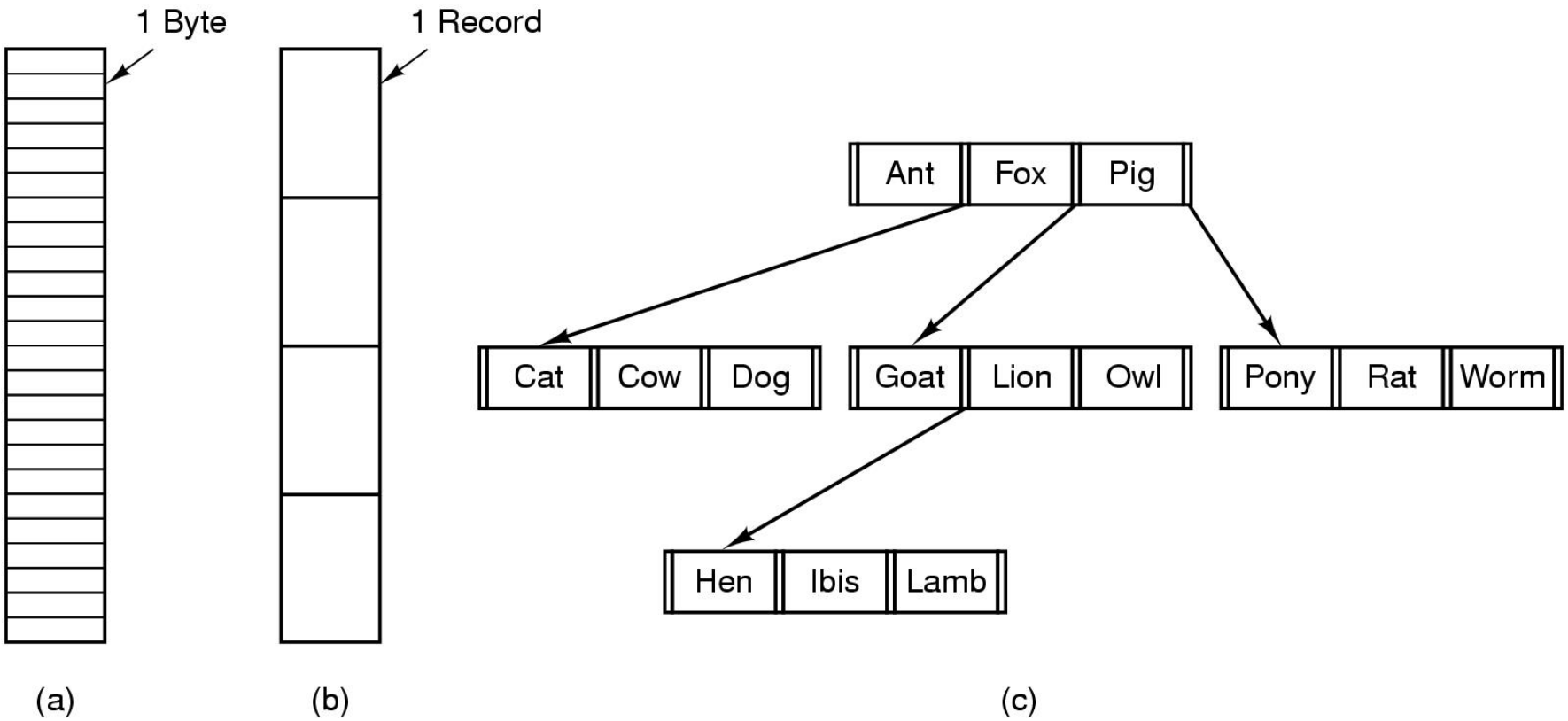
- None - just a sequence of bytes
- Simple record structure
  - Lines
  - Fixed length
  - Variable length
- Complex structures
  - Formatted document
  - Object file
  - Executable file
- Who decides?
  - Operating system
    - Structure of the executable file and shared library file

File structure (1/2)

- None - just a sequence of bytes
- Simple record structure
  - Lines
  - Fixed length
  - Variable length
- Complex structures
  - Formatted document
  - Object file
  - Executable file
- Who decides?
  - Operating system
    - Structure of the executable file and shared library file
  - Program

File structure (2/2)

File structure (2/2)



File attributes

File attributes

- Name

File attributes

- Name - only information kept in human-readable form

File attributes

- Name - only information kept in human-readable form
- Type

File attributes

- Name - only information kept in human-readable form
- Type - needed for systems that support different types

File attributes

- Name - only information kept in human-readable form
- Type - needed for systems that support different types
- Location

File attributes

- Name - only information kept in human-readable form
- Type - needed for systems that support different types
- Location - pointer to file location on the device

File attributes

- Name - only information kept in human-readable form
- Type - needed for systems that support different types
- Location - pointer to file location on the device
- Size

File attributes

- Name - only information kept in human-readable form
- Type - needed for systems that support different types
- Location - pointer to file location on the device
- Size - current file size

File attributes

- Name - only information kept in human-readable form
- Type - needed for systems that support different types
- Location - pointer to file location on the device
- Size - current file size
- Protection

File attributes

- Name - only information kept in human-readable form
- Type - needed for systems that support different types
- Location - pointer to file location on the device
- Size - current file size
- Protection - controls who can do reading, writing and executing

File attributes

- Name - only information kept in human-readable form
- Type - needed for systems that support different types
- Location - pointer to file location on the device
- Size - current file size
- Protection - controls who can do reading, writing and executing
- Time, date and user identification

File attributes

- Name - only information kept in human-readable form
- Type - needed for systems that support different types
- Location - pointer to file location on the device
- Size - current file size
- Protection - controls who can do reading, writing and executing
- Time, date and user identification - data for protection, security and usage monitoring

File attributes

- Name - only information kept in human-readable form
- Type - needed for systems that support different types
- Location - pointer to file location on the device
- Size - current file size
- Protection - controls who can do reading, writing and executing
- Time, date and user identification - data for protection, security and usage monitoring
- All these information about files are kept in the directory structure, which is maintained on the device

File operations

File operations

- Create, Open, Close, Read, Write, Seek, Delete

File operations

- Create, Open, Close, Read, Write, Seek, Delete
- Open files

File operations

- Create, Open, Close, Read, Write, Seek, Delete
- Open files
  - When a file has been opened, in addition to the information stored on the device, several pieces of data are needed to manage open files

File operations

- Create, Open, Close, Read, Write, Seek, Delete
- Open files
  - When a file has been opened, in addition to the information stored on the device, several pieces of data are needed to manage open files
    - File pointer

File operations

- Create, Open, Close, Read, Write, Seek, Delete
- Open files
  - When a file has been opened, in addition to the information stored on the device, several pieces of data are needed to manage open files
    - File pointer - pointer to last read/write location, per process that has the file open

File operations

- Create, Open, Close, Read, Write, Seek, Delete
- Open files
  - When a file has been opened, in addition to the information stored on the device, several pieces of data are needed to manage open files
    - File pointer - pointer to last read/write location, per process that has the file open
    - File-open count

File operations

- Create, Open, Close, Read, Write, Seek, Delete
- Open files
  - When a file has been opened, in addition to the information stored on the device, several pieces of data are needed to manage open files
    - File pointer - pointer to last read/write location, per process that has the file open
    - File-open count - counter of number of times a file is open to allow removal of data from open-file table when the last process closes it

File operations

- Create, Open, Close, Read, Write, Seek, Delete
- Open files
  - When a file has been opened, in addition to the information stored on the device, several pieces of data are needed to manage open files
    - File pointer - pointer to last read/write location, per process that has the file open
    - File-open count - counter of number of times a file is open to allow removal of data from open-file table when the last process closes it
    - Device location of the file

File operations

- Create, Open, Close, Read, Write, Seek, Delete
- Open files
  - When a file has been opened, in addition to the information stored on the device, several pieces of data are needed to manage open files
    - File pointer - pointer to last read/write location, per process that has the file open
    - File-open count - counter of number of times a file is open to allow removal of data from open-file table when the last process closes it
    - Device location of the file - cache of data access information

File operations

- Create, Open, Close, Read, Write, Seek, Delete
- Open files
  - When a file has been opened, in addition to the information stored on the device, several pieces of data are needed to manage open files
    - File pointer - pointer to last read/write location, per process that has the file open
    - File-open count - counter of number of times a file is open to allow removal of data from open-file table when the last process closes it
    - Device location of the file - cache of data access information
    - Access rights

File operations

- Create, Open, Close, Read, Write, Seek, Delete
- Open files
  - When a file has been opened, in addition to the information stored on the device, several pieces of data are needed to manage open files
    - File pointer - pointer to last read/write location, per process that has the file open
    - File-open count - counter of number of times a file is open to allow removal of data from open-file table when the last process closes it
    - Device location of the file - cache of data access information
    - Access rights - per-process access mode information

File types

File types

- The content of a file determines its type.

File types

- The content of a file determines its type.
- How does the user know the type of a file?

File types

- The content of a file determines its type.
- How does the user know the type of a file?
  - Include the type as part of the file name called the *extension*.

File types

- The content of a file determines its type.
- How does the user know the type of a file?
  - Include the type as part of the file name called the *extension*.

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

File access

File access

- Sequential access

File access

- Sequential access
  - read all bytes/records from the beginning

File access

- Sequential access
  - read all bytes/records from the beginning
  - cannot jump around, but could rewind or back up

File access

- Sequential access
  - read all bytes/records from the beginning
  - cannot jump around, but could rewind or back up
  - convenient when the device is magnetic tape

File access

- Sequential access
  - read all bytes/records from the beginning
  - cannot jump around, but could rewind or back up
  - convenient when the device is magnetic tape
- Random access

File access

- Sequential access
  - read all bytes/records from the beginning
  - cannot jump around, but could rewind or back up
  - convenient when the device is magnetic tape
- Random access
  - bytes/records read in any order

File access

- Sequential access
  - read all bytes/records from the beginning
  - cannot jump around, but could rewind or back up
  - convenient when the device is magnetic tape
- Random access
  - bytes/records read in any order
  - essential for database systems

Questions

- Any questions?

Directory structure

Directory structure

- File system resides on the devices, such as disk and tape.

Directory structure

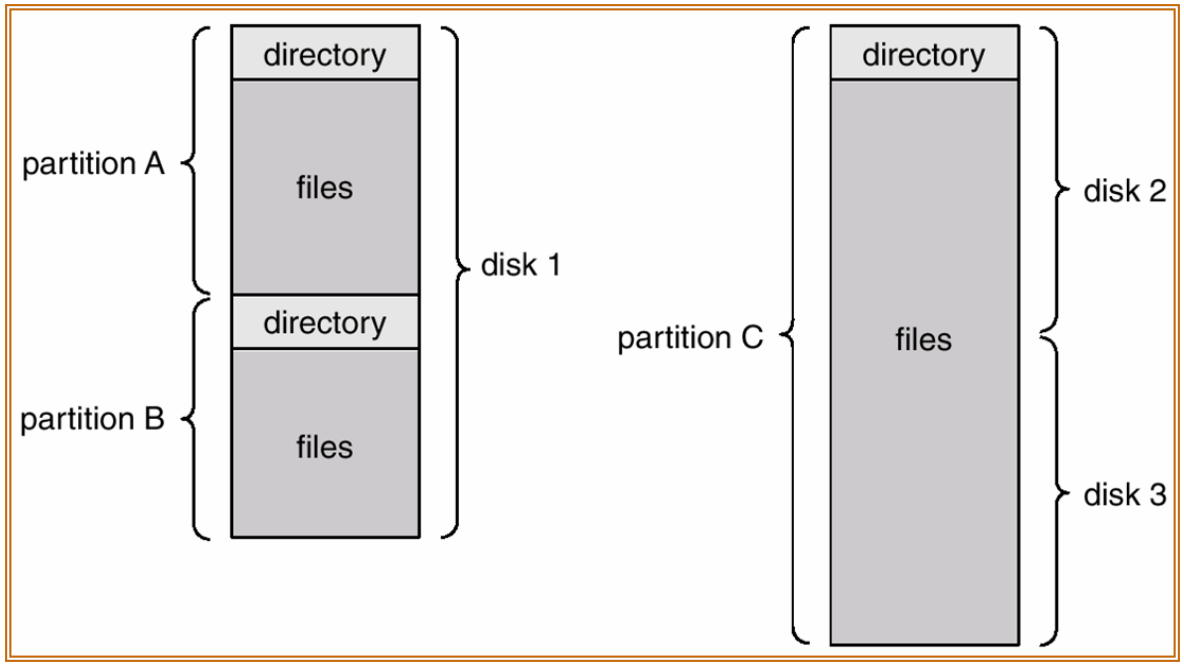
- File system resides on the devices, such as disk and tape.
  - Usually, devices are split into one or more *partitions* on which the file systems reside.

Directory structure

- File system resides on the devices, such as disk and tape.
  - Usually, devices are split into one or more *partitions* on which the file systems reside.
- To better organize files within a file system, *directory* is used to group files.

Directory structure

- File system resides on the devices, such as disk and tape.
  - Usually, devices are split into one or more *partitions* on which the file systems reside.
- To better organize files within a file system, *directory* is used to group files.



Directory operations

Directory operations

- Search for a file

Directory operations

- Search for a file
- Create a file

Directory operations

- Search for a file
- Create a file
- Delete a file

Directory operations

- Search for a file
- Create a file
- Delete a file
- List a directory

Directory operations

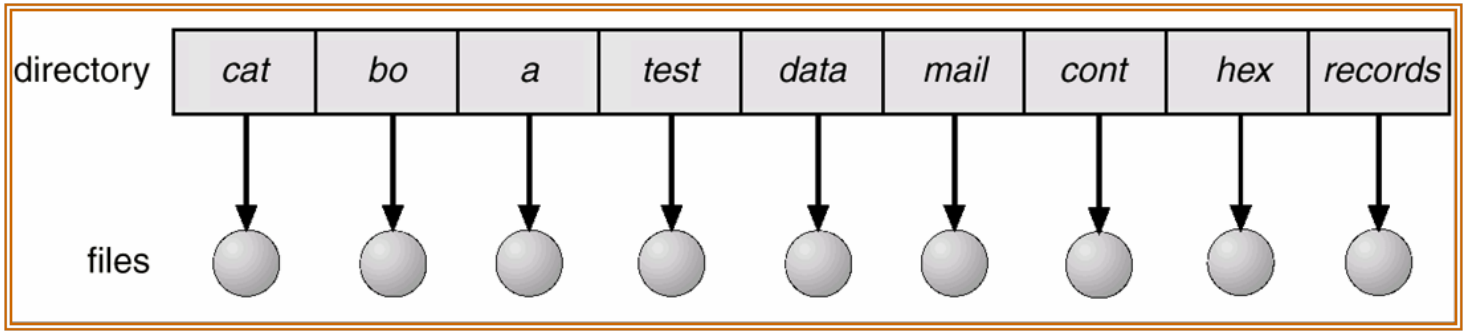
- Search for a file
- Create a file
- Delete a file
- List a directory
- Rename a file

Directory operations

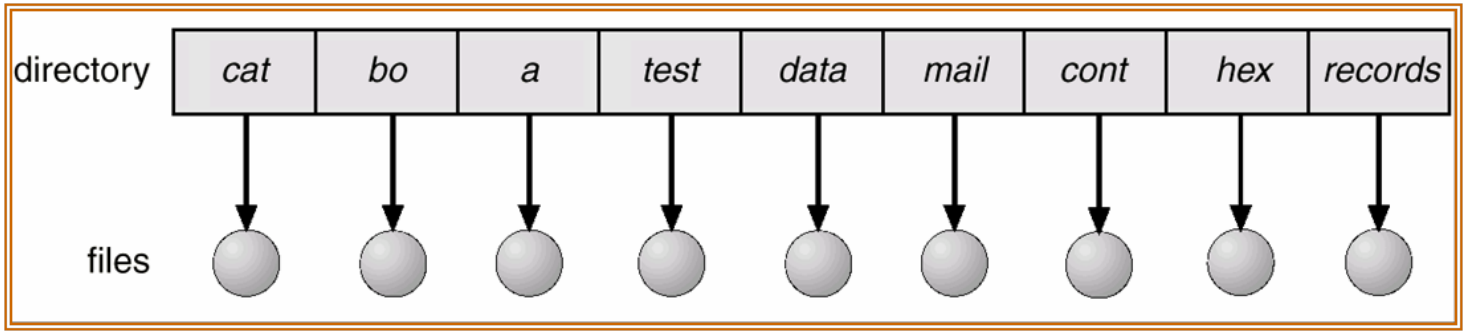
- Search for a file
- Create a file
- Delete a file
- List a directory
- Rename a file
- Traverse the entire file system

Single-level directory

Single-level directory

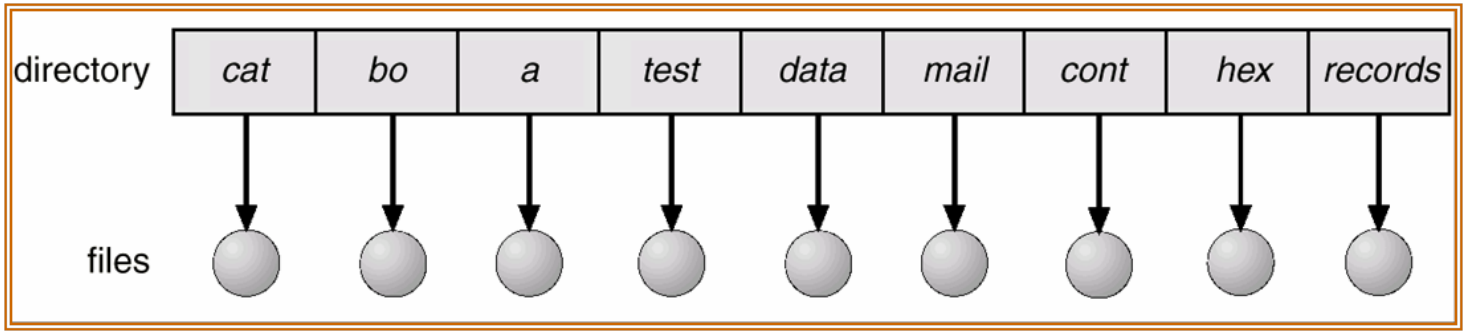


Single-level directory



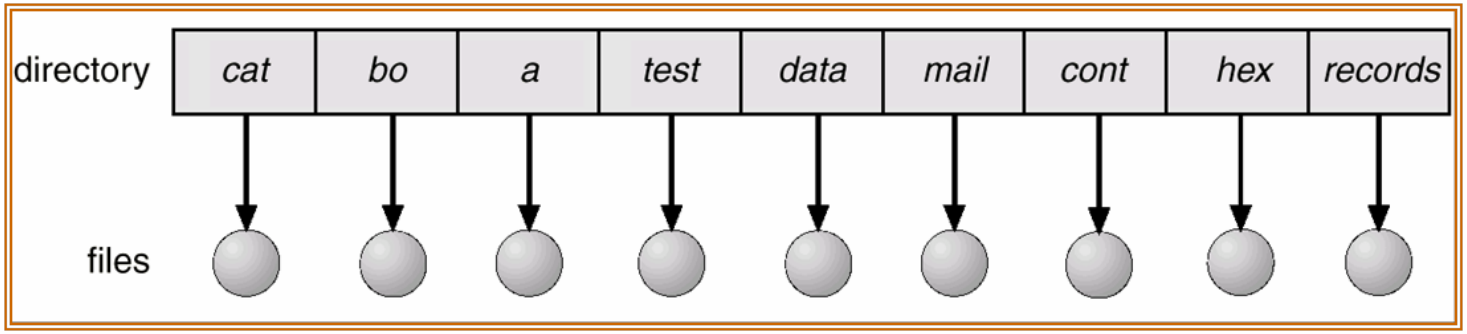
- Limitations

Single-level directory



- Limitations
  - Naming problem

Single-level directory



- Limitations
  - Naming problem
  - Grouping problem

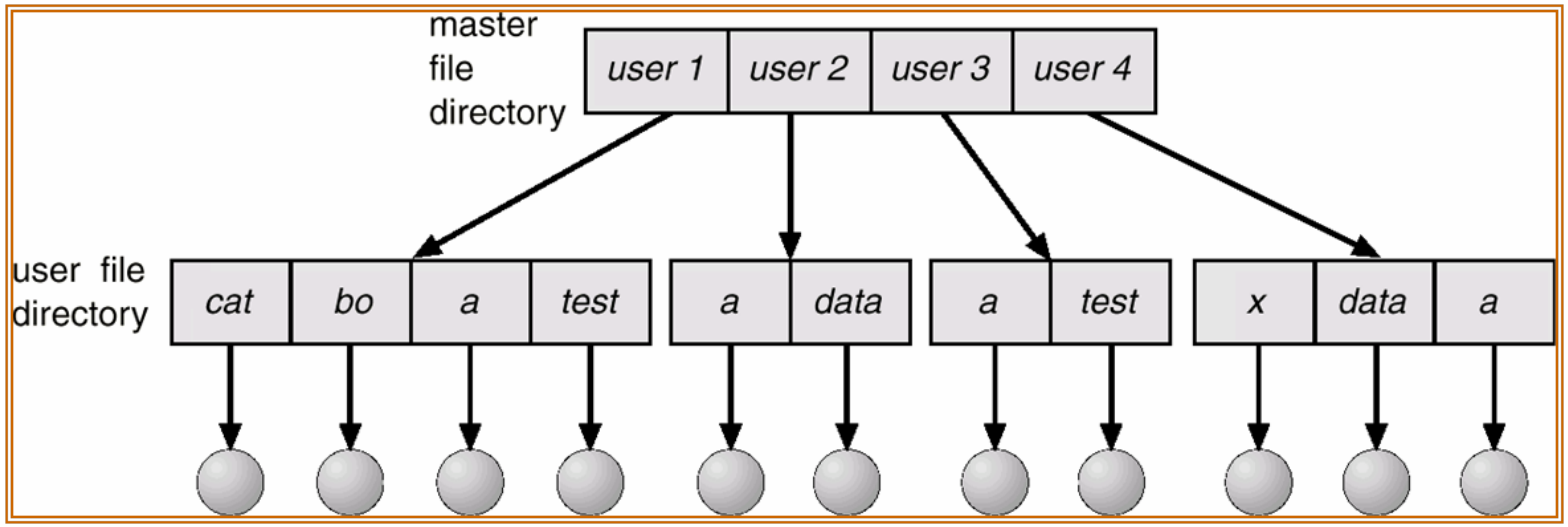
Two-level directory

Two-level directory

- Separate directory for each user

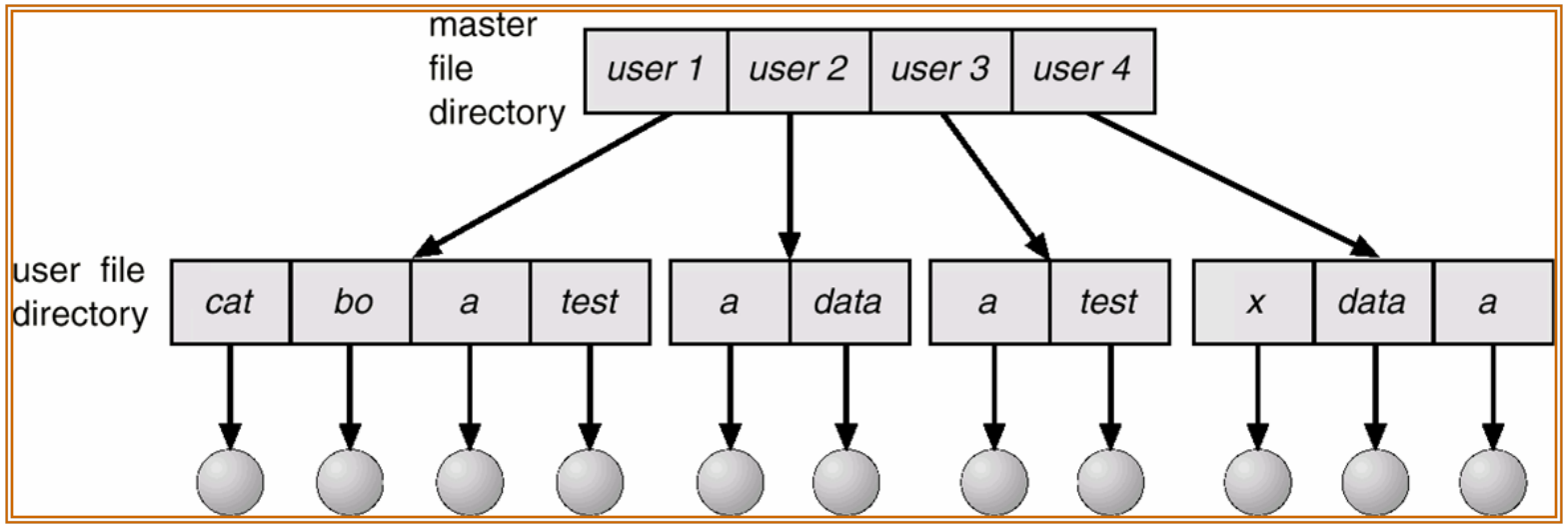
Two-level directory

- Separate directory for each user



Two-level directory

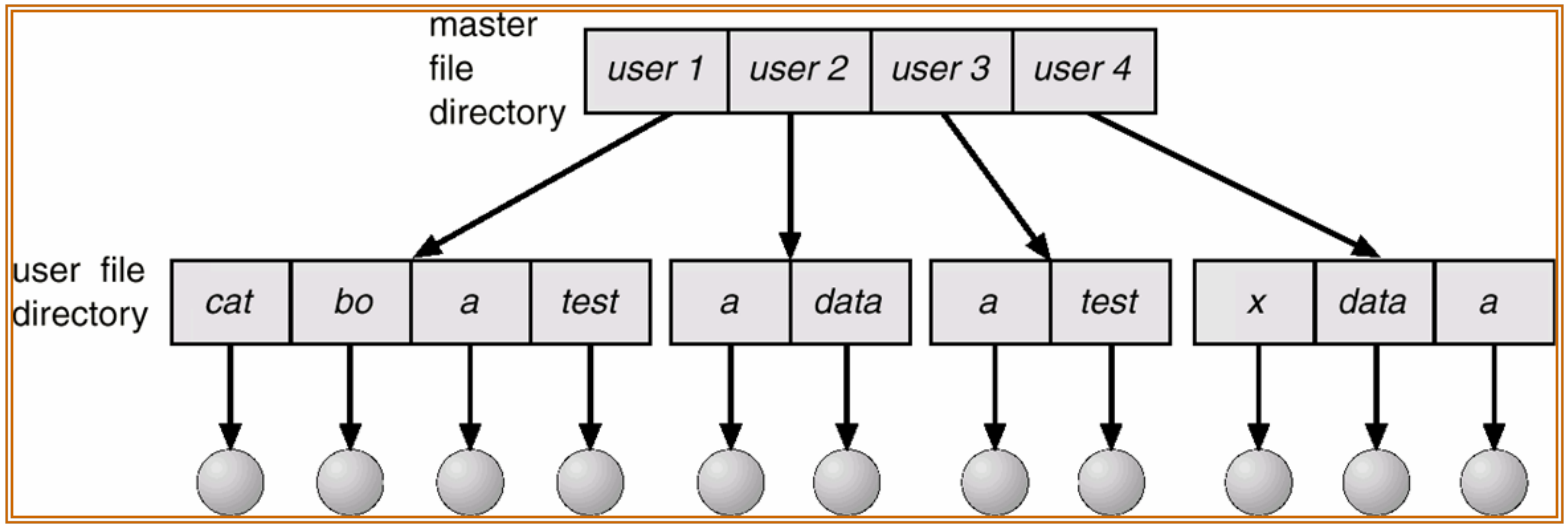
- Separate directory for each user



- Features

Two-level directory

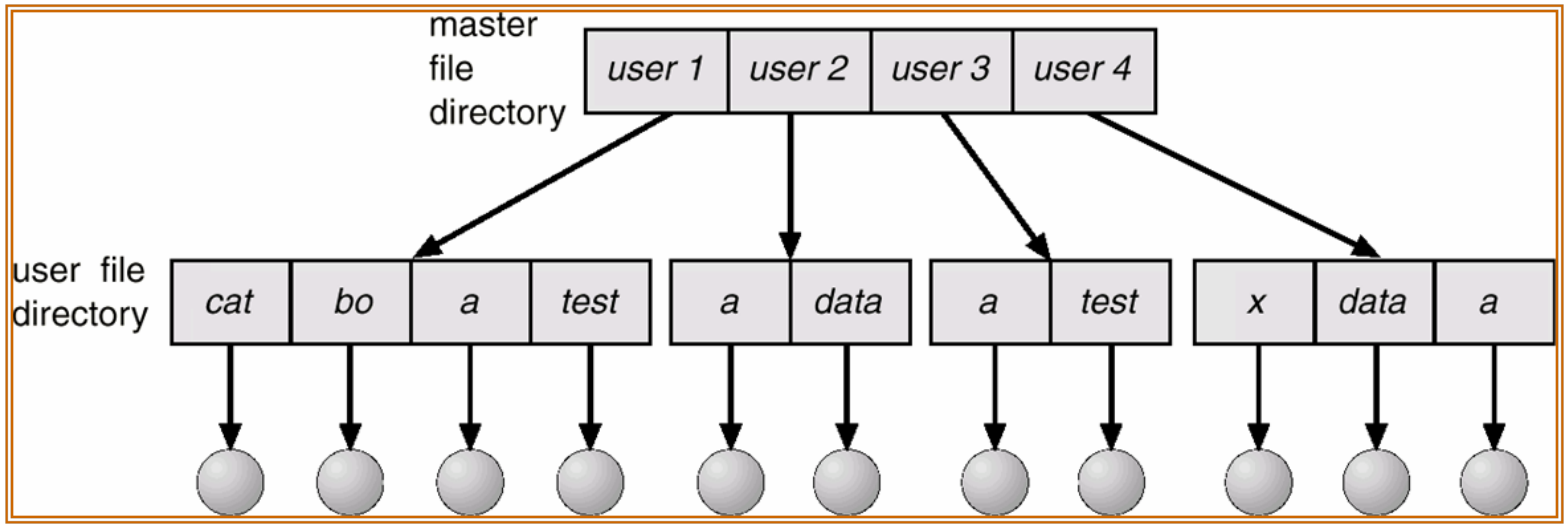
- Separate directory for each user



- Features
  - Files or directories can be located by its *path*

Two-level directory

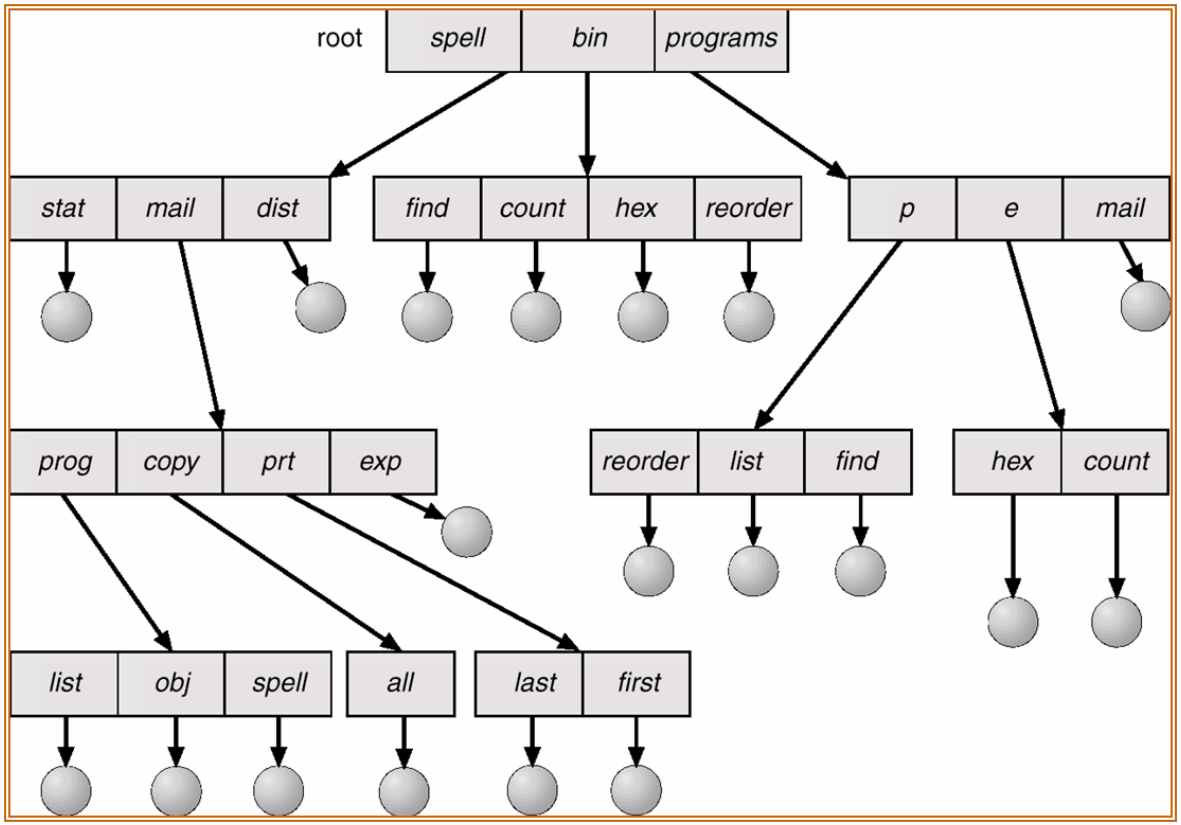
- Separate directory for each user



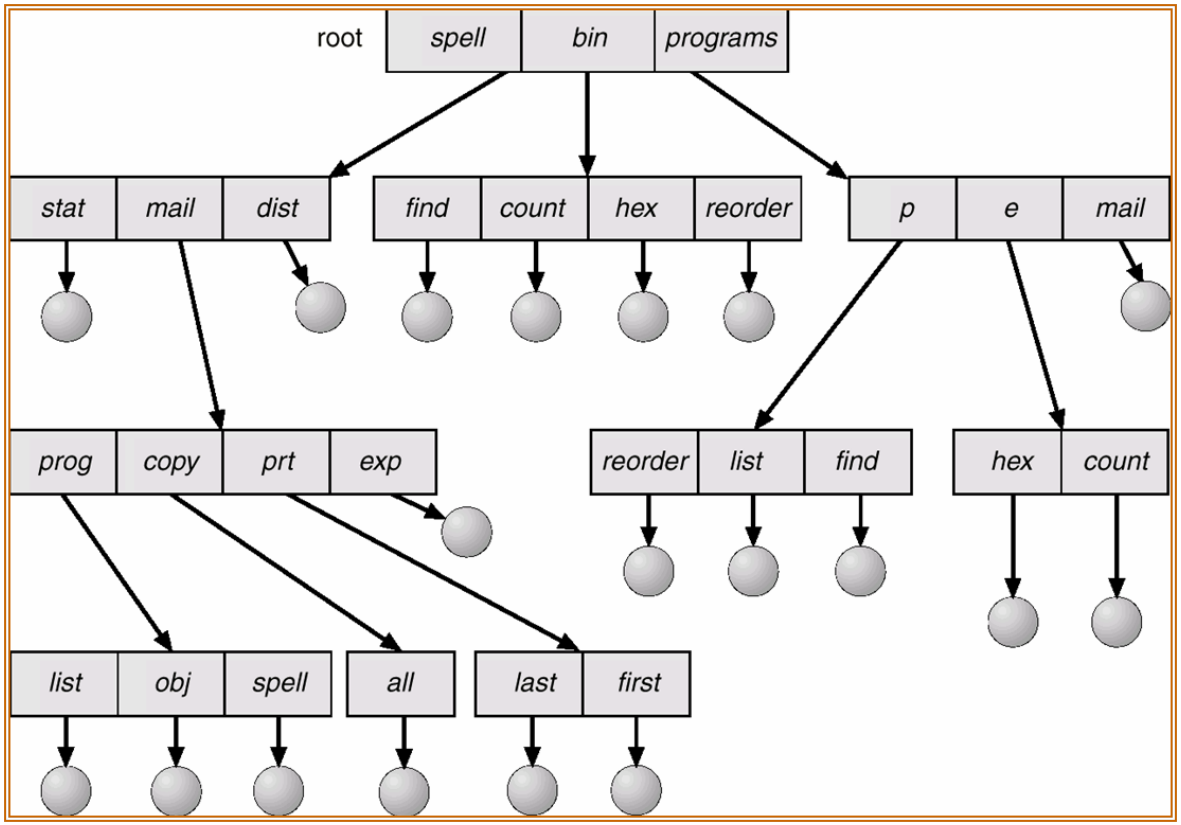
- Features
  - Files or directories can be located by its *path*
  - Can have the same file name for different users

Tree-structured directory

Tree-structured directory

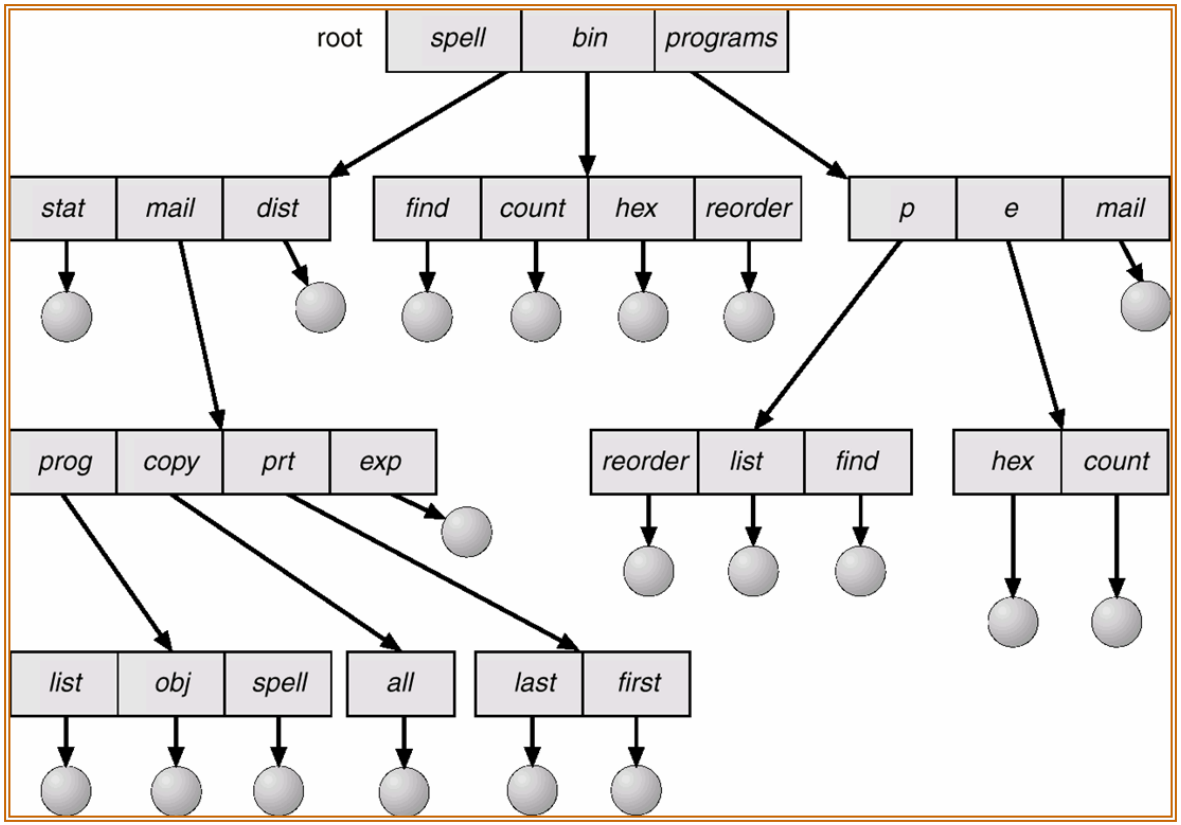


Tree-structured directory



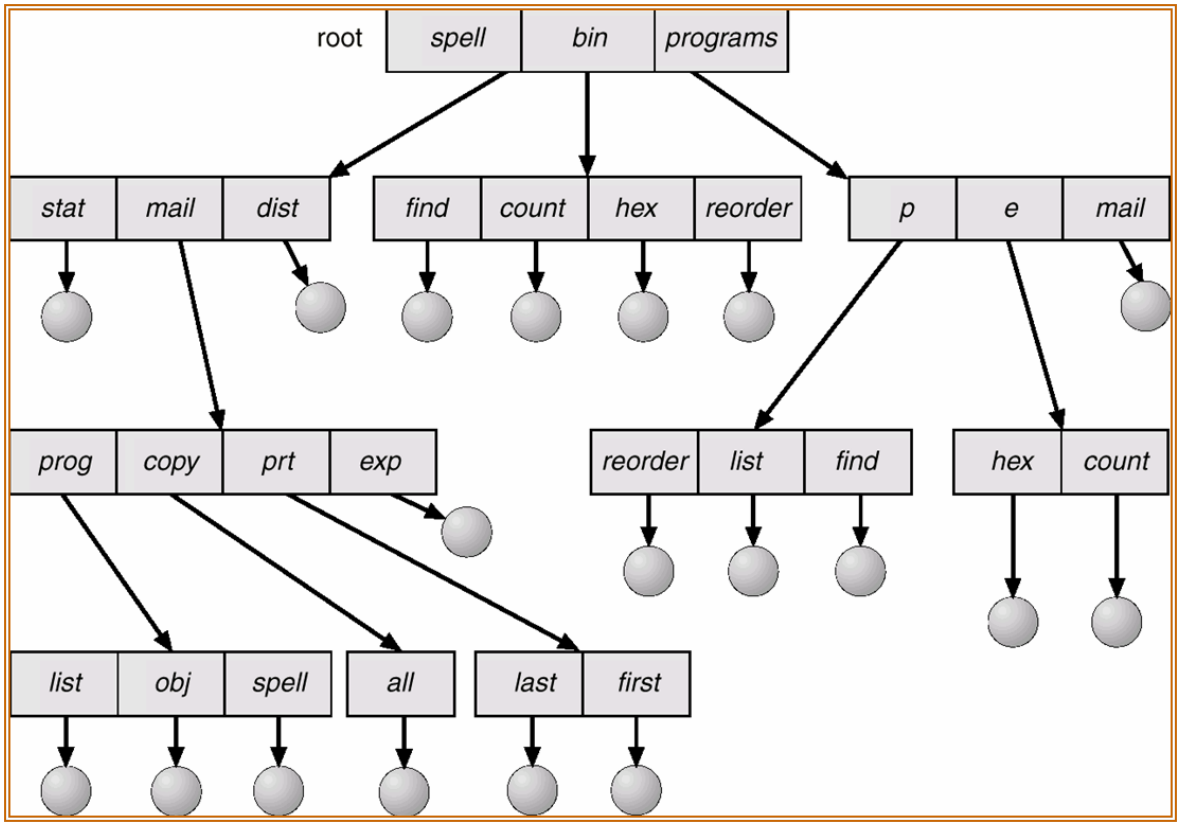
- Features

Tree-structured directory



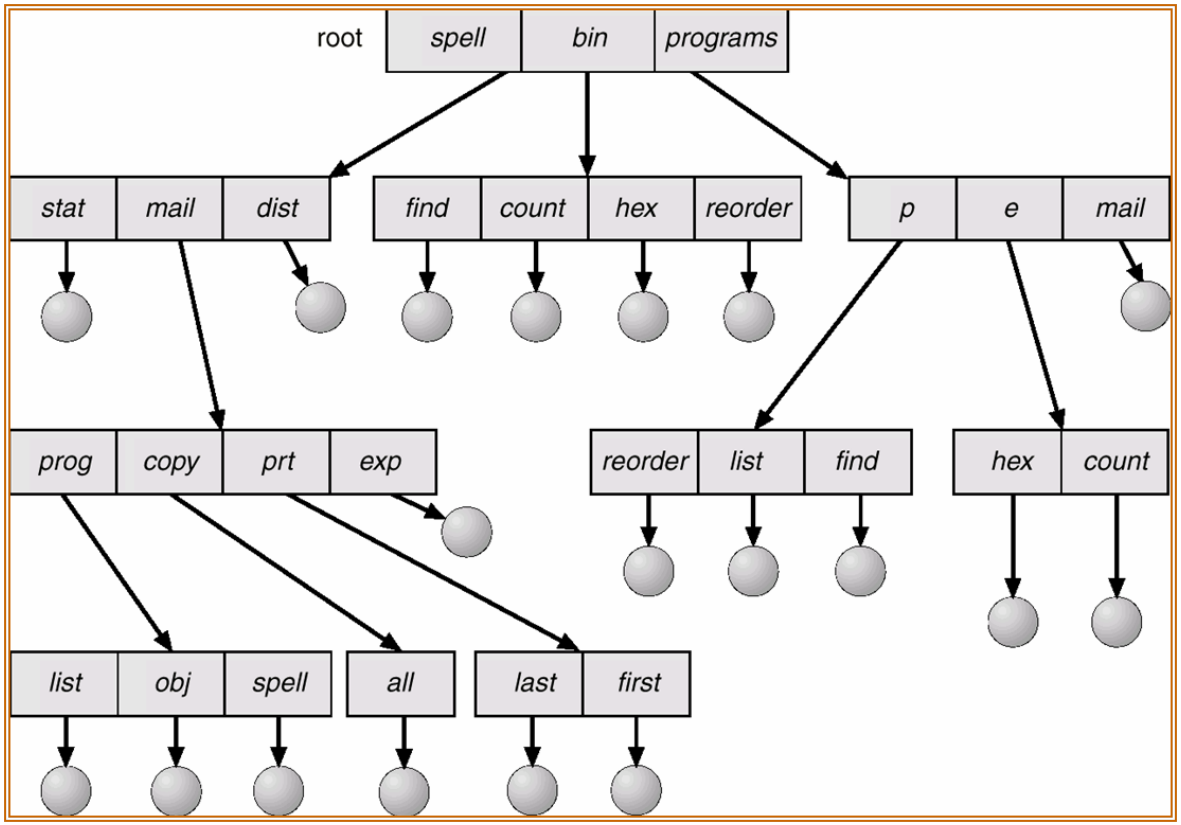
- Features
  - Grouping capability

Tree-structured directory



- Features
  - Grouping capability
  - *Absolute* or *relative* path

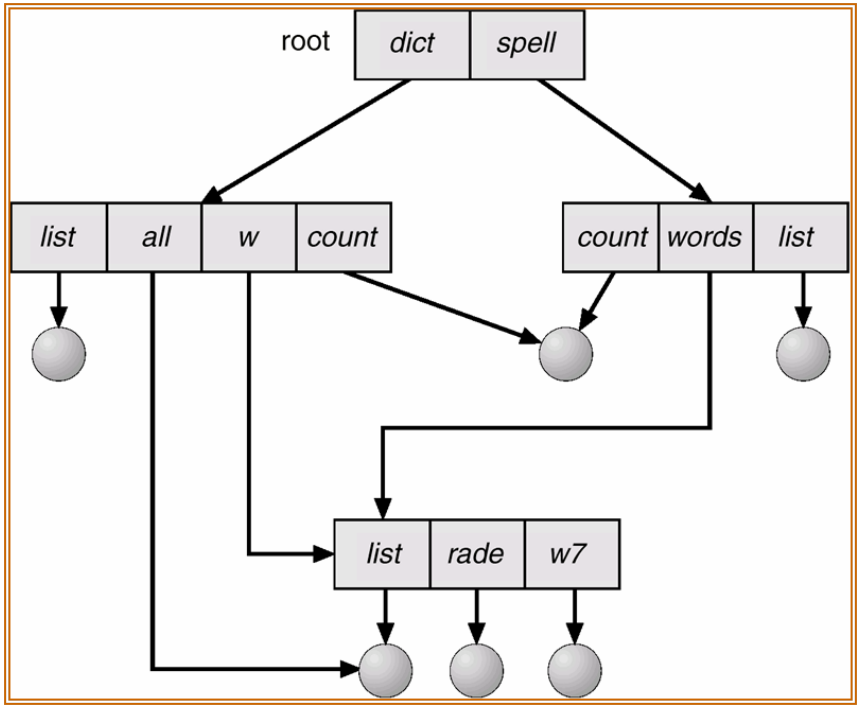
Tree-structured directory



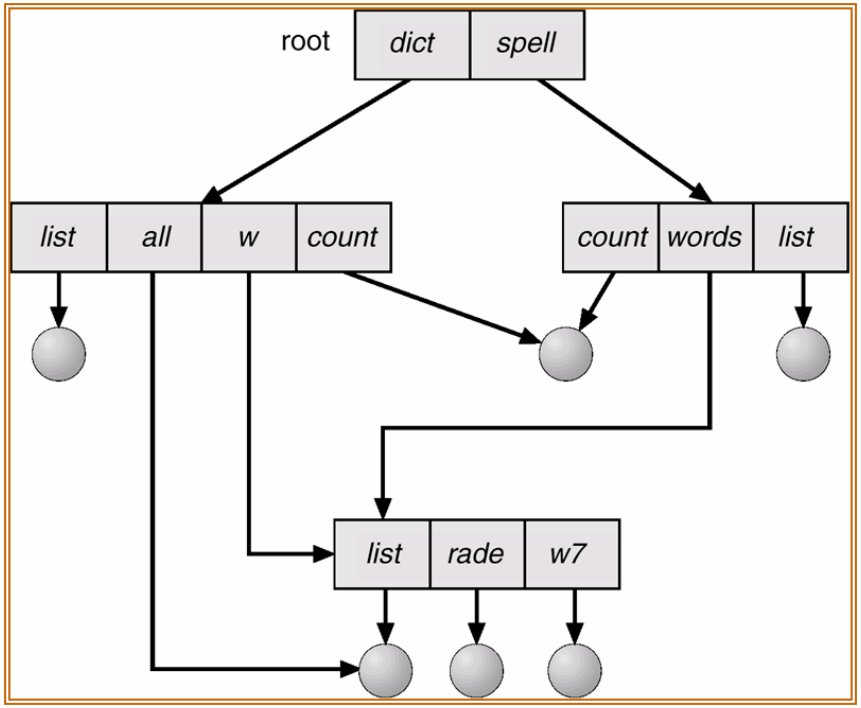
- Features
  - Grouping capability
  - *Absolute* or *relative* path
  - *Working* (*current*) directory of each process

Acyclic-graph directory

Acyclic-graph directory

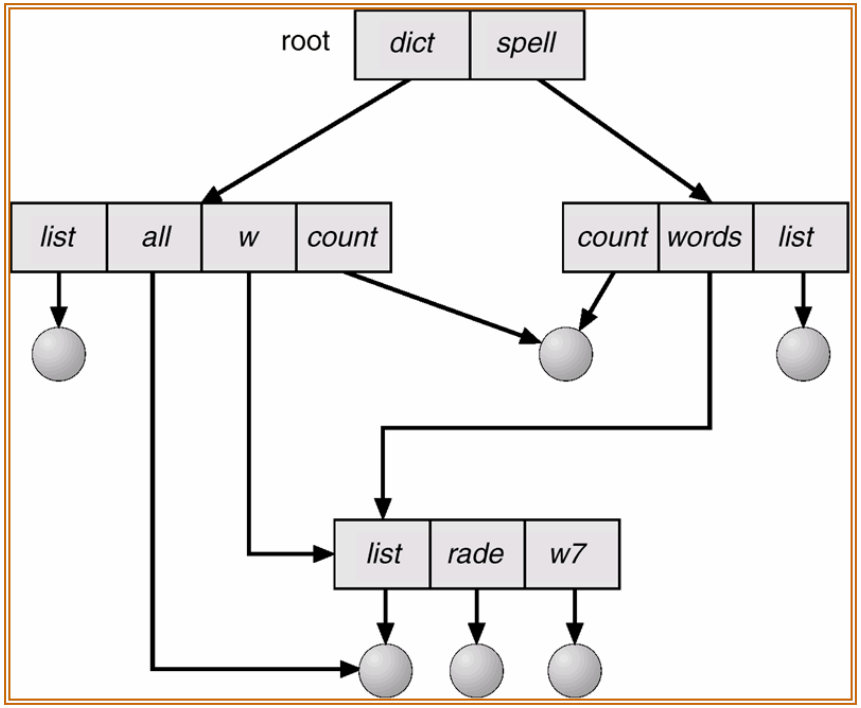


Acyclic-graph directory



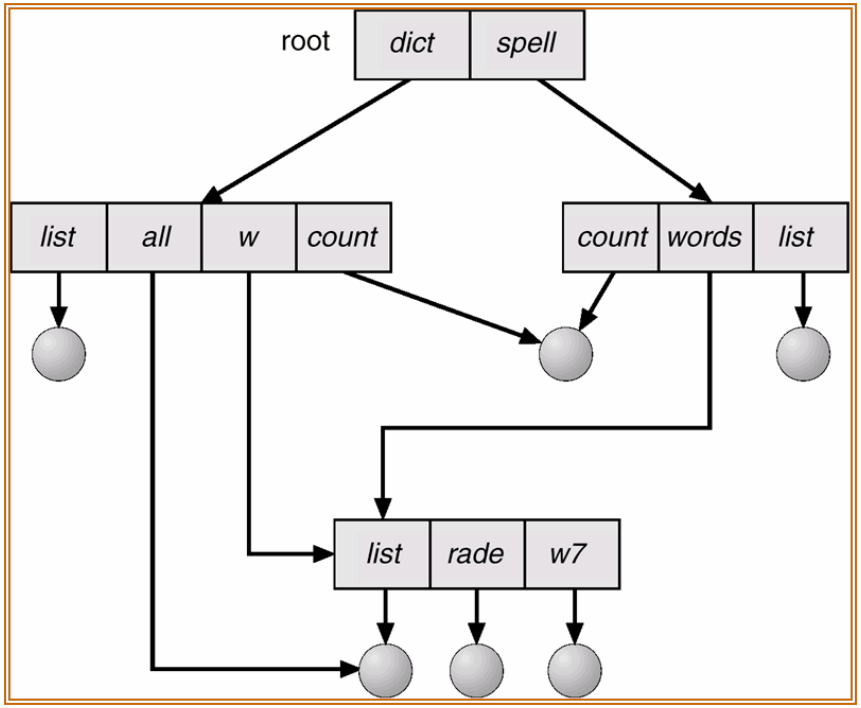
- Features

Acyclic-graph directory



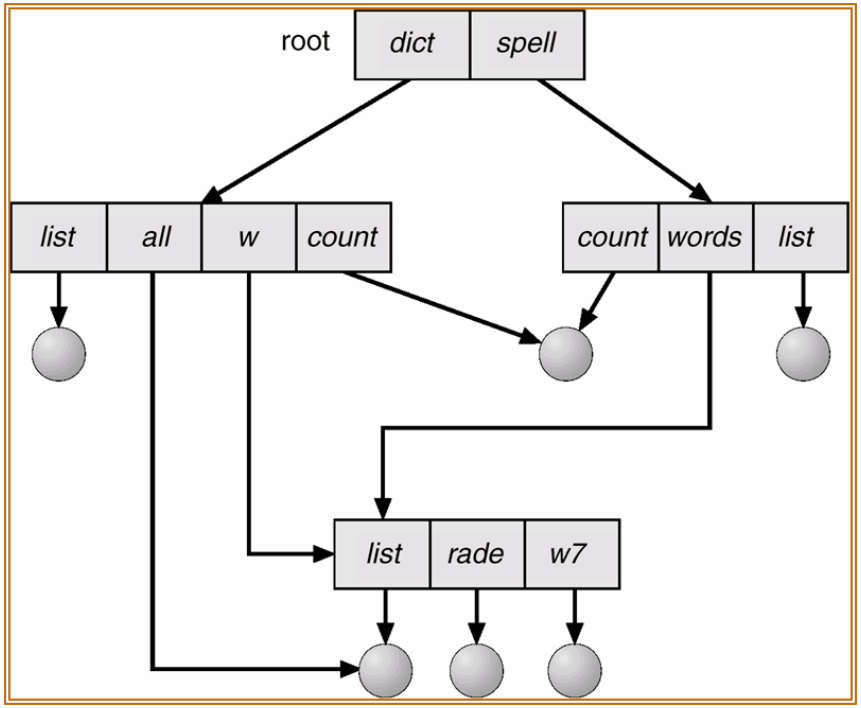
- Features
  - Two different names of the same file, i.e., *aliasing*

Acyclic-graph directory



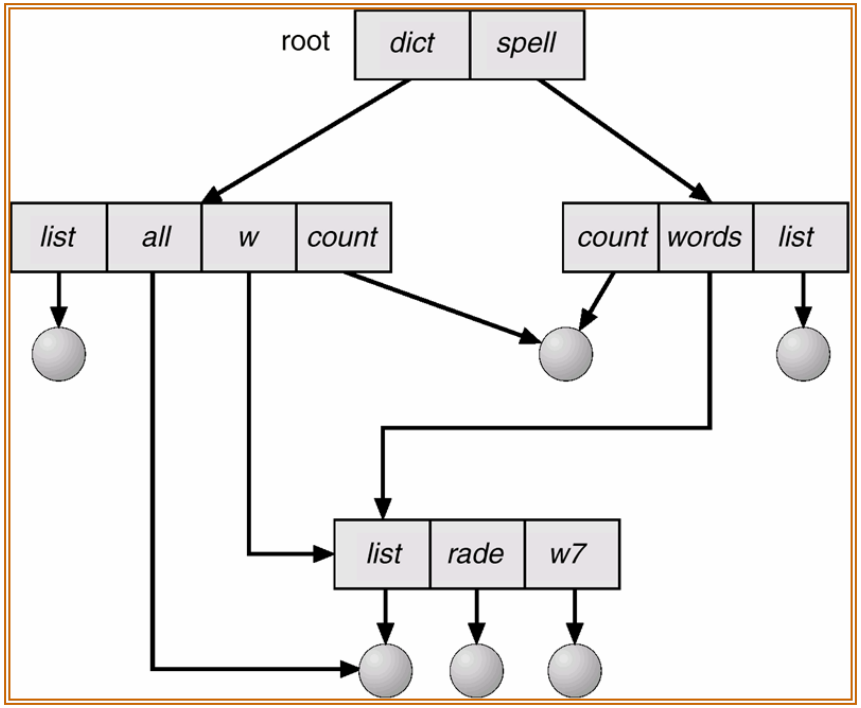
- Features
  - Two different names of the same file, i.e., *aliasing*
  - Dangling pointer problem

Acyclic-graph directory



- Features
  - Two different names of the same file, i.e., *aliasing*
  - Dangling pointer problem
    - Some operating systems do not support acyclic-graph directory, such as MS-DOS

Acyclic-graph directory



- Features
  - Two different names of the same file, i.e., *aliasing*
  - Dangling pointer problem
    - Some operating systems do not support acyclic-graph directory, such as MS-DOS
    - UNIX/LINUX and Windows (7+) support it via *symbol link*

Questions

- Any questions?

File system mounting (1/2)

File system mounting (1/2)

- A file system which resides on the device must be *mounted* to be accessed.

File system mounting (1/2)

- A file system which resides on the device must be *mounted* to be accessed.
  - The place where the file system is mounted is named *mount point*.

File system mounting (1/2)

- A file system which resides on the device must be *mounted* to be accessed.
  - The place where the file system is mounted is named *mount point*.
  - Typically, a mount point is an empty directory.

File system mounting (1/2)

- A file system which resides on the device must be *mounted* to be accessed.
  - The place where the file system is mounted is named *mount point*.
  - Typically, a mount point is an empty directory.
- For example, Windows operating systems mount any partitions which contain the FAT(-12, -16, -32) or NTFS file system at “C:”, “D:” and so on during the booting process.

File system mounting (1/2)

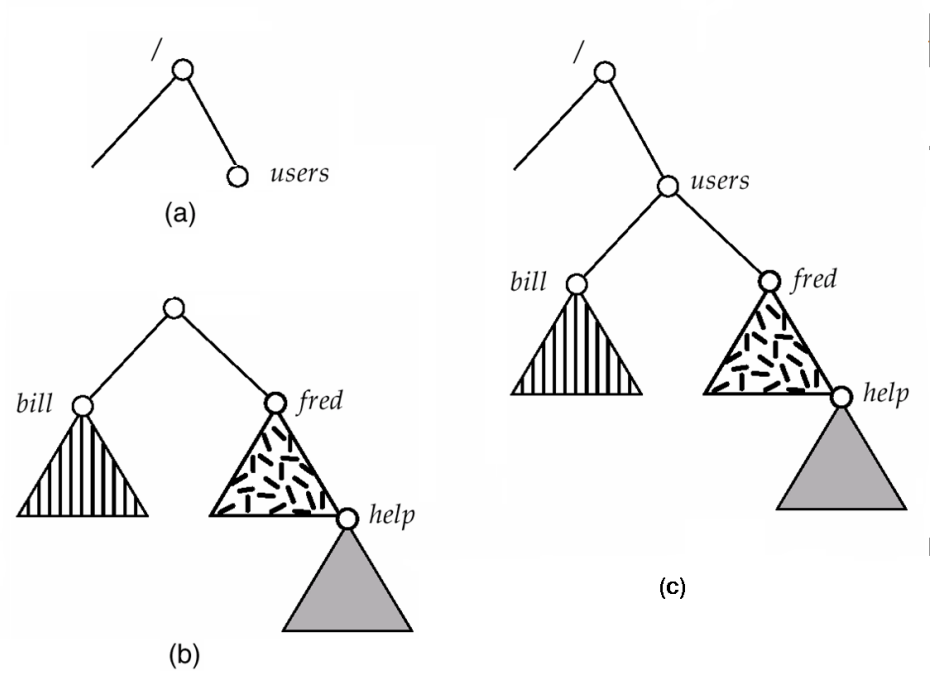
- A file system which resides on the device must be *mounted* to be accessed.
  - The place where the file system is mounted is named *mount point*.
  - Typically, a mount point is an empty directory.
- For example, Windows operating systems mount any partitions which contain the FAT(-12, -16, -32) or NTFS file system at “C:”, “D:” and so on during the booting process.
- While in the UNIX/Linux, system administrators must issue a command to mount a file system within a device.

File system mounting (1/2)

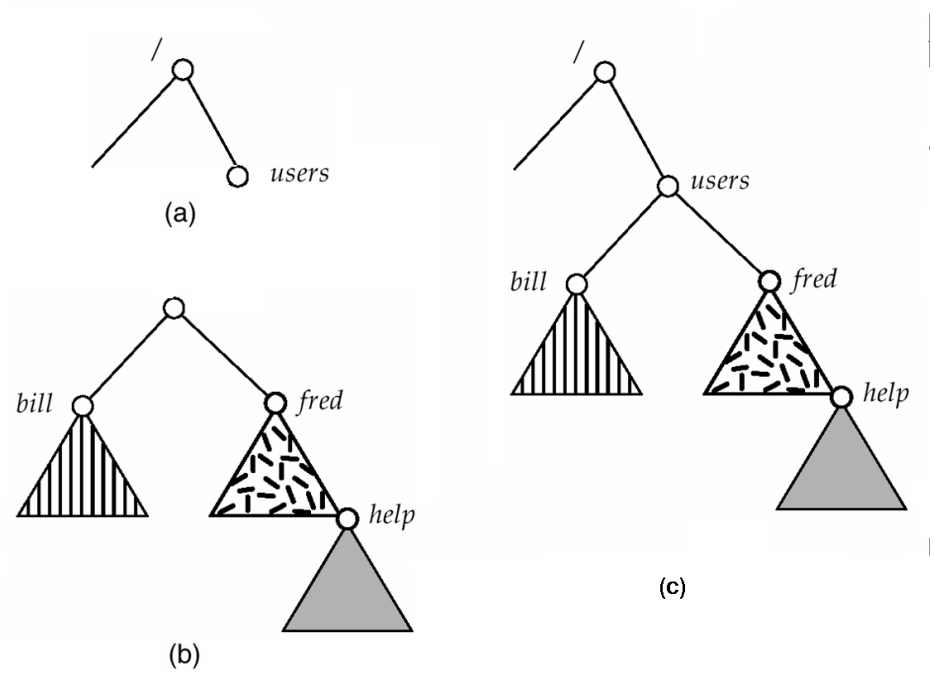
- A file system which resides on the device must be *mounted* to be accessed.
  - The place where the file system is mounted is named *mount point*.
  - Typically, a mount point is an empty directory.
- For example, Windows operating systems mount any partitions which contain the FAT(-12, -16, -32) or NTFS file system at “C:”, “D:” and so on during the booting process.
- While in the UNIX/Linux, system administrators must issue a command to mount a file system within a device.
  - *mount -t iso9660 /mnt/cdrom /dev/cdrom*

File system mounting (2/2)

File system mounting (2/2)

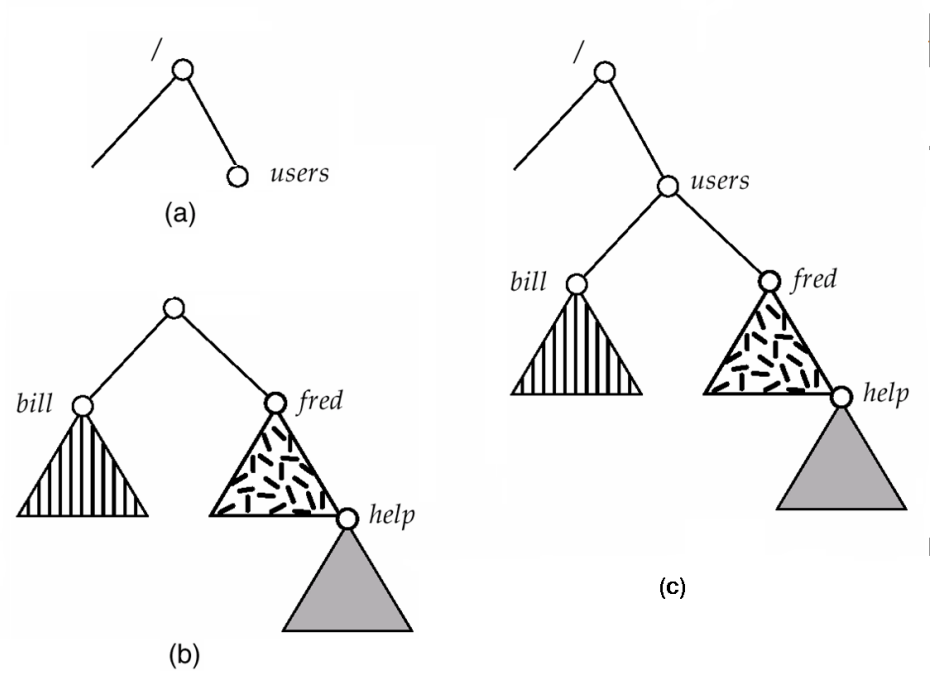


File system mounting (2/2)



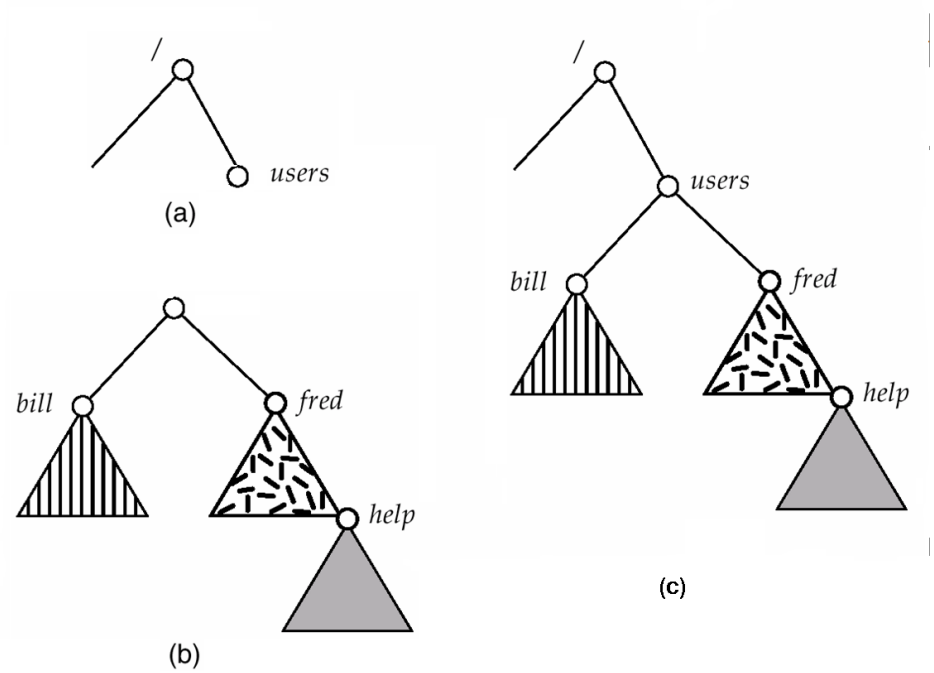
- Example

File system mounting (2/2)



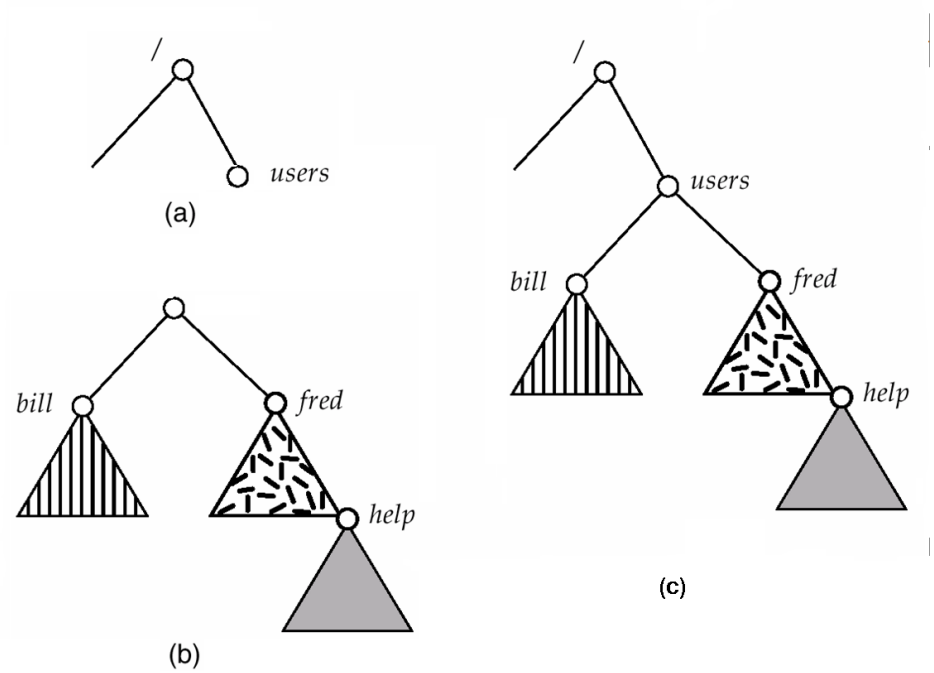
- Example
  - An existing mounted file system, figure (a)

File system mounting (2/2)



- Example
  - An existing mounted file system, figure (a)
  - An unmounted file system resides on a disk, figure (b)

File system mounting (2/2)



- Example
  - An existing mounted file system, figure (a)
  - An unmounted file system resides on a disk, figure (b)
  - After mounting the file system (b) on the directory of existing file system "/users", figure (c)

Questions

- Any questions?

File sharing

File sharing

- Sharing of files on multi-user systems is desirable.

File sharing

- Sharing of files on multi-user systems is desirable.
  - Most systems identify the users by its unique *user identification*, or *UID*

File sharing

- Sharing of files on multi-user systems is desirable.
  - Most systems identify the users by its unique *user identification*, or *UID*
  - In addition to UID, some systems also implement the *group* functionality

File sharing

- Sharing of files on multi-user systems is desirable.
  - Most systems identify the users by its unique *user identification*, or *UID*
  - In addition to UID, some systems also implement the *group* functionality
    - Each group is assigned an unique *group identification*, or *GID*

File sharing

- Sharing of files on multi-user systems is desirable.
  - Most systems identify the users by its unique *user identification*, or *UID*
  - In addition to UID, some systems also implement the *group* functionality
    - Each group is assigned an unique *group identification*, or *GID*
  - Every users can be in one or more groups

File sharing

- Sharing of files on multi-user systems is desirable.
  - Most systems identify the users by its unique *user identification*, or *UID*
  - In addition to UID, some systems also implement the *group* functionality
    - Each group is assigned an unique *group identification*, or *GID*
  - Every users can be in one or more groups
- When a file or directory is initially created, it is associated with the UID and GID of the user.

File sharing

- Sharing of files on multi-user systems is desirable.
  - Most systems identify the users by its unique *user identification*, or *UID*
  - In addition to UID, some systems also implement the *group* functionality
    - Each group is assigned an unique *group identification*, or *GID*
  - Every users can be in one or more groups
- When a file or directory is initially created, it is associated with the UID and GID of the user.
  - The user who owns a file is the *owner* of the file.

Protection

Protection

- File owner should be able to control

Protection

- File owner should be able to control
  - What can be done and

Protection

- File owner should be able to control
  - What can be done and
  - by whom

Protection

- File owner should be able to control
  - What can be done and
  - by whom
- Types of access

Protection

- File owner should be able to control
  - What can be done and
  - by whom
- Types of access
  - Read (R), Write (W), Execute (X), Append, Delete, List

UNIX/Linux file and directory protection

UNIX/Linux file and directory protection

- Three classes of users and their access rights
  - owner with her/his RWX

UNIX/Linux file and directory protection

- Three classes of users and their access rights
  - owner with her/his RWX
  - group with their RWX

UNIX/Linux file and directory protection

- Three classes of users and their access rights
  - owner with her/his RWX
  - group with their RWX
  - public with their RWX

UNIX/Linux file and directory protection

- Three classes of users and their access rights
  - owner with her/his RWX
  - group with their RWX
  - public with their RWX
- Example

UNIX/Linux file and directory protection

- Three classes of users and their access rights
  - owner with her/his RWX
  - group with their RWX
  - public with their RWX
- Example
  - *-rw-rw-r— 1 hmj devel 525 2007-04-27 23:52 vmmap.c*

Access-control list

Access-control list

- The *access-control list (ACL)* specifies the user name and the types of access allowed for each user.

Access-control list

- The *access-control list (ACL)* specifies the user name and the types of access allowed for each user.
  - It's used to enforce fine-grained file and directory protection.

Access-control list

- The *access-control list (ACL)* specifies the user name and the types of access allowed for each user.
  - It's used to enforce fine-grained file and directory protection.
- The main problem with ACL is their length.

Access-control list

- The *access-control list (ACL)* specifies the user name and the types of access allowed for each user.
  - It's used to enforce fine-grained file and directory protection.
- The main problem with ACL is their length.
  - So, the most common approach is to combine the UNIX-style protection with the ACL.

Access-control list

- The *access-control list (ACL)* specifies the user name and the types of access allowed for each user.
  - It's used to enforce fine-grained file and directory protection.
- The main problem with ACL is their length.
  - So, the most common approach is to combine the UNIX-style protection with the ACL.
  - For example, Windows NT or later and Solaris 2.6 or later use this combined approach.

Questions

- Any questions?

Outline

File system structure

File system structure

- File

File system structure

- File
  - Logical storage unit

File system structure

- File
  - Logical storage unit
  - Collection of related information

File system structure

- File
  - Logical storage unit
  - Collection of related information
  - *File Control Block (FCB)* contains all meta-information of a file

File system structure

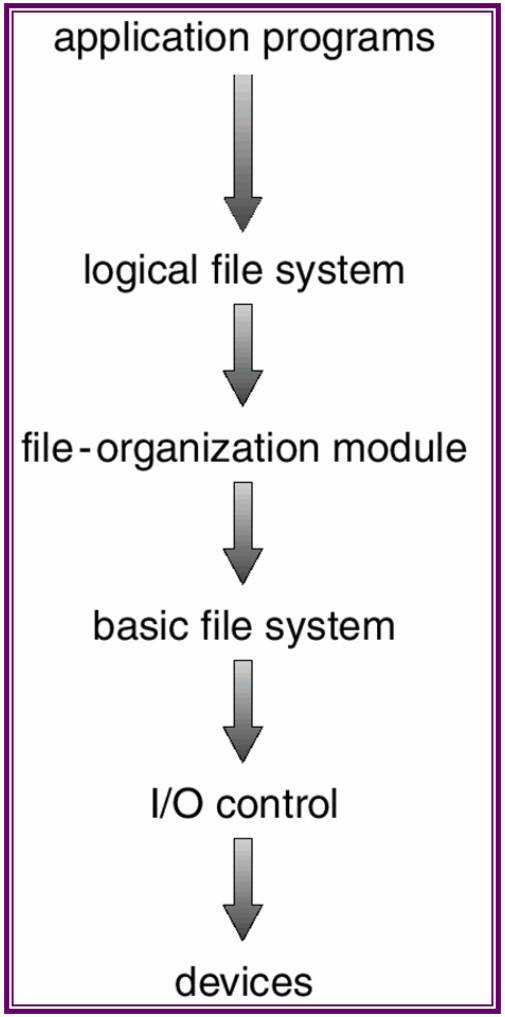
- File
  - Logical storage unit
  - Collection of related information
  - *File Control Block (FCB)* contains all meta-information of a file
    - In UNIX, FCB is usually called *inode*

File system structure

- File
  - Logical storage unit
  - Collection of related information
  - *File Control Block (FCB)* contains all meta-information of a file
    - In UNIX, FCB is usually called *inode*
- A file system resides on the secondary storages and is organized into layers

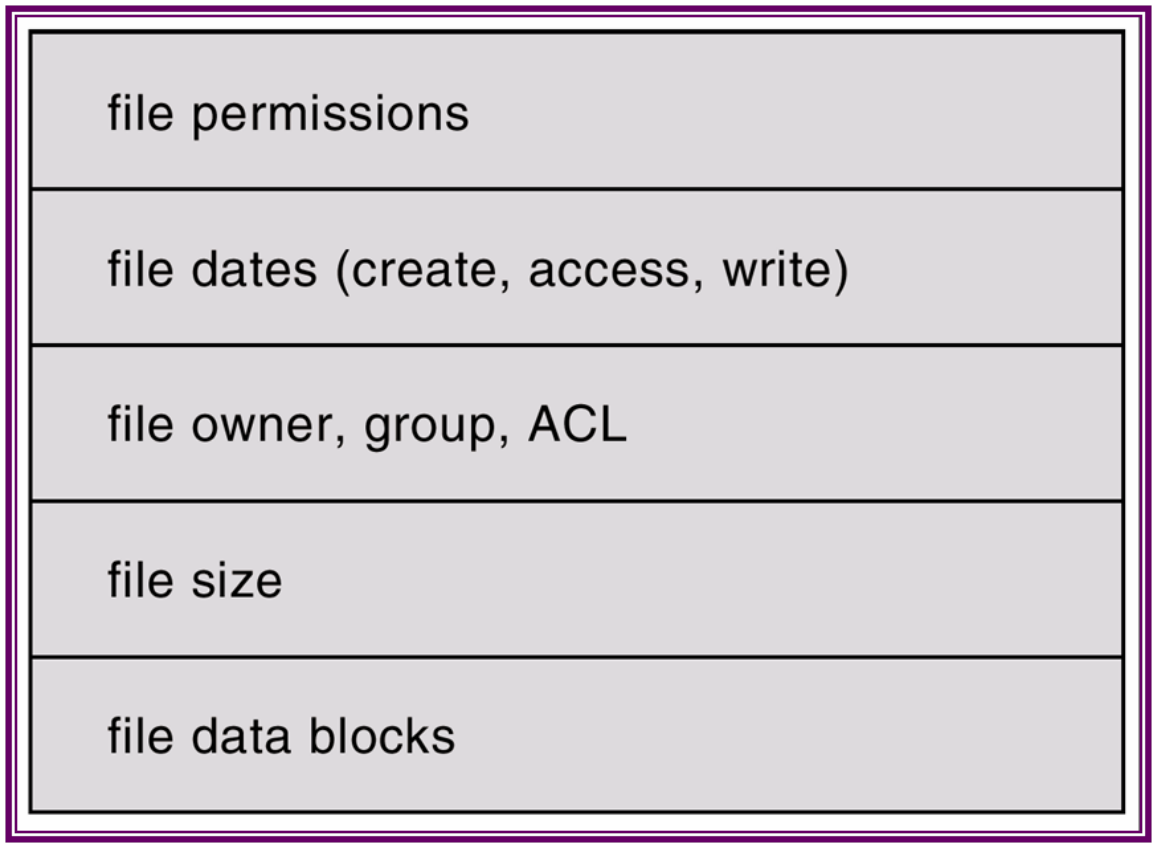
Layered File system

Layered File system



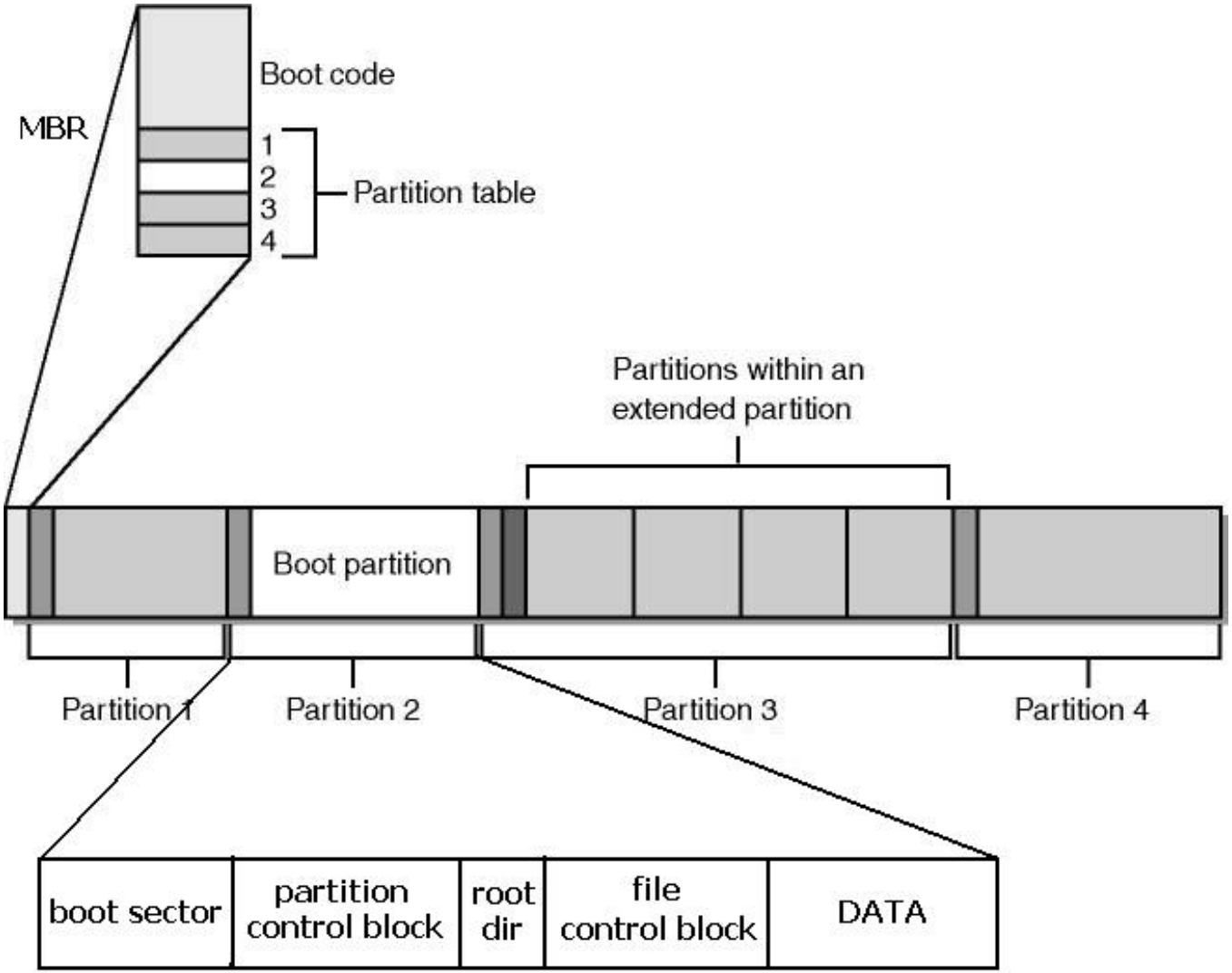
A typical file control block

A typical file control block



On-disk file system structure (1/2)

On-disk file system structure (1/2)



On-disk file system structure (2/2)

On-disk file system structure (2/2)

- The “boot sector” contains codes used to load the operating system kernel when booting;

On-disk file system structure (2/2)

- The “boot sector” contains codes used to load the operating system kernel when booting;
  - It’s also known as the “boot block” and loaded by the code within the MBR.

On-disk file system structure (2/2)

- The "boot sector" contains codes used to load the operating system kernel when booting;
  - It's also known as the "boot block" and loaded by the code within the MBR.
- The "partition control block" contains the partition details such as number of blocks in the partition, size of blocks, free-block count and free-block pointers, and free FCB count and free FCB pointers.

On-disk file system structure (2/2)

- The "boot sector" contains codes used to load the operating system kernel when booting;
  - It's also known as the "boot block" and loaded by the code within the MBR.
- The "partition control block" contains the partition details such as number of blocks in the partition, size of blocks, free-block count and free-block pointers, and free FCB count and free FCB pointers.
  - It's also known as "superblock" in UNIX or "Master File Table" in Windows NT.

In-memory file system structure

In-memory file system structure

- An in-memory partition table containing information about each mounted partition

In-memory file system structure

- An in-memory partition table containing information about each mounted partition
- An in-memory directory structure that holds the directory information of recently accessed directories

In-memory file system structure

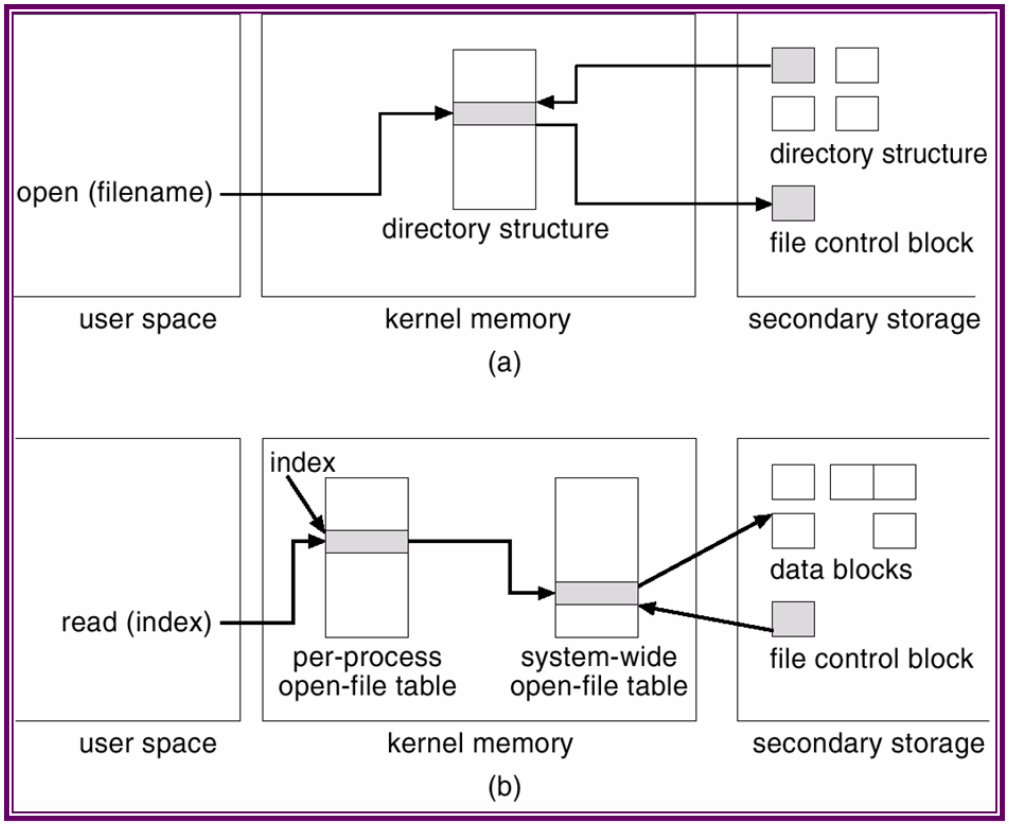
- An in-memory partition table containing information about each mounted partition
- An in-memory directory structure that holds the directory information of recently accessed directories
- The *system-wide open-file table* contains a copy of FCB of each open file, as well as other information

In-memory file system structure

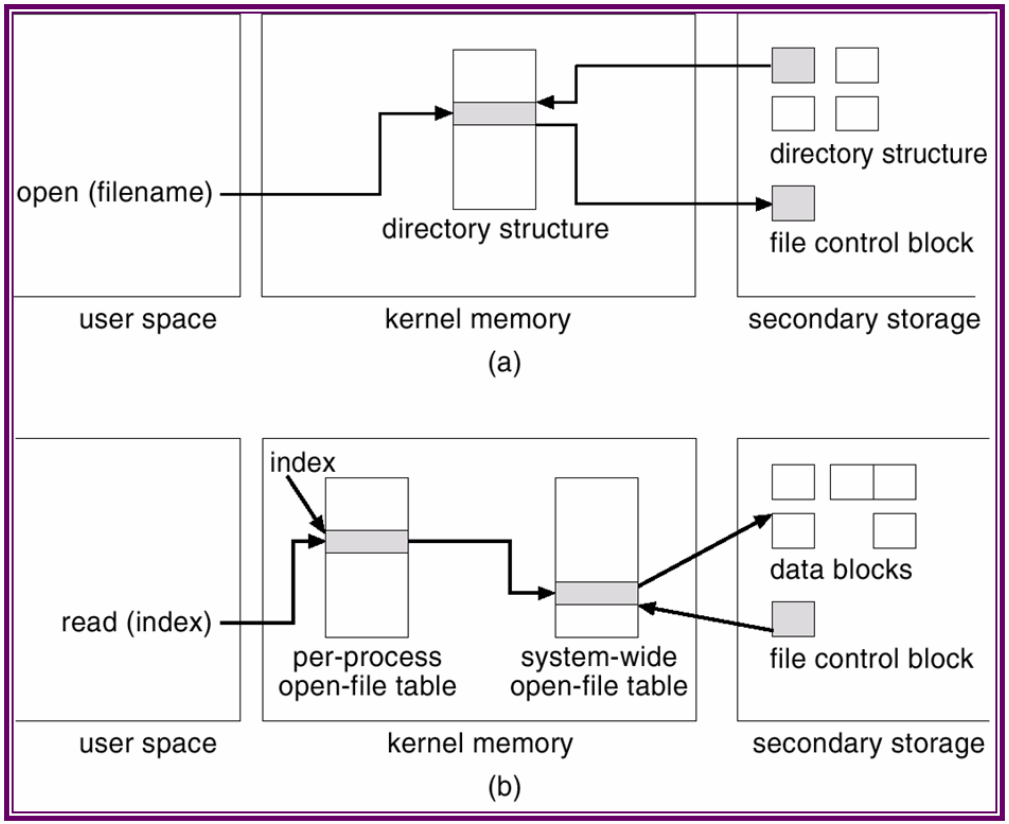
- An in-memory partition table containing information about each mounted partition
- An in-memory directory structure that holds the directory information of recently accessed directories
- The *system-wide open-file table* contains a copy of FCB of each open file, as well as other information
- The *per-process open-file table* contains a pointer to the appropriate entry in the system-wide open-file table, as well as other information

Open-file table

Open-file table

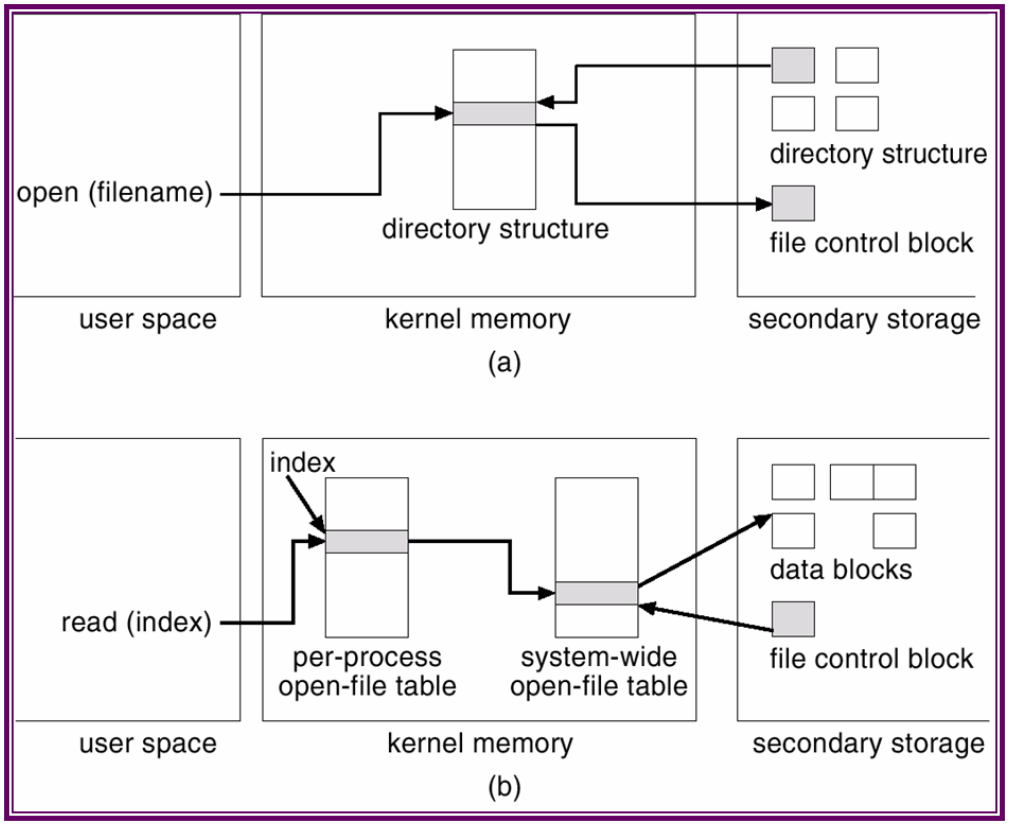


Open-file table



- The "index" in the above figure is known as *file descriptor* in UNIX or *file handle* in Windows NT.

Open-file table



- The "index" in the above figure is known as *file descriptor* in UNIX or *file handle* in Windows NT.
  - It's the value returned by the system call "open" in UNIX or "CreateFile" in Windows NT

Questions

- Any questions?

Virtual File System (VFS)

Virtual File System (VFS)

- Why VFS?

Virtual File System (VFS)

- Why VFS?
  - The types of file system are as many as the flavors of ice cream.

Virtual File System (VFS)

- Why VFS?
  - The types of file system are as many as the flavors of ice cream.
  - How does an operating system allow multiple types of file systems to be integrated into one directory structure?

Virtual File System (VFS)

- Why VFS?
  - The types of file system are as many as the flavors of ice cream.
  - How does an operating system allow multiple types of file systems to be integrated into one directory structure?
  - How can users seamlessly move between file system types as they navigate a directory?

Virtual File System (VFS)

- Why VFS?
  - The types of file system are as many as the flavors of ice cream.
  - How does an operating system allow multiple types of file systems to be integrated into one directory structure?
  - How can users seamlessly move between file system types as they navigate a directory?
- Most operating systems use object-oriented techniques to simplify, organize and modularize the implementation

Virtual File System (VFS)

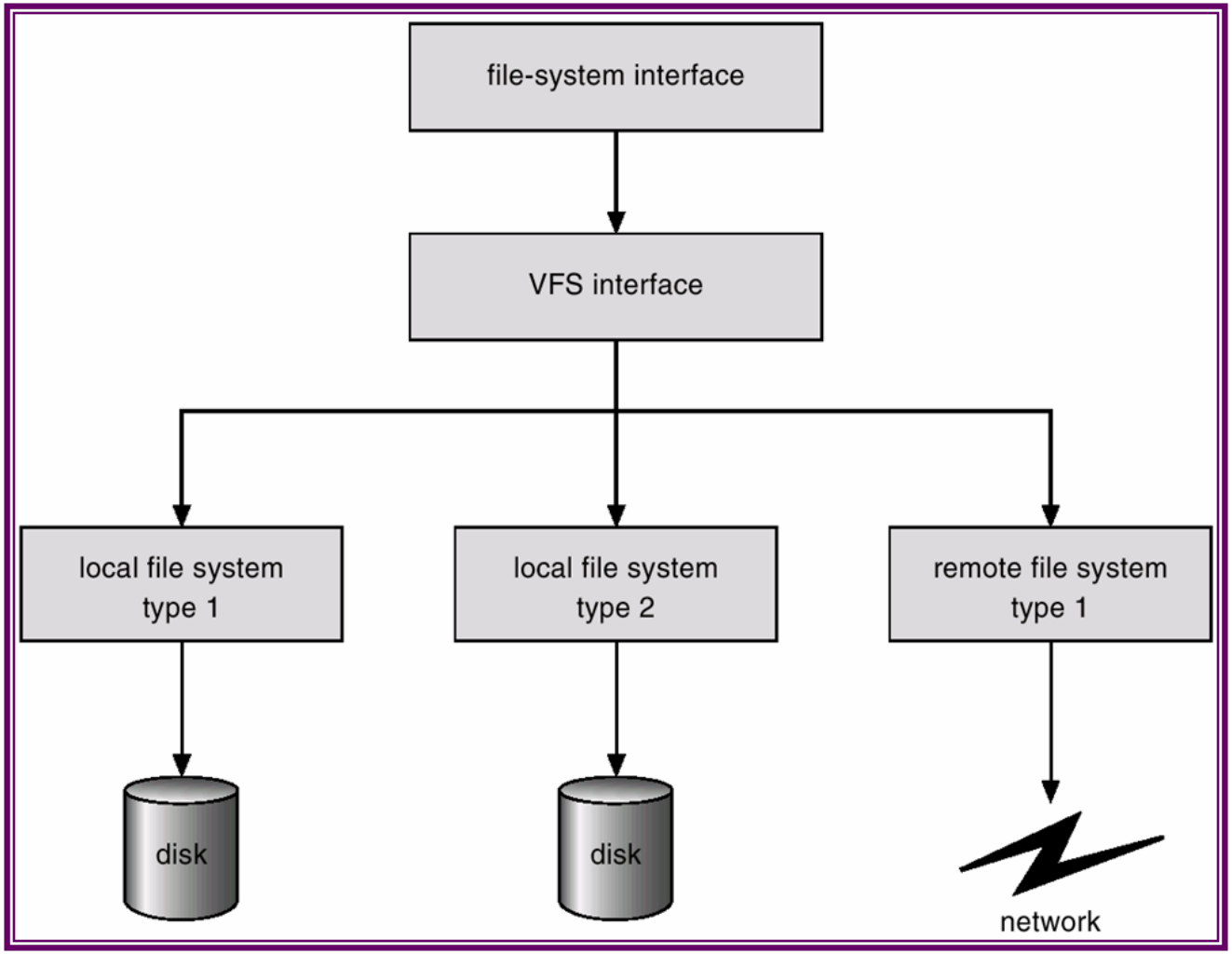
- Why VFS?
  - The types of file system are as many as the flavors of ice cream.
  - How does an operating system allow multiple types of file systems to be integrated into one directory structure?
  - How can users seamlessly move between file system types as they navigate a directory?
- Most operating systems use object-oriented techniques to simplify, organize and modularize the implementation
  - A common file system interface is separated from the file system implementations

Virtual File System (VFS)

- Why VFS?
  - The types of file system are as many as the flavors of ice cream.
  - How does an operating system allow multiple types of file systems to be integrated into one directory structure?
  - How can users seamlessly move between file system types as they navigate a directory?
- Most operating systems use object-oriented techniques to simplify, organize and modularize the implementation
  - A common file system interface is separated from the file system implementations
  - The file system interface contains the system calls such as “open”, “close”, “read”, “write” and “seek”, etc.

Schematic view of a VFS (1/2)

Schematic view of a VFS (1/2)



Schematic view of a VFS (2/2)

Schematic view of a VFS (2/2)

- The “VFS interface” serves two important functions:

Schematic view of a VFS (2/2)

- The "VFS interface" serves two important functions:
  - It separates file-system-generic operations from their implementation by defining a clean VFS interface

Schematic view of a VFS (2/2)

- The "VFS interface" serves two important functions:
  - It separates file-system-generic operations from their implementation by defining a clean VFS interface
  - The VFS is based on a file-representation structure, called a *vnode*, that contains a numerical designator for a network-wide unique file

Questions

- Any questions?

Directory implementation

Directory implementation

- Some operating systems, including UNIX, treat a directory exactly as a file

Directory implementation

- Some operating systems, including UNIX, treat a directory exactly as a file
  - It holds two pieces of information for each file or sub-directory within it: *file/subdir name* and *a pointer to the FCB of the file/subdir* (they are usually organized into a C "struct dirent")

Directory implementation

- Some operating systems, including UNIX, treat a directory exactly as a file
  - It holds two pieces of information for each file or sub-directory within it: *file/subdir name* and *a pointer to the FCB of the file/subdir* (they are usually organized into a C "struct dirent")
- A directory may contain a lot of files or subdirectories, how to organize these "dirent"?

Directory implementation

- Some operating systems, including UNIX, treat a directory exactly as a file
  - It holds two pieces of information for each file or sub-directory within it: *file/subdir name* and *a pointer to the FCB of the file/subdir* (they are usually organized into a C "struct dirent")
- A directory may contain a lot of files or subdirectories, how to organize these "dirent"?
  1. Linear list

Directory implementation

- Some operating systems, including UNIX, treat a directory exactly as a file
  - It holds two pieces of information for each file or sub-directory within it: *file/subdir name* and *a pointer to the FCB of the file/subdir* (they are usually organized into a C "struct dirent")
- A directory may contain a lot of files or subdirectories, how to organize these "dirent"?
  1. Linear list
     - Simple to program

Directory implementation

- Some operating systems, including UNIX, treat a directory exactly as a file
  - It holds two pieces of information for each file or sub-directory within it: *file/subdir name* and *a pointer to the FCB of the file/subdir* (they are usually organized into a C "struct dirent")
- A directory may contain a lot of files or subdirectories, how to organize these "dirent"?
  1. Linear list
     - Simple to program
     - Time-consuming to search

Directory implementation

- Some operating systems, including UNIX, treat a directory exactly as a file
  - It holds two pieces of information for each file or sub-directory within it: *file/subdir name* and *a pointer to the FCB of the file/subdir* (they are usually organized into a C "struct dirent")
- A directory may contain a lot of files or subdirectories, how to organize these "dirent"?
  1. Linear list
     - Simple to program
     - Time-consuming to search
  2. Hash table

Directory implementation

- Some operating systems, including UNIX, treat a directory exactly as a file
  - It holds two pieces of information for each file or sub-directory within it: *file/subdir name* and *a pointer to the FCB of the file/subdir* (they are usually organized into a C "struct dirent")
- A directory may contain a lot of files or subdirectories, how to organize these "dirent"?
  1. Linear list
     - Simple to program
     - Time-consuming to search
  2. Hash table
     - Decrease search time

Directory implementation

- Some operating systems, including UNIX, treat a directory exactly as a file
  - It holds two pieces of information for each file or sub-directory within it: *file/subdir name* and *a pointer to the FCB of the file/subdir* (they are usually organized into a C "struct dirent")
- A directory may contain a lot of files or subdirectories, how to organize these "dirent"?
  1. Linear list
     - Simple to program
     - Time-consuming to search
  2. Hash table
     - Decrease search time
     - Difficult to grow

Directory implementation

- Some operating systems, including UNIX, treat a directory exactly as a file
  - It holds two pieces of information for each file or sub-directory within it: *file/subdir name* and *a pointer to the FCB of the file/subdir* (they are usually organized into a C "struct dirent")
- A directory may contain a lot of files or subdirectories, how to organize these "dirent"?
  1. Linear list
     - Simple to program
     - Time-consuming to search
  2. Hash table
     - Decrease search time
     - Difficult to grow
- Most operating systems use the "Linear list" to organize a directory.

Questions

- Any questions?

Allocation methods

Allocation methods

- How to allocate space to files so that disk space is utilized effectively and files can be accessed quickly?

Allocation methods

- How to allocate space to files so that disk space is utilized effectively and files can be accessed quickly?
- Three major methods of allocating

Allocation methods

- How to allocate space to files so that disk space is utilized effectively and files can be accessed quickly?
- Three major methods of allocating
  1. Contiguous allocation

Allocation methods

- How to allocate space to files so that disk space is utilized effectively and files can be accessed quickly?
- Three major methods of allocating
  1. Contiguous allocation
  2. Linked allocation

Allocation methods

- How to allocate space to files so that disk space is utilized effectively and files can be accessed quickly?
- Three major methods of allocating
  1. Contiguous allocation
  2. Linked allocation
  3. Indexed allocation

Contiguous Allocation

Contiguous Allocation

- Each file occupies a set of contiguous blocks on the disk

Contiguous Allocation

- Each file occupies a set of contiguous blocks on the disk
- Pros and cons

Contiguous Allocation

- Each file occupies a set of contiguous blocks on the disk
- Pros and cons
  - Simple: only starting location (block #) and length (number of blocks) are saved in FCB

Contiguous Allocation

- Each file occupies a set of contiguous blocks on the disk
- Pros and cons
  - Simple: only starting location (block #) and length (number of blocks) are saved in FCB
  - Support random access and cache friendly

Contiguous Allocation

- Each file occupies a set of contiguous blocks on the disk
- Pros and cons
  - Simple: only starting location (block #) and length (number of blocks) are saved in FCB
  - Support random access and cache friendly
  - Suffer from the *external fragmentation*

Contiguous Allocation

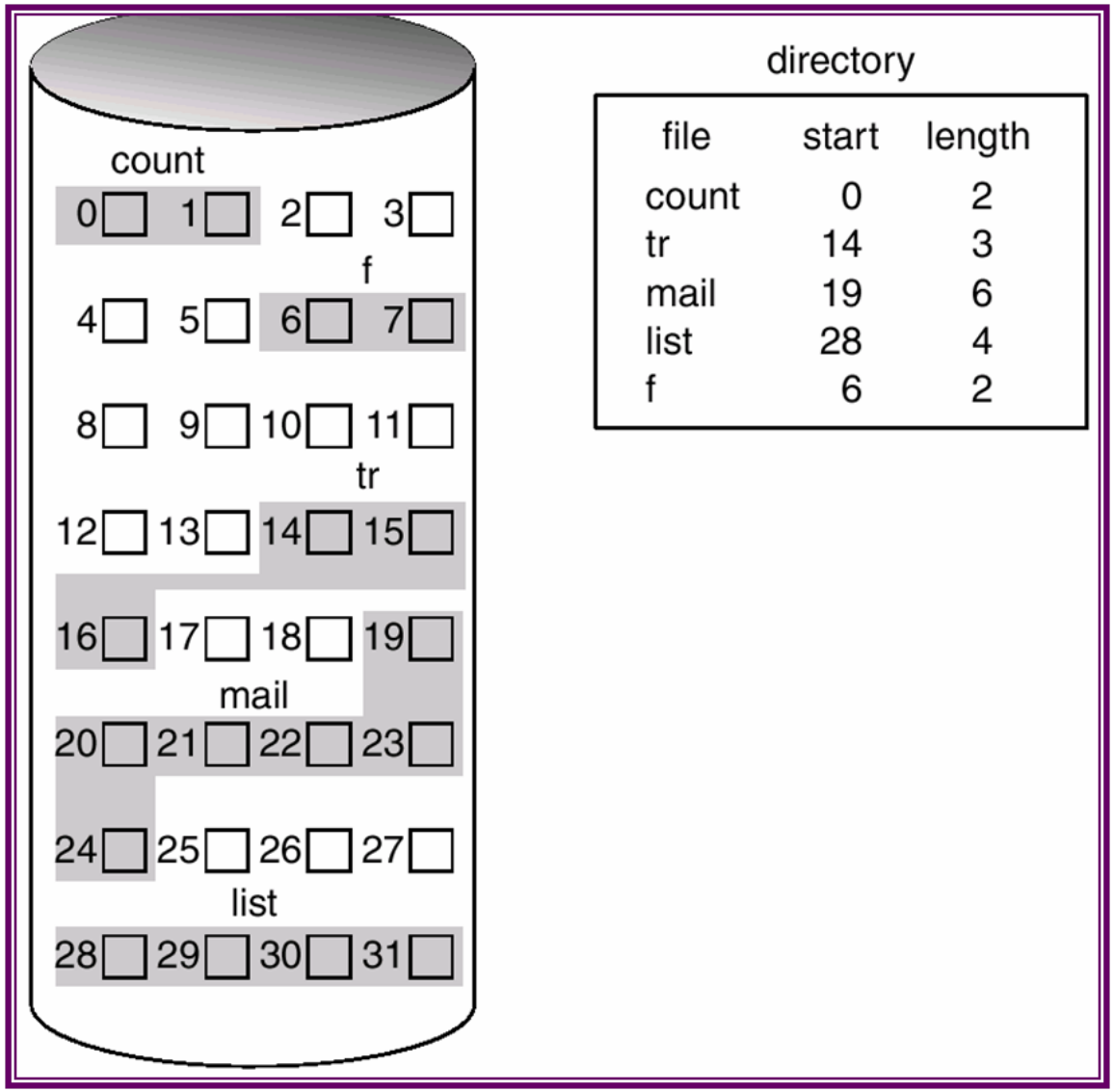
- Each file occupies a set of contiguous blocks on the disk
- Pros and cons
  - Simple: only starting location (block #) and length (number of blocks) are saved in FCB
  - Support random access and cache friendly
  - Suffer from the *external fragmentation*
    - The problem of *dynamic storage allocation*

Contiguous Allocation

- Each file occupies a set of contiguous blocks on the disk
- Pros and cons
  - Simple: only starting location (block #) and length (number of blocks) are saved in FCB
  - Support random access and cache friendly
  - Suffer from the *external fragmentation*
    - The problem of *dynamic storage allocation*
  - Difficult to grow a file

An example

An example


count
0 1 2 3
f
4 5 6 7
8 9 10 11
tr
12 13 14 15
16 17 18 19
mail
20 21 22 23
24 25 26 27
list
28 29 30 31
directory
file start length
count 0 2
tr 14 3
mail 19 6
list 28 4
f 6 2

Linked Allocation

Linked Allocation

- Each file is a linked list of disk blocks

Linked Allocation

- Each file is a linked list of disk blocks
  - The FCB contains a pointer to the first and last blocks of the file

Linked Allocation

- Each file is a linked list of disk blocks
  - The FCB contains a pointer to the first and last blocks of the file
  - Each block contains a pointer to the next block

Linked Allocation

- Each file is a linked list of disk blocks
  - The FCB contains a pointer to the first and last blocks of the file
  - Each block contains a pointer to the next block
    - These pointers are not visible to the user

Linked Allocation

- Each file is a linked list of disk blocks
  - The FCB contains a pointer to the first and last blocks of the file
  - Each block contains a pointer to the next block
    - These pointers are not visible to the user
    - Thus, if each block is 512 bytes and a disk address (the pointer) requires 4 bytes, then the user sees blocks of 508 bytes

Linked Allocation

- Each file is a linked list of disk blocks
  - The FCB contains a pointer to the first and last blocks of the file
  - Each block contains a pointer to the next block
    - These pointers are not visible to the user
    - Thus, if each block is 512 bytes and a disk address (the pointer) requires 4 bytes, then the user sees blocks of 508 bytes
- Pros and cons

Linked Allocation

- Each file is a linked list of disk blocks
  - The FCB contains a pointer to the first and last blocks of the file
  - Each block contains a pointer to the next block
    - These pointers are not visible to the user
    - Thus, if each block is 512 bytes and a disk address (the pointer) requires 4 bytes, then the user sees blocks of 508 bytes
- Pros and cons
  - Simple: need only starting address

Linked Allocation

- Each file is a linked list of disk blocks
  - The FCB contains a pointer to the first and last blocks of the file
  - Each block contains a pointer to the next block
    - These pointers are not visible to the user
    - Thus, if each block is 512 bytes and a disk address (the pointer) requires 4 bytes, then the user sees blocks of 508 bytes
- Pros and cons
  - Simple: need only starting address
  - No waste of space

Linked Allocation

- Each file is a linked list of disk blocks
  - The FCB contains a pointer to the first and last blocks of the file
  - Each block contains a pointer to the next block
    - These pointers are not visible to the user
    - Thus, if each block is 512 bytes and a disk address (the pointer) requires 4 bytes, then the user sees blocks of 508 bytes
- Pros and cons
  - Simple: need only starting address
  - No waste of space
  - Extra space required for the pointer

Linked Allocation

- Each file is a linked list of disk blocks
  - The FCB contains a pointer to the first and last blocks of the file
  - Each block contains a pointer to the next block
    - These pointers are not visible to the user
    - Thus, if each block is 512 bytes and a disk address (the pointer) requires 4 bytes, then the user sees blocks of 508 bytes
- Pros and cons
  - Simple: need only starting address
  - No waste of space
  - Extra space required for the pointer
  - No random access

Linked Allocation

- Each file is a linked list of disk blocks
  - The FCB contains a pointer to the first and last blocks of the file
  - Each block contains a pointer to the next block
    - These pointers are not visible to the user
    - Thus, if each block is 512 bytes and a disk address (the pointer) requires 4 bytes, then the user sees blocks of 508 bytes
- Pros and cons
  - Simple: need only starting address
  - No waste of space
  - Extra space required for the pointer
  - No random access
  - Bad reliability because of scattered pointers

Linked Allocation

- Each file is a linked list of disk blocks
  - The FCB contains a pointer to the first and last blocks of the file
  - Each block contains a pointer to the next block
    - These pointers are not visible to the user
    - Thus, if each block is 512 bytes and a disk address (the pointer) requires 4 bytes, then the user sees blocks of 508 bytes
- Pros and cons
  - Simple: need only starting address
  - No waste of space
  - Extra space required for the pointer
  - No random access
  - Bad reliability because of scattered pointers
- The overhead of the pointers can be decreased by collecting several blocks into one larger block called a *cluster*

File Allocation Table (FAT)

File Allocation Table (FAT)

- To solve the problem of simple linked allocation, a section of disk at the beginning of each partition is set aside to contain a table which contains all the pointers of the file system.

File Allocation Table (FAT)

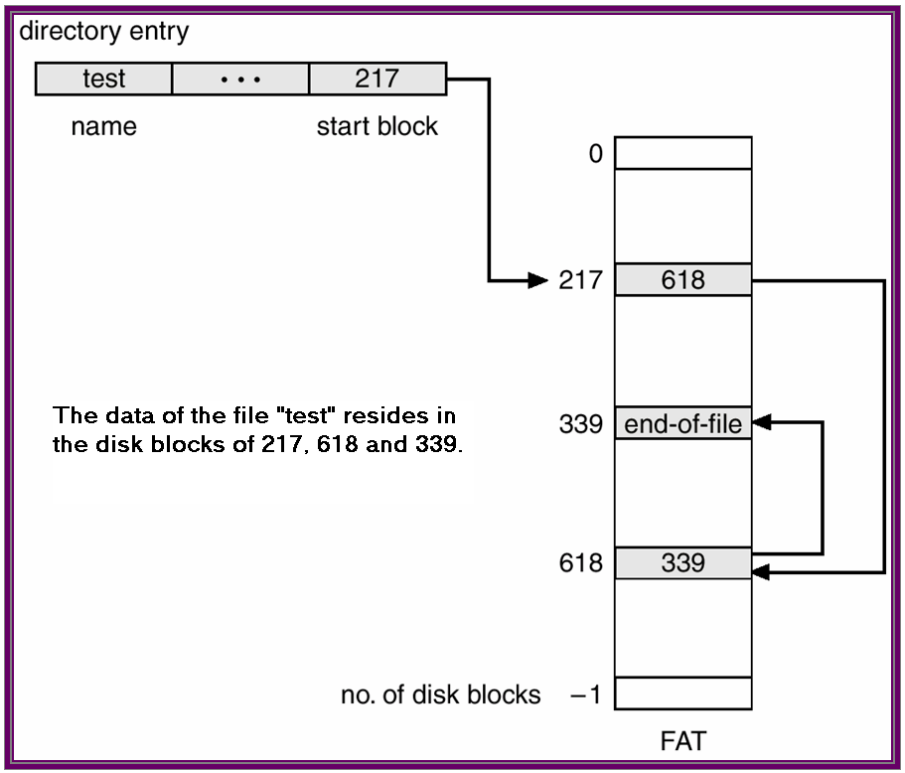
- To solve the problem of simple linked allocation, a section of disk at the beginning of each partition is set aside to contain a table which contains all the pointers of the file system.
  - This table is called *File Allocation Table (FAT)*

File Allocation Table (FAT)

- To solve the problem of simple linked allocation, a section of disk at the beginning of each partition is set aside to contain a table which contains all the pointers of the file system.
  - This table is called *File Allocation Table (FAT)*
- An example

File Allocation Table (FAT)

- To solve the problem of simple linked allocation, a section of disk at the beginning of each partition is set aside to contain a table which contains all the pointers of the file system.
  - This table is called *File Allocation Table (FAT)*
- An example



FAT file system

FAT file system

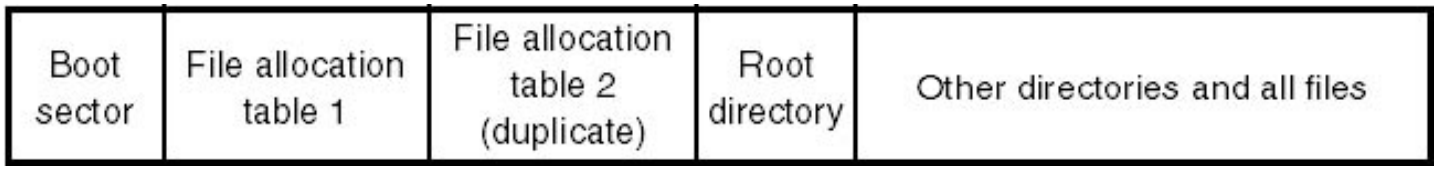
- The above linked allocation with FAT is used by the MS-DOS and OS/2 operating systems.

FAT file system

- The above linked allocation with FAT is used by the MS-DOS and OS/2 operating systems.
  - It's well known as the FAT file system

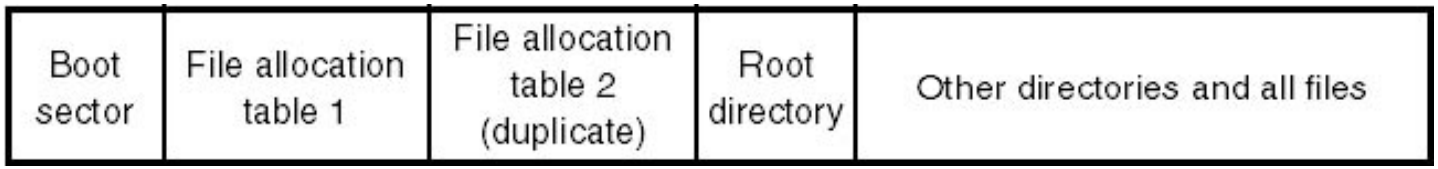
FAT file system

- The above linked allocation with FAT is used by the MS-DOS and OS/2 operating systems.
  - It's well known as the FAT file system



FAT file system

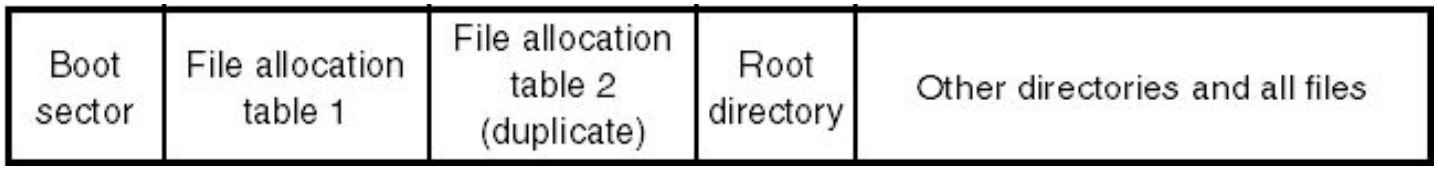
- The above linked allocation with FAT is used by the MS-DOS and OS/2 operating systems.
  - It's well known as the FAT file system



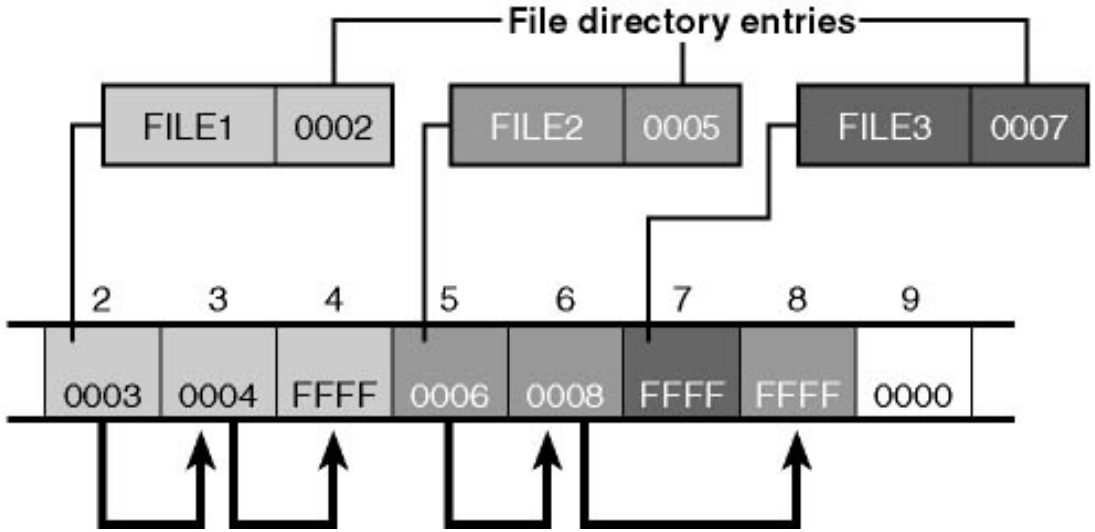
- An example (FAT16)

FAT file system

- The above linked allocation with FAT is used by the MS-DOS and OS/2 operating systems.
  - It's well known as the FAT file system



- An example (FAT16)



Questions

- Any questions?

Indexed Allocation

Indexed Allocation

- Bring all pointers to disk blocks of a file together into one location: *index block*

Indexed Allocation

- Bring all pointers to disk blocks of a file together into one location: *index block*
  - The index block holds an array of disk-block addresses

Indexed Allocation

- Bring all pointers to disk blocks of a file together into one location: *index block*
  - The index block holds an array of disk-block addresses
- Pros and cons

Indexed Allocation

- Bring all pointers to disk blocks of a file together into one location: *index block*
  - The index block holds an array of disk-block addresses
- Pros and cons
  - Need index table

Indexed Allocation

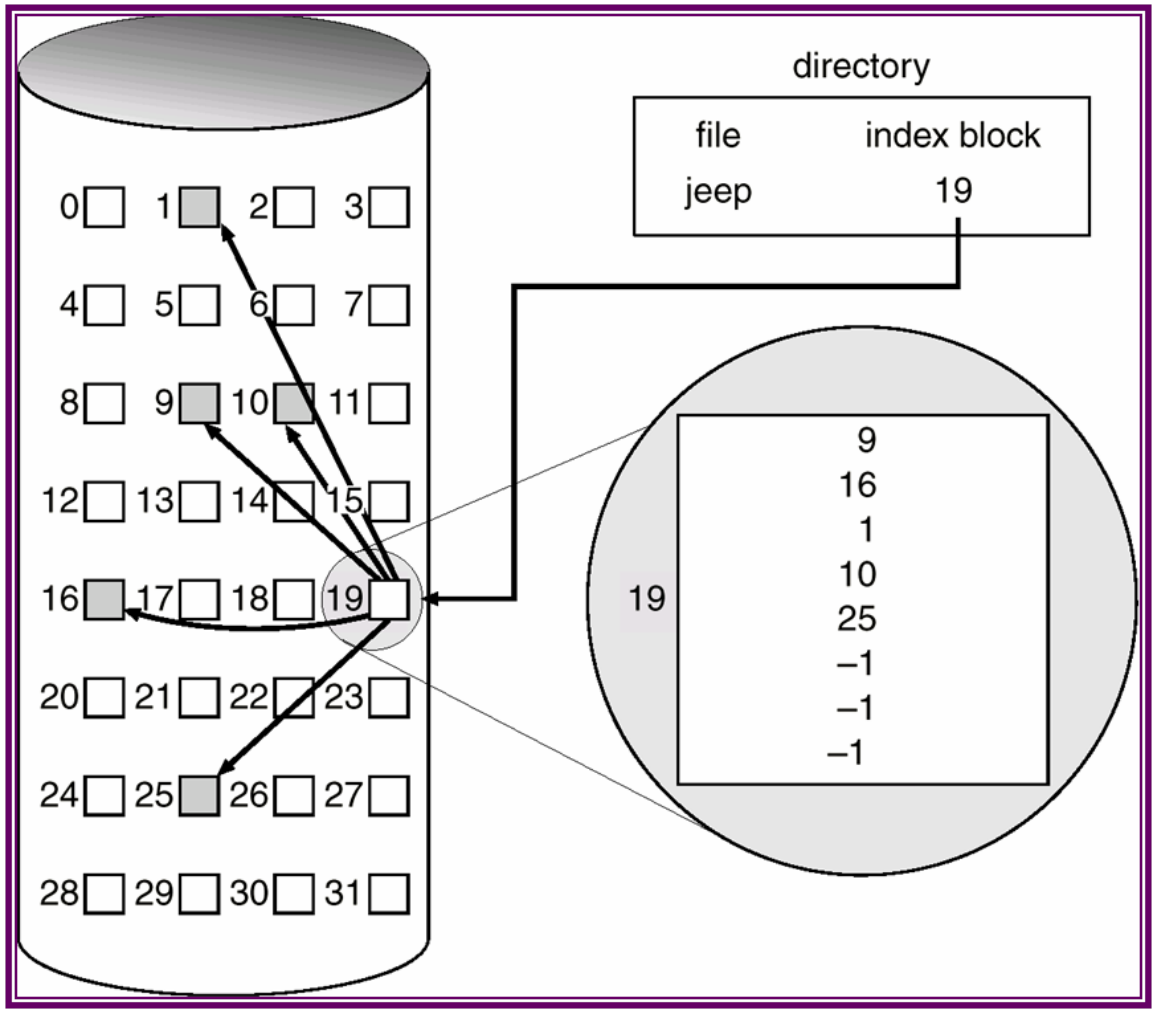
- Bring all pointers to disk blocks of a file together into one location: *index block*
  - The index block holds an array of disk-block addresses
- Pros and cons
  - Need index table
  - Random access

Indexed Allocation

- Bring all pointers to disk blocks of a file together into one location: *index block*
  - The index block holds an array of disk-block addresses
- Pros and cons
  - Need index table
  - Random access
  - Dynamic access without external fragmentation, but have overhead of index block.

An example of indexed allocation

An example of indexed allocation



The index block

The index block

- How large should the index block be?

The index block

- How large should the index block be?
- Three schemes:

The index block

- How large should the index block be?
- Three schemes:
  1. Linked scheme

The index block

- How large should the index block be?
- Three schemes:
  1. Linked scheme
  2. Multi-level index

The index block

- How large should the index block be?
- Three schemes:
  1. Linked scheme
  2. Multi-level index
  3. Combined scheme of above two

The index block

- How large should the index block be?
- Three schemes:
  1. Linked scheme
  2. Multi-level index
  3. Combined scheme of above two
     - This is scheme used by most UNIX file systems

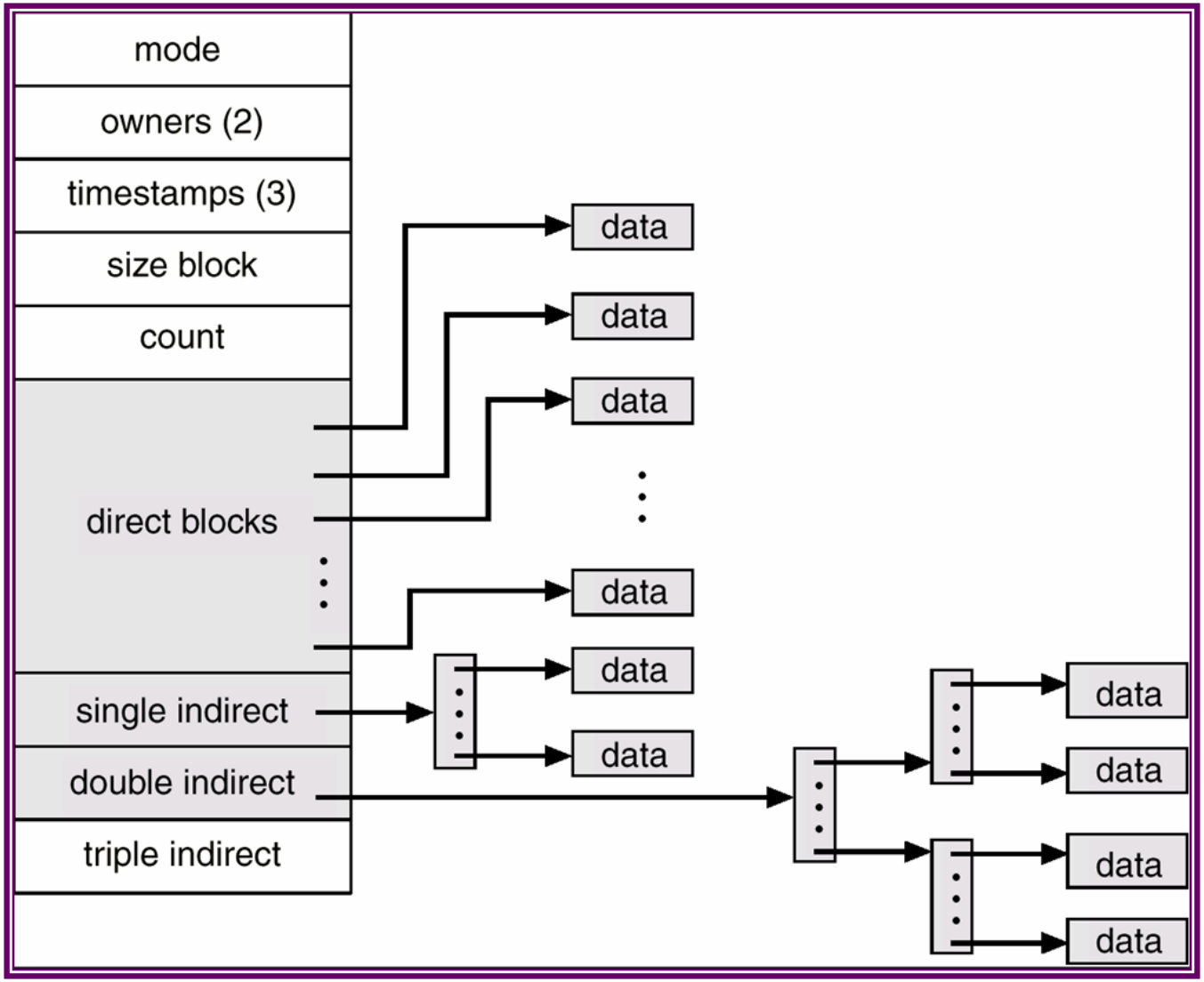
An example

An example

- The FCB of the UNIX file system, i.e., the *inode*

An example

- The FCB of the UNIX file system, i.e., the *inode*



Questions

- Any questions?

Free space management

Free space management

- Since disk space is limited, we need to reuse the space from deleted files for new files, if possible.

Free space management

- Since disk space is limited, we need to reuse the space from deleted files for new files, if possible.
  - Hence, the free space within a file system must be recorded for allocation to new files

Free space management

- Since disk space is limited, we need to reuse the space from deleted files for new files, if possible.
  - Hence, the free space within a file system must be recorded for allocation to new files
- Usually, one of the two methods is used

Free space management

- Since disk space is limited, we need to reuse the space from deleted files for new files, if possible.
  - Hence, the free space within a file system must be recorded for allocation to new files
- Usually, one of the two methods is used
  1. Bit vector

Free space management

- Since disk space is limited, we need to reuse the space from deleted files for new files, if possible.
  - Hence, the free space within a file system must be recorded for allocation to new files
- Usually, one of the two methods is used
  1. Bit vector
  2. Linked list

Bit vector

Bit vector

- It's also known as *bit map*

Bit vector

- It's also known as *bit map*
- Each block is represented by 1 bit

Bit vector

- It's also known as *bit map*
- Each block is represented by 1 bit
  - If the block is free, the bit is 1

Bit vector

- It's also known as *bit map*
- Each block is represented by 1 bit
  - If the block is free, the bit is 1
  - If the block is allocated, the bit is 0

Bit vector

- It's also known as *bit map*
- Each block is represented by 1 bit
  - If the block is free, the bit is 1
  - If the block is allocated, the bit is 0
- Pros and cons

Bit vector

- It's also known as *bit map*
- Each block is represented by 1 bit
  - If the block is free, the bit is 1
  - If the block is allocated, the bit is 0
- Pros and cons
  - Simple and efficient in finding the first free block, or n consecutive free blocks on the disk

Bit vector

- It's also known as *bit map*
- Each block is represented by 1 bit
  - If the block is free, the bit is 1
  - If the block is allocated, the bit is 0
- Pros and cons
  - Simple and efficient in finding the first free block, or n consecutive free blocks on the disk
    - Because many CPUs supply bit-manipulation instructions

Bit vector

- It's also known as *bit map*
- Each block is represented by 1 bit
  - If the block is free, the bit is 1
  - If the block is allocated, the bit is 0
- Pros and cons
  - Simple and efficient in finding the first free block, or n consecutive free blocks on the disk
    - Because many CPUs supply bit-manipulation instructions
  - Easy to get contiguous blocks
  - Bit vector must be kept in the memory to be efficient and in the disk to be persistent

Bit vector

- It's also known as *bit map*
- Each block is represented by 1 bit
  - If the block is free, the bit is 1
  - If the block is allocated, the bit is 0
- Pros and cons
  - Simple and efficient in finding the first free block, or n consecutive free blocks on the disk
    - Because many CPUs supply bit-manipulation instructions
  - Easy to get contiguous blocks
  - Bit vector must be kept in the memory to be efficient and in the disk to be persistent
    - Assume block size $= 2^{12}$ bytes and disk size $= 2^{30}$ bytes (1 gigabyte), then n=$2^{30}/2^{12} = 2^{18}$ (32K bytes)

Questions

- Any questions?

Linked list

Linked list

- Link together all the free disk blocks and keep a pointer to the first free block in a special location on the disk and cache it in memory for efficiency

Linked list

- Link together all the free disk blocks and keep a pointer to the first free block in a special location on the disk and cache it in memory for efficiency
- Pros and cons

Linked list

- Link together all the free disk blocks and keep a pointer to the first free block in a special location on the disk and cache it in memory for efficiency
- Pros and cons
  - The link pointer (except the link head) is saved within the free blocks, low space overhead

Linked list

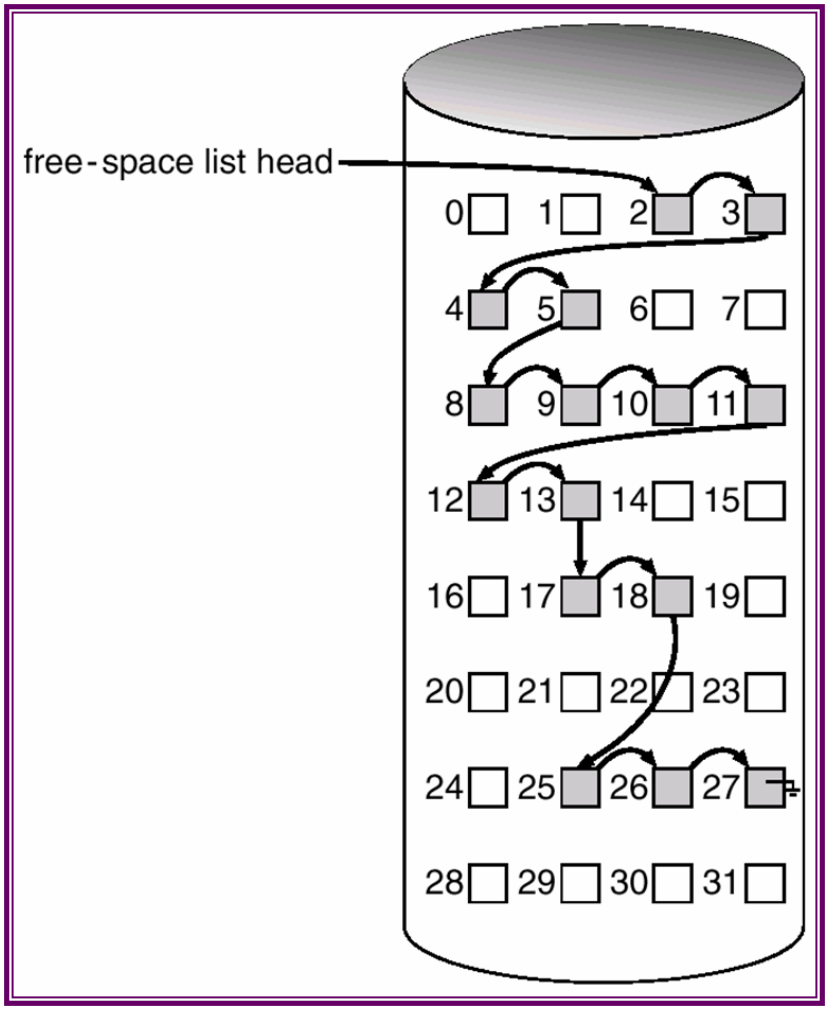
- Link together all the free disk blocks and keep a pointer to the first free block in a special location on the disk and cache it in memory for efficiency
- Pros and cons
  - The link pointer (except the link head) is saved within the free blocks, low space overhead
  - It's not efficient to traverse the list to get free blocks

Linked list

- Link together all the free disk blocks and keep a pointer to the first free block in a special location on the disk and cache it in memory for efficiency
- Pros and cons
  - The link pointer (except the link head) is saved within the free blocks, low space overhead
  - It's not efficient to traverse the list to get free blocks
    - Cache several addresses of the free blocks at beginning

An example

An example


free-space list head
0
1
2
3
4
5
6
7
8
9
10
11
12
13
14
15
16
17
18
19
20
21
22
23
24
25
26
27
28
29
30
31

Questions

- Any questions?

Performance

Performance

- If every disk access requires the disk operations, such as rotation, seek, access and transfer,

Performance

- If every disk access requires the disk operations, such as rotation, seek, access and transfer,
  - system performance must be bad

Performance

- If every disk access requires the disk operations, such as rotation, seek, access and transfer,
  - system performance must be bad
  - disk should be busy and its lifetime be short

Performance

- If every disk access requires the disk operations, such as rotation, seek, access and transfer,
  - system performance must be bad
  - disk should be busy and its lifetime be short
- We need caches at following three levels:

Performance

- If every disk access requires the disk operations, such as rotation, seek, access and transfer,
  - system performance must be bad
  - disk should be busy and its lifetime be short
- We need caches at following three levels:
  1. In the disk controller: *track cache*

Performance

- If every disk access requires the disk operations, such as rotation, seek, access and transfer,
  - system performance must be bad
  - disk should be busy and its lifetime be short
- We need caches at following three levels:
  1. In the disk controller: *track cache*
  2. In the operating system: *disk cache*

Performance

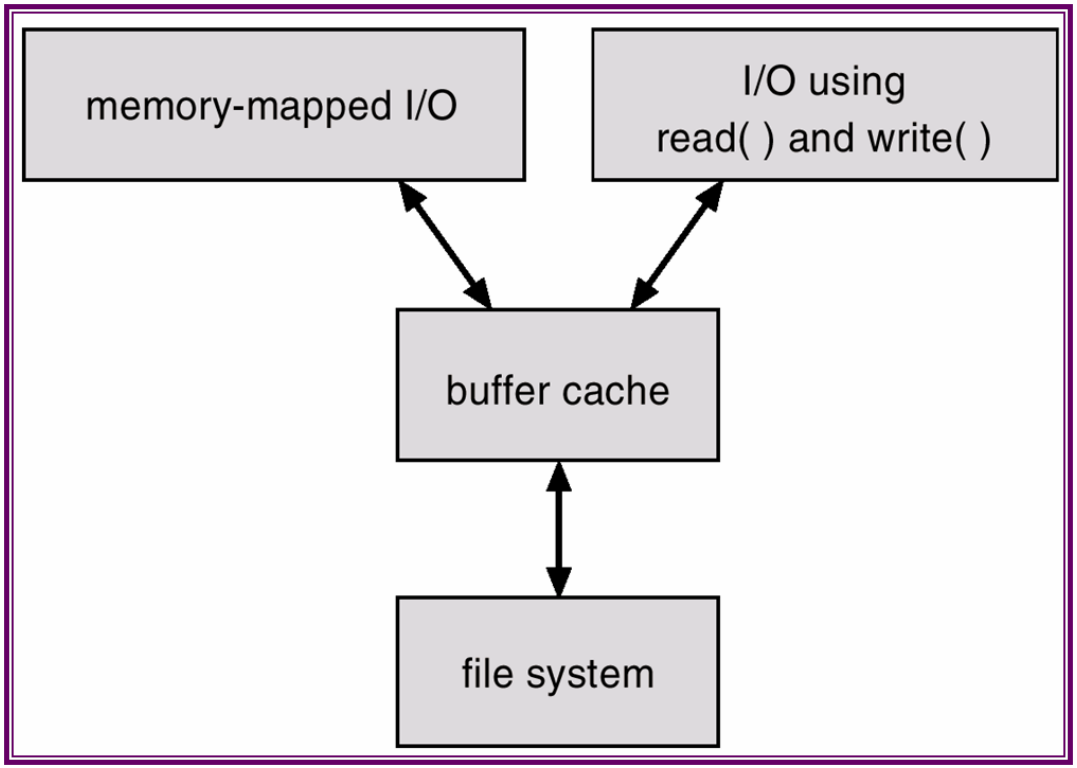
- If every disk access requires the disk operations, such as rotation, seek, access and transfer,
  - system performance must be bad
  - disk should be busy and its lifetime be short
- We need caches at following three levels:
  1. In the disk controller: *track cache*
  2. In the operating system: *disk cache*
     - Best known as the *buffer cache*

Performance

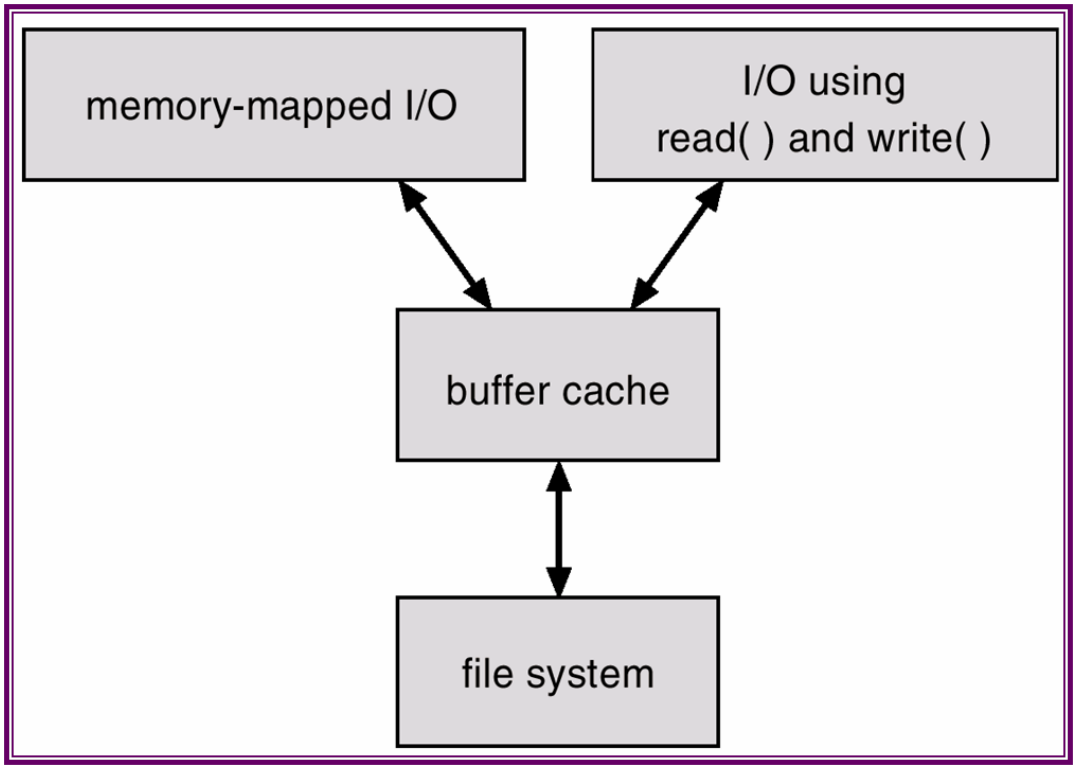
- If every disk access requires the disk operations, such as rotation, seek, access and transfer,
  - system performance must be bad
  - disk should be busy and its lifetime be short
- We need caches at following three levels:
  1. In the disk controller: *track cache*
  2. In the operating system: *disk cache*
     - Best known as the *buffer cache*
  3. In the user program: *RAM disk*

Buffer cache

Buffer cache

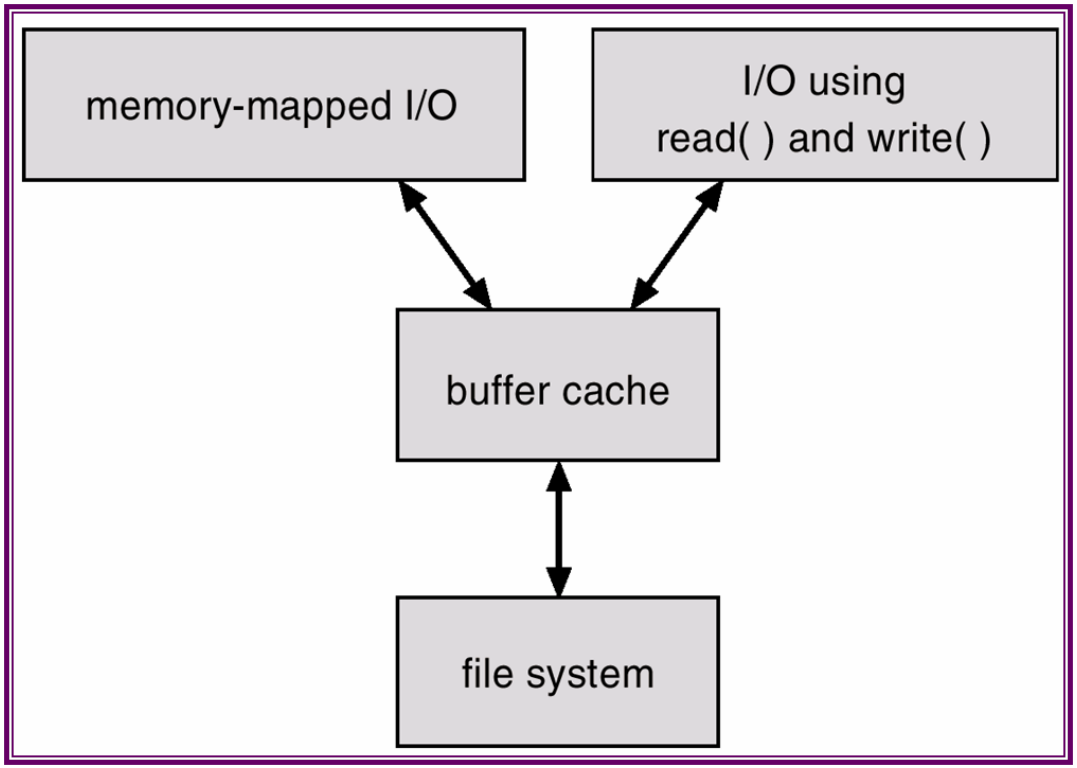


Buffer cache



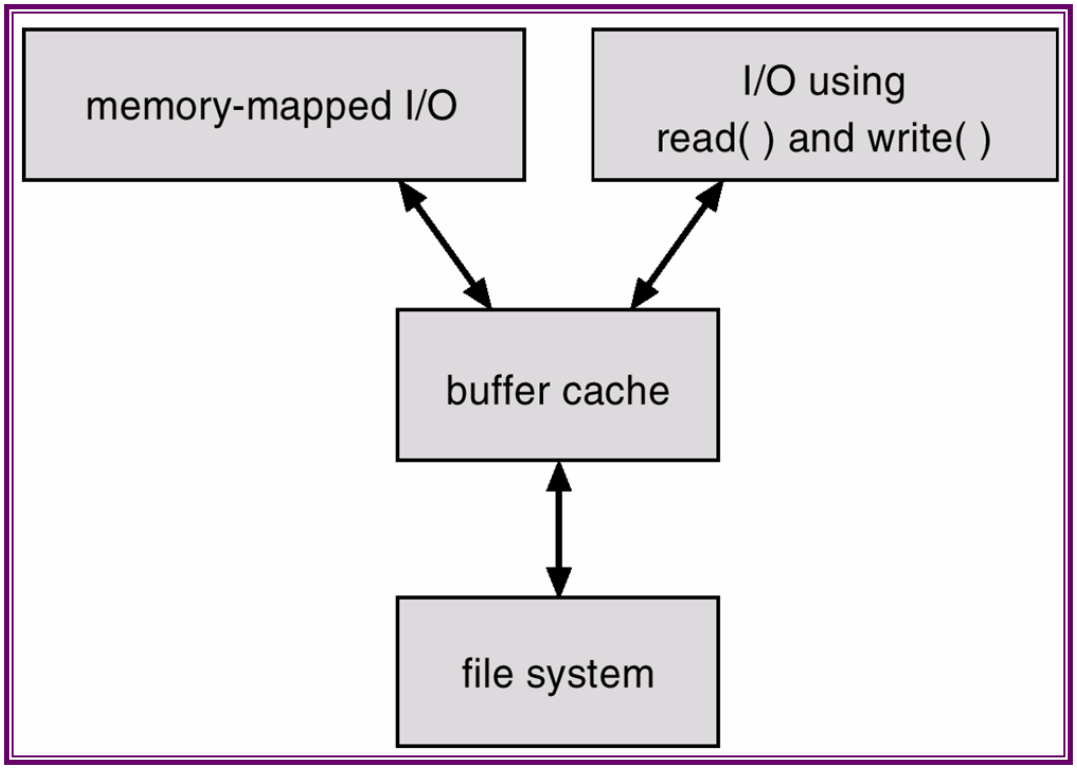
- It's typically a separate portion of the main memory

Buffer cache



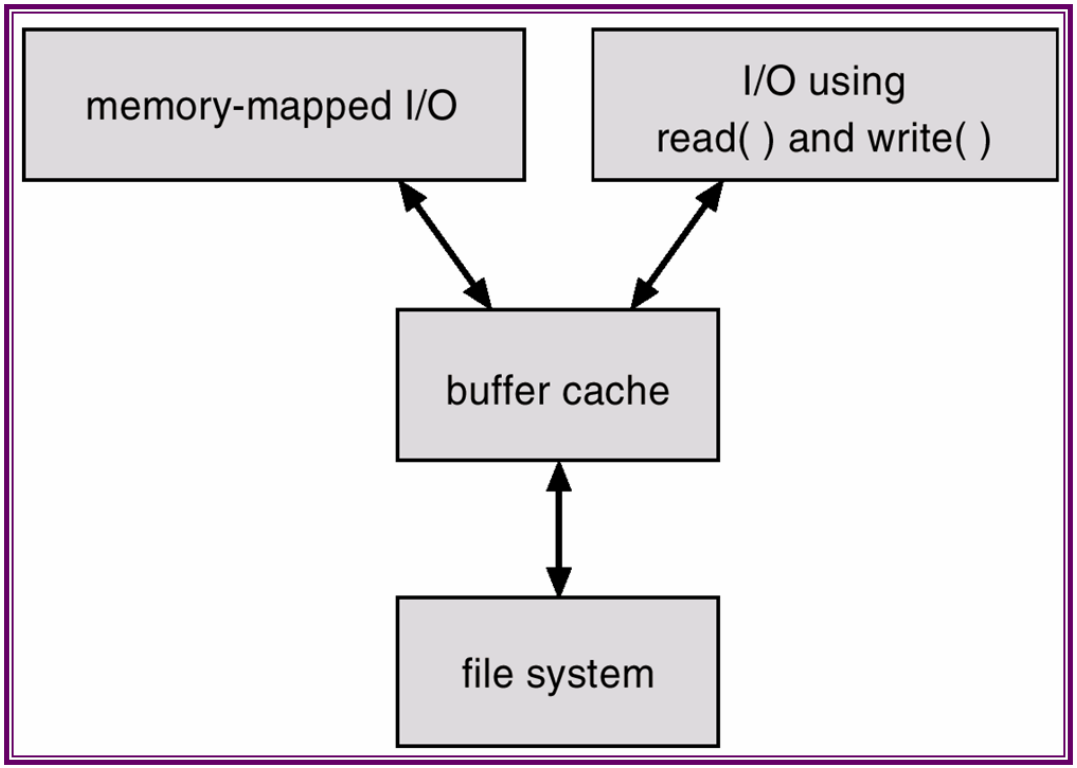
- It's typically a separate portion of the main memory
- It controls performance by selecting replacement algorithms

Buffer cache



- It's typically a separate portion of the main memory
- It controls performance by selecting replacement algorithms
  - *read-ahead*: sequential pre-fetch

Buffer cache



- It's typically a separate portion of the main memory
- It controls performance by selecting replacement algorithms
  - *read-ahead*: sequential pre-fetch
  - *free-behind*: remove a block as soon as its access is completed

RAM disk

RAM disk

- A portion of memory is set aside and treated as a *virtual disk*

RAM disk

- A portion of memory is set aside and treated as a *virtual disk*
  - The RAM disk device driver accepts all the standard disk operations, but performs those operations on the memory, instead of on a disk

RAM disk

- A portion of memory is set aside and treated as a *virtual disk*
  - The RAM disk device driver accepts all the standard disk operations, but performs those operations on the memory, instead of on a disk
- Unfortunately, RAM disks are only useful for temporary storage, since its contents will be lost between reboots

Questions

- Any questions?