1 File Systems

1.1 Files

- A file is a named collection of related information, usually as a sequence of bytes, with two views:
  - Logical (programmer's) view, as the users see it.
  - Physical (operating system) view, as it actually resides on secondary storage.
- What is the difference between a file and a data structure in memory? Basically,
  - files are intended to be non-volatile; hence in principle, they are long lasting,
  - files are intended to be moved around (i.e., copied from one place to another), accessed by different programs and users, and so on.
- File lifetime is independent of process lifetime
- Used to share data between processes
- Input to applications is by means of a file
- **File Management**; File management system is considered part of the operating system
  - Manages a trusted, shared resource
  - Bridges the gap between:
    - lowlevel disk organization (an array of blocks),
    - and the user's views (a stream or collection of records)
  - Also includes tools outside the kernel; E.g. formatting, recovery, defrag, consistency, and backup utilities.
  - Objectives for a File Management System;
    - Provide a convenient naming system for files
    - Provide uniform I/O support for a variety of storage device types
    - Provide a standardized set of I/O interface routines
    - Guarantee that the data in the file are valid
    - Optimize performance

Extension	Meaning
file.bak	Backup file
file.c	C source program
file.gif	Compuserve Graphical Interchange Format image
file.hlp	Help file
file.html	World Wide Web HyperText Markup Language document
file.jpg	Still picture encoded with the JPEG standard
file.mp3	Music encoded in MPEG layer 3 audio format
file.mpg	Movie encoded with the MPEG standard
file.o	Object file (compiler output, not yet linked)
file.pdf	Portable Document Format file
file.ps	PostScript file
file.tex	Input for the TEX formatting program
file.txt	General text file
file.zip	Compressed archive

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

Figure 1: Some typical file extensions.

* Minimize or eliminate the potential for lost or destroyed data
* Provide I/O support and access control for multiple users
* Support system administration (e.g., backups)

1.1.1 File Naming

- File system must provide a convenient naming scheme
- Textual Names (see Fig. 1)
- May have restrictions
  - Only certain characters, E.g. no '/' characters
  - Limited length
  - Only certain format, E.g DOS, 8 + 3
- Case (in)sensitive
- Names may obey conventions (.c files or C files)
  - Interpreted by tools (UNIX)
  - Interpreted by operating system (Windows)

1.1.2 File Structure; From OS's perspective

- **Stream of Bytes** (see Fig. 2)
  - OS considers a file to be unstructured

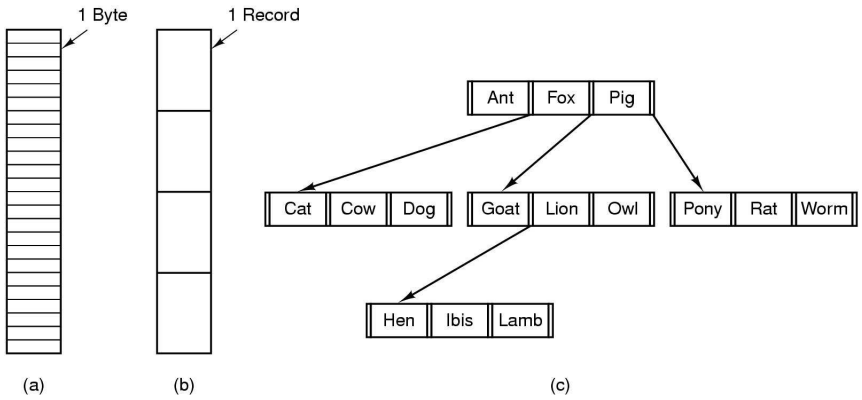

Figure 2: Three kinds of files. (a) byte sequence. (b) record sequence. (c) tree.

  - Simplifies file management for the OS
  - Applications can impose their own structure
  - Used by UNIX, Windows, most modern OSes

- **Records** (see Fig. 2)
  - Collection of bytes treated as a unit; Example: employee record
  - Operations at the level of records (read_rec, write_rec)
  - File is a collection of similar records
  - OS can optimize operations on records

- **Tree of Records** (see Fig. 2)
  - Records of variable length
  - Each has an associated key
  - Record retrieval based on key

1.1.3 File Types

- Regular files
- Directories
- Device Files

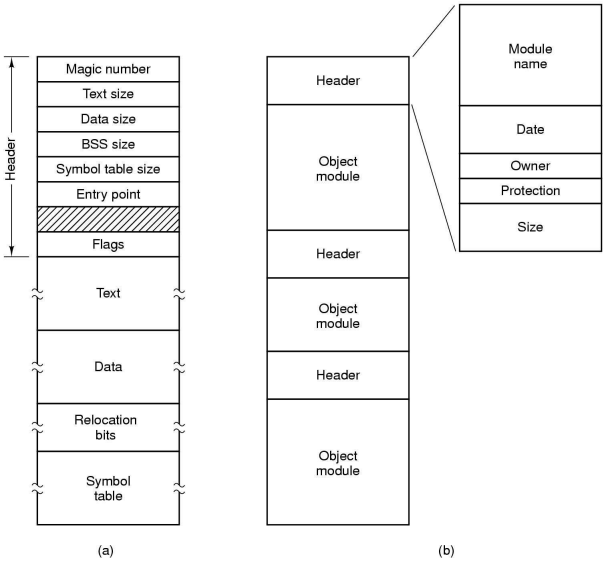

Figure 3: (a) An executable file (b) An archive.

  - Character Devices
  - Block Devices

- Some systems distinguish between regular file types; ASCII text files, binary files
- A common implementation technique (as organizational help with consistent usage) is to include the type as an extension to the file name (see Fig. 1)
- Files are structured internally to meet the expectations of the program(s) that manipulate them.
- All systems recognize their own executable file format; May use a magic number (see Fig. 3)

1.1.4 File Access

- The information stored in a file can be accessed in a variety of methods:
  - Sequential access
    * in order, one record after another

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

Figure 4: Some possible file attributes.

        - read all bytes/records from the beginning
        - cannot jump around, could rewind or back up
        - convenient when medium was mag tape
    - Random (Direct) access
        - bytes/records read in any order skipping the previous records
        - essential for data base systems
        - read can be
            - move file pointer (seek), then read or
            - each read specifies the file pointer
    - Keyed; in any order, but with particular value(s); e.g., hash table or dictionary. TLB lookup is one example of a keyed search

- Other access methods, such as indexed, can be built on top of the above basic techniques.

1.1.5 File Attributes(see Fig. 4)

- Information about files kept in directory structure maintained on disk
- Each file is associated with a collection of information, known as attributes:

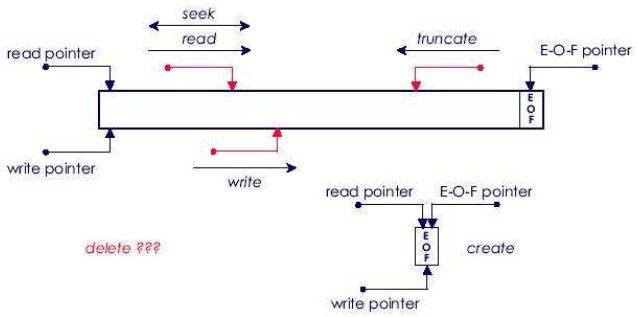

Figure 5: File operations.

- name, owner, creator, only information in human-readable form
- type, (e.g., source, data, binary) needed if system supports different types
- location, (e.g., I-node or disk address) pointer to file location on device
- organization, (e.g., sequential, indexed, random)
- time and date, (creation, modification, and last accessed)
- size, current file size
- protection, who can do reading, writing, executing
- variety of other (e.g., maintenance) information.

1.1.6 File Operations

- There are six basic operations (not all) for file manipulation: create, write, read, delete, reposition r/w pointer (a.k.a. seek), and truncate (not very common.) (see Fig. 5)
- Open(Fi) search directory structure on disk for entry Fi, move content of entry to memory
- Close (Fi) move content of entry Fi in memory to directory structure on disk

1.1.7 An Example Program Using File System Calls (see Fig. 6)

- copyfile **abc** *xyz*; where argv[0]="copyfile", argv[1]="abc", argv[2]="xyz"

```
/* 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 */

    /* 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 */
}
```

Figure 6: A simple program to copy a file.

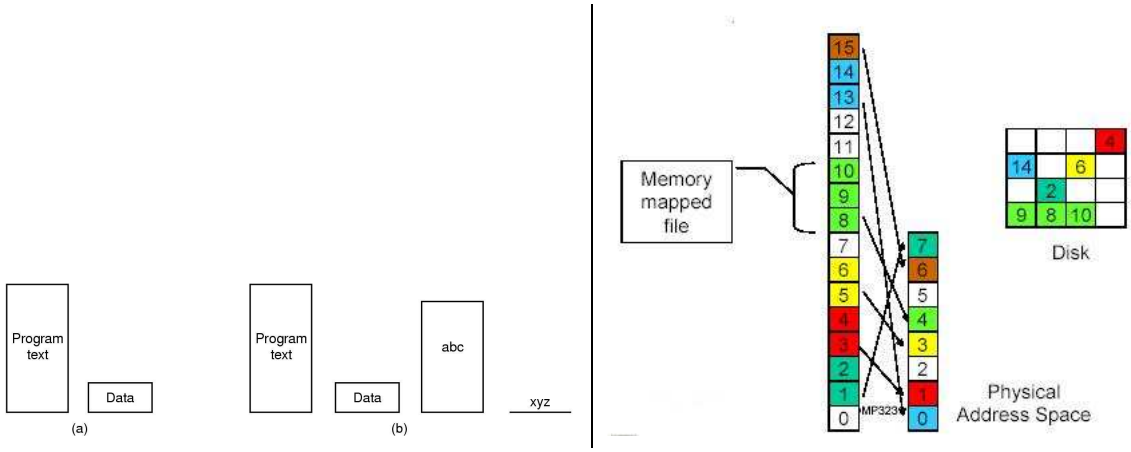

Figure 7: Left: (a) Segmented process before mapping files into its address space (b) Process after mapping existing file abc into one segment creating new segment for xyz. Right: Memory mapped files and paging

1.1.8 Memory–Mapped Files (see Fig. 7)

- Avoids translating from on-disk format to in-memory format (and vice versa)
  - Supports complex structures
  - No read/write systems calls
  - File simply (paged or swapped) to file system
  - Unmap when finished
- Problems
  - Determining actual file size after modification; Round to nearest whole page (even if only 1 byte file)
  - Care must be taken if file is shared; E.g. one process memorymapped and one process read/write syscalls
  - Large files may not fit in the virtual address space

1.1.9 File Organization and Access; Programmer's Perspective

- One of the key elements of a file system is the way the files are organized. File organization is the logical structuring as well as the access method(s) of files.

- Given an operating system supporting unstructured files that are stream-of-bytes, how should one organize the contents of the files?
- Performance considerations:
  - File system performance affects overall system performance
  - Organization of the file system affects performance
  - File organization (data layout) affects performance; depends on access patterns
- Possible access patterns:
  - Read the whole file
  - Read individual blocks or records from a file
  - Read blocks or records preceding or following the current one
  - Retrieve a set of records
  - Write a whole file sequentially
  - Insert/delete/update records in a file
  - Update blocks in a file
- Criteria for File Organization
  - Rapid access
    - Needed when accessing a single record
    - Not needed for batch mode
  - Ease of update; File on CDROM will not be updated, so this is not a concern
  - Economy of storage
    - Should be minimum redundancy in the data
    - Redundancy can be used to speed access such as an index
  - Simple maintenance
  - Reliability
- Fundamental File Organizations; Common file organization schemes are:
  - Pile
  - Sequential

  - Indexed Sequential
  - Indexed
  - Direct or Hashed

- **Pile** (see Fig. 8)
  - Data are collected in the order they arrive
  - Purpose is to accumulate a mass of data and save it
  - Records may have different fields
  - No structure
  - Record access is by exhaustive search
  - Update:
    * Same size record; okay
    * Variable size; poor
  - Retrieval:
    * Single record; poor
    * Subset; poor
    * Exhaustive; okay

- **Sequential** (see Fig. 8)
  - Fixed format used for records
  - Records are the same length
  - Field names and lengths are attributes of the file
  - One field is the key filed
    * Uniquely identifies the record
    * Records are stored in key sequence
  - New records are placed in a log file or transaction file
  - Batch update is performed to merge the log file with the master file
  - Update:
    * Same size record; good
    * Variable size; No
  - Retrieval:
    * Single record; poor

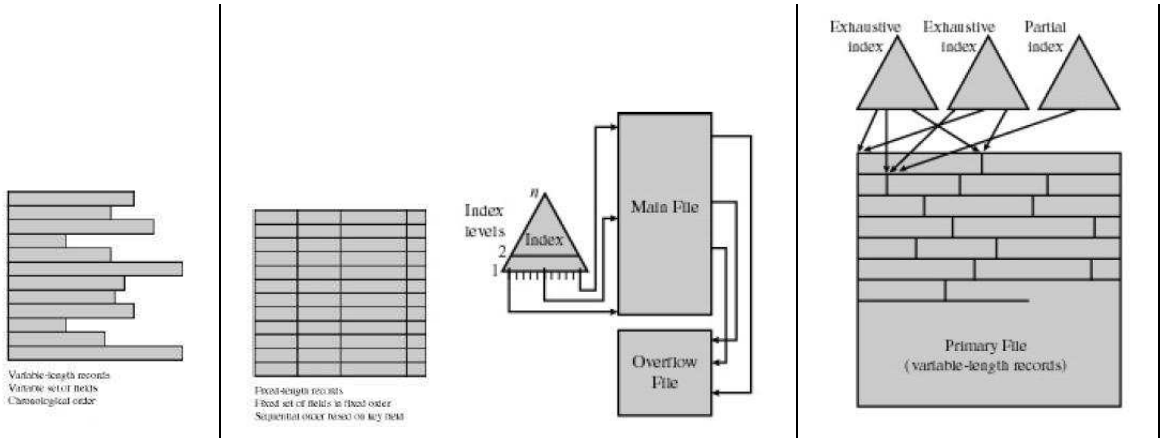

Figure 8: Fundamental File Organizations; (a) Pile (b) Sequential (c) Indexed Sequential (d) Indexed.

    - Subset; poor
    - Exhaustive; okay

- **Indexed Sequential** (see Figs. 8,9)
    - Index provides a lookup capability to quickly reach the vicinity of the desired record
        - Contains key field and a pointer to the main file
        - Indexed is searched to find highest key value that is equal or less than the desired key value
        - Search continues in the main file at the location indicated by the pointer
    - New records are added to an overflow file
    - Record in main file that precedes it is updated to contain a pointer to the new record
    - The overflow is merged with the main file during a batch update
    - Update:
        - Same size record; good
        - Variable size; No
    - Retrieval:
        - Single record; good
        - Subset; poor

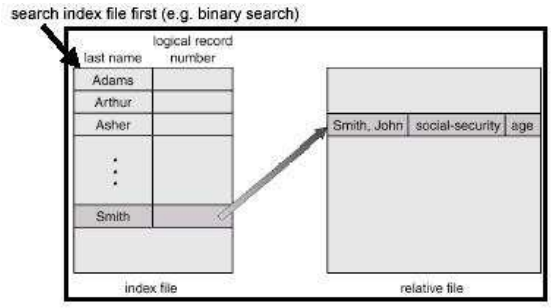

Figure 9: IBM indexed-sequential access method (ISAM).

    - Exhaustive; okay

- **Comparison of sequential and indexed sequential lookup**
  - Example: a file contains 1 million records
  - On average 500,00 accesses are required to find a record in a sequential file
  - If an index contains 1000 entries, it will take on average 500 accesses to find the key, followed by 500 accesses in the main file. Now on average it is 1000 accesses

- **Indexed File** (see Fig. 8)
  - Uses multiple indexes for different key fields
  - May contain an exhaustive index that contains one entry for every record in the main file
  - May contain a partial index
  - Update:
    - Same size record; good
    - Variable size; good
  - Retrieval:
    - Single record; good
    - Subset; good (Assuming the selecting attribute is indexed on)
    - Exhaustive; okay

- **The Direct, or Hashed File**

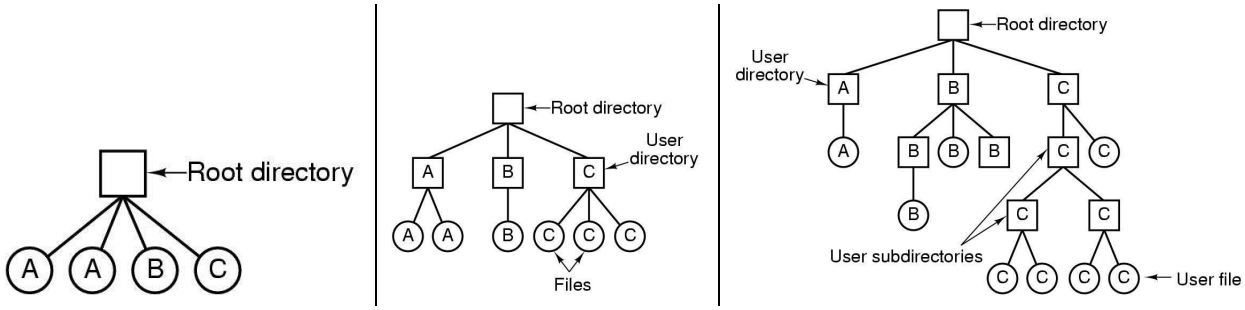

Figure 10: (a) Single-Level Directory Systems (b) Two-Level Directory Systems (c) Hierarchical Directory Systems.

- Key field required for each record
- Key maps directly or via a hash mechanism to an address within the file
- Directly access a block at a the known address
- Update:
  - Same size record; good
  - Variable size; No (Fixed sized records used)
- Retrieval:
  - Single record; excellent
  - Subset; poor
  - Exhaustive; poor

1.2 Directories

- A directory is a symbol table, which can be searched for information about the files. Also, it is the fundamental way of organizing files. Usually, a directory is itself a file
- A typical directory entry contains information (attributes, location, ownership) about a file. Directory entries are added as files are created, and are removed when files are deleted.
- Provides mapping between file names and the files themselves
- Goals in Organization of Directory
  - **Efficiency**; locate file quickly
  - **Naming**; convenient to users,

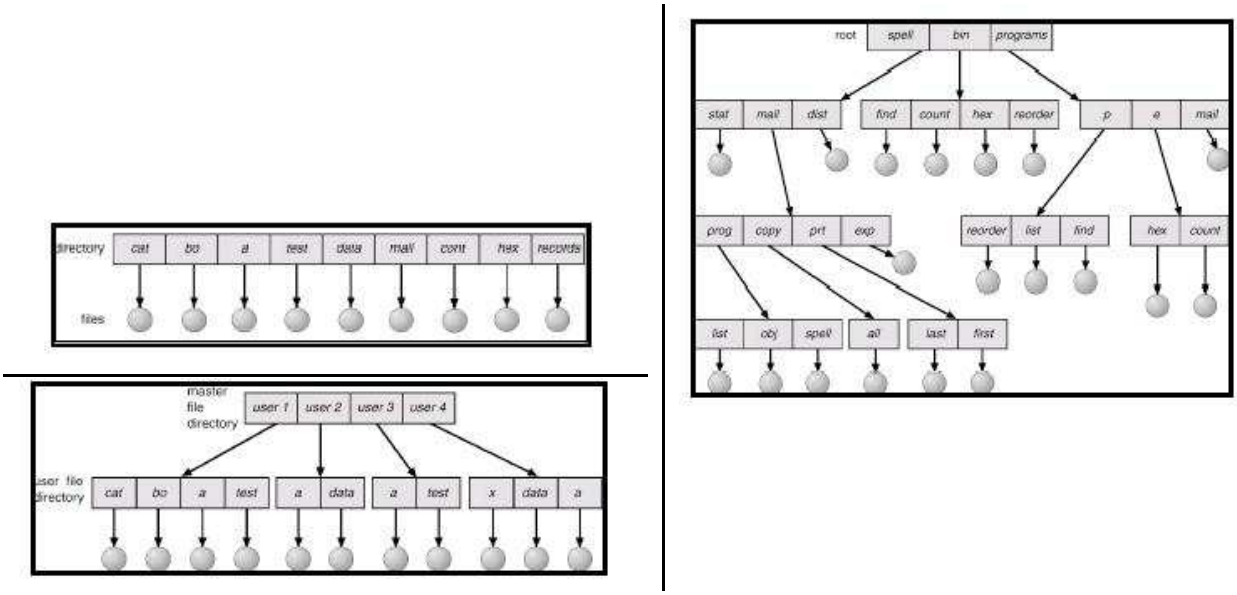

Figure 11: Example to (a) Single-Level Directory Systems (b) Two-Level Directory Systems (c) Hierarchical Directory Systems.

    - ∗ 2 users can use same name for different files
    - ∗ Same file can have several different names
  - **Grouping**; logical grouping of files by attributes, (e.g., all Java programs, all games, . . .)

- Single–Level Directory Systems (see Figs. 10,11)
  - List of entries, one for each file
  - Sequential file with the name of the file serving as the key
  - Provides no help in organizing the files
  - Forces user to be careful not to use the same name for two different files

- Two–Level Directory Systems (see Figs. 10,11)
  - One directory for each user and a master directory
  - Master directory contains entry for each user; Provides access control information
  - Each user directory is a simple list of files for that user
  - Still provides no help in structuring collections of files

- Hierarchical, or Tree-Structured Directory Systems (see Figs. 10,11)
  - Files can be located by following a path from the root, or master, directory down various branches; This is the absolute pathname for the file
  - Can have several files with the same file name as long as they have unique path names

1.2.1 Path Names

- Always specifying the absolute pathname for a file is tedious!
- Introduce the idea of a working directory; Files are referenced relative to the working directory
- Example: cwd = /home/dizin, profile = /home/dizin/.profile
- Absolute pathname; A path specified from the root of the file system to the file
- A Relative pathname; A pathname specified from the cwd
- Note: '.' (dot) and '..' (dotdot) refer to current and parent directory
  - Example: cwd = /home/dizin
  - ../../etc/passwd
  - /etc/passwd
  - ../dizin/../../etc/passwd
  - Are all the same file

1.2.2 File Sharing

- In multiuser system, allow files to be shared among users
- Two issues
  - Access rights. Allowing users to share files raises a major issue: protection. A general approach is to provide controlled access to files through a set of operations such as read, write, delete, list, and append. Then permit users to perform one or more operations. One popular protection mechanism is a condensed version of access list, where the system recognizes three classifications of users with each file and directory: user, group, other

  - Management of simultaneous access

- **Access Rights**
  - None
    * User may not know of the existence of the file
    * User is not allowed to read the user directory that includes the file
  - Knowledge; User can only determine that the file exists and who its owner is
  - Execution; The user can load and execute a program but cannot copy it
  - Reading; The user can read the file for any purpose, including copying and execution
  - Appending; The user can add data to the file but cannot modify or delete any of the file's contents
  - Updating; The user can modify, deleted, and add to the file's data. This includes creating the file, rewriting it, and removing all or part of the data
  - Changing protection; User can change access rights granted to other users
  - Deletion; User can delete the file
  - Owners
    * Has all rights previously listed
    * May grant rights to others using the following classes of users; Specific user, User groups, All for public files

```
total 1704
drwxr-x--- 3 user group 4096 Oct 14 08:13 .
drwxr-x--- 3 user group 4096 Oct 14 08:14 ..
drwxr-x--- 2 user group 4096 Oct 14 08:12 backup
-rw-r----- 1 user group 141133 Oct 14 08:13 eniac3.jpg
-rw-r----- 1 user group 1580544 Oct 14 08:13 wk11.ppt
```

  - First letter: file type
    * **d** for directories
    * **-** for regular files
  - Three user categories; user, group, and other

  - Three access rights per category; read, write, and execute
  - d**rwx***rwx*rwx; **user** *group* other
  - Execute permission for directory? Permission to access files in the directory
  - To list a directory requires read permissions
  - What about drwxr-x–x ?
  - Problematic example
    * A owns file foo.bar
    * A wishes to keep his file private
    * Inaccessible to the general public
    * A wishes to give B read and write access
    * A wishes to give C readonly access
    * ???????

- **Management of Simultaneous Access**
  - Most Oses provide mechanisms for users to manage concurrent access to files; Example: lockf(), flock() system calls
  - Typically
    * User may lock entire file when it is to be updated
    * User may lock the individual records during the update
  - Mutual exclusion and deadlock are issues for shared access