File Management

Tanenbaum, Chapter 4

COMP3231
Operating Systems

Kevin Elphinstone

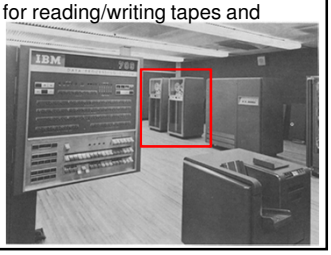
Outline

- Files and directories from the programmer (and user) perspective
- Files and directories internals – the operating system perspective

A brief history of file systems

Early batch processing systems
- –No OS
- –I/O from/to punch cards
- –Tapes and drums for external storage, but no FS
- –Rudimentary library support for reading/writing tapes and drums

IBM 709 [1958]



A brief history of file systems

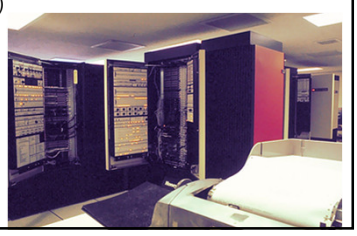
- The first file systems were single-level (everything in one directory)
- Files were stored in contiguous chunks
  - –Maximal file size must be known in advance
- Now you can edit a program and save it in a named file on the tape!

PDP-8 with DECTape [1965]

A brief history of file systems

- Time-sharing OSs
  - –Required full-fledged file systems
- MULTICS
  - –Multilevel directory structure (keep files that belong to different users separately)
  - –Access control lists
  - –Symbolic links

Honeywell 6180 running MULTICS [1976]



A brief history of file systems

- UNIX
  - –Based on ideas from MULTICS
  - –Simpler access control model
  - –Everything is a file!

PDP-7

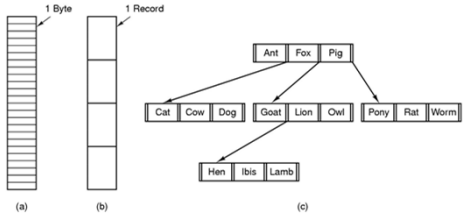
Overview of the FS abstraction

User's view	Under the hood
Uniform namespace	Heterogeneous collection of storage devices
Hierarchical structure	Flat address space (block numbers)
Arbitrarily-sized files	Fixed-size blocks
Symbolic file names	Numeric block addresses
Contiguous address space inside a file	Fragmentation
Access control	No access control
Tools for • Formatting • Defragmentation • Backup • Consistency checking	

File Names

- File system must provide a convenient naming scheme
  - Textual Names
  - 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)

File Structure Abstractions



- Three kinds of files
  - byte sequence
  - record sequence
  - key-based, tree structured
- e.g. IBM's indexed sequential access method (ISAM)

File Structure Abstractions

Stream of Bytes

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

Records

- 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 optimise operations on records

File Structure Abstractions

- Tree of Records
  - Records of variable length
  - Each has an associated key
  - Record retrieval based on key
  - Used on some data processing systems (mainframes)
- Mostly incorporated into modern databases

File Types

- Regular files
- Directories
- Device Files
  - May be divided into
    - Character Devices – stream of bytes
    - Block Devices
- Some systems distinguish between regular file types
  - ASCII text files, binary files

File Access Types

- Sequential access
  - –read all bytes/records from the beginning
  - –cannot jump around, could rewind or back up
  - –convenient when medium was magnetic tape
- Random access
  - –bytes/records read in any order
  - –essential for data base systems
  - –read can be …
    - move file pointer (seek), then read or
      - –lseek(location,…);read(…)
    - each read specifies the file pointer
      - –read(location,…)

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

Typical File Operations

- Create
- Delete
- Open
- Close
- Read
- Write
- Append
- Seek
- Get attributes
- Set Attributes
- 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 Organisation and Access
Programmer's Perspective

- Given an operating system supporting unstructured files that are a *stream-of-bytes,*

how can one organise the contents of the files?

File Organisation and Access
Programmer's Perspective

- 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

Programmers are free to structure the file to suit the application.

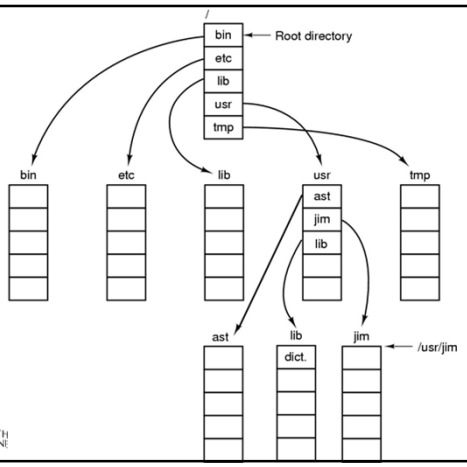
Criteria for File Organization

Things to consider when designing file layout

- Rapid access
  - Needed when accessing a single record
  - Not needed for batch mode
    - read from start to finish
- Ease of update
  - File on CD-ROM 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

File Directories

- Provide mapping between file names and the files themselves
- Contain information about files
  - Attributes
  - Location
  - Ownership
- Directory itself is a file owned by the operating system



Hierarchical (Tree-Structured) Directory

- 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

Current *Working Directory*

- 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/kevine

  .profile = /home/kevine/.profile

Relative and Absolute Pathnames

- 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/kevine

```
../../etc/passwd
/etc/passwd
../kevine/../.././etc/passwd
```

Are all the same file

Typical Directory Operations

- Create
- Delete
- Opendir
- Closedir
- Readdir
- Rename
- Link
- Unlink

Nice properties of UNIX naming

- Simple, regular format
  - Names referring to different servers, objects, etc., have the same syntax.
    - Regular tools can be used where specialised tools would be otherwise be needed.
- Location independent
  - Objects can be distributed or migrated, and continue with the same names.

Where is /home/kevine/.profile?

You only need to know the name!

An example of a bad naming convention

- From, Rob Pike and Peter Weinberger, "The Hideous Name", Bell Labs TR

UCBVAX::SYS$DISK:[ROB.BIN]CAT_V.EXE;13

File Sharing

- In multiuser system, allow files to be shared among users
- Two issues
  - Access rights
  - Management of simultaneous access

Access Rights

- None
  - User may not know of the existence of the file
  - User is not allowed to read the directory that includes the file
- Knowledge
  - User can only determine that the file exists and who its owner is

Access Rights

- 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

Access Rights

- 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

Access Rights

- 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

Case Study: UNIX Access Permissions

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

- First letter: file type
  - ***d*** for directories
  - - for regular files
- Three user categories
  - ***u***ser, ***g***roup, and ***o***ther

UNIX Access Permissions

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

- Three access rights per category
  - ***r***ead, ***w***rite, and e***x***ecute

drwxrwxrwx

user group other

UNIX Access Permissions

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

- Execute permission for directory?
  - –Permission to access files in the directory
- To list a directory requires read permissions
- What about **drwxr-x--x**?

UNIX Access Permissions

- Shortcoming
  - The three user categories are rather coarse
- Problematic example
  - Joe owns file **`foo.bar`**
  - Joe wishes to keep his file private
    - Inaccessible to the general public
  - Joe wishes to give Bill read and write access
  - Joe wishes to give Peter read-only access
  - How????????

Simultaneous Access

- Most OSes provide mechanisms for users to manage concurrent access to files
  - Example: flock(), lockf(), system calls
- Typically
  - User may lock entire file when it is to be updated
  - User may lock the individual records (i.e. ranges) during the update
- Mutual exclusion and deadlock are issues for shared access