Lecture 11:

File Management

www.cl.cam.ac.uk/Teaching/2001/OSFounds/

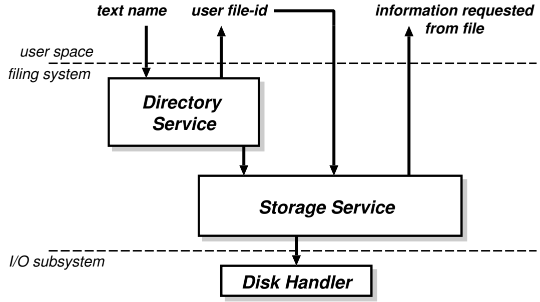
Lecture 11: Monday 29th October 2001

Today's Lecture

Today we'll cover:

- How does OS present a uniform logical view of information storage?
  - Files and directories,
  - Namespaces,
  - Sharing of files and directories, and
  - Operations on files.

File Management



Filing systems have two main components:

1. **Directory Service**
   - maps from names to file identifiers.
   - handles access & existence control
2. **Storage Service**
   - provides mechanism to store data on disk
   - includes means to implement directory service

File Concept

What is a file?

- Basic abstraction for non-volatile storage.
- Typically comprises a single contiguous logical address space.
- Internal structure:
  1. None (e.g. sequence of words, bytes), or
  2. Simple record structures
     - lines
     - fixed length
     - variable length
  3. Complex structures
     - formatted document
     - relocatable object file
- Can simulate last two with first method by inserting appropriate control characters.
- All a question of who decides:
  - operating system
  - program(mer).

Naming Files

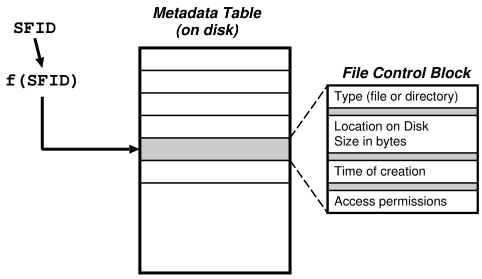
Files usually have at least two kinds of 'name':

1. **System file identifier (SFID)**:
   - (typically) a unique integer value associated with a given file
   - SFIDs are the names used within the filing system itself
2. "Human" name, e.g. `hello.java`
   - What users like to use
   - Mapping from human name to SFID is held in a **directory**, e.g.

Name	SFID
hello.java	12353
Makefile	23812
README	9742

   - Directories also non-volatile ⇒ must be stored on disk along with files.
3. Frequently also get user file identifier (UFID).
   - used to identify *open* files (see later)

File Meta-data



In addition to their contents and their name(s), files typically have a number of other attributes, e.g.

- Location: pointer to file location on device
- Size: current file size
- Type: needed if system supports different types
- Protection: controls who can read, write, etc.
- Time, date, and user identification: data for protection, security and usage monitoring.

Together this information is called **meta-data**.
It is contained in a **file control block**.

Directory Name Space

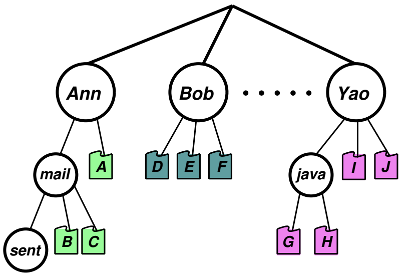
What are the requirements for our name space?

- Efficiency: locating a file quickly.
- Naming: user convenience
  - allow two (or more generally N) users to have the same name for different files
  - allow one file have several different names
- Grouping: logical grouping of files by properties (e.g. all Java programs, all games, . . .)

First attempts:

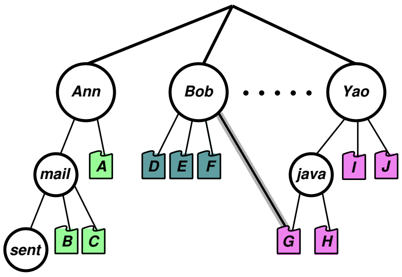
- Single-level: one directory shared between all users
  - ⇒ naming problem
  - ⇒ grouping problem
- Two-level directory: one directory per user
  - access via *pathname* (e.g. `bob:hello.java`)
  - can have same filename for different user
  - but still no grouping capability.

Directory Name Space cont.



- Get more flexibility with a general **hierarchy**.
  - directories hold files or [further] directories
  - create/delete files relative to a given directory
- Human name is full path name, but can get long: e.g. /usr/groups/X11R5/src/mit/server/os/4.2bsd/utils.c
  - offer relative naming
  - login directory
  - current working directory
- What does it mean to delete a [sub]-directory?

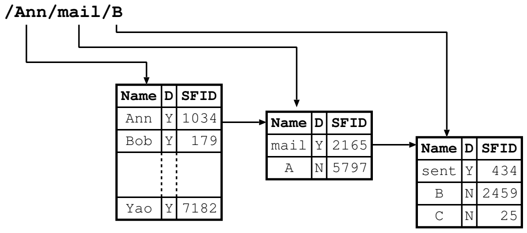
Directory Name Space cont.



- Hierarchy good, but still only one name per file.

⇒ extend to directed acyclic graph (DAG) structure:
  - allow **shared** subdirectories and files.
  - can have multiple **aliases** for the same thing
- Problem: dangling references
- Solutions:
  - back-references (but variable size records)
  - reference counts.
- Problem: cycles. . .

Directory Implementation



Name	D	SFID
Ann	Y	1034
Bob	Y	179
⋮	⋮	⋮
Yao	Y	7182

Name	D	SFID
mail	Y	2165
A	N	5797

Name	D	SFID
sent	Y	434
B	N	2459
C	N	25

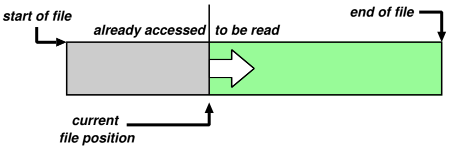
- Directories are non-volatile ⇒ store as "files" on disk, each with own SFID.
- Must be different *types* of file (for traversal)
- Explicit directory operations include:
  - create directory
  - delete directory
  - list contents
  - select current working directory
  - insert an entry for a file (a "link")

File Operations

UFID	*SFID*	*File Control Block (Copy)*
1	*23421*	*location on disk, size,...*
2	*3250*	*" "*
3	*10532*	*" "*
4	*7122*	*" "*
⋮	⋮	⋮

- Opening a file: `UFID = open(<pathname>)`
  1. directory service recursively searches directories for components of `<pathname>`
  2. if all goes well, eventually get `SFID` of file.
  3. copy file control block into memory.
  4. create new `UFID` and return to caller.
- Create a new file: `UFID = create(<pathname>)`
- Once have `UFID` can read, write, etc.
  - various modes (see next slide)
- Closing a file: `status = close(UFID)`
  1. copy [new] file control block back to disk.
  2. invalidate `UFID`

File Operations cont.



- Associate a **cursor** or **file position** with each open file (viz. UFID), initialised to start of file.
- Basic operations: *read next* or *write next*, e.g.
  - `read(UFID, buf, nbytes)`, or
  - `read(UFID, buf, nrecords)`
- Sequential Access: above, plus `rewind(UFID)`.
- Direct Access: *read* N or *write* N
  - allow "random" access to any part of file.
  - can implement with `seek(UFID, pos)`
- Other forms of data access possible, e.g.
  - append-only (may be faster)
  - indexed sequential access mode (ISAM)

Other Filing System Issues

- Access Control: file owner/creator should be able to control what can be done, and by whom.
  - access control normally a function of directory service ⇒ checks done at file *open* time
  - various types of access, e.g.
    - read, write, execute, (append?),
    - delete, list, rename
  - more advanced schemes possible (see later)
- Existence Control: what if a user deletes a file?
  - probably want to keep file in existence while there is a valid pathname referencing it
  - plus check entire FS periodically for garbage
  - existence control can also be a factor when a file is renamed/moved.
- Concurrency Control: need some form of **locking** to handle simultaneous access
  - may be mandatory or advisory
  - locks may be shared or exclusive
  - granularity may be file or subset

Summary

You should now understand:

- How files can be structured,
- How a directory can be represented,
- Directory hierarchies,
- Sharing of files/directories,
- Simple operations provided.

Next lecture: Unix (Part I)

Background Reading:

- Silberschatz et al.: – Chapter 11