Lecture 12:

Unix I: History and File Management

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

Lecture 12: Wednesday 31st October 2001

Today's Lecture

Today we'll cover:

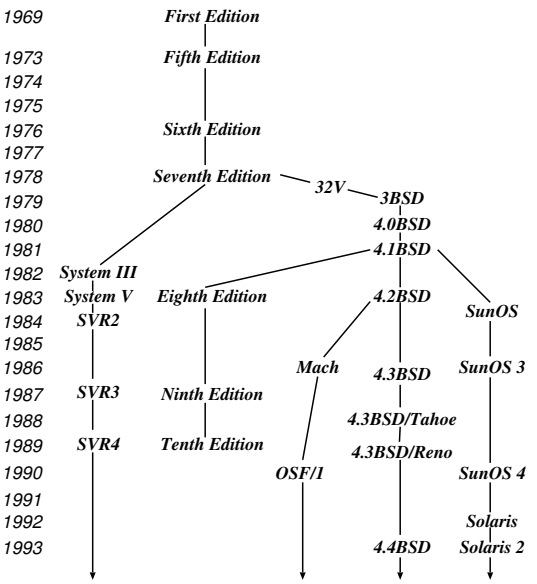
- Case Study: Unix
  - History,
  - Design features,
  - File and Directories,
  - Access Control.

Unix: Introduction

- Unix first developed in 1969 at Bell Labs (Thompson & Ritchie)
- Originally written in PDP-7 asm, but then (1973) rewritten in the 'new' high-level language C

  $\Rightarrow$ easy to port, alter, read, etc.
- 6^{th} edition ("V6") was widely available (1976).
  - source avail $\Rightarrow$ people could write new tools.
  - nice features of other OSes rolled in promptly.
- By 1978, V7 available (for both the 16-bit PDP-11 and the new 32-bit VAX-11).
- Since then, two main families:
  - AT&T: "System V", currently SVR4.
  - Berkeley: "BSD", currently 4.3BSD/4.4BSD.
- Standardisation efforts (e.g. POSIX, X/OPEN) to homogenise.
- Best known "UNIX" today is probably **linux**, but also get FreeBSD, NetBSD, and (commercially) Solaris, OSF/1, IRIX, and Tru64.

Unix Family Tree (Simplified)



Design Features

Ritchie and Thompson writing in CACM, July 74, identified the following (new) features of UNIX:

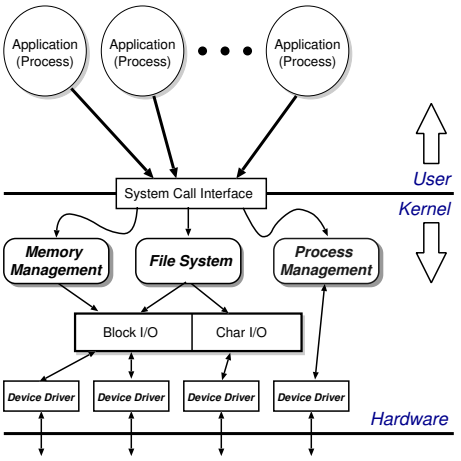
1. A hierarchical file system incorporating demountable volumes.
2. Compatible file, device and inter-process I/O.
3. The ability to initiate asynchronous processes.
4. System command language selectable on a per-user basis.
5. Over 100 subsystems including a dozen languages.
6. A high degree of portability.

Features which were not included:

- real time
- multiprocessor support

Fixing the above is pretty hard.

Structural Overview

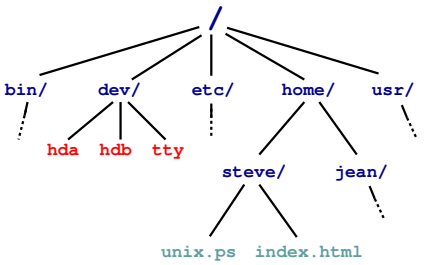


- Clear separation between user and kernel portions.
- Processes are unit of scheduling and protection.
- All I/O looks like operations on **files**.

File Abstraction

- A file is an unstructured sequence of bytes.
- Represented in user-space by a **file descriptor** (fd)
- Operations on files are:
  - *fd* = **open** (*pathname*, *mode*)
  - *fd* = **creat**(*pathname*, *mode*))
  - bytes = **read**(*fd*, *buffer*, *nbytes*)
  - count = **write**(*fd*, *buffer*, *nbytes*)
  - reply = **seek**(*fd*, *offset*, *whence*)
  - reply = **close**(*fd*)
- Devices represented by **special files**:
  - support above operations, although perhaps with bizarre semantics.
  - also have `ioctl`'s: allow access to device-specific functionality.
- Hierarchical structure supported by **directory files**.

Directory Hierarchy



- Directories map names to files (and directories).
- Have distinguished *root directory* called '/'
- Fully qualified pathnames ⇒ perform traversal from root.
- Every directory has '.' and '..' entries: refer to self and parent respectively.
- Shortcut: current working directory (*cwd*).
- In addition **shell** provides access to **home directory** as ~*username* (e.g. `~jean/`)

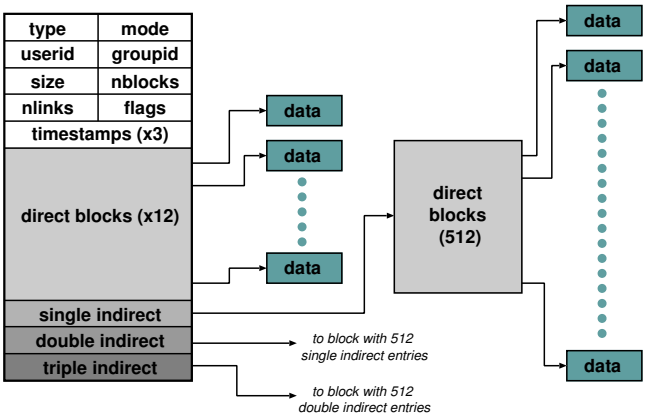
Aside: Password File

- `/etc/passwd` holds list of password entries.
- Each entry roughly of the form:

  user-name:encrypted-passwd:home-directory:shell

- Use **one-way function** to encrypt passwords.
  - i.e. a function which is easy to compute in one direction, but has a hard to compute inverse.
- To login:
  1. Get user name
  2. Get password
  3. Encrypt password
  4. Check against version in `/etc/password`
  5. If ok, instantiate login shell.
- Publicly readable since lots of useful info there.
- Problem: off-line attack.
- Solution: *shadow passwords* (`/etc/shadow`)

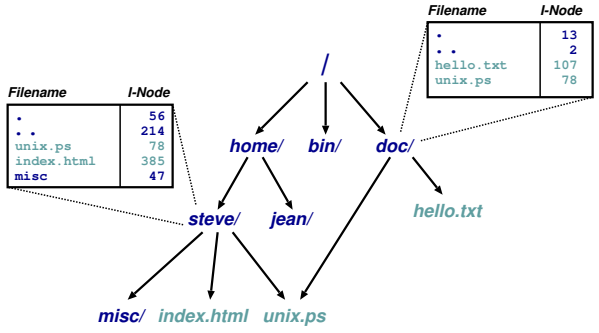
File System Implementation



- Inside kernel, a file is represented by a data structure called an index-node or **inode**.
- Holds file **meta-data**:

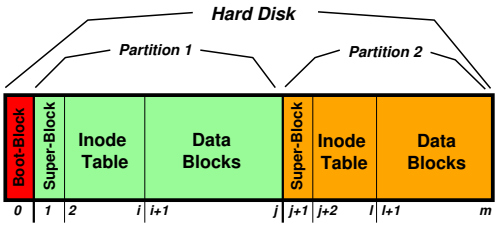
  a) Owner, permissions, reference count, etc.
  b) Location on disk of actual data (file contents).
- Where is the filename kept?

Directories and Links



- Directory is a file which maps filenames to inodes.
- An instance of a file in a directory is a **(hard) link**. (this is why have reference count in inode).
- Directories can have at most 1 (real) link. Why?
- Also get **soft-** or **symbolic**-links: a 'normal' file which contains a pathname.

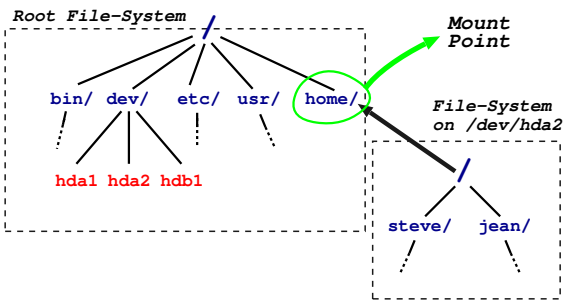
On-Disk Structures



- A disk is made up of a **boot block** followed by one or more **partitions**.

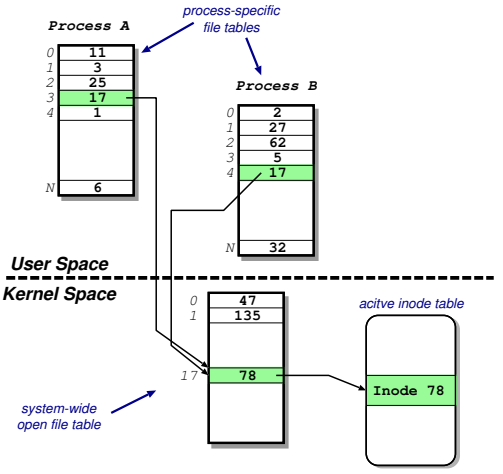
  (a partition is just a contiguous range of N fixed-size blocks of size k for some N and k).
- A Unix file-system resides within a partition.
- **Superblock** contains info such as:
  - number of blocks in file-system
  - number of free blocks in file-system
  - start of the free-block list
  - start of the free-inode list.
  - various bookkeeping information.

Mounting File-Systems



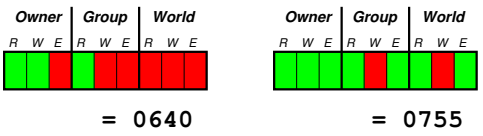
- Entire file-systems can be **mounted** on an existing directory in an already mounted filesystem.
- At very start, only '/' exists ⇒ need to mount a *root file-system*.
- Subsequently can mount other file-systems, e.g. `mount("/dev/hda2", "/home", options)`
- Provides a **unified name-space**: e.g. access `/home/jean/` directly.
- Cannot have hard links across mount points: why?
- What about soft links?

In-Memory Tables



- Recall: process sees files as file descriptors
- In implementation these are just indices into **process-specific open file table**
- Entries point to **system-wide open file table**. Why?
- These in turn point to (in memory) **inode table**.

Access Control



- Access control information held in each inode.
- Three bits for each of **owner**, **group** and **world**: read, write and execute.
- What do these mean for directories?
- In addition have **setuid** and **setgid** bits:
  - normally processes inherit permissions of invoking user.
  - setuid/setgid allow user to "become" someone else when running a given program.
  - e.g. `prof` owns both executable `test` (0711 and setuid), and `score` file (0600)
    ⇒ anyone user can run it.
    ⇒ it can update `score` file.
    ⇒ but users can't cheat.
- And what do *these* mean for directories?

Consistency Issues

- To delete a file, use the `unlink` system call.
- From the shell, this is `rm <filename>`
- Procedure is:
  1. check if user has sufficient permissions on the file (must have *write* access).
  2. check if user has sufficient permissions on the directory (must have *write* access).
  3. if ok, remove entry from directory.
  4. Decrement reference count on inode.
  5. if now zero:
     a. free data blocks.
     b. free inode.
- If crash: must check entire file-system:
  - check if any block unreferenced.
  - check if any block double referenced.

Summary

You should now understand:

- Files are unstructured byte streams.
- Everything is a file: 'normal' files, directories, symbolic links, special files.
- Hierarchy built from root ('/').
- Unified name-space (multiple file-systems may be mounted on any leaf directory).
- Low-level implementation based around inodes.
- Disk contains list of inodes (along with, of course, actual data blocks).
- Processes see file descriptors: small integers which map to system file table.
- Permissions for owner, group and everyone else.
- Setuid/setgid allow for more flexible control.
- Care needed to ensure consistency.

Next lecture: Unix Part II: Processes

Background Reading:

- Silberschatz et al.: – Appendix A (on web)
- Leffler et al.: – Part 3
- Ritchie & Thompson: – Original paper (on web)