File I/O
File Systems

Adapted by Joseph Lunderville
from slides by Dr. Brian Fraser
and course material by Dr. Steve Ko

Topics

- Can we use a file system for more than just reading and writing data files?
- How are file systems organized?
- What are hard/soft links?

The Universality of I/O

Everything is a File

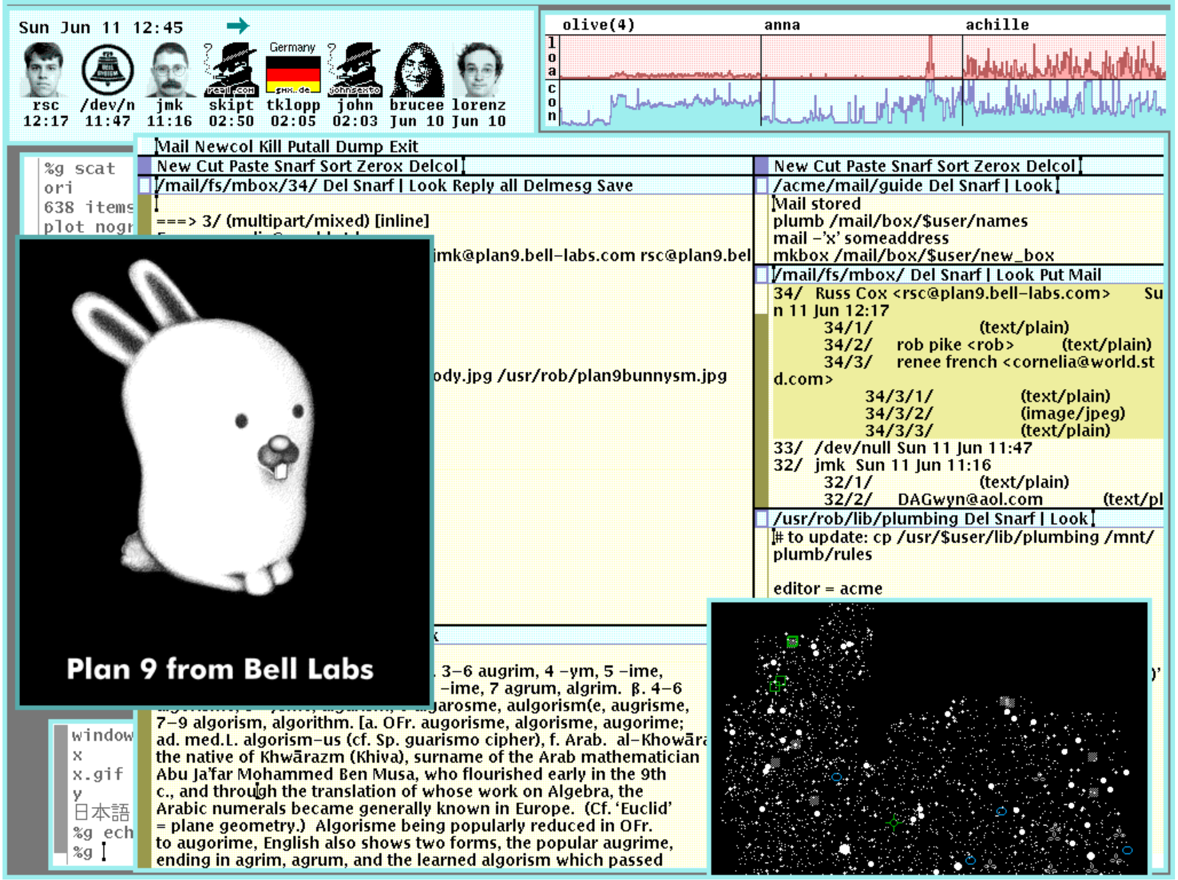
- *UNIX I/O model gives access to many things via files*
  - Actual files!
  - Devices: disks, sensors, keyboards, …
  - Networking
  - Process information
- *`/sys`, `/proc` File Systems*
  - Shows system and process information using `open()`, `read()`, etc.
  - Kernel dynamically populates information in form of files
  - Not “real files”: don’t consume disk space, not persistent

Example: /proc file system

• /proc/PID/status	general
• /proc/PID/fd	open files
• /proc/PID/task/TID	threads

Footnote - Plan 9 from Bell Labs

- Actually, not *everything* is a file…
- **What if it was?!**
- You get an obscure research OS!
- Everything a file, and the FS hierarchy defines isolation between environments
- Foreshadows containerization
- Highly distributed (this was a primary goal)
- Trivia: UTF-8 encoding was invented by Ken Thompson for Plan 9



```
Sun Jun 11 12:45
rsc    /dev/n  jmk    skipt  tklopp  john   brucee  lorenz
12:17  11:47   11:16  02:50  02:05   02:03  Jun 10  Jun 10

olive(4)          anna          achille

Mail Newcol Kill Putall Dump Exit
New Cut Paste Snarf Sort Zerox Delcol
/mail/fs/mbox/34/ Del Snarf | Look Reply all Delmesg Save
===> 3/ (multipart/mixed) [inline]
jmk@plan9.bell-labs.com rsc@plan9.bel
ody.jpg /usr/rob/plan9bunnysm.jpg

New Cut Paste Snarf Sort Zerox Delcol
/acme/mail/guide Del Snarf | Look
Mail stored
plumb /mail/box/$user/names
mail -'x' someaddress
mkbox /mail/box/$user/new_box
/mail/fs/mbox/ Del Snarf | Look Put Mail
34/  Russ Cox <rsc@plan9.bell-labs.com>   Su
n 11 Jun 12:17
     34/1/              (text/plain)
     34/2/    rob pike <rob>      (text/plain)
     34/3/    renee french <cornelia@world.st
d.com>
          34/3/1/            (text/plain)
          34/3/2/            (image/jpeg)
          34/3/3/            (text/plain)
33/  /dev/null Sun 11 Jun 11:47
32/  jmk  Sun 11 Jun 11:16
     32/1/              (text/plain)
     32/2/    DAGwyn@aol.com          (text/pl
/usr/rob/lib/plumbing Del Snarf | Look
# to update: cp /usr/$user/lib/plumbing /mnt/
plumb/rules

editor = acme

%g scat
ori
638 items
plot nogr

3-6 augrim, 4 -ym, 5 -ime,
-ime, 7 agrum, algrim. β. 4-6
-arosme, aulgorism(e, augrisme,
7-9 algorism, algorithm. [a. OFr. augorisme, algorisme, augorime;
ad. med.L. algorism-us (cf. Sp. guarismo cipher), f. Arab. al-Khowāra
the native of Khwārazm (Khiva), surname of the Arab mathematician
Abu Ja'far Mohammed Ben Musa, who flourished early in the 9th
c., and through the translation of whose work on Algebra, the
Arabic numerals became generally known in Europe. (Cf. 'Euclid'
= plane geometry.) Algorisme being popularly reduced in OFr.
to augorime, English also shows two forms, the popular augrime,
ending in agrim, agrum, and the learned algorism which passed

window
x
x.gif
y
日本語
%g ech
%g
```

The Terminal Is A File

- *Universality of file IO: Terminal*
  - 3 standard file descriptors that are always open
  - These are opened by the init process
  - fork() clones (most) open file descriptors; so child processes also have them

File Descriptor	Purpose	POSIX Name	stdio stream
0	Standard Input	STDIN_FILENO	stdin
1	Standard Output	STDOUT_FILENO	stdout
2	Standard Error	STDERR_FILENO	stderr

Devices Are Files

- *Many devices have a "device file" in /dev/*
  - This is called a node.
- *Some are real devices*
  - e.g., a mouse, a disk, USB serial adapter
- *Some are virtual devices*
  - `/dev/null` is a “black hole” which forgets all data
  - `/dev/zero` provides infinite null characters
  - `/dev/random` and `/dev/urandom` are (pseudo-) random number generators

    ```
    $ od -vAn -N2 -tu2 < /dev/urandom
    ```

/sys Kernel Controls Are Files

- *File IO in `/sys` file system*
  - `/sys` allows access to kernel internal
  - E.g., device or subsystem status, configuration…
- *Examples*
  - Checking disk health
  - Setting maximum file watchers
  - Listing devices on a bus
- *`ioctl()` syscall*
  - Extra syscall for I/O for things outside of the “normal” universal I/O model
    - But you *just said* the model was “universal”?!
  - Pragmatic: send out-of-band messages to drivers or subsystems
  - E.g., change the speed of a serial port

Disk Partitions

Disk Partitions

- A disk is divided into partitions.
  - Appear in /sys/block/ as subdirectories (among other places)
  - In Windows, partitions get drive letters: C:, D: , etc.
- *A partition is typically used to hold a file system*
  - A file system is a system that manages files and directories
  - Many different types of file systems
  - Each partition can have a different file system
- *Most PCs have at least 2 partitions on their primary disks*
  - One is usually FAT32, holding a (small) filesystem used by EFI to begin the boot process
  - The other is OS-specific: for Linux, probably EXT4, and it stores all other OS and user files
  - Partitions are often nested to support features like RAID or full-disk encryption: not our problem

Disk Partitions (cont’d)

- *User's perspective*
  - File system is a file tree; starts with root directory /
  - Each partition contains a different tree (but more later when talking about mounting)
- *Swap partition*
  - A partition may be used as a swap space for memory management, i.e., paging
  - Nowadays more common to configure a swap *file* stored in the root filesystem
    - This is more complex, and requires coordination between the virtual memory system and the filesystem, but is more flexible
  - If a system is blessed with lots of memory, swap may not be configured at all

Inodes

Inodes

- A file is associated with an inode
  - An inode contains metadata about the file
  - E.g., file type, permissions, owner, timestamps, etc.
  - An inode is identified by a number:

    `ls -li` (or just `ls -i`) shows inode numbers
- `stat()`, `lstat()`, and `fstat()`
  - Functions that work with file metadata mostly from the i-node
  - Read `man 2 stat` and `man 3 stat` for more details

Activity - Inode

- Activity: use `stat()` to display if path is file or directory (15m)
  - Use command line argument to get filename (`arg[1]` likely)
  - Read `man  inode`, especially about `st_mode`
    - Also `S_ISREG(...)`, `S_ISDIR(...)`
  - Print "Regular file" if it's a file
  - Print "Directory" if its a directory
  - Print "Other" otherwise

Hard and Soft Links

Hard Links

- ***Hard links***
  - We can give many names to the same file
  - A hard link is giving another name to an existing file
- *Hard link limitations*
  - Cannot hard link a directory
  - This prevents circular links, i.e., a child directory that links to the parent directory
  - Hard links should be within the same file system, because a hard link is giving another name to an existing file
  - This has security implications

Activity - Hard Links

- Activity: use ln to create a hard link to a file (5m)
  - Read man ln to figure out how to create a hard link
  - Run ls -li for both the original file and the hard link (they're exactly the same)
    - ls -li shows the number of links as well (the third column)
    - # links should increase as more hard links are created
  - Modify content of original file
    - Check contents of the hard link (and vice versa)
    - They should be the same

Footnote: What rm Does

- rm only deletes the hard link
  - rm is actually “unlink” (i.e., the system call `unlink()`)
  - (There's also a more convenient one, `remove()`)
  - File data is only removed when it is not referenced
    - No links left
    - File is not held open (you can delete an open file: why?)

Soft (Or Symbolic) Links

- *Soft links (also called “symbolic” or symlinks)*
  - Unlike a hard link, a soft link is an independent file
  - The content of the file is the path to the original file
  - Similar to a Windows shortcut, but kernel functions interpret the symlink
    - (...also, Windows has actual symlinks, but they are still not conventionally used, I think)
  - There's a system call `symlink()`
- *No limitations like hard links*
  - Symlinks are allowed for directories
  - Symlinks do not have to be within the same file system
  - This also has security implications

Activity - Soft Links

- Activity: create a symlink with ln -s (5m)
  - Use ln -s to create a symlink to a file
  - Run ls -li
    - They each have a unique i-node number, meaning they are two different files
    - The hard link count does not change even if you create a symlink: no new reference to the inode
  - The symlink will point to nothing if the original gets deleted
    - This is called a dangling link

Optional:
Bits - setuid, setguid, sticky

Setuid / Setguid Bits

- *Program Permissions*
  - Normally, programs you run will run with your permission
- **Setuid bit:** if set, the user that runs the program can act as the owner of the program
  - E.g., passwd sets a user's password:
    - It must write to the password file (/etc/shadow), which is owned by the root
    - So, use the setuid bit!
    - When a user runs passwd, the program can act as root to modify the password file
    - Similar to how a syscall implicitly transitions to kernel mode
- **Setgid bit:** if set, the user that runs the program can act as if the user belonged to the group of the program

Sticky Bit

- **Sticky bit:**
  - Can be set on a *shared directory* for better control.
  - When set, only able to delete/rename file if:

    1) you own it

    2) you have write permission for it

    (affects the directory, not the file access permissions)

Sticky Example

- *Situation 1: Regular Directory*
  - Create a shared_photos/ directory that is write-open for others (e.g., rw-rw-rw-).
  - User dr-evil creates a file selfie.jpg in it.
  - User boogieman can delete selfie.jpg.
- *Situation 2: Sticky Bit!*
  - Set sticky bit on shared_photos/

    ```
    chmod +t shared_photos/
    ```
  - User dr-evil creates a file selfie.jpg in it.
  - User boogieman cannot delete selfie.jpg.

VFS - Virtual File System, Mount/Unmount

VFS (Virtual File System)

- *VFS (Virtual File System)*
  - defines an interface that different file systems can implement
  - Interface includes: open, read, write, close, etc.; VFS in kernel defines a function to handle each
  - It's not a file system of real files, just software pretending to be a file system
- If a file system implements this interface, it can be used as a Linux file system
  - This is how we implement `/sys`, `/proc`, `/dev`, …

Mounting

- *Linux presents all file systems as a **single tree***
  - Starts at root directory /
  - In reality, **this single file tree is actually multiple file trees combined together**
- *Recall*
  - A partition contains a file tree
  - There can be multiple partitions on a single disk
  - There can be multiple disks for a single machine

Mounting and Unmounting

- Mounting combines multiple file trees into one
  - All file systems (from different partitions/disks) are mounted and form a single file tree
- **`mount`** command mounts a file tree (a file system) to a specific directory
  - This target directory is called a mount point
  - The **`mount`** command also shows the current setup (shows the same information as /proc/mounts)
  - The **`umount`** command unmounts a file system

Summary

- *Everything is a file*
  - Use file operations to access almost anything
  - /proc for process info
  - /dev for devices
  - /sys for system info
- **Partitions** split up disks
- **Inodes** used for meta data about each file/directory
- **Hard and soft links** allow two entries for one file
- **Mounting** places one file tree inside another