Take-away points

UNIX background

UNIX file system implementation

- **On-disk structures: directory, i-node, data block**
- **Kernel data structures: file descriptor, open-file table, buffer cache**

Process

Shell

UNIX Background

Earliest: 1969-1970 on PDP-7/9

Usage of PDP-11

☺ textual material; trouble data; recording/checking phone service orders.

Demonstrate: not expensive ($40,000 machine, 2 man years)

Simplicity, elegance, and ease of use.

A lot of programs available on UNIX: …….

System environment

PDP-11/45: 16-bit word (8b byte) computer, 144KB core memory, ~x0M disk

UNIX is written in C!!!

(Pilot is written in AlGOL; nucleus ?; Multics written in PL/I; Pilot written in Mesa)

File System

Ordinary files

Directory files

i-node

Linking (link can only be done on non-directory files)
    Command: link (hard link!)
Directory structure has to be a rooted TREE (no cycle; each dir has only one parent)
    Why? Easy

Special files

    /dev (read write would lead to device activation, device type and sequence number; the former shows what function to use; the latter helps identify the device)

    Advantage and disadvantage

    + file and I/O are similar

    +file name and device names are similar

    +regular and special files have similar protection mechanism

How to locate a data block

    Root

    Directory file structure

    i-node array

    i-node structure: owner; protection bits; physical address/location of the file; file size;time of last modification; #links to file; directory/special file/large-or-small;

    8 pointers to 8 blocks (each block is 512-bytes) [**block size!!** Logically divided]

    Difference between small/large file's i-node …

    Caching effects:

        Per-process file descriptor (points to open file table, inherited by children)

        Global open-file table (every open corresponds to one; read-write index to the file, each descriptor can be shared across processes)

Buffer cache: write/read, always first check the buffer in the system; there is also data structure to bookmark `dirty' block, those blocks will be written back to the disk at some time later on

Protection

User-owner

7-bit protection

Setuid (this is patented by AT&T)

Sudo is implemented upon setuid ☺

Enable much more capable acces scontrol!!

Can be done by chmod4xxx; chmod6xxx

How to implement mount

Essentially, mount is just to maintain a system table (across process)

<i-number, device-name> → <device-name>

During file-access, at each point, the mount table is looked up to make sure whether it is necessary to transfer a different device' root

i.e., whether needs to change i-number to 1 and device-name to the value device

significance of i-node

1. **An easy, short, unambiguous name**
2. **Easy to go through (better than diretory)**

Multics file system

tree structure, directories and files

each file and directory is a segment

dir seg holds array of "branches"

name, length, ACL, array of block #s, "active"

unique ROOT directory path names: ROOT > A > B

note there are no inodes, thus no i-numbers

Comment [s1]: This is a huge difference. Because there is no i-node; a lot of things are different; without this layer of indirection (i-node); file BELONGS to a directory; no way to establish link (almost no way)!!!

so "real name" for a file is the complete path name

o/s tables have path name where unix would have i-number

presumably makes renaming and removing active files awkward

no hard links

(it is hard to have hard link, because of using symbolic ...)

Processes

Each process has one continuous virtual space (different from multics)

Three logical segments.

Lowest: code segment

Above code segment (at 8K aligned boundary): heap

Highest: stack

Fork (label)

Note: ther eis a label, slight different from fork current

Jump to label.

The whole core image is copied!!

File descriptors and open fiels are shared!

Pipe

Pipe return file descriptor (normal file descriptor!!)

Inherited by child processes

When processes have the same ancestor, they can use pipe to communicate

Execute

Code, data are all replaced; only file descriptors are unchanged

Wait; exit

shell

command line interpretor

redirection ☺ by default, shell program has 0, 1, 2 files initialized (0 is stdin, 1 is stdout, 2 is stderr); <, > will help to close original 0, 1 and change these descriptor to the specified files.

Help programming: programming always only need to program 0,1,2

Based on redirection, filter implementation is obvious

Init

Init process creates shell process; if shell dies (e.g., exit or password does not match), simply kills the shell and restart a new one.