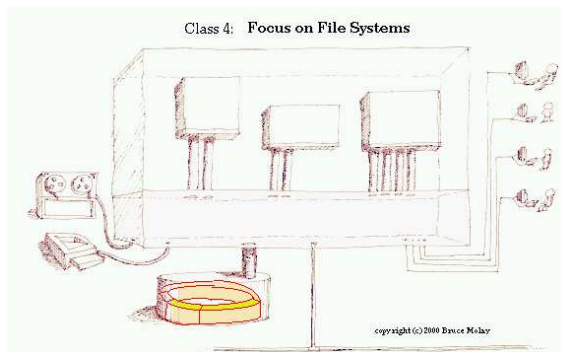


page000.html

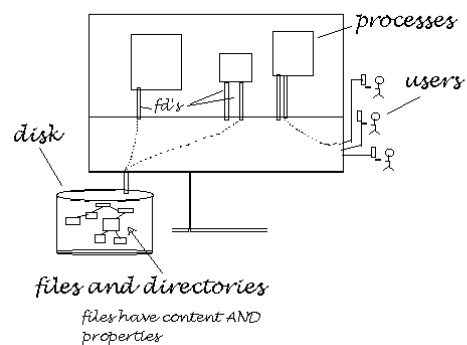
000



page001.html

Directory Trees and Disks

001

4) Focus on File Systems*The story so far*

page002.html

002

Tonight: focus on file systemsWhat users see: one big tree of dirs and files*The tree contains files, file info, and directories*The reality: one or more stacks of magnetic disks or memory chips.*How can multiple stacks of disks be made to look like a single tree of directories and files?*

- What is the internal structure of a 'file system'?
- How do multiple disks merge into one tree?

project: write pwd

page003.html

perror and errno

003

2) But first, a brief interruption to discuss error handling

<code>errno</code> error code	<code>perror(char *)</code> prints a message	<code>strerror(errno)</code> returns a string
----------------------------------	---	--

Fact: system calls return -1 on error

But: what *is* the specific error?

example: `open()` can fail for many reasons

Q: How can the program determine which error it is?

A: The kernel sets a global value, known as `errno`, to a code in `/usr/include/errno.h`

Q2: How can a program tell the user what is the cause of the error?

A2: `perror(str)` prints "str: description"

Q3: What to do?

A: it depends

`perror_demo.c`

page004.html

User View of Directories

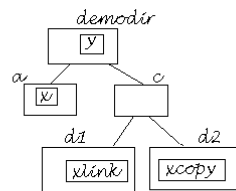
004

3) User view and Commands for file system

first, let's become familiar with the major fs-related commands by building a demo directory tree

commands

<code>mkdir</code>	<code>cat</code>
<code>rmdir</code>	<code>cp</code>
<code>cd</code>	<code>rm</code>
<code>pwd</code>	<code>mv</code>
	<code>ln</code>
<code>du</code>	
<code>ls -la</code>	
<code>ls -R</code>	



page005.html

005

4) Almost no limits to tree structure

- directories can contain lots of files
- directory depths can exceed the capacity of most fs commands

Ex. `dig`
`while true`
`do`
 `mkdir s`
 `cd s`
`done`

What does `du` do with this directory structure? Or `tree`
`ls -lR`, `find`, ...?

Note: some system maintenance tools might fail on these.

see also: `dig_verbose`



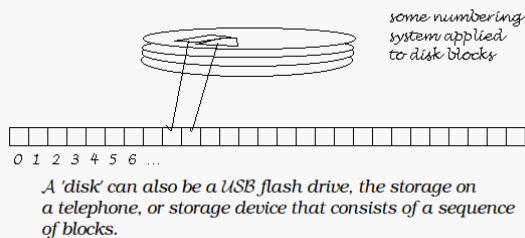
page006.html

Structure of a File System

006

5) Face Reality: a disk is not a tree
we are not 'in' a directory

6) A disk is a stack of magnetic platters,
 disks, tracks, sectors, viewable as a
 sequence of DISK BLOCKS

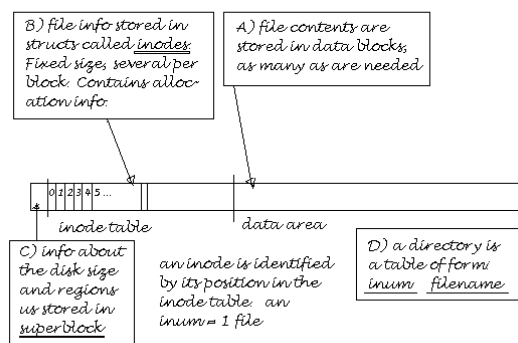


page007.html

007

7) How can a numbered sequence of blocks
store files, file info, and directories ?

Ans: divide the disk into three regions; structure them sp:



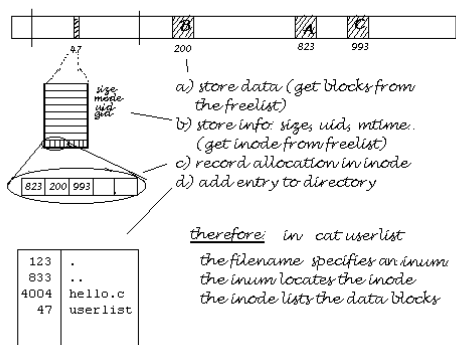
page008.html

Block Allocation Lists

008

8) Using this model to understand
creating a file: e.g. who > userlist

say the file requires three disk blocks of storage...



page009.html

Details of Allocation Lists

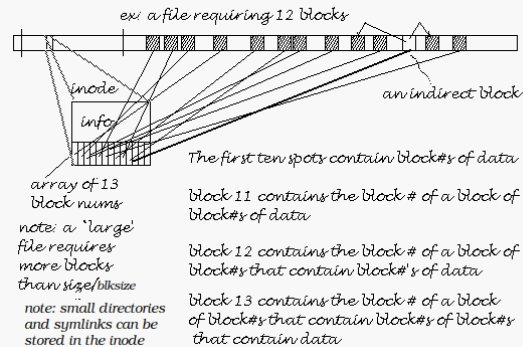
009

Details of recording disk allocation

fact a large file is stored in many blocks

fact the inode holds the info about which blocks contain the data

problem: how can a fixed-sized inode store a *long* block list?

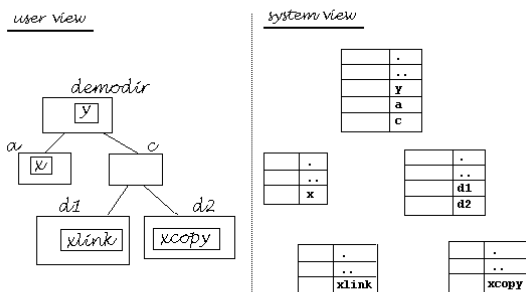


page010.html

010

9) Using this model to revisit demodir

now that we know the internal structure of dirs, we can see what is *really* going on in our demodir structure

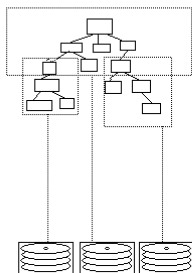


page011.html

Multiple disks, one tree

011

Question2: How can many disks form one tree?



Each disk has a device ID.

Ans: Each disk contains a tree structure. That tree can be 'mounted' on another tree at any directory.

stat() info includes the inode *and* the device. That pair uniquely identifies the file.

Each file on a disk has an inode number.

The 'mount' command reports what disks are attached to the tree at what directories. The directories are called 'mount-points'.

page012.html

012

10) The System Calls for standard file ops

command	syscall	action
rm	unlink()	removes a link if links == 0, deallocate
rmdir	rmdir()	delinks a directory
ln	link()	creates a new link
mv	link() then unlink() now rename()	
mkdir	mkdir()	creates new directory

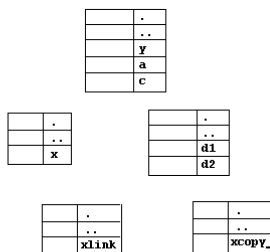
page013.html

Writing pwd

013

11) writing pwd

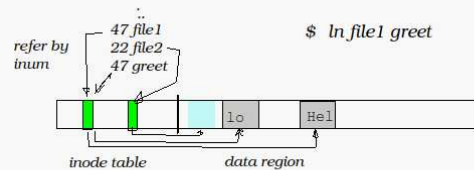
pwd prints the path to the current directory. But, the current directory only knows itself as "." How can its location in the tree be determined?



page014.html

Multiple Names for Same File

014

Hard Links: Different Names for the Same File

One file: two dir entries (i.e. two links)

You can use either name. `chmod o+x file1`, `chmod u-r greet`

Q: What if you `rm file1`? What happens to `greet`?

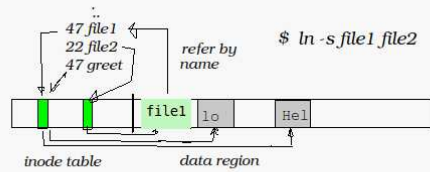
A:

Note: In a city, each street has its own list of house numbers
Each file system has its own inode table (inums)

But: A hard link cannot refer to an inode on a different
file system. 123 Main St vs 123 Maple St

page015.html

015

Symbolic Links: Another Way for Different Names, Same File

There are two files : the symlink is a file that contains the name of its referent

You can use either name : cat file1, cat file2

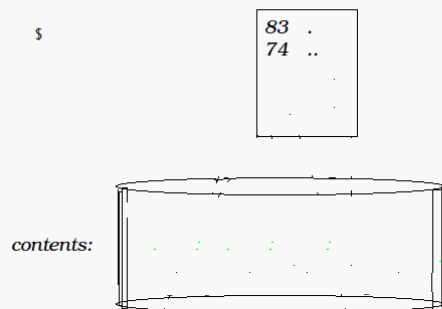
Q: What if you rm file1? What happens to file2 ?

A:

But: A symlink can contain the name of a file/dir on a different filesystem.

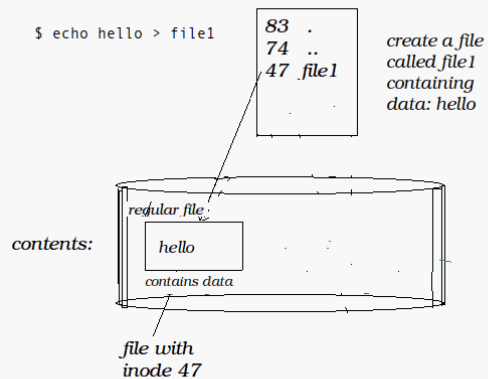
page016.html

016

Making Links: Time Lapse Images

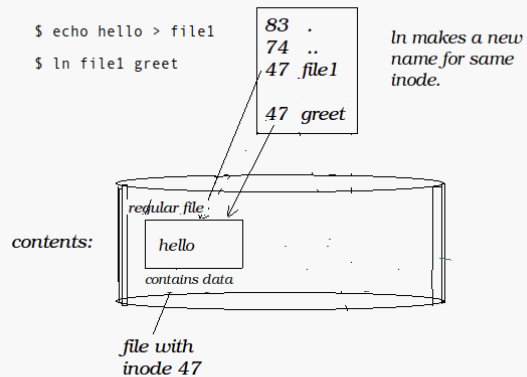
page017.html

017

Making Links: Time Lapse Images

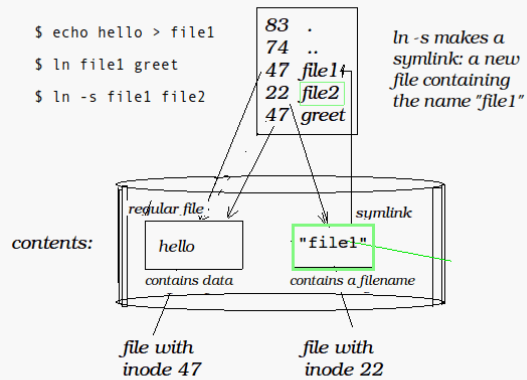
page018.html

018

Making Links: Time Lapse Images

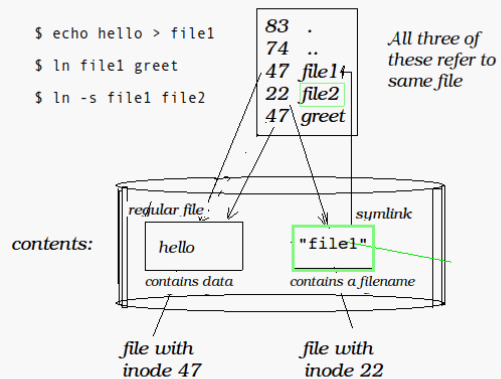
page019.html

019

Making Links: Time Lapse Images

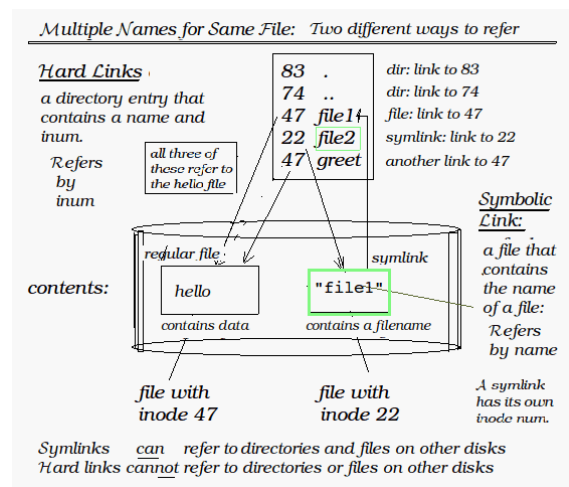
page020.html

020

Making Links: Time Lapse Images

page021.html

021



page022.html

022

