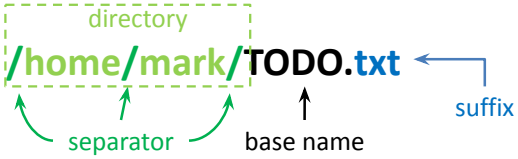
File Names and Directories

- 11B Name-space Semantics
- 11E Name-space Representation
- 11L Disk Partitioning and Volumes

File Names and Name Binding

- file system knows files by internal descriptors
- users know files by names
  - names more easily remembered than disk addresses
  - names can be structured to organize millions of files
- file system responsible for name-to-file mapping
  - associating names with new files
  - changing names associated with existing files
  - allowing users to search the name space
- there are many ways to structure a name space

What is in a Name?



- suffixes and file types
  - file-to-application binding often based on suffix
    - defined by system configuration registry
    - configured per user, or per directory
  - suffix may define the file type (e.g. Windows)
  - suffix may only be a hint (magic # defines type)

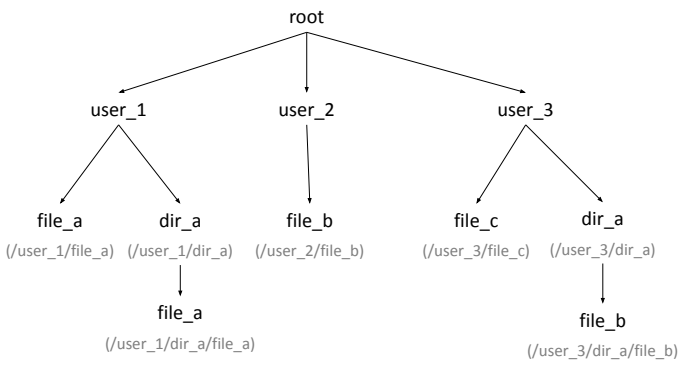
Flat Name Spaces

- there is one naming context per file system
  - all file names must be unique within that context
- all files have exactly one true name
  - these names are probably very long
- file names may have some structure
  - e.g. CAC101.CS111.SECTION1.SLIDES.LECTURE_13
  - this structure may be used to optimize searches
  - the structure is very useful to users
  - the structure has no meaning to the file system

Hierarchical Namespaces

- directory
  - a file containing references to other files
  - it can be used as a naming context
    - each process has a *current working directory*
    - names are interpreted relative to directory
- nested directories can form a tree
  - file name is a path through that tree
  - directory tree expands from a *root* node
    - *fully qualified* names begin from the root
  - may actually form a directed graph

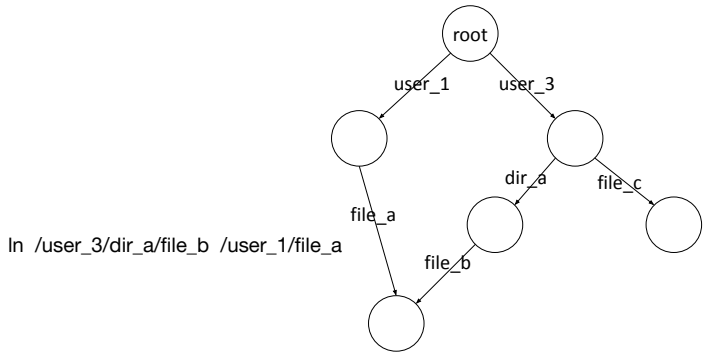
A rooted directory tree



True Names vs. Path Names

- Some file systems have “true names”
- DOS and ISO9660 have a single “path name”
  - files are described by directory entries
  - data is referred to by exactly one directory entry
  - each file has only one (character string) name
- Unix (and Linux) … have named links
  - files are described by I-nodes (w/unique I#)
  - directories associate names with I-node numbers
  - many directory entries can refer to same I-node

Hard Links: example

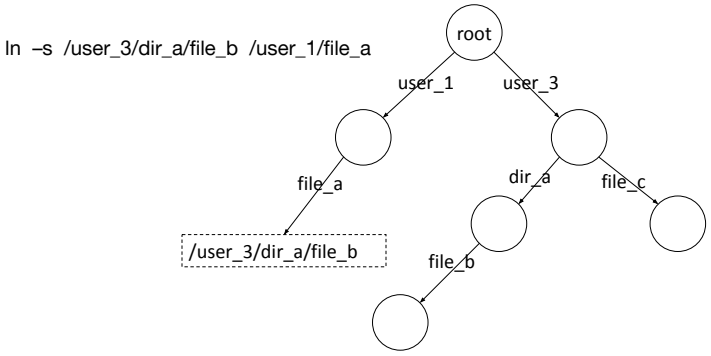


Both names now refer to the same I-node

Unix-style Hard Links

- all protection information is stored in the file
  - file owner sets file protection (e.g. read-only)
  - all links provide the same access to the file
  - anyone with read access to file can create new link
  - but directories are protected files too
    - not everyone has read/search access to every directory
- all links are equal
  - there is nothing special about the owner‘s link
  - file is not deleted until no links remain to file
  - reference count keeps track of references

Symbolic Links: example



/user_1/file_a is now a macro for /user_3/dir_a/file_b

Symbolic Links

- another type of special file
  - an indirect reference to some other file
  - contents is a path name to another file
- Operating System recognizes symbolic links
  - automatically opens associated file instead
  - if file is inaccessible or non-existent, the open fails
- symbolic link is not a reference to the I-node
  - symbolic links will not prevent deletion
  - doesn’t guarantee ability to follow specified path
  - Internet URLs are similar to symbolic links

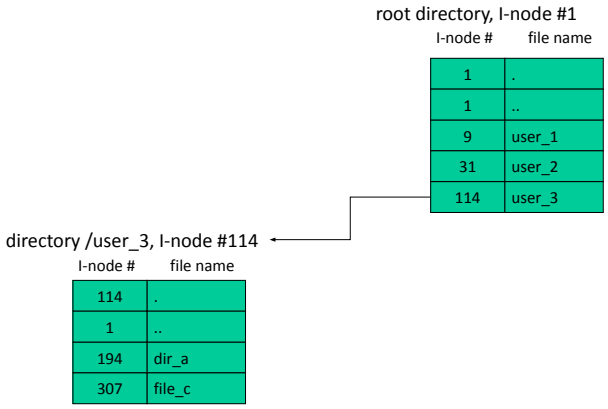
Generalized Directories: Issues

- Can there be multiple links to a single file?
  - who can create new references to a file?
  - if first reference is deleted, does the file go away?
  - can the file's owner cancel other peoples' access?
- Can clients work their way back up the tree?
- Is the namespace truly a tree
  - or is it an a-cyclic directed graph
  - or an arbitrary directed graph
- Does namespace span multiple file systems?

Directories are usually files

- directories are a special type of file
  - used by OS to map file names into the associated files
- a directory contains multiple directory entries
  - each directory entry describes one file and its name
- user applications are allowed to "read" directories
  - to get information about each file
  - to find out what files exist
  - perhaps through abstracted interfaces - readdir(2)
- only Operating System is allowed to write them
  - the file system depends on the integrity of directories

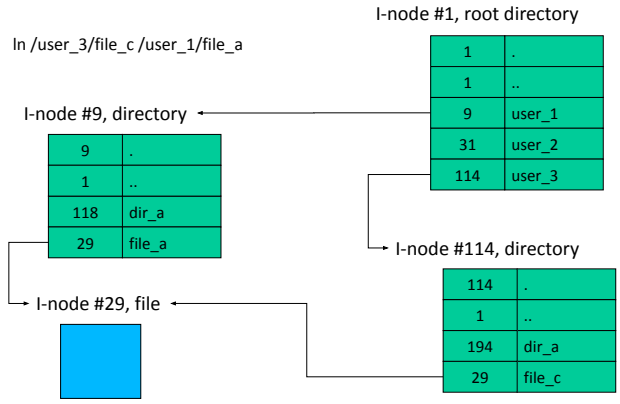
UNIX Directories



(Example: UNIX Directories)

- file names separated by slashes
  - e.g. /user_3/dir_a/file_b
- the actual file descriptors are the I-nodes
  - directory entries only point to I-nodes
  - association of name with I-node called a "link"
  - multiple entries can point to same I-node
- contents of a Unix directory entry
  - name (relative to this directory)
  - pointer to the I-node of the associated file

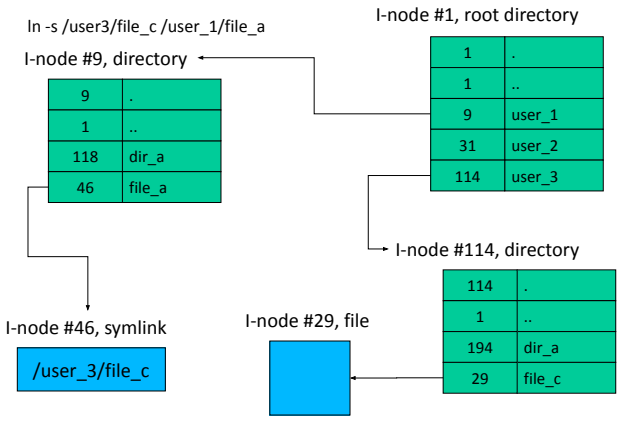
Hard Links - example



Hard Links and De-allocation

- there can be many links to a single I-node
  - all links are equivalent
  - no link enjoys a preferential (e.g. master) status
- file exists as long as at least one link exists
- most easily implemented w/reference counts
  - increment with every link creation
  - decrement with every unlink deletion
  - delete file when reference count goes to zero
- more efficient than garbage collection

Symbolic Links - example



DOS Directories

root directory, starting in cluster #1

file name	type	length	…	1st cluster
user_1	DIR	256 bytes	…	9
user_2	DIR	512 bytes	…	31
user_3	DIR	284 bytes	…	114

Directory /user_3, starting in cluster #114

file name	type	length	…	1st cluster
..	DIR	256 bytes	…	1
dir_a	DIR	512 bytes	…	62
file_c	FILE	1824 bytes	…	102

(Example: DOS Directories)

- File/dir names separated by back-slashes
  - e.g. \user_3\dir_a\file_b
- directory entries are the file descriptors
  - as such, only one entry can refer to each file
- contents of a DOS directory entry
  - name (relative to this directory)
  - type (ordinary file, directory, ...)
  - location of first cluster of file (rest via FAT)
  - length of file in bytes
  - other privacy and protection attributes

File Systems and Disks

- independence of multiple disks
  - they can be turned on and off independently
  - they can be backed-up and restored independently
  - some are physically removable (diskette, CD, Flash)
- file system spanning multiple (non RAID) disks is risky
  - losing one disk could lose parts of many files
  - better to lose all of some files and none of others
  - disks can be checked, repaired, restored independently
- people do put multiple file systems on a single disk
  - partitioning a physical disk into multiple logical disks

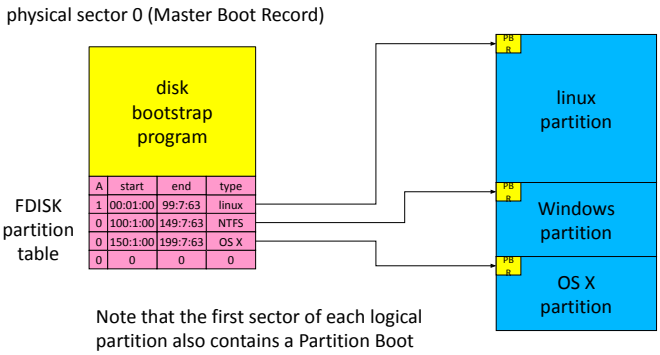
Logical Disk Partitioning

- divide physical disk into multiple logical disks
  - perhaps in disk driver, perhaps through meta-driver
  - rest of system sees partitions as separate devices
- typical motivations
  - permit multiple OS to coexist on a single disk
    - e.g. a notebook that can boot either Windows or Linux
  - fire-walls for installation, back-up and recovery
    - e.g. separate personal files from the installed OS file system
  - fire-walls for free-space
    - running out of space on one FS doesn't affect others
  - want different types of file systems
    - performance differences, inter-operability

Disk Partitioning Mechanisms

- some designed for use by a particular OS
  - e.g. Linux LVM partitions, understood by GRUB
- some designed to support multiple OS
  - e.g. DOS FDISK partitions, understood by BIOS
- there may be hierarchical partitioning
  - e.g. logical volumes within an FDISK partition
- should be possible to boot from any partition
  - direct from BIOS, or w/help from L2 bootstrap

example: FDISK Disk Partitioning



Note that the first sector of each logical partition also contains a Partition Boot Record, which will be used to boot the operating system for that partition.

A Simple Bootstrap Procedure

1. BIOS tries designated devices in a configurable order
2. Read MBR (Block 0) from chosen device and check for validity, and branch to it.
3. Scan FDISK table to find ACTIVE partition, read PBR (block 0) from chosen partition, and branch to it.
4. PBR loader finds and loads 2nd level bootstrap (e.g. GRUB), and branches to it.
5. 2nd level bootstrap (often a very sophisticated program) finds and loads operating system.
6. all early-stage loading is done by calls to BIOS … until OS device drivers are loaded and configure

Working with multiple file systems

- finding files is easy if there is only one file system
  - any file we want must be on that one file system
  - directories enable us to name files within a file system
- what if there are multiple file systems available?
  - somehow, we have to say which one our file is on
- how do we specify which file system to use?
  - one way or another, it must be part of the file name
  - it may be implicit (e.g. same as current directory)
  - we need some way of specifying which file system

Specifying Which File System

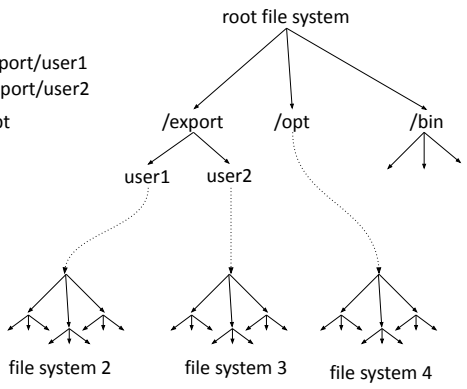
- we could specify physical device it resides on
  - e.g. "/devices/pci/pci1000,4/disk/lun1/partition2"
    - that would get old real quick
- we could assign logical names to our partitions
  - e.g. "A:", "C:", "D:"
    - you only have to think physical when you set them up
    - but you still have to be aware multiple volumes exist
- we could weave multi-file-system name space
  - e.g. Unix mounts

Unix File System Mounts

- goal
  - make many file systems appear to be one giant
  - users need not be aware of file system boundaries
- mechanism
  - /etc/mount *device* on *directory*
  - creates a warp from the named directory to the top of the file system on the specified device
  - any file name beneath that directory is interpreted relative to the root of the mounted file system

Unix - Mounted File Systems

mount filesystem2 on /export/user1
mount filesystem3 on /export/user2
mount filesystem4 on /opt



Reading and Assignments

Reading:

- Arpaci C41 … Fast File System implementation
- FUSE … file systems implemented in user-mode

Projects:

- get project 3A mostly working