

Focus on File Systems

Writing `pwd`

Prof. Seokin Hong

Kyungpook National University

Fall 2018

Objectives

■ Ideas and Skills

- User's view of the Unix file system tree
- Internal structure of Unix file system: inodes and data blocks
- How directories are connected
- Hard links, symbolic links: ideas and system calls
- How pwd works
- Mounting file systems

■ System Calls and Functions

- mkdir, rmdir, chdir
- link, unlink, rename, symlink

■ Commands

- pwd

Contents

- 4.2 User's View of File System
- 4.3 Internal Structure of Unix File System
- 4.4 Understanding Directories
- 4.5 Writing pwd
- 4.6 Multiple File Systems: A Tree of Trees

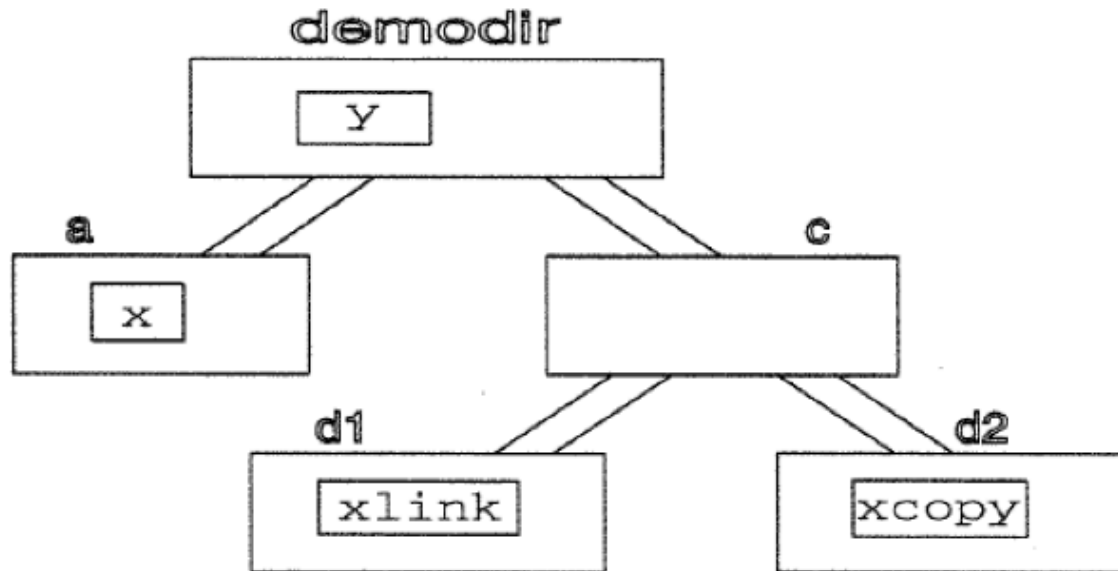
File System ?

- A *file system* is the methods and data structures that an operating system uses to keep track of files on a disk or partition
- Aspect of file systems
 - **Space management** : The file system is responsible for organizing files and directories and keeping track of which areas of the media belong to which file and which are not being used.
 - **Filenames** : used to identify a storage location in the file system.
 - **Directories** : allow the user to group files into separate collections.
 - **Metadata** : other bookkeeping information (e.g., file size, last modified time, owner ID, etc.) associated with each file within a file system.



Directories and Files

- What users see: one big tree of directories and files
 - Each directory can contain files or directories.



Directory Commands

\$ mkdir demodir

\$ cd demodir

mkdir: create a new directory with a specified name

rmdir: removes a directory

mvdir : rename a directory. move a directory from one place to another

cd : move from you from one directory to another

\$ mkdir b oops

\$ mv b c

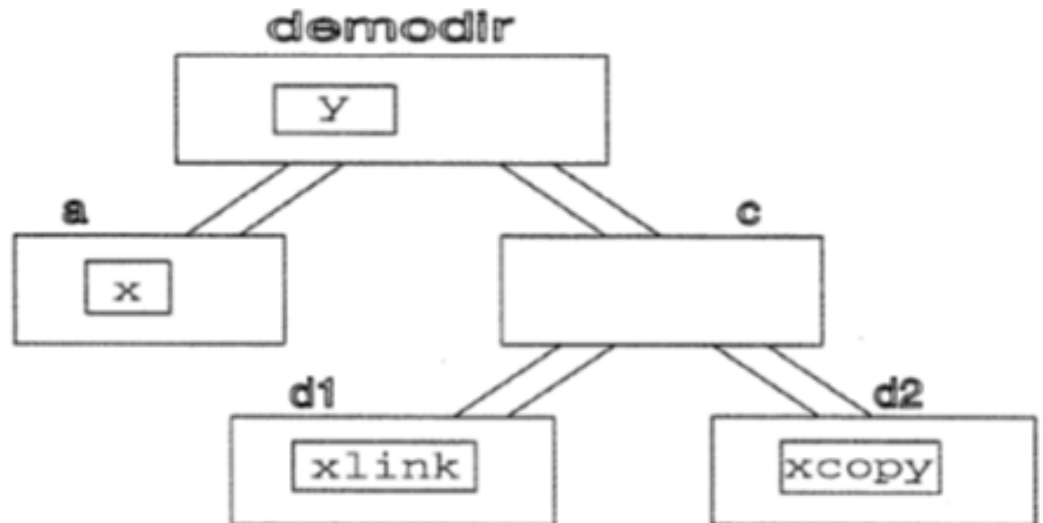
\$ rmdir oops

\$ cd c

\$ mkdir d1 d2

\$ cd ../../

\$ mkdir demodir/a



File Commands

```
$ cd demodir
```

```
$ cp /etc/group x
```

```
$ cp x copy.of.x
```

```
$ mv copy.of.x y
```

```
$ mv x a
```

```
$ cd c
```

```
$ cp ../a/x d2/xcopy
```

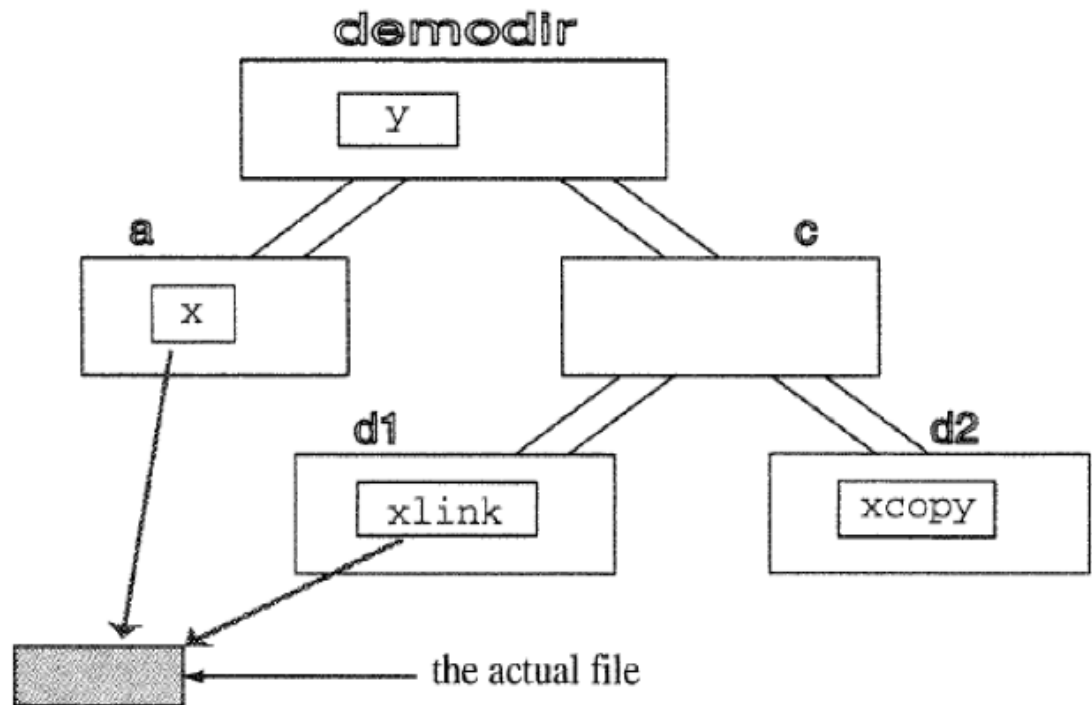
```
$ ln ../a/x d1/xlink
```

```
$ ls > d1/xlink
```

```
$ cd ../..
```

```
$ cat demodir/a/x
```

ln : make a **link** which is a pointer to a file



Unix Commands operate on entire tree structure

■ ls -R

- Lists the contents of the specified directory and all its subdirectories
- Ex) \$ ls -R demodir/

■ chmod -R

- Changes the permission bits of files in applying the changes to all files in subdirectories.
- Ex) \$ chmod -R 755 directory-name/

Summary of the User's View of the UNIX File System

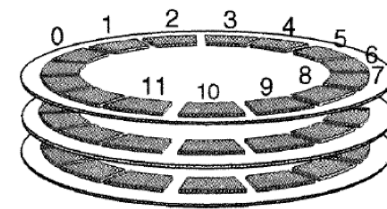
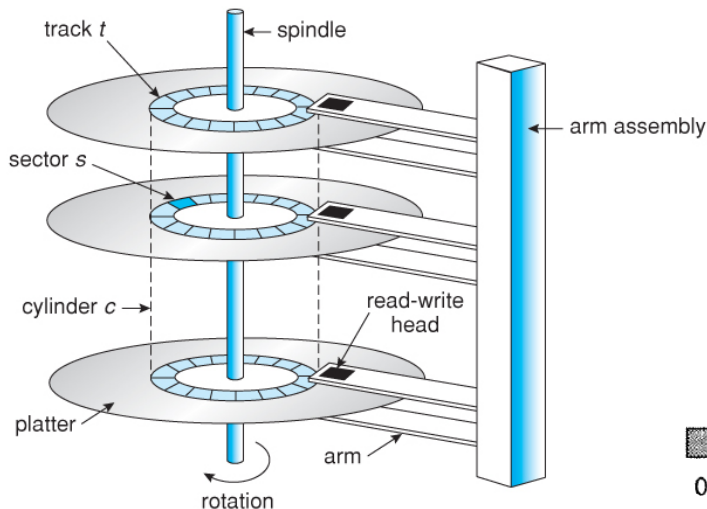
- The disk appears as a tree structure of directories
- All files on a Unix system reside in this structure
- Unix includes many programs for working with the objects of this structure

Contents

- 4.2 User's View of File System
- 4.3 Internal Structure of Unix File System
- 4.4 Understanding Directories
- 4.5 Writing pwd
- 4.6 Multiple File Systems: A Tree of Trees

From Platters to an Array of Blocks

- A disk is a stack of magnetic platters
 - Each platter is organized into **tracks**
 - Track is divided into **sectors** that is the smallest physical storage unit on disk. Each sector stores some number of bytes (ex: 512 bytes)
 - A block is a group of sectors that the operating system can address. A block might be one sector or it might be several sectors
- Assigning numbers to the blocks makes a disk look like an array



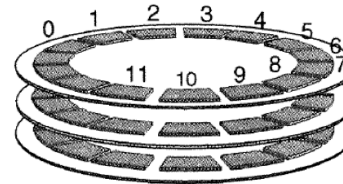
Assigning numbers to disk blocks makes a disk look like an array.



From an Array of Blocks to Three Regions

■ Divide the array of blocks into tree sections:

- **Superblock**
- **inode Table**
- **Data Area (Data blocks)**



Assigning numbers to disk blocks makes a disk look like an array.

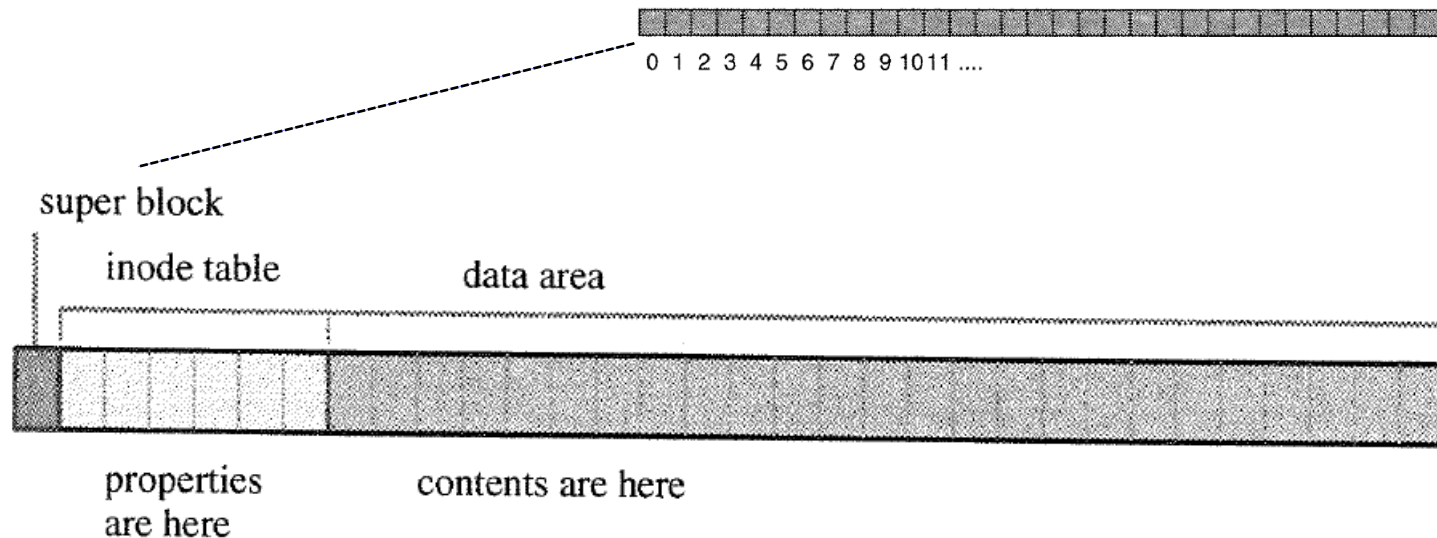


FIGURE 4.4

The three regions of a file system.

From an Array of Blocks to Three Regions

■ Divide the array of blocks into tree sections:

○ **Superblock**

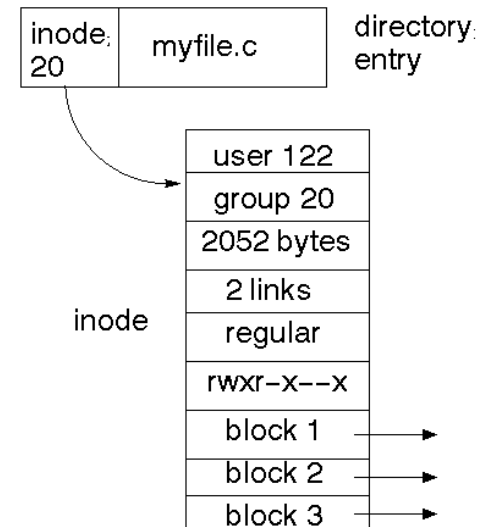
- Contains information about the organization of the file system
- Ex) size of each area, location of unused data block

○ **inode Table** : an array of inodes.

- **inode** : a structure containing the properties of a file such as size, user ID of the owner, the time of last modification.
- Every file stored in the file system has one inode in the table
- Each inode is identified by its position (e.g., inode 2 is the third struct in the inode table of the file system)

○ **Data Area (Data blocks)**

- Contains contents of files
- If a file is larger than a block size, the contents of the file are stored in as many blocks as are needed.

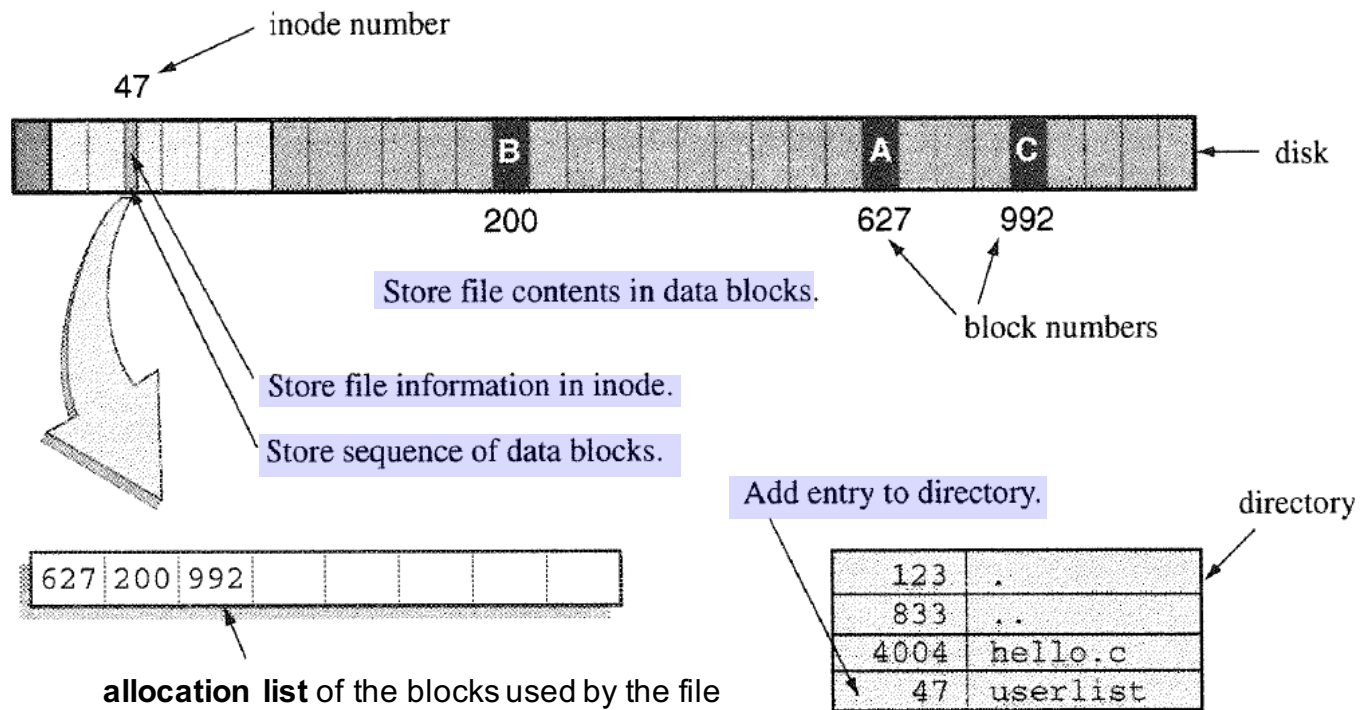


File System in Practice: Creating a File

- Create a file : `$ who > userlist`
- What happens when the userlist file is created?
 - 1) **Store properties:** Kernel allocates a free inode and records the properties in the inode
 - 2) **Store data:** Kernel allocates free blocks and copies file data from kernel buffers into the allocated blocks
 - 3) **Record allocation:** Kernel records the sequence of the allocated block numbers in the disk allocation section of the inode
 - 4) **Add filename to directory:** Kernel adds an entry (a pair of the inode number and filename) to the directory for the file

File System in Practice: Creating a File

- Create a file : \$ who > userlist
- What happens when it is created the userlist file?



File System in Practice: How Directories Work

■ Internal structure of directory

- Directory is a kind of file that contains a list of names of files
- The internal structure of directory is a kind of table that contains pairs of inode number and filename

i-num	filename
2342	.
43989	..
3421	hello.c
533870	myls.c

■ We can find the inode numbers of files with “ls -li” command

```
$ ls -li demodir
```

```
177865 .  
529193 ..  
588277 a  
200520 c  
204491 y
```

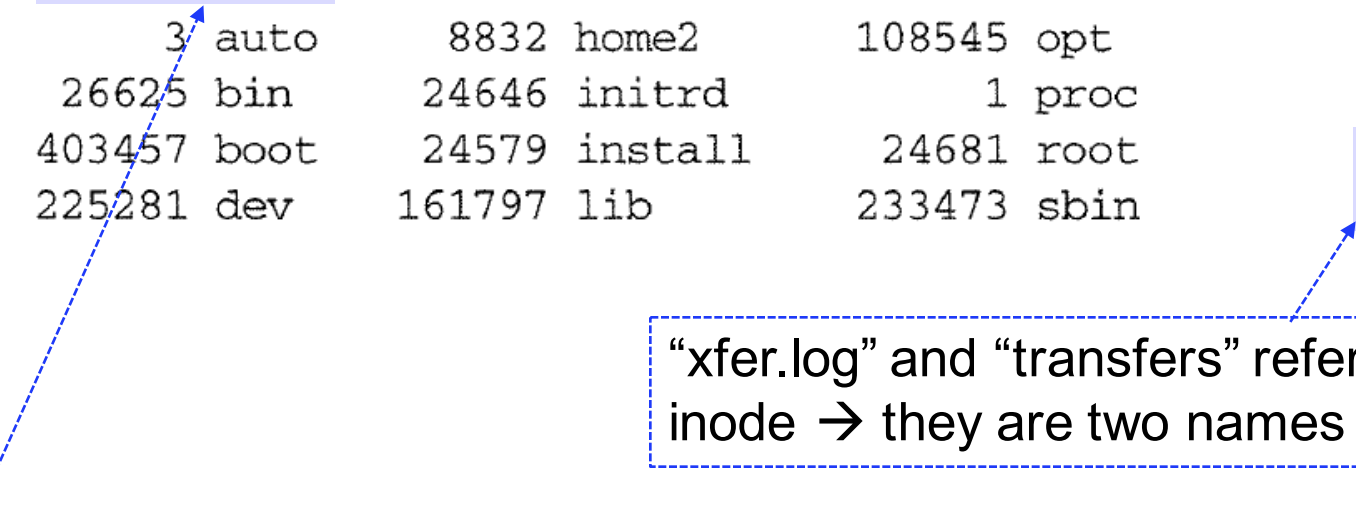
```
$
```

The filename y refers to inode number 204491 which means that the properties of the file is stored in 204492th inode in the inode table

File System in Practice: How Directories Work

- There can be multiple links to the same file

```
$ ls -la /
2 .          28673 etc          11 lost+found 438292 shlib
2 ..         311297 home        4097 mnt       40961 tmp
3 auto       8832 home2        108545 opt     18433 usr
26625 bin    24646 initrd       1 proc        10241 var
403457 boot  24579 install      24681 root     183 xfer.log
225281 dev   161797 lib         233473 sbin    183 transfers
```



“xfer.log” and “transfers” refer to the same inode → they are two names for the same file.

For the root directory (/), “.” and “..” refer to the same inode
→ they refer to the same directory

How?

When unix command `mkfs` creates a file system, `mkfs` sets *the parent of the root directory to point to itself*

File System in Practice: How “cat” Works

- Read a file : `$ cat userlist`
- What happen when you read a file?
 - 1) **Search the directory for the filename:** Kernel searches the directory for the entry containing the string `userlist` and get the associated inode number from the entry
 - 2) **Read inode and get a list of data blocks:** Kernel locates the corresponding inode with the inode number in inode region of the file system, and get a list of data blocks
 - Locating an inode requires a simple calculation, because all inodes are the same size and each block contains a fixed number of inodes
 - 3) **Go to the data blocks and read contents:** Kernel steps through the data blocks, copying bytes from the disk to the kernel buffers and back to the array (buffer) in user space

File System in Practice: How “cat” Works

- Read a file : `$ cat userlist`
- What happen when you read a file?

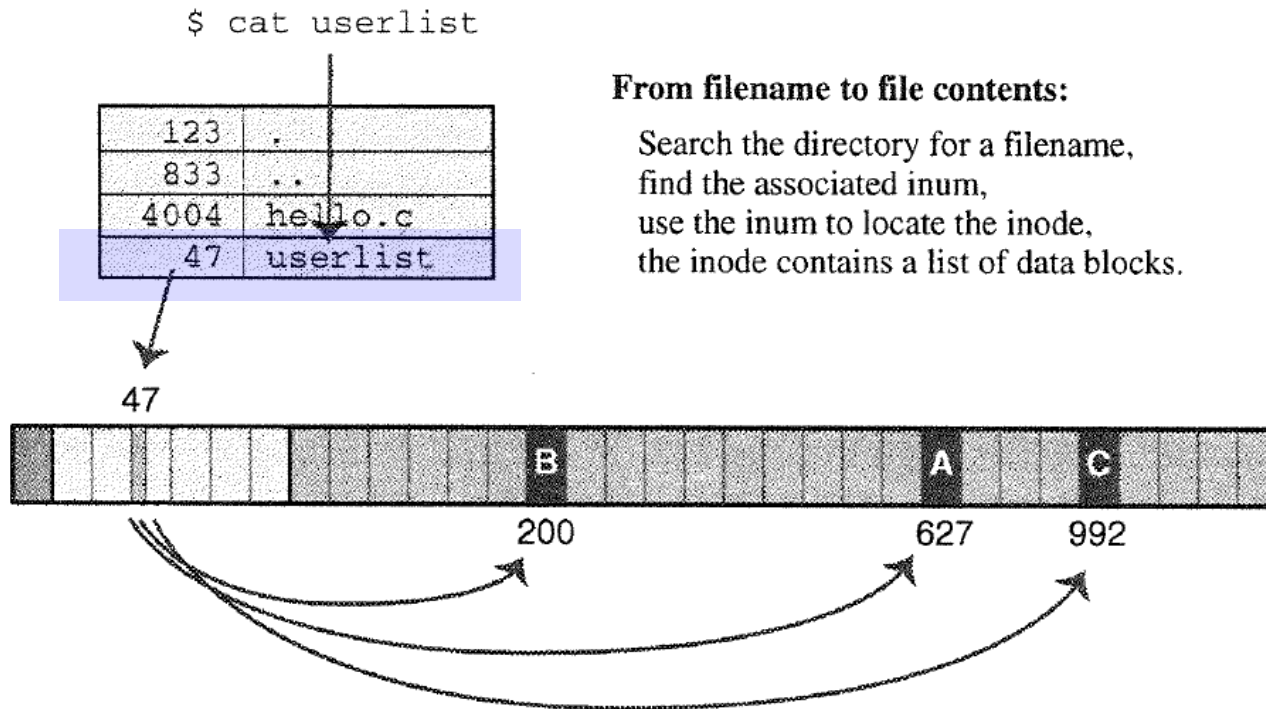


FIGURE 4.6

From filename to disk blocks.

File System in Practice: How “cat” Works

- What happen when you attempt to **open** a file that you do not have read or write permission?
 - 1) Kernel uses the filename to find the inode number
 - 2) Kernel uses the inode number to locate the inode
 - 3) Kernel finds the permission bits and the user ID of the owner of the file
 - 4) If your user ID, the ower ID of the file, and the permission bits do not allow access, then the “open()” system call returns -1 and sets errno to EPERM.

inodes and Big Files

■ Situation

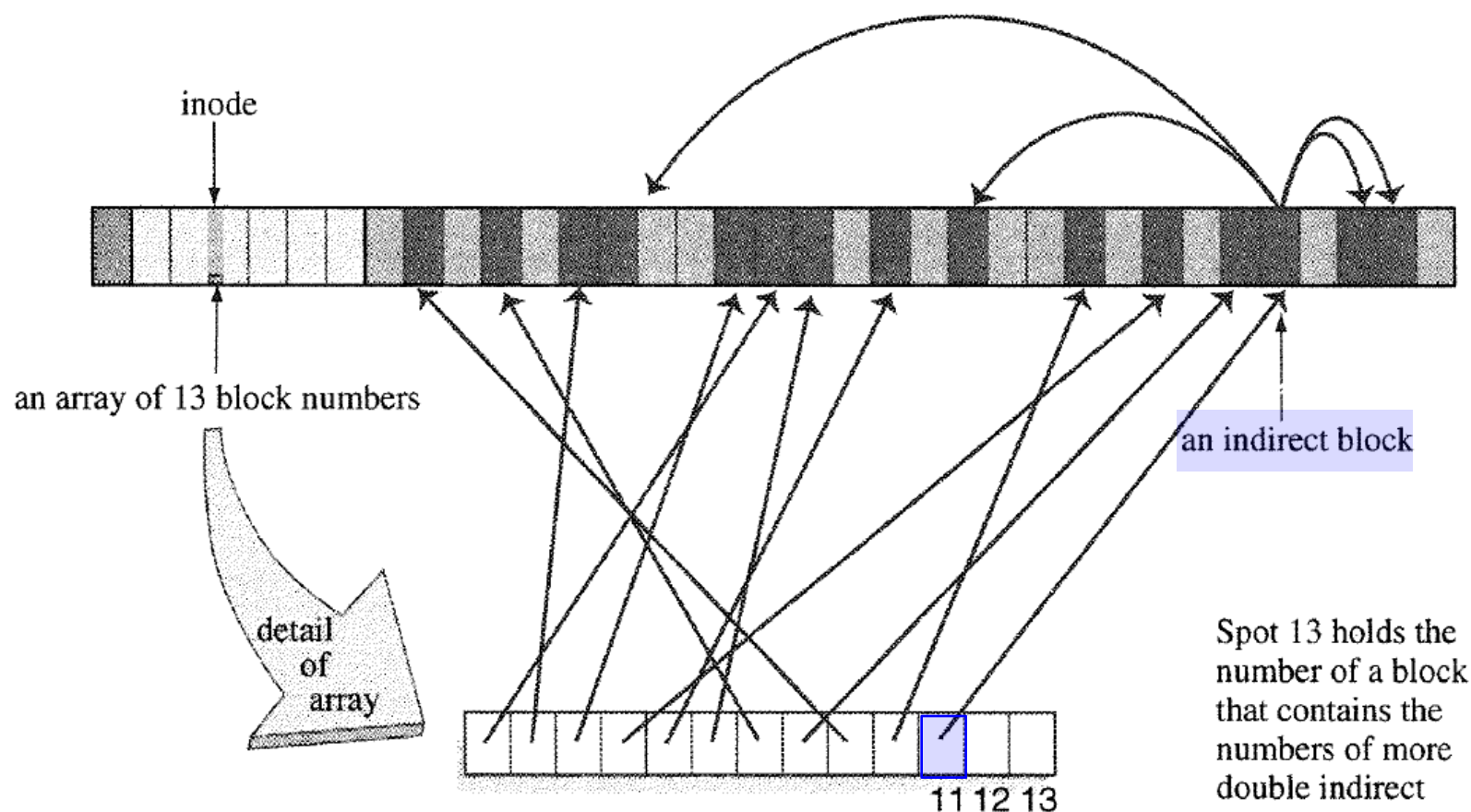
- A large file requires many blocks
- The inode stores an allocation list of the blocks

■ **Problem:** How can a fixed-sized inode store a long allocation list?

■ **Solution:** store most of the allocation list in data blocks and leave pointers to those blocks in the inode

- Say, an inode has an allocation array with only 13 spots and we want to store a list of 14 block number
- Put the first 10 block numbers in the allocation array of the inode
- Put the last 4 numbers in a block placed in the data area
- The block number of the block holding the extra 4 block numbers is stored in the 11th spot in the inode
- **indirect block** : block that contains the part of the allocation list

Inodes and Big Files



The allocation list starts in the first 10 spots in the array in the inode.

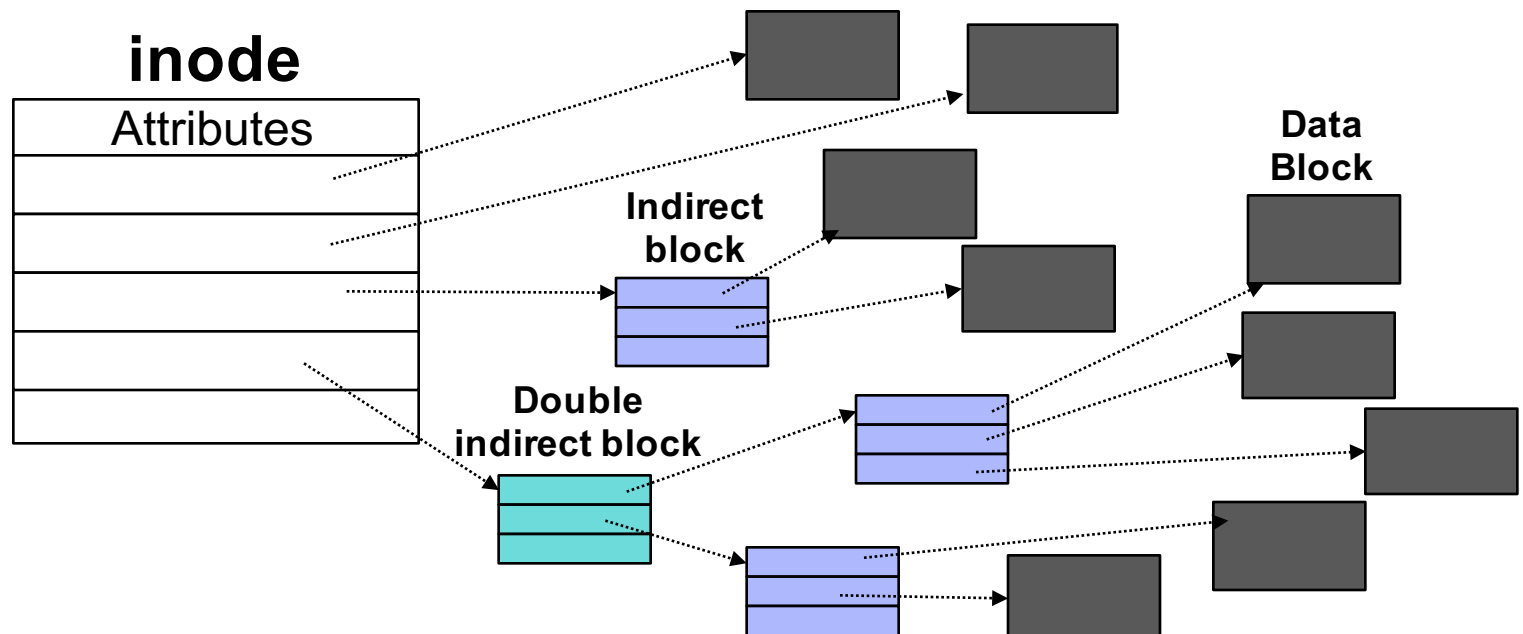
The allocation list continues in an indirect block. Spot 11 holds the number of that block.

Spot 12 holds the number of a block that contains the numbers of more indirect blocks. This block is called a double indirect block.

Spot 13 holds the number of a block that contains the numbers of more double indirect blocks. This block is called a triple indirect block.

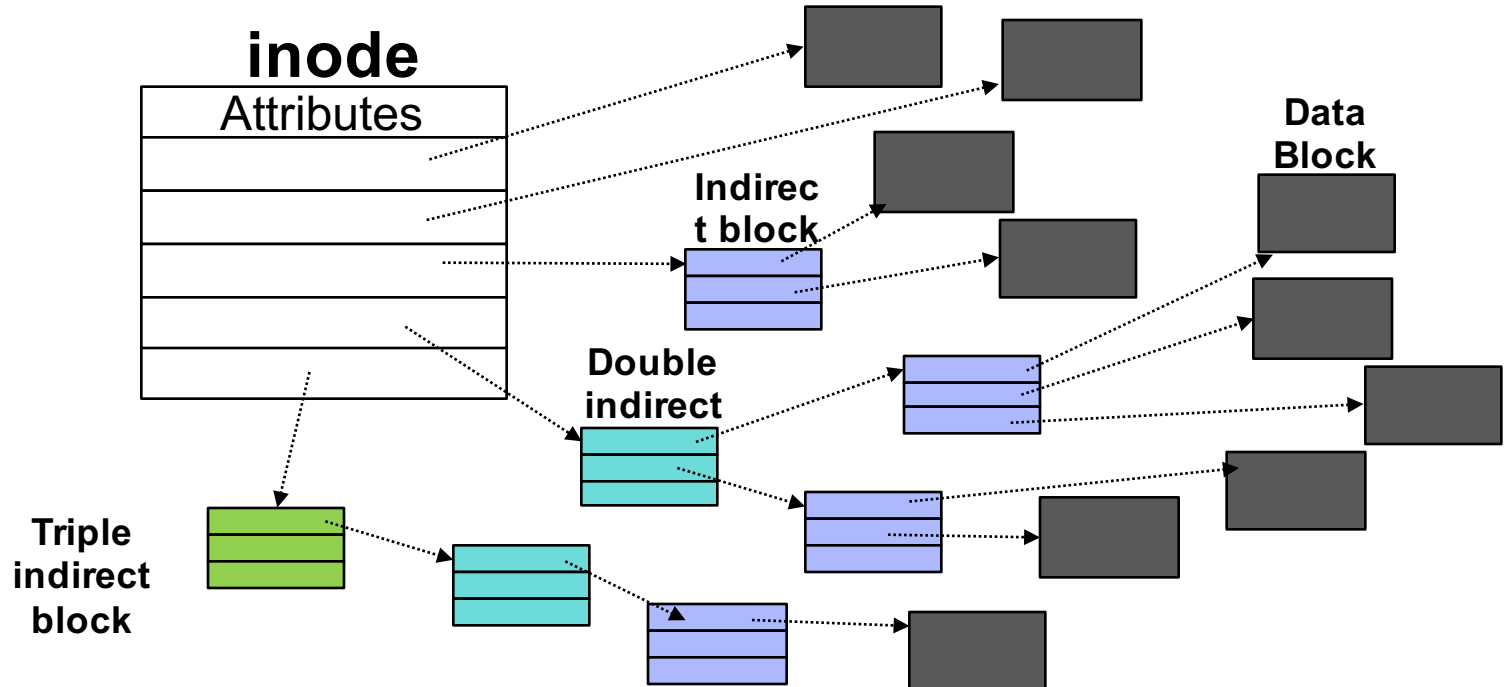
Inodes and Big Files

- What happens when the indirect block is full?
 - As more bytes are added to the file, kernel allocates more data blocks
→ allocation list gets longer and requires more storage
 - When the allocation list overflows the indirect block,
 - Kernel stores the block number of a block (called **a *double indirect block***), which contains a list of the indirect blocks, in 12th spot in the inode



Inodes and Big Files

- What happens when the double indirect block is full?
 - Kernel creates new double indirect blocks
 - Kernel creates **a triple indirect block** to hold the block numbers of the new double indirect blocks
 - The block number of the triple indirect block is stored in the last spot of the allocation array in the inode



Inodes and Big Files

- What happens when the triple indirect block is full?
 - The file has reached its limit
 - If we want to use really big files, we need to set up a file system with larger blocks.
 - When we create the file system, we can specify the size of block
 - A block might be one sector (512 Bytes) or it might be several sectors

```
Last login: Sun Sep 30 09:11:00 2018 from 192.177.100
[seokin@compasslab1:~]$ mkfs -help

Usage:
mkfs [options] [-t <type>] [fs-options] <device> [<size>]

Make a Linux filesystem.

Options:
-t, --type=<type>    filesystem type; when unspecified, ext2 is used
fs-options           parameters for the real filesystem builder
<device>             path to the device to be used
<size>               number of blocks to be used on the device
-V, --verbose         explain what is being done;
                      specifying -V more than once will cause a dry-run
-h, --help            display this help
-V, --version         display version

For more details see mkfs(8).
seokin@compasslab1:~$
```

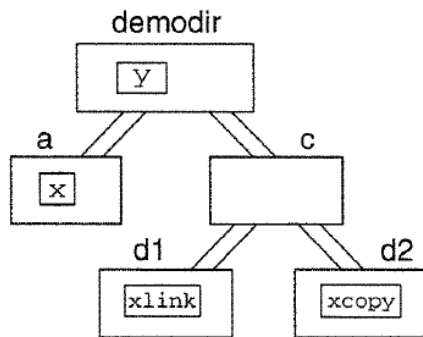
Contents

- 4.2 User's View of File System
- 4.3 Internal Structure of Unix File System
- 4.4 Understanding Directories
- 4.5 Writing pwd
- 4.6 Multiple File Systems: A Tree of Trees

Understanding Directory Structure

- Users see the file system as a collection of directories, subdirectories, and files
- Internally, a directory is a file that contains filename and inode number

user view



system view

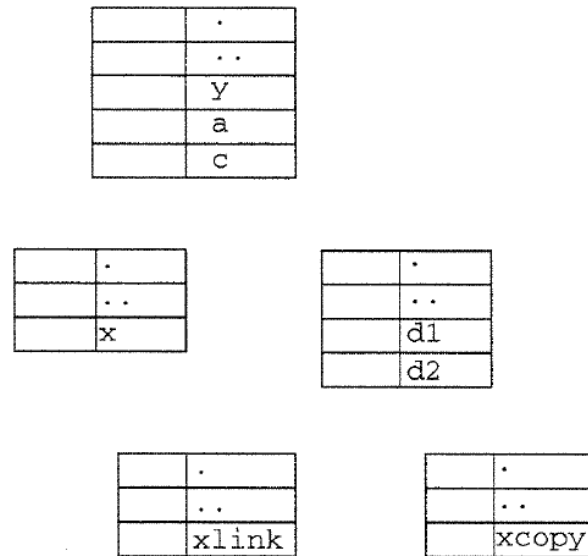


FIGURE 4.8

Two views of a directory tree.

How can we translate from one diagram to the other?

- Let's fill the inode number in the system view diagram.

system view

	.
	..
	y
	a
	c

	.
	..
	x

	.
	..
	d1
	d2

	.
	..
	xlink

	.
	..
	xcopy

```
$ ls -laR demodir
865 .      193 ..    277 a       520 c       491 y
demodir/a:
277 .      865 ..    402 x
demodir/c:
520 .      865 ..    651 d1      247 d2
demodir/c/d1:
651 .      520 ..    402 xlink
demodir/c/d2:
247 .      520 ..    680 xcopy
$
```

How can we translate from one diagram to the other?

- Let's fill the inode number in the system view diagram.

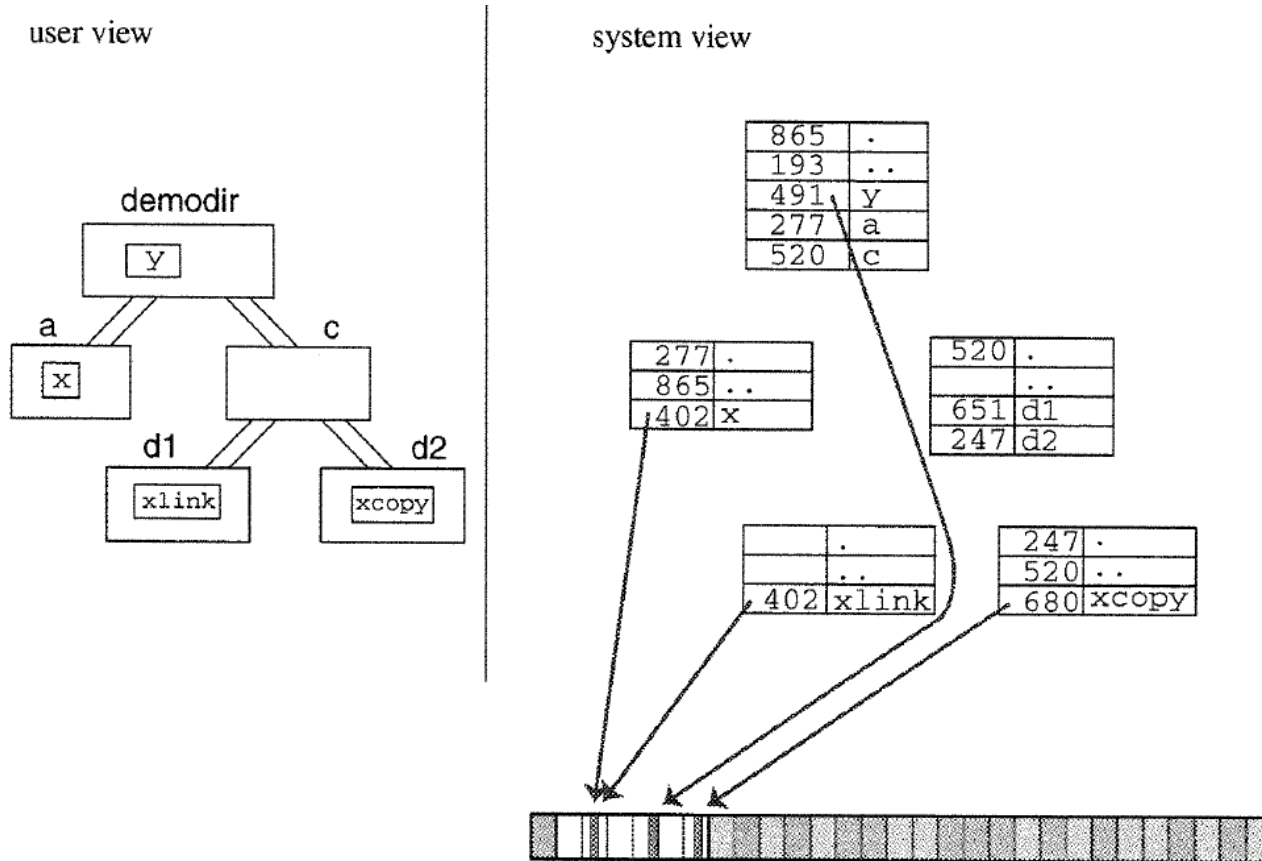


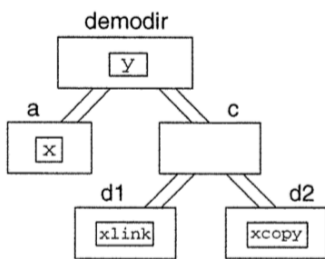
FIGURE 4.9

Filenames and pointers to files.

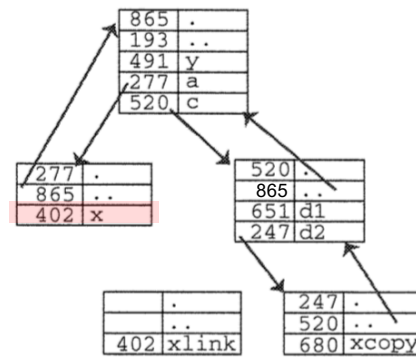
Real Meaning of “A file is in a directory”

- In the system view, we see the directory contains an entry with filename “**y**” and inode number 491
- “file **x** is in directory **a**” means there is a **link** to inode 402 in the directory called “**a**”, and the filename attached to that **link** is “**x**”

user view



system view



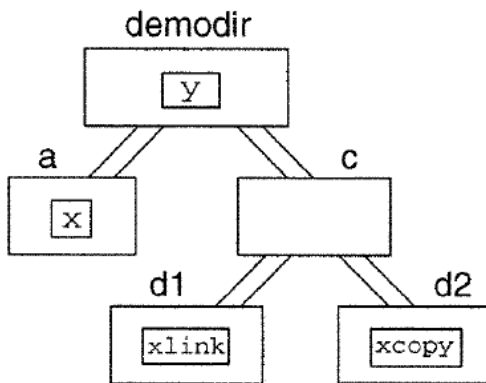
In short,

- Directories contain references to files, each of these reference is called a link.
- The content of the file are in data blocks
- The properties of the file are recorded in an inode struct in the inode table
- The inode number and a name are stored in a directory

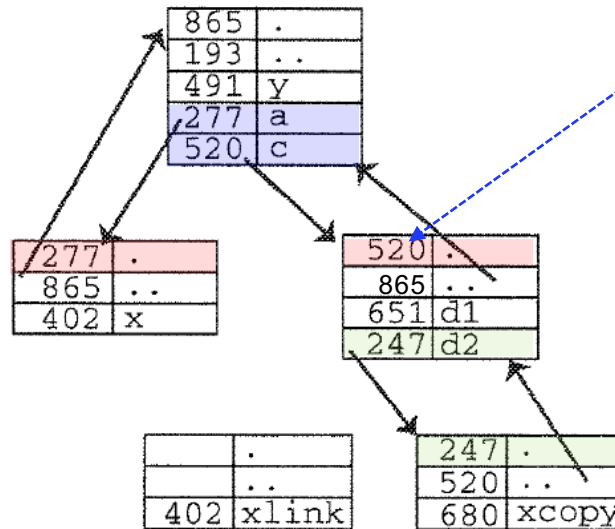
Real Meaning of “A directory contains a subdirectory”

- The directory called “c” is a subdirectory of the “**demodir**” directory → means that demodir contains a link to the inode for that subdirectory

user view



system view

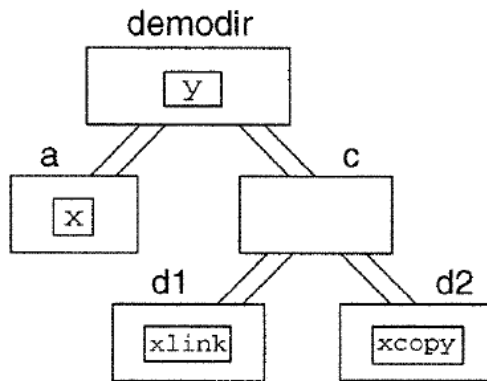


Kernel installs an entry (.) for its own inode in every directory

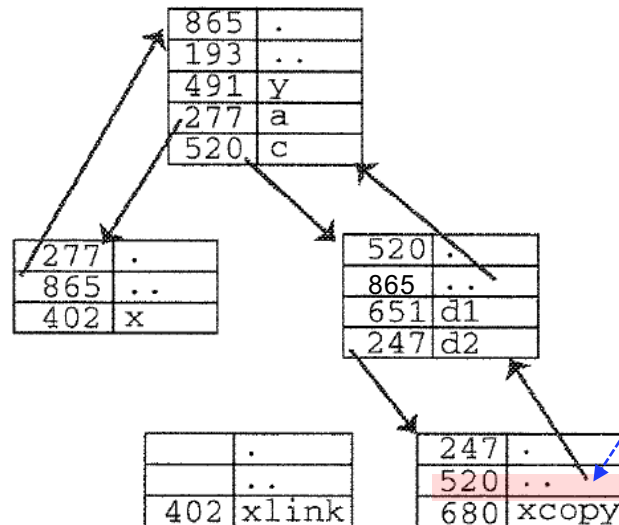
Real Meaning of “A directory has a parent directory”

- The directory called “**d2**” has a parent directory → means that “**d2**” contains a link to the inode for the parent directory
 - A directory contains a link (inode number) for the parent directory named “..”

user view



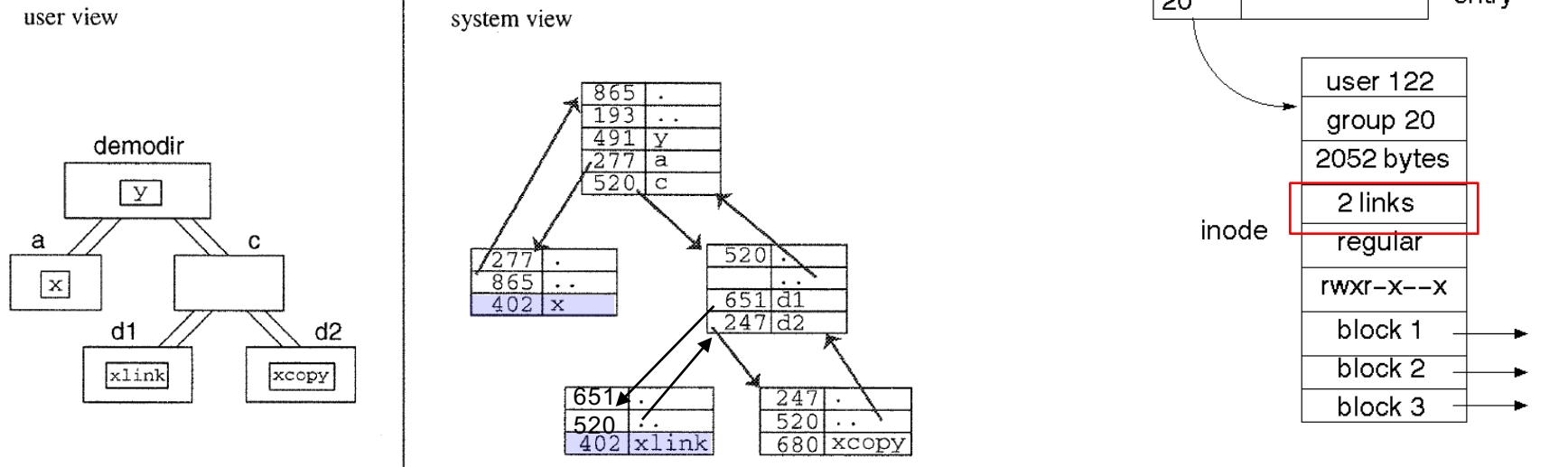
system view



Kernel installs an entry (..) for its parent directory in every directory

Multiple Links, Link Counts

- inode 402 has two links.



- The two links (***called hard link***) refer the same inode
- The file is an inode and a bunch of data blocks; a link is a reference to an inode. We can make as many links to a file as you like
- Kernel records the number of links in the inode. The link count is one of members of the struct stat returned by the stat system call.

Commands and System Calls for Directory Trees

- **mkdir** : Creates new directories
 - Uses `mkdir()` system call
 - `mkdir()` creates and links a new dir node to the directory tree;
 - creates the inode for the directory;
 - allocates a disk block for its contents;
 - installs the two entries in the directory: `.(dot)` and `..(dotdot)` with inode numbers
 - adds a link to that node to its parent directory.

Commands and System Calls for Directory Trees

- **rmdir** : Deletes a directory
 - Uses `rmdir()` system call:
 - `rmdir()` removes a dir node from the directory tree.
 - The directory must be empty
 - The link to the directory is removed from its parent directory
 - If the directory itself is not in use by another process, the inode and data blocks are freed.

Commands and System Calls for Directory Trees

- **rm** : Removes entries from a directory
 - Uses `unlink()` system call
 - `unlink()` deletes a directory entry;
 - It decrements the link count for the corresponding inode.
 - If the link count for the inode becomes zero, the data blocks and inode are freed.

Commands and System Calls for Directory Trees

- In : Creates a link to a file
 - Uses `link()` system call:
 - `link()` makes a new link to an inode.
 - The new link contains the inode number of the original link and has the name specified.

Commands and System Calls for Directory Trees

- mv : Changes the name or location of a file or directory
 - Uses rename() system call
 - rename("y","y.old") changes the name of the file.
 - rename("y","c/d2/y.old") changes the name and the location of the file.
 - rename can be also used for directories.

Commands and System Calls for Directory Trees

- How does rename move a file to another directory?
 - Files are not really in directories, links to files are in directories
 - rename() moves the link from one directory to another
 - Step 1. Copy original link to new name and/or location
 - Step 2. Delete original link

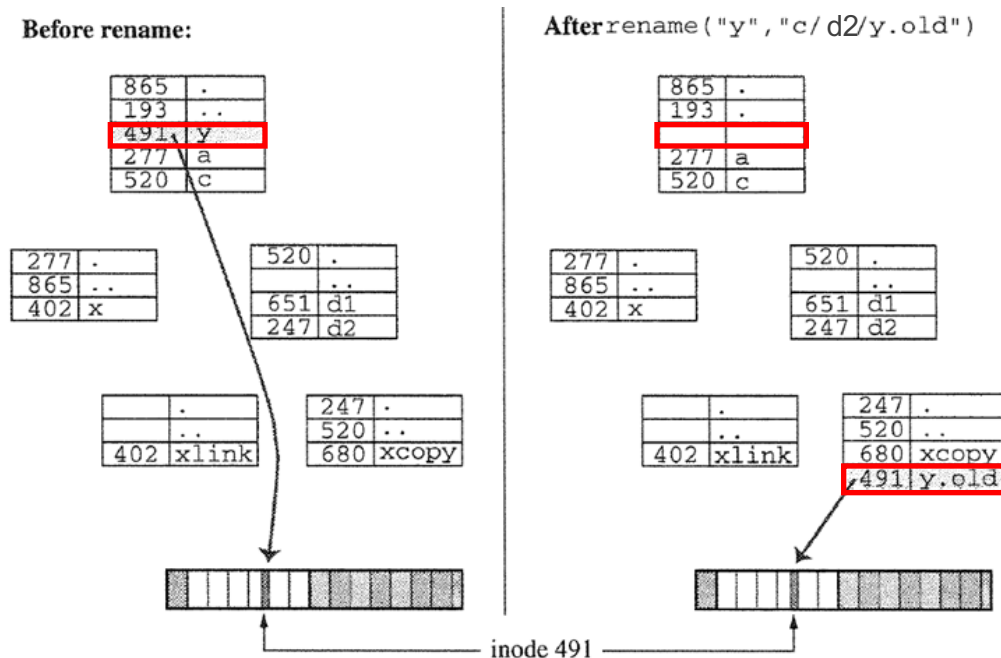


FIGURE 4.11

Moving a file to a new directory.

Commands and System Calls for Directory Trees

- `cd` : Changes the current directory of a process
 - Uses the `chdir()` system call
 - Each running program on Unix has a current directory;
 - Internally, process keeps a variable that stores the inode number of the current directory

Contents

- 4.2 User's View of File System
- 4.3 Internal Structure of Unix File System
- 4.4 Understanding Directories
- 4.5 Writing pwd
- 4.6 Multiple File Systems: A Tree of Trees

How pwd Works

- pwd : prints the path to the current directory

```
$ pwd
```

```
/home/yourname/experiments/demodir/c/d2
```

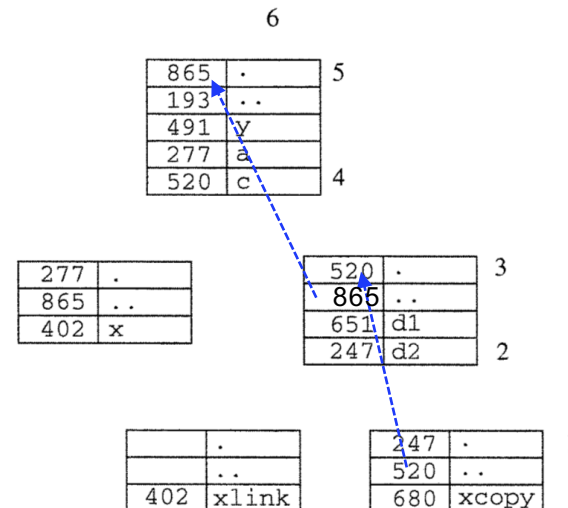
- **How?**

- follows the links and read the directories by climbing up the directory tree until it reaches the top of the tree

- **Algorithm**

- 1. Get the inode number (say, n) for "."
- 2. Go to the parent directory with chdir() system call
- 3. Find the name for the link with inode n
- Repeat 1~3 until you reach the top of the directory tree

Computing pwd:



1. "." is 247
chdir ..
2. 247 is called "d2"
3. "." is 520
chdir ..
4. 520 is called "c"
5. "." is 865
chdir ..
6. 865 is called "demodir"
7. "." is 193
chdir ..

How pwd Works

- Q1: How do we know when we reach the top of the tree?
 - In the root directory, “.” and “..” point to the same inode
 - repeat until it gets to a directory in which inode numbers for “.” and “..” are equal

- Q2: How do we print the directory names in the correct order?
 - Make a recursive program that winds up to the top of the directory tree and prints the directory names, one by one, as it unwinds.

```
$ pwd
```

```
/home/yourname/experiments/demodir/c/d2
```

A Version of pwd: spwd.c

```
/* spwd.c: a simplified version of pwd
 *
 *      starts in current directory and recursively
 *      climbs up to root of filesystem, prints top part
 *      then prints current part
 *
 *      uses readdir() to get info about each thing
 *
 *      bug: prints an empty string if run from "/"
 **/
#include      <stdio.h>
#include      <sys/types.h>
#include      <sys/stat.h>
#include      <dirent.h>
#include      <stdlib.h>
#include      <string.h>

ino_t  get_inode(char *);
void    printpathto(ino_t);
void    inum_to_name(ino_t , char *, int );

int main()
{
    printpathto( get_inode( "." ) );          /* print path to here    */
    putchar('\n');                           /* then add newline      */
    return 0;
}
```

A Version of pwd: spwd.c

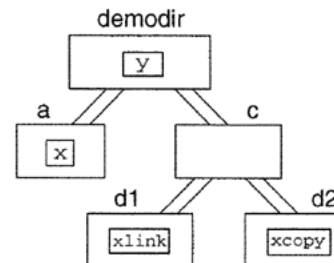
```
ino_t get_inode( char *fname )
/*
 *      returns inode number of the file
 */
{
    struct stat info;
    if ( stat( fname , &info ) == -1 ){
        fprintf(stderr, "Cannot stat ");
        perror(fname);
        exit(1);
    }
    return info.st_ino;
}
```

A Version of pwd: spwd.c

```
void printpathto( ino_t this_inode )
/*
 *   prints path leading down to an object with this inode
 *   kindof recursive
 */
{
    ino_t    my_inode ;
    char     its_name[BUFSIZ];

    if ( get_inode("../") != this_inode )
    {
        chdir( "../" );                /* up one dir */

        inum_to_name(this_inode,its_name,BUFSIZ); /* get its name */
        my_inode = get_inode( "." );          /* print head */
        printpathto( my_inode );              /* recursively */
        printf("/%s", its_name );            /* now print */
                                           /* name of this */
    }
}
```



6

865	.	5
193	..	
491	y	
277	a	
520	c	4

277	.
865	..
402	x

3

520	.	2
	..	
651	d1	
247	d2	

	.
	..
402	xlink

1

247	.
520	..
680	xcopy

A Version of pwd: spwd.c

```
void inum_to_name(ino_t inode_to_find , char *namebuf, int buflen)
/*
 *      looks through current directory for a file with this inode
 *      number and copies its name into namebuf
 */
{
    DIR          *dir_ptr;                /* the directory */
    struct dirent *direntp;               /* each entry */
    dir_ptr = opendir( "." );
    if ( dir_ptr == NULL ){
        perror( "." );
        exit(1);
    }
    /*
     * search directory for a file with specified inum
     */
    while ( ( direntp = readdir( dir_ptr ) ) != NULL )
        if ( direntp->d_ino == inode_to_find )
        {
            strncpy( namebuf, direntp->d_name, buflen);
            namebuf[buflen-1] = '\0';    /* just in case */
            closedir( dir_ptr );
            return;
        }
    fprintf(stderr, "error looking for inum %d\n", inode_to_find);
    exit(1);
}
```

Contents

- 4.2 User's View of File System
- 4.3 Internal Structure of Unix File System
- 4.4 Understanding Directories
- 4.5 Writing pwd
- 4.6 Multiple File Systems: A Tree of Trees

Multiple File System: a Tree of Trees

- What if a system has two disks or partitions?
 - Approach 1 (e.g., Windows): assign drive letters to each disk or partition and use the letter for a part of the full path to a file
 - Approach 2: assign block numbers across all disks to create a virtual disk
- Approach of Unix
 - Each disk/partition has its own file system tree.
 - When there is more than one file system on a computer, Unix provides a way to graft these trees into one larger tree.

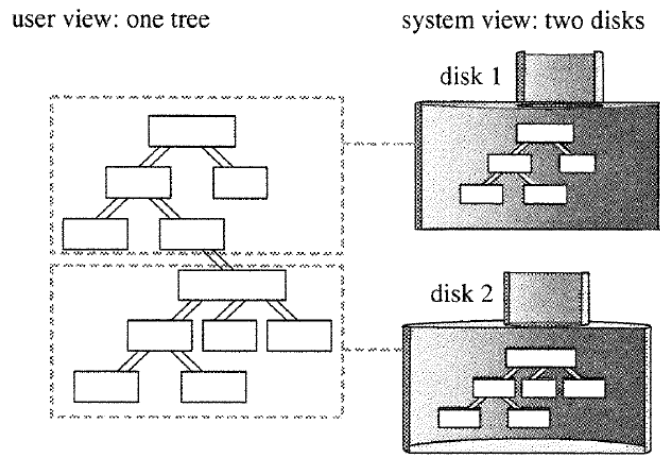


FIGURE 4.13
Tree grafting.

- User sees a seamless tree of directories even there are two trees (one on disk 1 and one on disk 2)
- One file system is designated the **root filesystem**
- The other file system is attached to some subdirectory of the root file system

Mount Points

- The directory to which the subtree is attached is called the mount point for that second file system.
- **mount** : lists currently mounted file systems and their mount points

```
$ mount
/dev/hda1 on / type ext2 (rw)
/dev/hda6 on /home type ext2 (rw)
none on /proc type proc (rw)
none on /dev/pts type devpts (rw,mode=0620)
$
```

The diagram illustrates the output of the `mount` command. Three blue arrows point from labels at the bottom to specific fields in the command output:

- An arrow from **device name** points to `/dev/hda1` in the first line.
- An arrow from **mount point** points to `/` in the first line.
- An arrow from **file system** points to `ext2` in the first line.

Limitation of Hard Links

- Problem: we cannot make links (hard link) to the same file from different file systems.
 - Under Unix, every file in a file system has an inode number
 - This is because each file system has its own inode table, so sharing inodes using inode numbers will not work

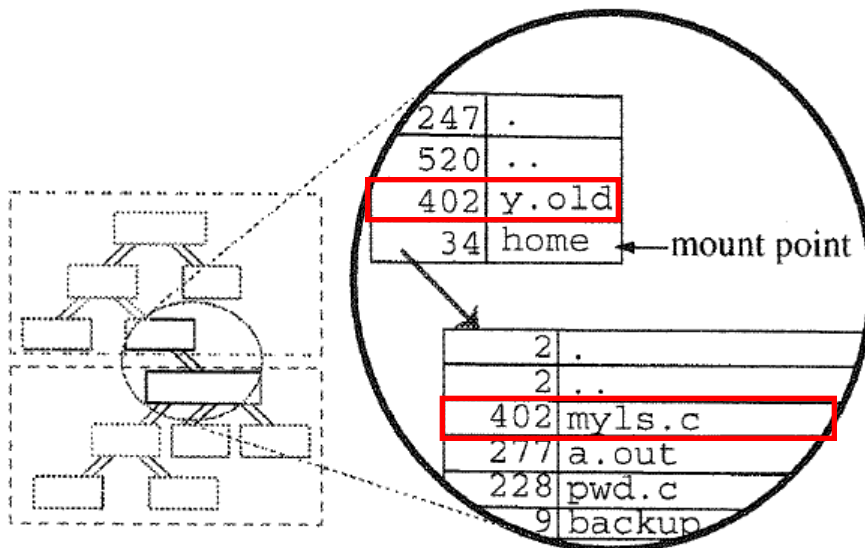
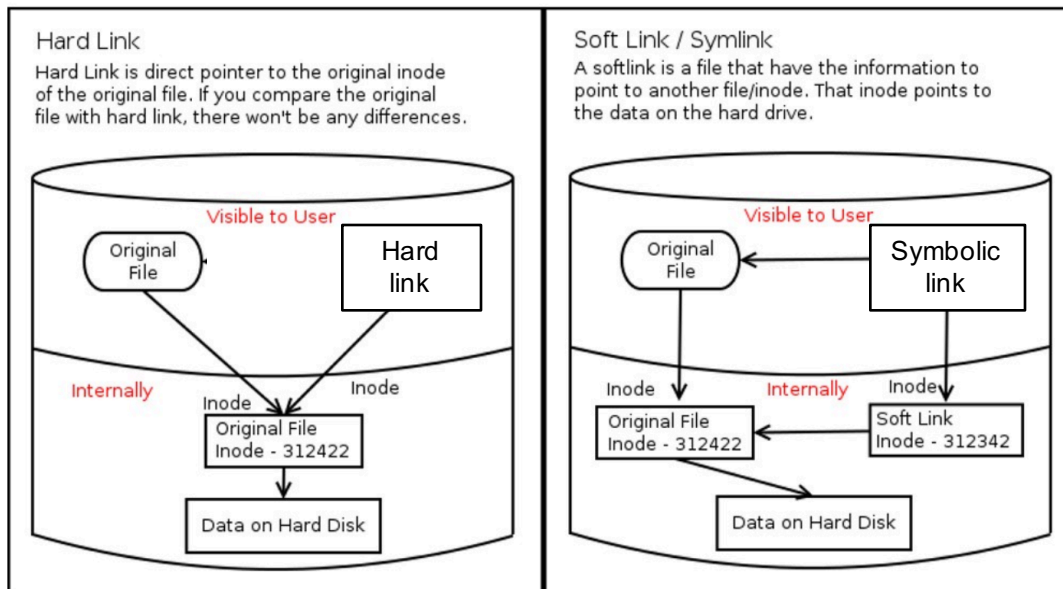


FIGURE 4.14

Inode numbers and file systems.

Symbolic Links

- To point to directories or to files on other file systems, Unix supports another kind of link: **symbolic link**
- A symbolic link refers to a file by name, not by inode number
- A symbolic link points to different inode through the link data
- A symbolic link has its own inode



Source:
<https://hashrootz.blogspot.com/2015/11/soft-link-and-hard-link.html>

Hard link vs Symbolic link

Hard link

Pros

- Deleting the original file does not cause the link to break or the content to disappear.
- No separate inode is required as it is shared between the links

Cons

- Cannot make a link across file systems.
- Cannot make a link for a directory

Symbolic link

Pros

- Can create symbolic links to almost all file system objects including files, directories, devices, and etc.
- Links can be created across file systems and even across disks or mounts.
- Can have different metadata such as file permissions for the symbolic link

Cons

- Deleting or moving the files causes the symbolic links to break.
- Some softwares do not work with symbolic links
- Consume more storage space because it requires its dedicated inode and data block

Hard link vs Symbolic link

■ Hard link

```
$ who > whoson
$ ln whoson ulist ※ makes a hard link ulist
$ ls -li whoson ulist
377 -rw-r--r--    2 bruce    users    235 Jul 16 09:42 ulist
377 -rw-r--r--    2 bruce    users    235 Jul 16 09:42 whoson
```

Same inode and properties!

■ Symbolic link

```
$ ln -s whoson users ※ makes a symbolic link users
$ ls -li whoson ulist users
377 -rw-r--r--    2 bruce    users    235 Jul 16 09:42 ulist
289 lrwxrwxrwx    1 bruce    users      6 Jul 16 09:43 users -> whoson
377 -rw-r--r--    2 bruce    users    235 Jul 16 09:42 whoson
```

File type l(el) is a symbolic link

Different inodes and properties from original file!

Summary

- Unix organizes disk storage into file systems.
 - File system is a collection of files and directories.
 - Directory is a list of names and pointers. Each entry in a directory points to a file or directory
- A Unix file system is comprised of three main parts : superblock, inode table, a data region.
 - File contents are stored in data blocks.
 - File attributes are stored in an inode. The position of the inode in the inode table is called the inode number of the file
- The same inode number may appear in several directories with various names.
 - Each entry is called a hard link of a file.
 - A symbolic link is a link that refers a file by name instead of by inode number
- Several file systems can be connected into one tree.
 - The kernel operation that connects a directory of one file system to the root file system is called mounting