Focus on File Systems

Writing `pwd`

류은경
ekryu@knu.ac.kr

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`**

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

Directories and Files

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

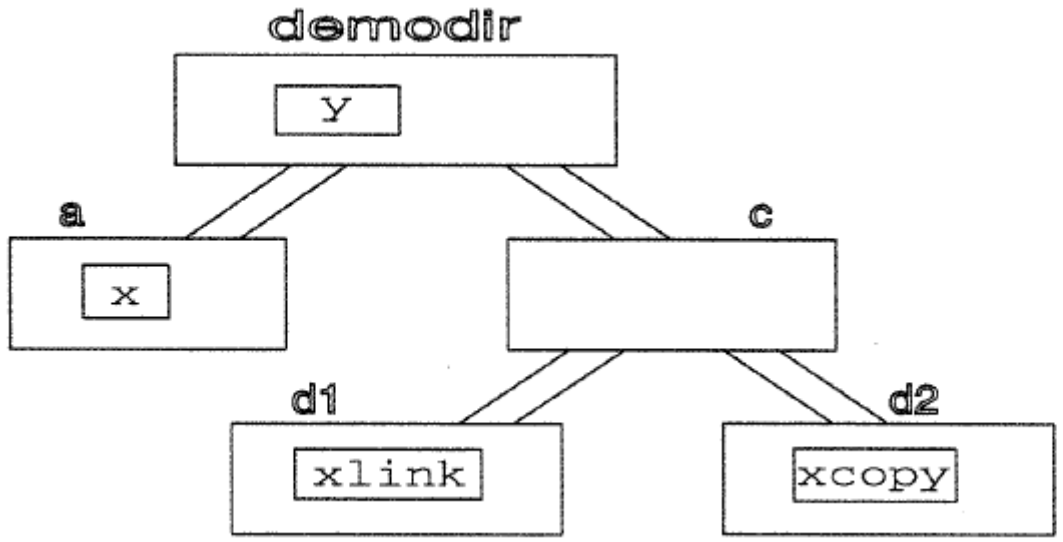
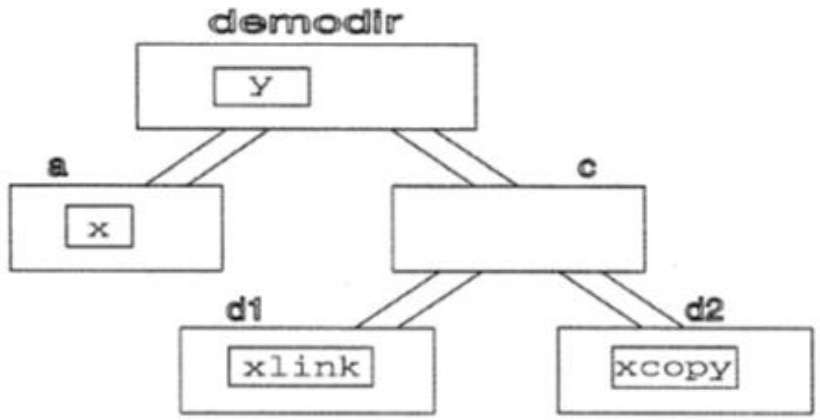


FIGURE 4.1

A tree of directories.

Directory Commands

```
$ mkdir demodir
$ cd demodir
$ pwd
…
$ mkdir b oops
$ mv b c
$ rmdir oops
$ cd c
$ mkdir d1 d2
$ cd ../..
$ mkdir demodir/a
```



명령어	설명	예제
`pwd`	현재 작업 디렉토리 출력	`pwd`
`cd <디렉토리>`	특정 디렉토리로 이동	`cd Documents`
`cd ..`	한 단계 상위 디렉토리로 이동	`cd ..`
`cd ~` 또는 `cd`	홈 디렉토리로 이동	`cd ~`
`cd -`	이전 디렉토리로 이동	`cd -`
`mkdir <디렉토리>`	새 디렉토리 생성	`mkdir new_folder`
`mkdir -p <경로>`	부모 디렉토리가 없으면 함께 생성	`mkdir -p dir1/dir2`
`rmdir <디렉토리>`	비어 있는 디렉토리 삭제	`rmdir empty_folder`
`rm -r <디렉토리>`	디렉토리 및 내부 파일/폴더 삭제 (주의)	`rm -r my_folder`
`ls -ld <디렉토리>`	디렉토리 자체 정보 출력	`ls -ld my_folder`
`du -sh <디렉토리>`	디렉토리 크기 확인	`du -sh my_folder`

File Commands

```
$ cd demodir
$ cp /etc/group x
$ cat 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
$ cp d1/xlink z
$ rm ../../demodir/c/d2/../z
$ cd ../..
$ cat demodir/a/x
```

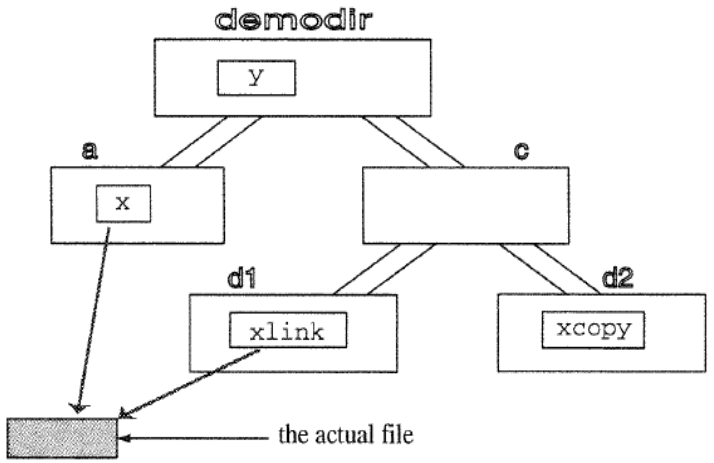


FIGURE 4.2
Two links to the same file.

Tree Commands

- **`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/`
- **`du`**
  - `Ex) $ du -h`
- **`find`**
  - `Ex) $ find . -name 'my*'`

4.2 User's View of File System

4.3 Internal Structure of Unix File System

4.4 Understanding Directories

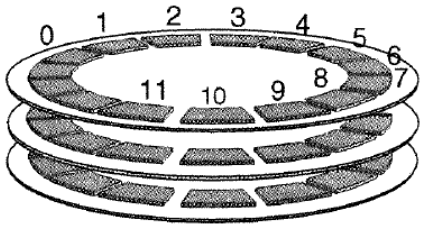
4.5 Writing pwd

4.6 Multiple File Systems: A Tree of Trees

Overview of Unix File System

- A *disk* is ***a stack of magnetic platters***
  - *Tracks*
  - *Sectors:* basic unit of storage on the disk
    - Each sector stores some number of bytes (ex: 512 bytes)

- ***A numbering system for disk sectors*** **lets us treat *a disk as an array of blocks***



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

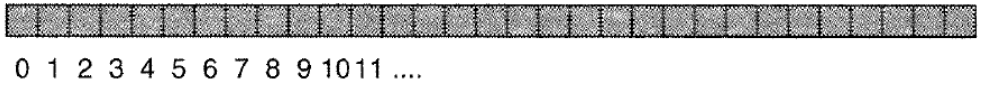


FIGURE 4.3
Assigning numbers to disk blocks.

From an Array of Blocks to Three Regions

- Divide the **array of blocks** into tree sections:
  - Superblock contains **info about file system**
    - Size of each area, location of unused data block
  - Inode Table has **properties of files**
    - An array of structs, fiexed size
  - Data blocks

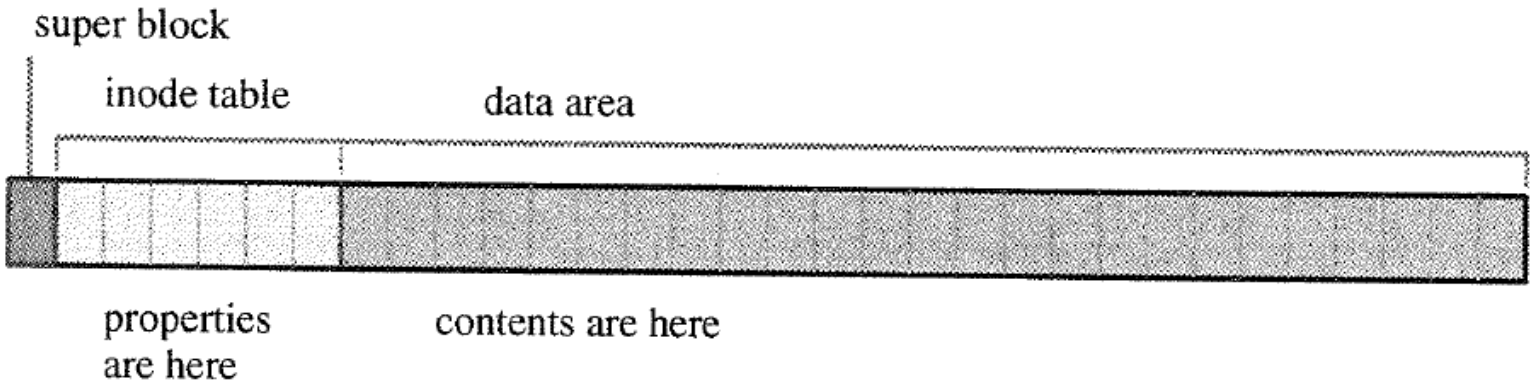


FIGURE 4.4

The three regions of a file system.

File System in Practice: Creating a File

- **Consider : `$ who > userlist`**

The file has **contents**, and the file has **properties**.

The **kernel** has to store: (1)(2)(3)

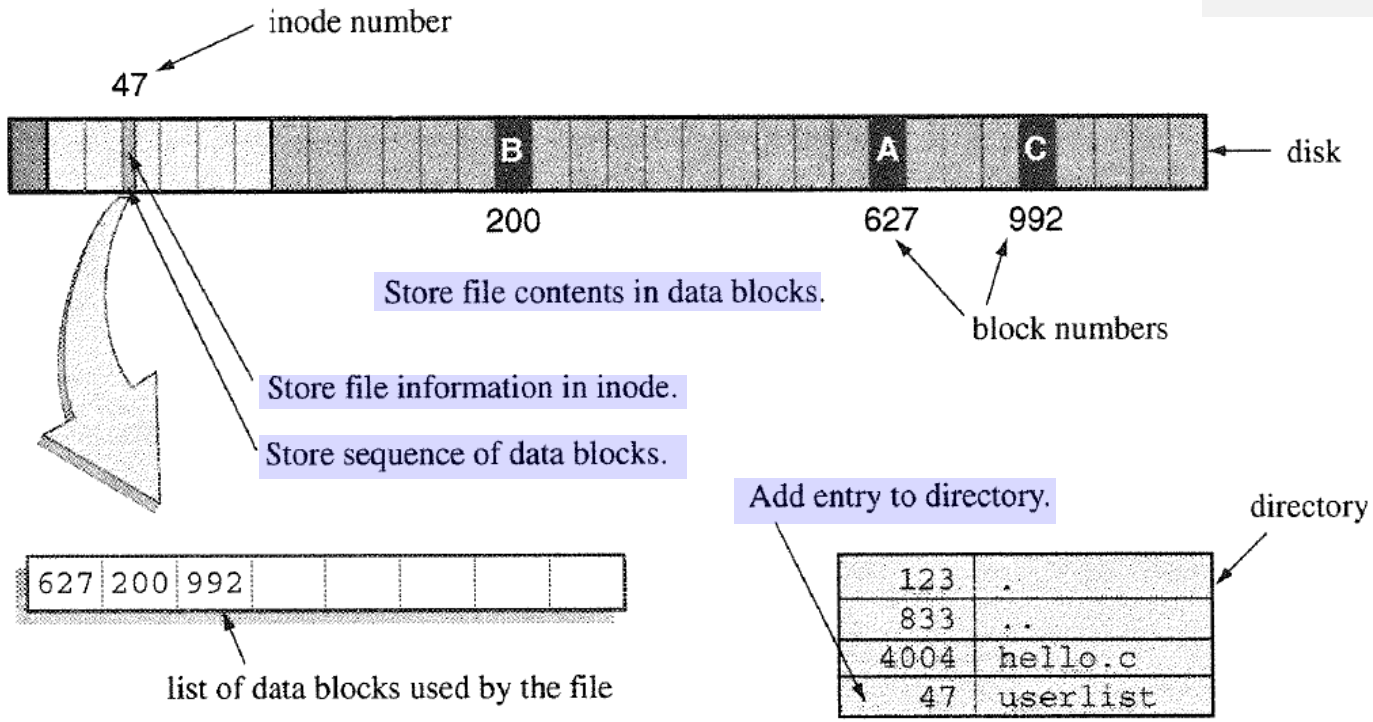


FIGURE 4.5

Internal structure of a file.

File System in Practice: How Directories Work

- **Internal structure of directory**

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

- **Looking Inside a Directory**

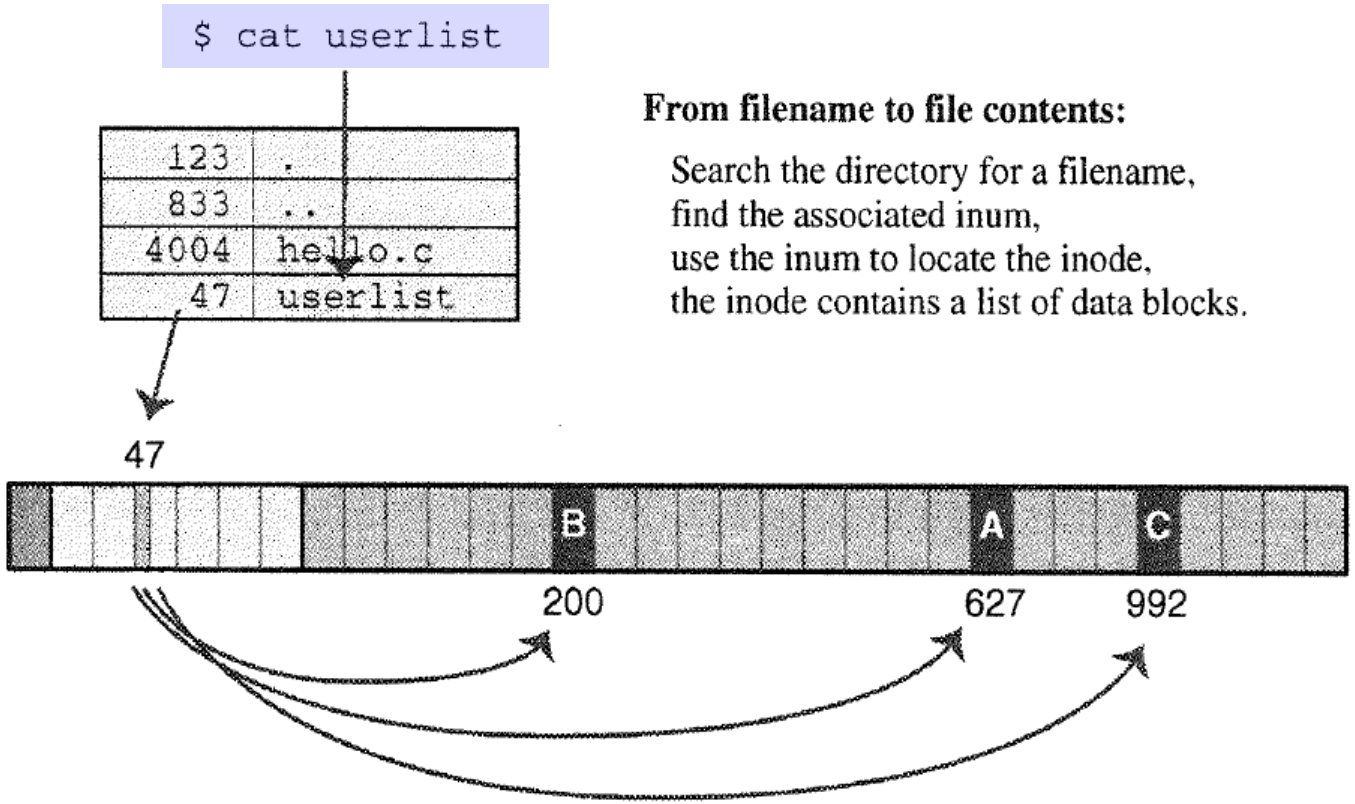
```
$ ls -1ia demodir
 177865 .
 529193 ..
 588277 a
 200520 c
 204491 y
$
```

Multiple Links to the Same File

```
$ ls -ia /
      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
```

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



From filename to file contents:

Search the directory for a filename,
find the associated inum,
use the inum to locate the inode,
the inode contains a list of data blocks.

FIGURE 4.6

From filename to disk blocks.

Inodes and Big Files

fact 1	A large file requires many disk blocks.
fact 2	The inode stores the disk block allocation list.
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.

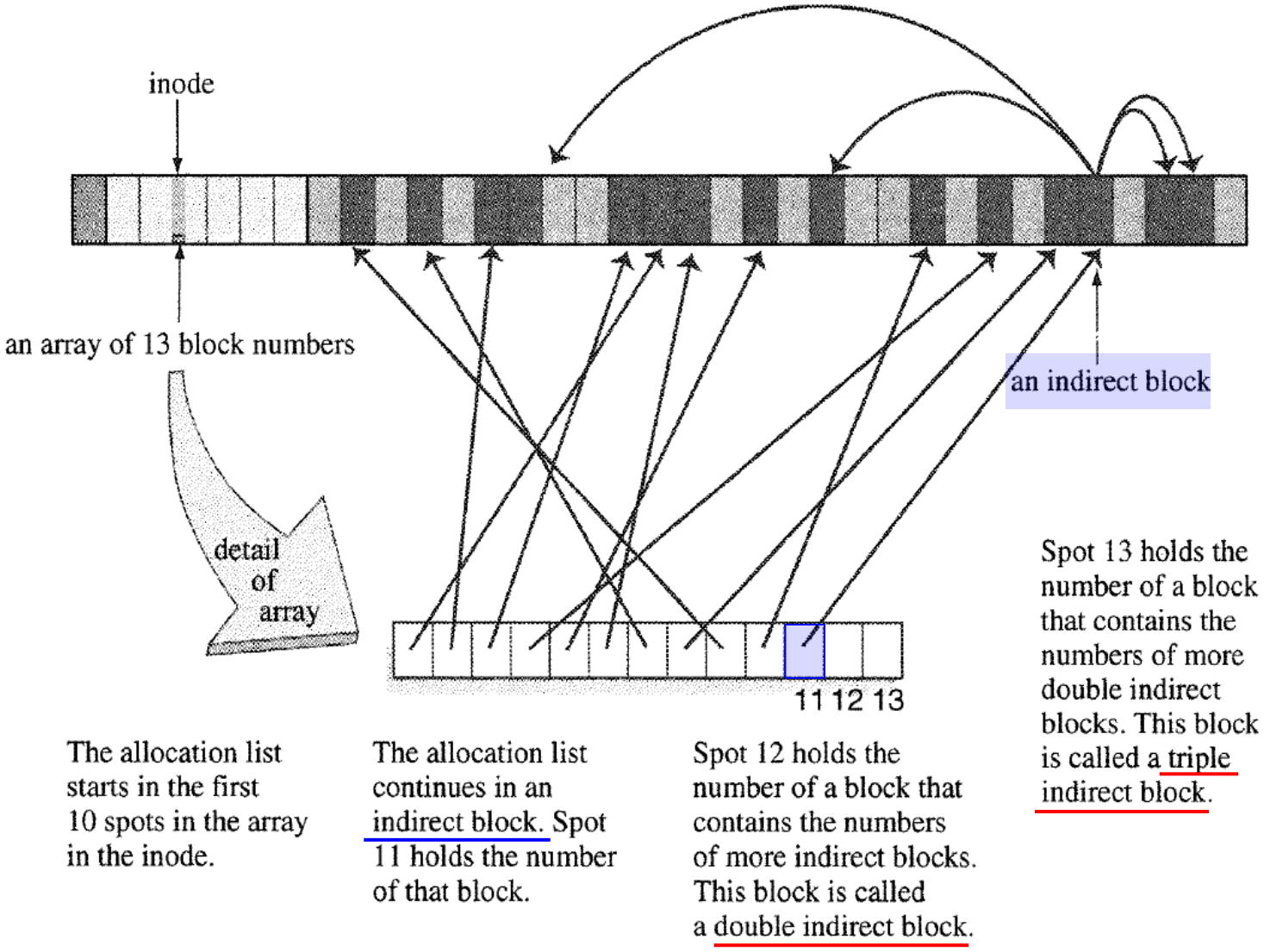


FIGURE 4.7

Block allocation list continues in data region.

Understanding Directory Structure

- **Internally, a directory is a file that contains filename and inode #**

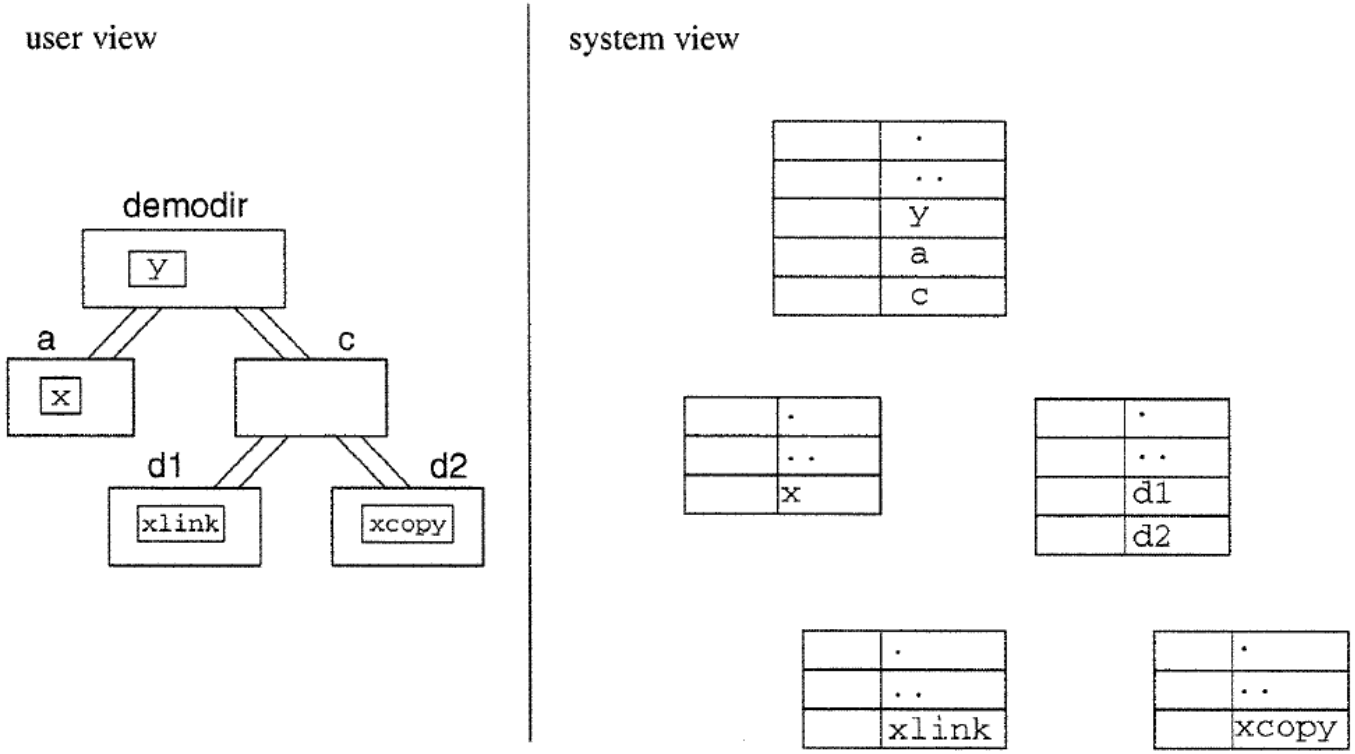


FIGURE 4.8

Two views of a directory tree.

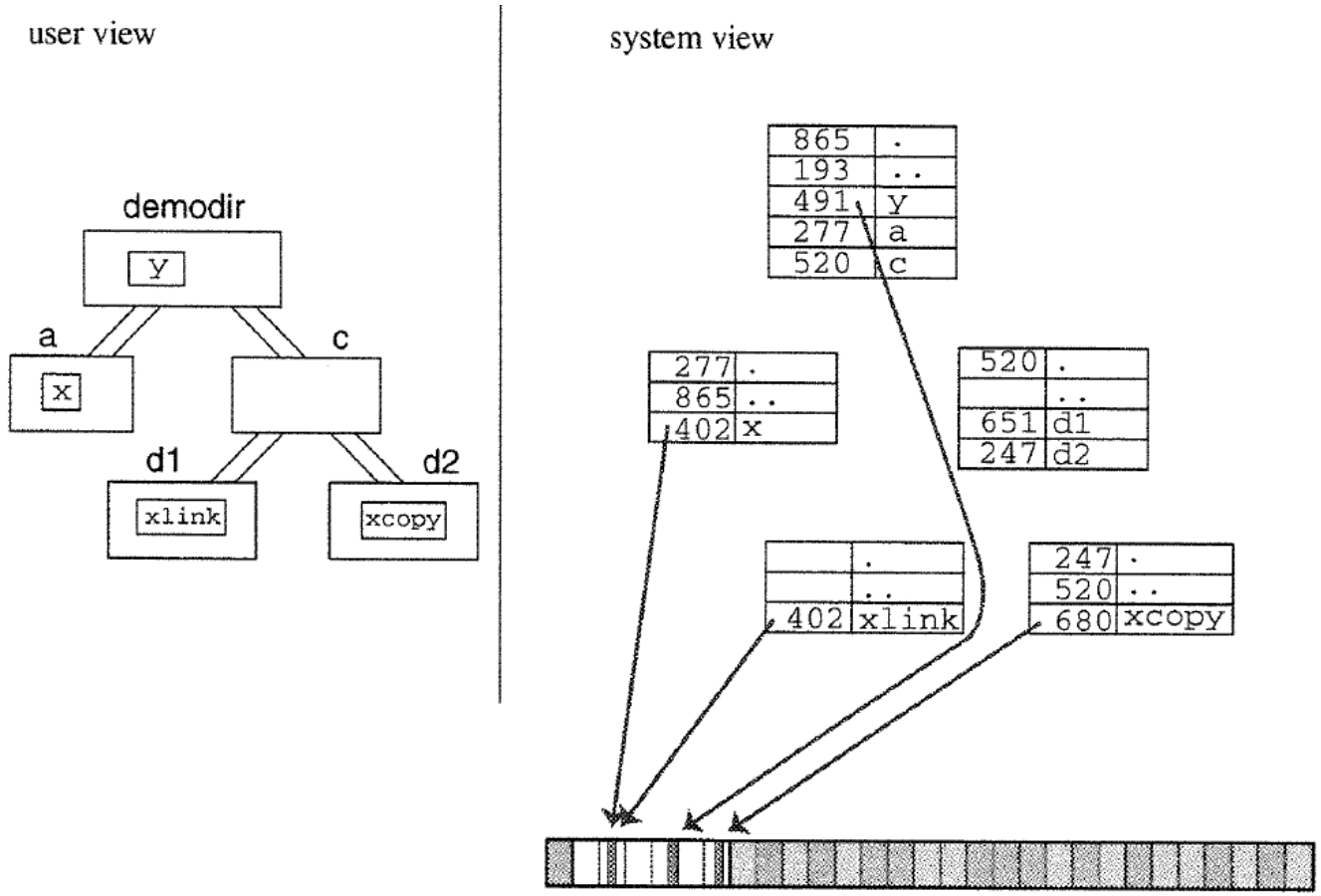
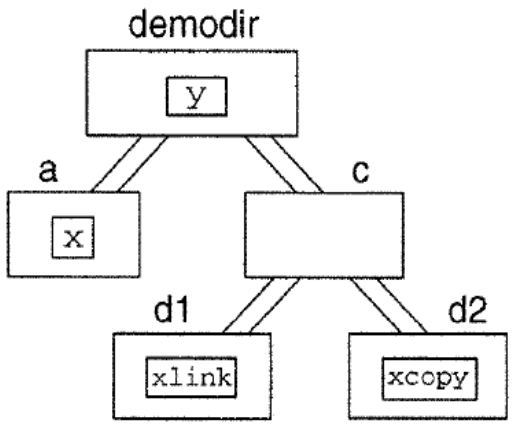


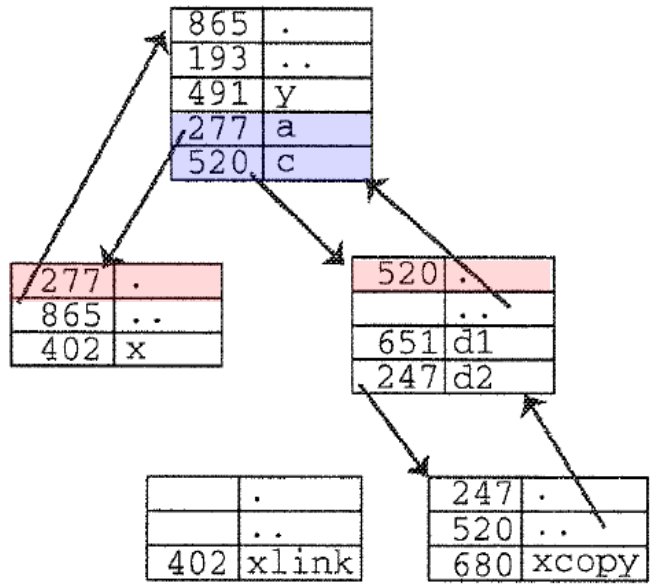
FIGURE 4.9

Filenames and pointers to files.

user view

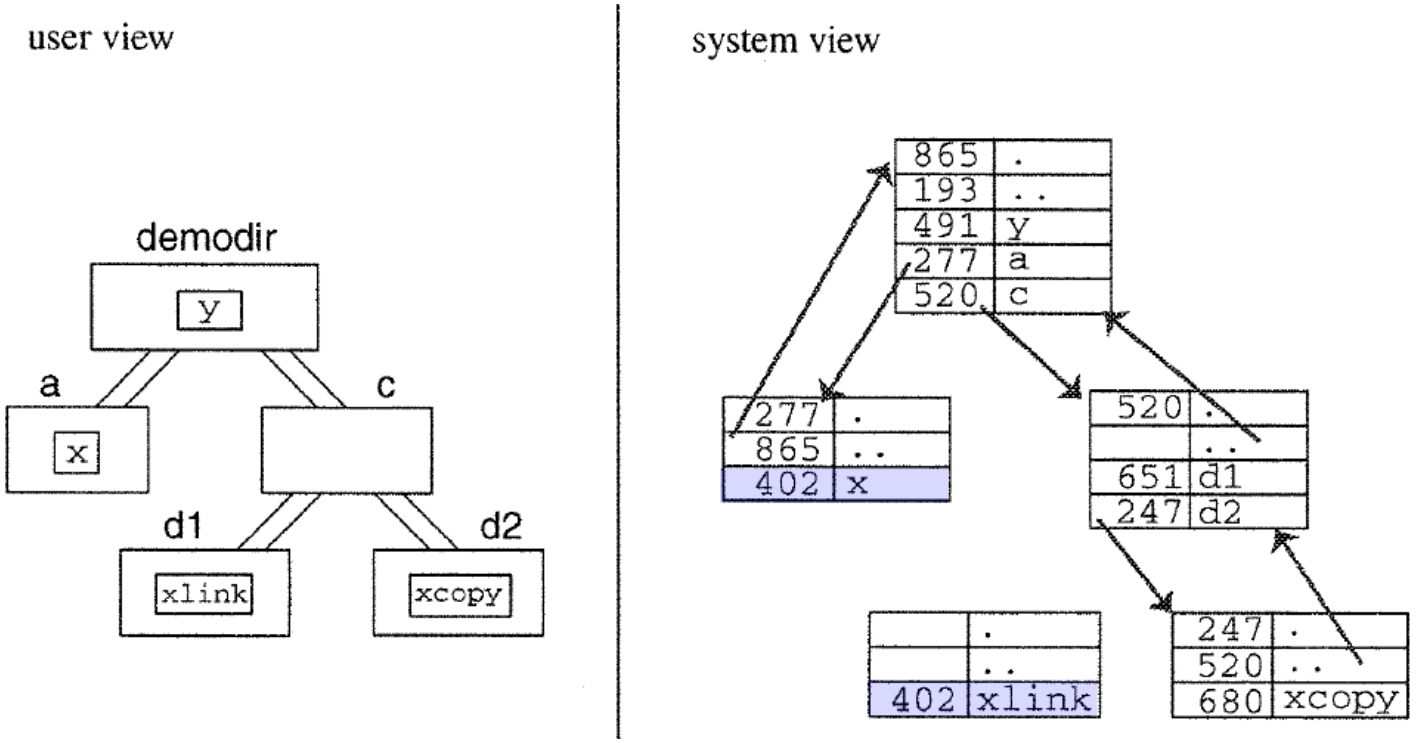


system view

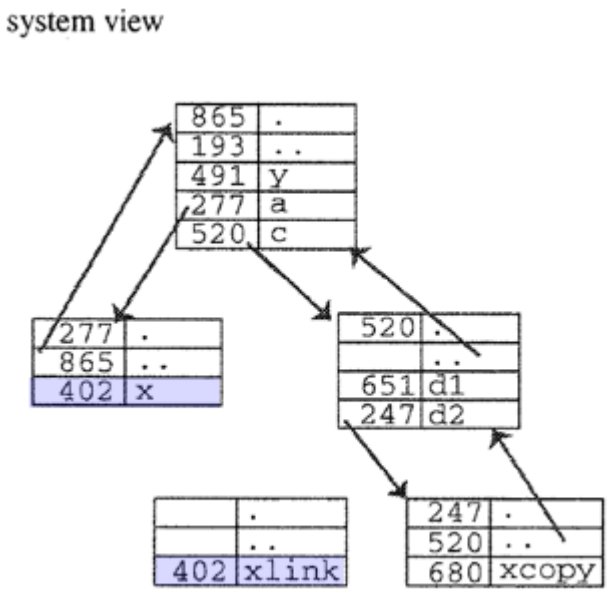



```
$ ls -iaR demodir
 865 .    193 ..   277 a    520 c    491 y

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
$
```

- **Multiple Links, Link Counts**
  - inode 402 has *two links*. (***hard link***)
  - ***link count*** is stored *in the inode* as one of the members of the struct **`stat`**



- **Filenames**
  - The **file names** that users see are actually **links** to inode numbers.
  - **Files are identified by inode numbers**



Directory Trees : Commands and System Calls

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

mkdir		
PURPOSE	Create a directory	
INCLUDE	#include <sys/stat.h> #include <sys/types.h>	
USAGE	int result = mkdir(char *pathname, mode_t mode)	
ARGS	pathname mode	name of new directory mask for permission bits
RETURNS	-1 0	if error if success

- **`rmdir` : Deletes a directory**
  - **`rmdir()`** removes a dir node from a dir 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.

rmdir	
PURPOSE	Delete a directory. The directory must be empty.
INCLUDE	#include <unistd.h>
USAGE	int result = rmdir(const char *path);
ARGS	path name of directory
RETURNS	-1 if error 0 if success

- **`rm` : Removes entries from a dir**
  - 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.

unlink	
PURPOSE	Remove a directory entry
INCLUDE	#include <unistd.h>
USAGE	int result = unlink(const char *path);
ARGS	path name of directory entry to remove
RETURNS	-1 if error 0 if success

- **`ln` : Creates a link to a file**
  - **Uses `link()` system call:**
  - **`link()`** makes a new link to an inode.

```
root@goorm:/workspace/sys_pro# ln a.txt b.txt
root@goorm:/workspace/sys_pro# ls -il
합계 28
393222 -rw-r--r-- 1 root root 2000  1월 10  2021 README.md
393249 -rw-rw-r-- 2 root root    0  9월 22 02:29 a.txt
393249 -rw-rw-r-- 2 root root    0  9월 22 02:29 b.txt
```

link	
PURPOSE	Make a new link to a file
INCLUDE	#include <unistd.h>
USAGE	int result = link(const char *orig, const char *new);
ARGS	orig name of original link new name of new link
RETURNS	-1 if error 0 if success

- **`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.

rename	
PURPOSE	Rename or move a link
INCLUDE	`#include <unistd.h>`
USAGE	`int result = rename(const char *from, const char *to);`
ARGS	from — name of original link to — name of new link
RETURNS	-1 — if error 0 — if success

- **How `rename` Works:**

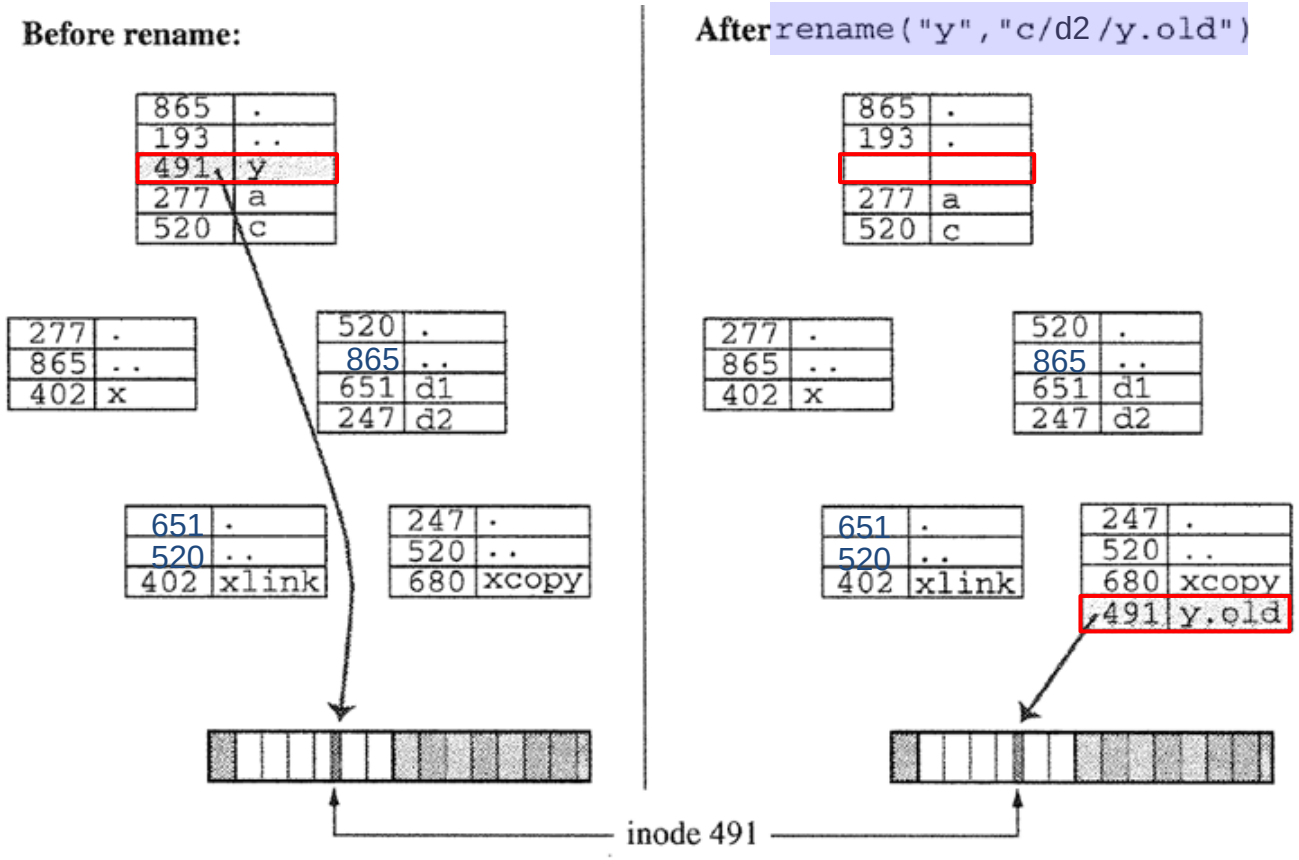


FIGURE 4.11

Moving a file to a new directory.

- **How `rename` Works**
  - Basic logic:

```
copy original link to new name and/or location
delete original link
```

  - **`rename("x", "z");`** works like this

```
if ( link("x","z") != -1 )
      unlink("x");
```

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

chdir	
PURPOSE	Change current directory of calling process
INCLUDE	#include <unistd.h>
USAGE	int result = chdir(const char *path);
ARGS	path path to new directory
RETURNS	-1 if error 0 if success

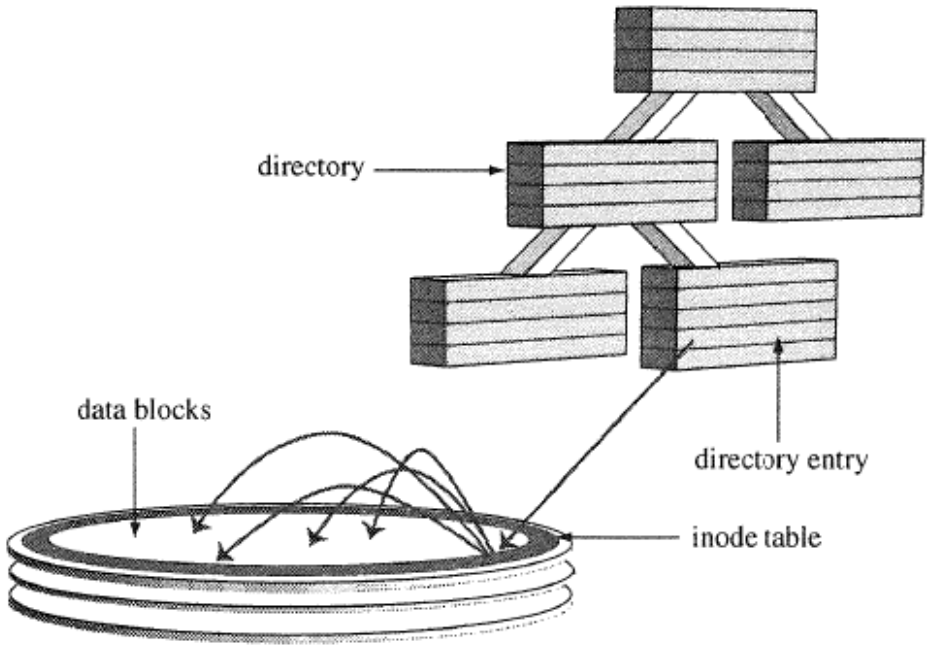


FIGURE 4.15

Inodes, data blocks, directories, pointers.

File Systems

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

Computing pwd:

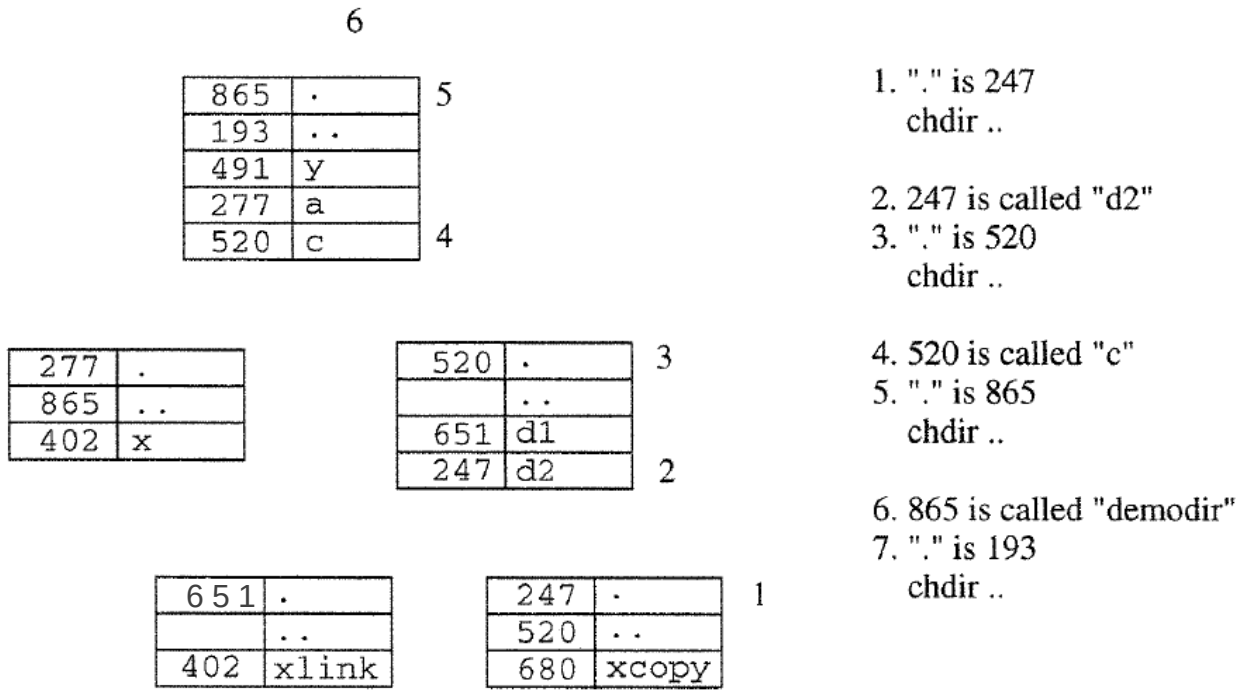


FIGURE 4.12

Computing the current path.

- **Algorithm**

1. Note the inode number for ".", call it *n* (use `stat`)
2. `chdir ..` (use `chdir`)
3. Find the name for the link with inode *n* (use `opendir`,`readdir`,`closedir`)
   Repeat (until you reach the top of the tree).

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

- **Q1: How do we know when we reach the top of the tree? …**
- **Q2: How do we print the directory names in the correct order? …**

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;
}
```

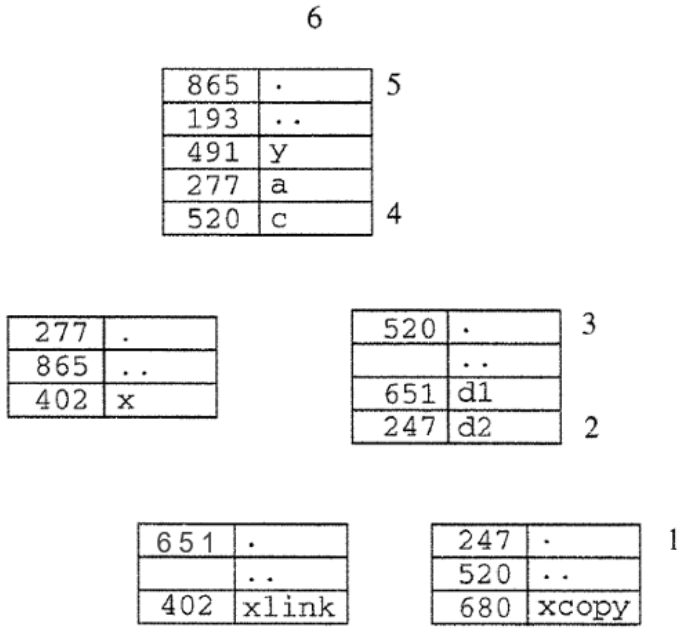
```
$ spwd
/bruce/experiments/demodir/c/d2
$
```

①

```
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;
}
```

```
$ spwd
/bruce/experiments/demodir/c/d2
$
```

```
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 )
        {
            ② chdir( ".." );                          /* up one dir    */
            ③ 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 */
        }
}
```



```
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);
}
```

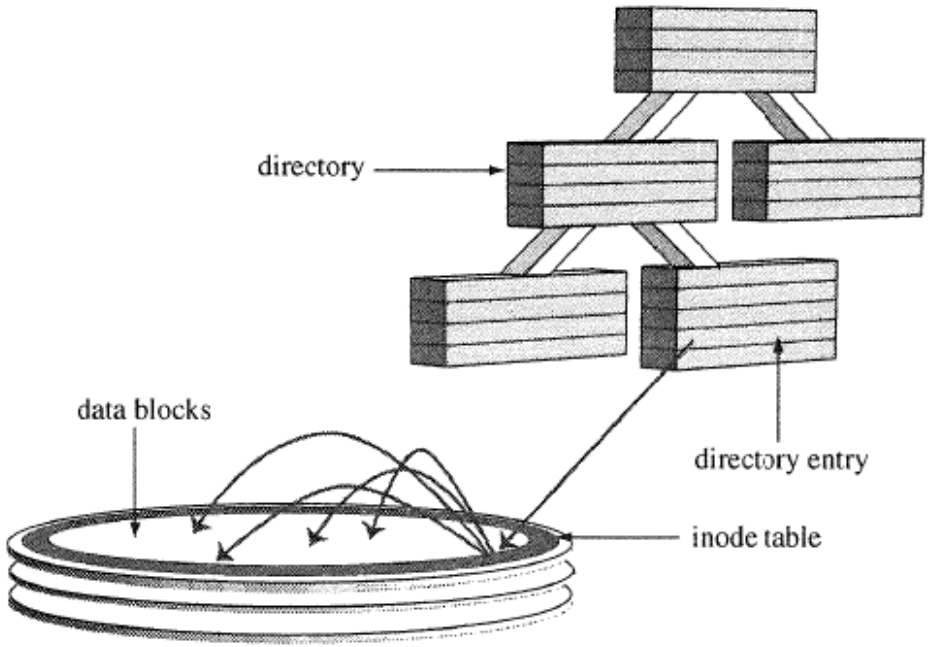


FIGURE 4.15

Inodes, data blocks, directories, pointers.

Multiple File Systems

- **What if** a Unix system has **two disks or partitions**?
  - 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*.

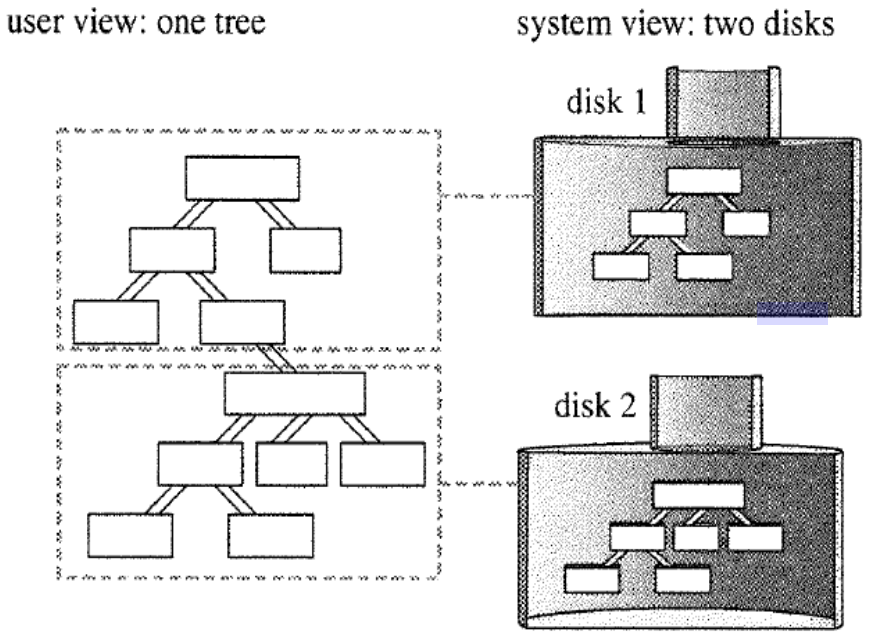


FIGURE 4.13

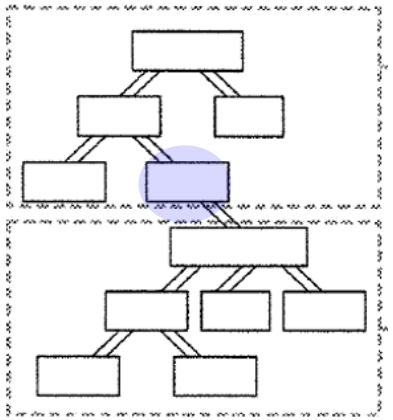
Tree grafting.

Mount Points

- **`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)
$
```

device name　　mount point　　file system (currently ext4)



```
# /dev/sdb1을 /mnt에 마운트
$ mount /dev/sdb1 /mnt

# /mnt 마운트 해제
$ umount /mnt

# 마운트된 파일 시스템 확인
$ mount      # 모든 마운트 정보 출력
$ df -h      # 디스크 사용량과 마운트 정보 확인
```

Duplicate Inode Numbers and Cross-Device Links

- Under Unix, **every file** has an **inode** #

- Problem:
  Two different disks may have different files with inode # 402.

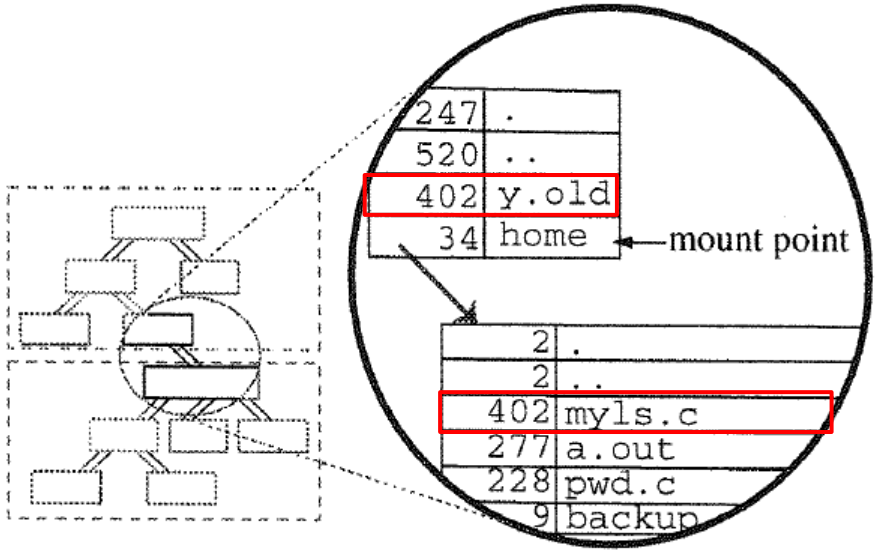


FIGURE 4.14

Inode numbers and file systems.

- Can I make **links to the same file from different file systems?**
  - …

- **System Calls** know about this :
  - **`link()`** → *Refuses* to create **`cross-device links`**
  - **`rename()`** → *Refuses* to transfer an inode # across FSs.

Symbolic Links

- ***Hard links* cannot point to inodes in other file systems**

- **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 #

- **Comparison:**
  - Hard link

```
$ who > whoson
$ ln whoson 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
$ 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!

- **This file, `users`,** is not the original file `whoson`
  - but it behaves like the original file when programs read from or write to it.

```
$ wc -l whoson users
      5 whoson
      5 users
     10 total
$ diff whoson users
$
```

※ wc reads files and counts lines.

※ diff reads files and compares contents.

- **System Calls** for Symbolic Links
  - **`symlink()`** creates a symbolic link.
  - **`readlink()`** obtains the name of the original file.
  - **`lstat()`** obtains information about the original file.

File Systems

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

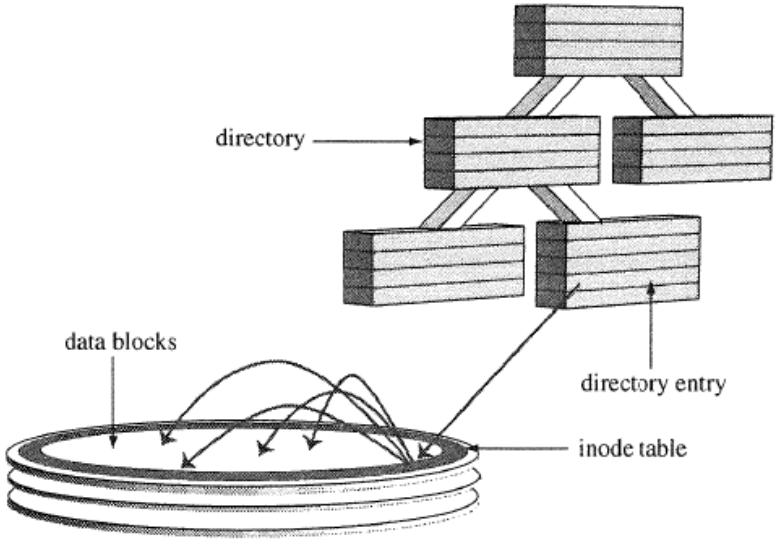


FIGURE 4.15

Inodes, data blocks, directories, pointers.