The UNIX File System

- File Systems and Directories
- UNIX inodes
- Accessing directories
- Understanding links in directories

- Reading: R&R, Ch 5

Directories

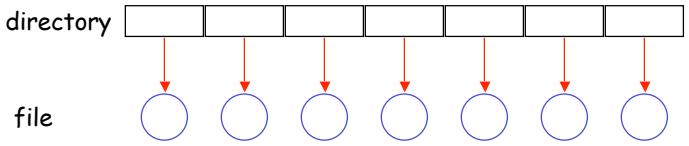
- Large amounts of data: Partition and structure for easier access.

- High-level structure:
  - *partitions* in MS-DOS
  - *minidisks* in MVS/VM
  - *file systems* in UNIX

- **Directories**: Map file name to directory entry (basically a symbol table).

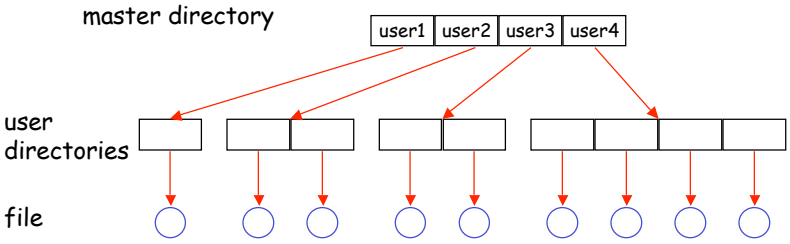
Directory Structures

- Single-level directory:



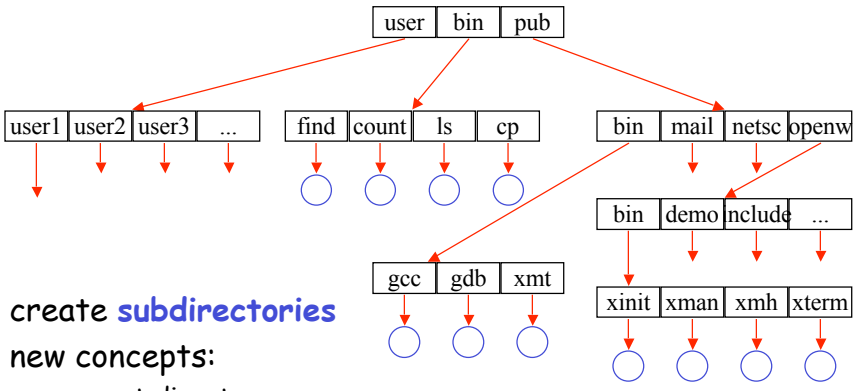
  - Problems:
    - limited-length file names
    - multiple users?
    - not composable (how do you combine multiple file systems?)

Two-Level Directories



- Path names
- Location of system files
  - special directory
  - search path

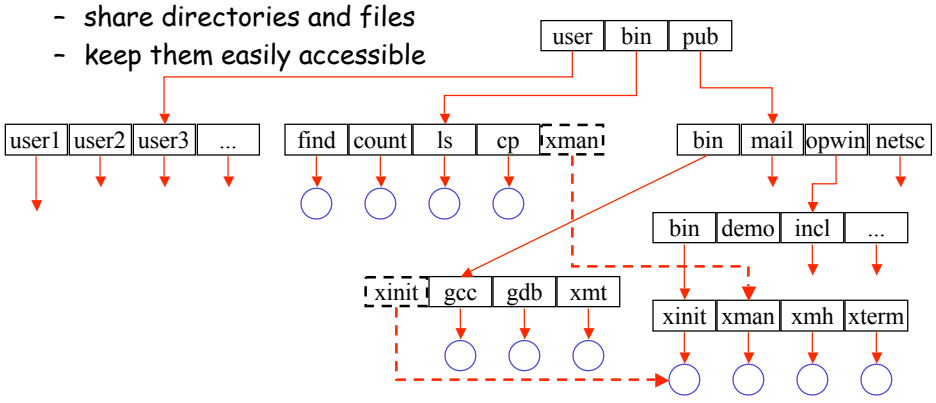
Tree-Structured Directories



- create **subdirectories**
- new concepts:
  - current directory
  - path names: complete vs. relative
  - search paths

Generalized Tree Structures

- share directories and files
- keep them easily accessible



- Links: File name that, when referred, affects file to which it was linked. (hard links, symbolic links)
- Problems:
  - consistency, deletion
  - Why links to directories only allowed for system managers?

UNIX Directory Navigation: current directory

```
#include <unistd.h>

char * getcwd(char * buf, size_t size);
/* get current working directory */
```

Example:

```
void main(void) {
    char mycwd[PATH_MAX];

    if (getcwd(mycwd, PATH_MAX) == NULL) {
        perror ("Failed to get current working directory");
        return 1;
    }
    printf("Current working directory: %s\n", mycwd);
    return 0;
}
```

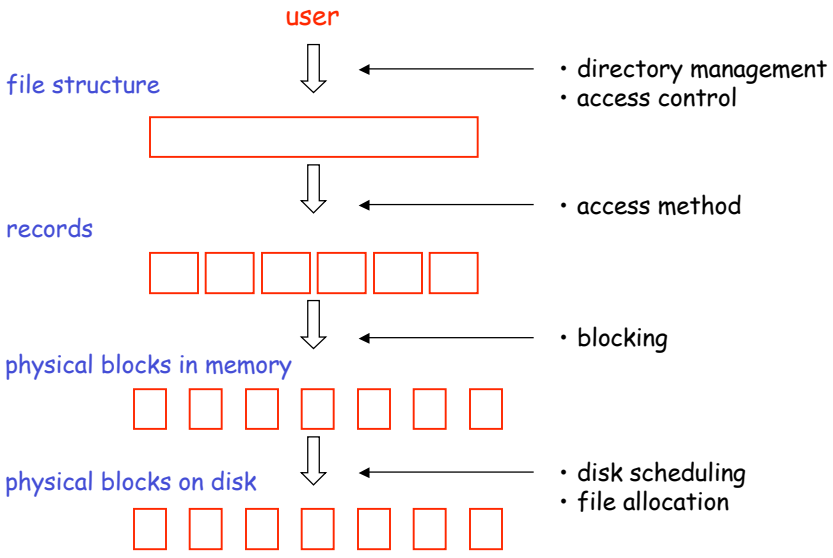
UNIX Directory Navigation: directory traversal

```
#include <dirent.h>

DIR           * opendir(const char * dirname);
        /* returns pointer to directory object          */
struct dirent * readdir(DIR * dirp);
        /* read successive entries in directory 'dirp'  */
int             closedir(DIR *dirp);
        /* close directory stream                       */
void            rewinddir(DIR * dirp);
        /* reposition pointer to beginning of directory */
```

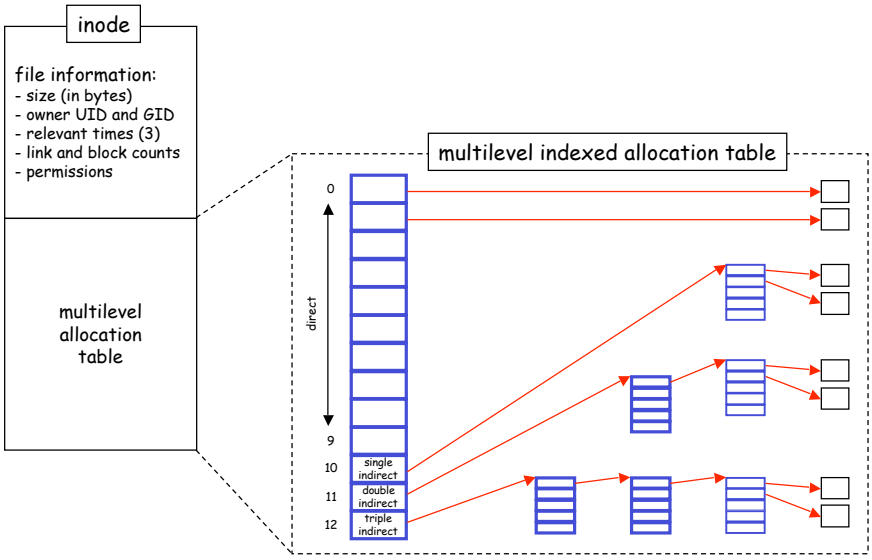
Directory Traversal: Example

```
#include <dirent.h>

int main(int argc, char * argv[]) {
    struct dirent * direntp;
    DIR           * dirp;

    if (argc != 2) {
        fprintf(stderr, "Usage: %s directory_name\n", argv[0]);
        return 1;
    }

    if ((dirp = opendir(argv[1])) == NULL) {
        perror("Failed to open directory");
        return 1;
    }

    while ((dirent = readdir(dirp)) != NULL)
        printf(%s\n", direntp->d_name);
    while((closedir(dirp) == -1) && (errno == EINTR));
    return 0;
}
```

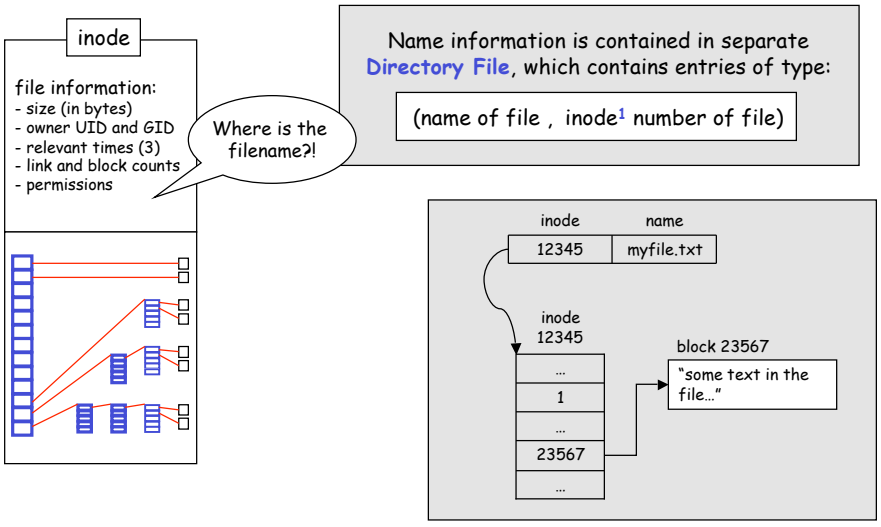
Interlude: Logical View of File Management



Unix File System Implementation: inodes

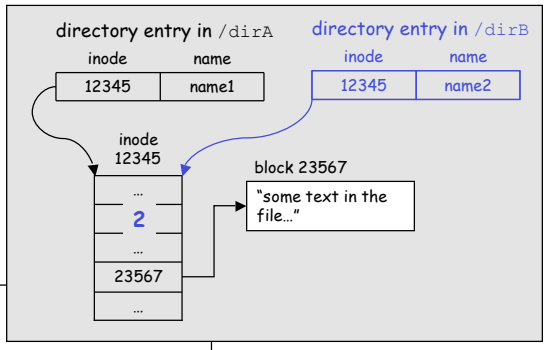


Directory Implementation



[1] More precisely: Number of block that contains inode.

Hard Links

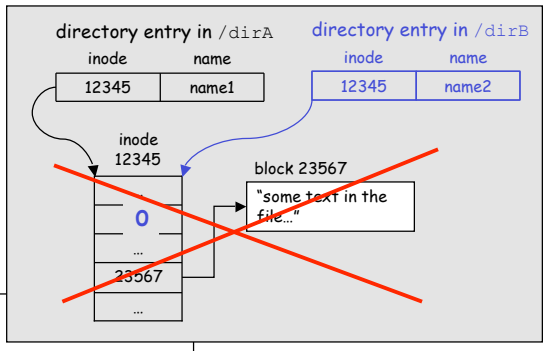


shell command
`ln /dirA/name1 /dirB/name2`

is typically implemented using the link system call:

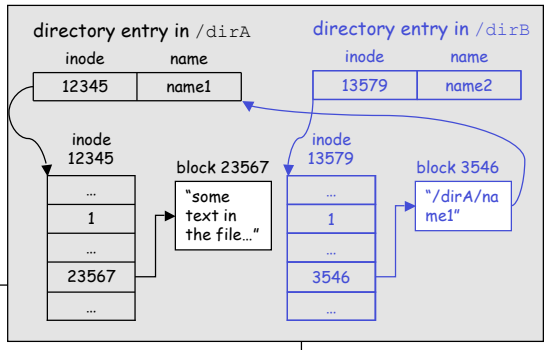
```
#include <stdio.h>
#include<unistd.h>

if (link("/dirA/name1", "/dirB/name2") == -1)
  perror("failed to make new link in /dirB");
```

Hard Links: unlink




```
#include <stdio.h>
#include<unistd.h>

if (unlink("/dirA/name1") == -1)
  perror("failed to delete link in /dirA");


if (unlink("/dirB/name2") == -1)
  perror("failed to delete link in /dirB");
```

Symbolic (Soft) Links



shell command
`ln -s /dirA/name1 /dirB/name2`

is typically implemented using the link system call:

```
#include <stdio.h>
#include<unistd.h>

if (symlink("/dirA/name1", "/dirB/name2") == -1)
  perror("failed to create symbolic link in /dirB");
```