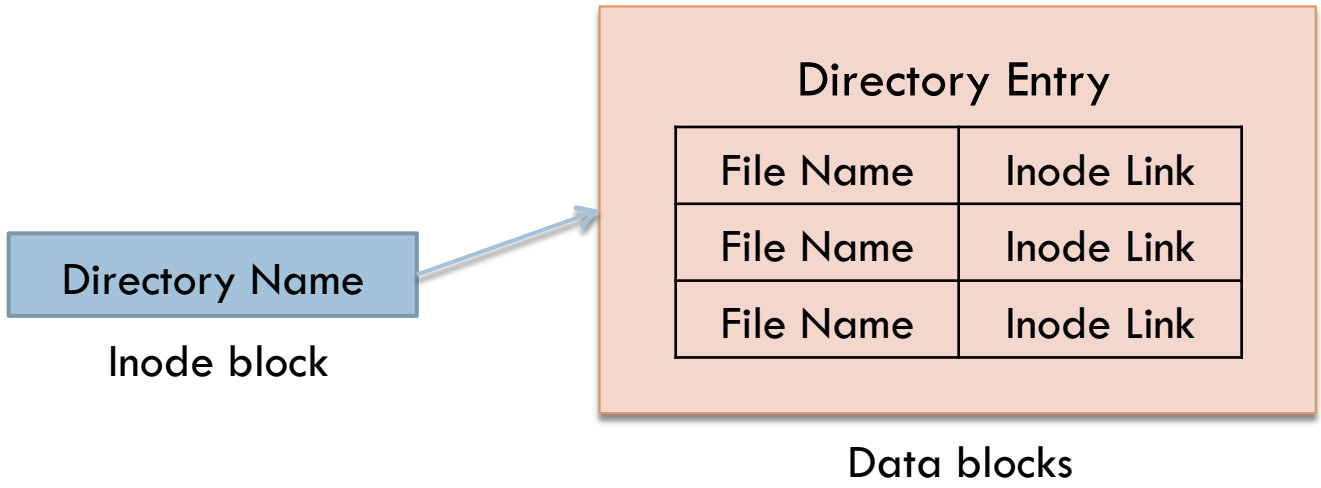
DIRECTORY MANAGEMENT

UNIX Programming 2014 Fall by Euiseong Seo

Directory Definition

- A directory is a file
  - Composed of directory entries
  - Each entry links filename to corresponding inode
  - You can directly read a directory file
    (Never do this!)



Directory Format

- Internal structure of a directory file may differ depending on types of file systems
- You should use POSIX directory access routines
- These routines use dirent structure
  - Each dirent represents an entry in a directory

```
struct dirent {
    ino_t          d_ino;       /* inode number */
    off_t          d_off;       /* not an offset; see NOTES */
    unsigned short d_reclen;    /* length of this record */
    unsigned char  d_type;      /* type of file; not supported
                                   by all filesystem types */
    char           d_name[256]; /* filename */
};
```

Accessing a Directory

- To determine if a filename represents a directory, use stat(), then S_ISDIR() on st_mode
- Directory access routines are modeled after standard I/O library
  1. Open directory
  2. Read directory entries sequentially
  3. Close directory
- DIR data type is used like FILE type

Accessing a Directory

- Prototype

```
#include <dirent.h>

DIR *opendir(const char *dirname);
DIR *fdopendir(int fd);
// fd should not be used after fdopendir

struct dirent *readdir(DIR *dirp);
int closedir(DIR *dirp);


long telldir(DIR *dirp);
void seekdir(DIR *dirp, long loc);
void rewinddir(DIR *dirp);
```

Accessing a Directory

- Example

```
DIR *dirp;
struct dirent *dent;

if((dirp = opendir(argv[1])) == NULL) {
    perror("opendir");
    exit(1);
}

printf("Contents\n========\n");
while(dent = readdir(dirp)) {
    printf("%s\n", dent->d_name);
}

(void)closedir(dirp);
```

Directory Creation and Deletion

- Directories are created with mkdir(2)
- rmdir(2) deletes a directory
  - Erase all directory entries but "." and ".." before deletion
- Prototype

```
#include <sys/types.h>
#include <sys/stat.h>
int mkdir(const char *path, mode_t mode);
int rmdir(const char *path);
```

Hard and Symbloc Links

- When you create a file you are creating a hard link
- You can create more links to an existing file with link(2) system call
- Hard links can only be created to files on the same file system (or partition)

Hard and Symbolic Links

- Symbolic link is a special type of file that has a file name as its contents
- symlink(2) creates a symbolic link
- You can create a symbolic link to a non-existing file
  - Broken symbolic link
- You can create a symbolic link across file systems
- A symbolic link can also point to other symbolic links
  - More than 20 links during name resolution → ELOOP

Hard and Symbolic Links

- Prototype

```
int link(const char *oldpath, const char *newpath);
int symlink(const char *oldpath, const char *newpath);
```

- Most system calls receiving file names will follow symbolic links
- Some calls do not (unlink()), or have other forms that do not follow symbolic links (lstat())

Hard and Symbolic Links

- open() will follow symbolic link
- How can you read contents of a symbolic link file?
  - ls shows contents of symbolic links
- Prototype

```
#include <unistd.h>
ssize_t readlink(const char *path, char *buf, size_t bufsize);
```

Hard and Symbolic Links

```
char *buf
struct stat statbuf;
int n;

if(lstat(argv[1], &statbuf) == -1)
    {perror("lstat"); exit(1);}

if(!S_ISLNK(statbuf.st_mode)) {
    fprintf(stderr, "%s is not a symbolic link.\n", argv[1]);
    exit(1);
}

buf = (char *)malloc(statbuf.st_size+1);
if(buf == NULL)
    {fprintf(stderr, "Out of memory.\n"); exit(1)}

n = readlink(argv[1], buf, statbuf.st_size+1);
if(n==-1) {perror("readlink"); exit(1);}
buf[n] = '\0';
printf("%s\n", buf);
```

Renaming a File

- rename(2) renames a file or a directory
- Prototype

```
#include <stdio.h>

int rename(const char *oldpath, const char *newpath);
```

- rename() does not follow symbolic links
- rename() will delete newpath if it exists
  - if newpath is an non-empty directory name, rename() will fail