

Files

&



Directories



Patrício R. Domingues

Departamento de Eng Informática
ESTG/IPLeiria

- File has content
 - Data: the data of the file
 - Metadata: the data associated with the characteristics of file
 - Owner, group
 - Dates
 - Create time, Access time, Modify time
 - Size
 - Permissions
 - ...

Query metadata of files

- `stat` family of functions to access metadata of a file
- `int stat(const char *path, struct stat *buf);`
- `int fstat(int fd, struct stat *buf);`
- `int lstat(const char *path, struct stat *buf);`
- All the `stat` functions fill in a `struct stat` regarding the file
- The functions differ in how the file is specified
 - `stat`: path of file
 - `fstat`: descriptor of the file
 - `lstat`: identical to `stat`, but symbolic link-aware

`struct stat >>`

struct stat

```
struct stat {  
    dev_t st_dev; ➔ ID of device containing file  
    ino_t st_ino; ➔ inode number  
    mode_t st_mode; ➔ permissions and type  
    nlink_t st_nlink; ➔ number of hard links  
    uid_t st_uid; ➔ userID of owner  
    gid_t st_gid; ➔ group ID of owner  
    dev_t st_rdev; ➔ device ID (if special file)  
    off_t st_size; ➔ total size in bytes  
    blksize_t st_blksize; ➔ blocksize for I/O  
    blkcnt_t st_blocks; ➔ number of blocks  
    time_t st_atime; ➔ last access time  
    time_t st_mtime; ➔ last modification time  
    time_t st_ctime; ➔ last status change time  
};
```

stat – st_mode field

```
struct stat sb;
stat("file.txt", &sb);
switch (sb.st_mode & S_IFMT) {  /* & ➔ binary AND */
    case S_IFBLK:
        printf("block device node\n");break;
    case S_IFCHR:
        printf("character device node\n");break;
    case S_IFDIR:
        printf("directory\n");break;
    case S_IFIFO:
        printf("FIFO\n");break;
    case S_IFLNK:
        printf("symbolic link\n");break;
    case S_IFREG:
        printf("regular file\n");break;
    case S_IFSOCK:
        printf("socket\n");break;
    default:
        printf("unknown\n");
        break;
}
```

Unix stat command (#1)

- Unix has a `stat` command
 - It accesses the metadata of a filesystem entry
- Example

```
stat SO
File: `SO'
Size: 4096          Blocks: 8          IO Block: 4096   directory
Device: 811h/2065d Inode: 3637503      Links: 3
Access: (0755/drwxr-xr-x)  Uid: ( 1000/   user)   Gid: ( 1000/   user)
Access: 2016-04-19 00:09:52.883037341 +0100
Modify: 2016-04-19 00:09:52.059043339 +0100
Change: 2016-04-19 00:09:52.059043339 +0100
Birth: -
```

`stat` with formatted output >>

Unix stat command (#2)

- The `stat` command has the `-c` option for formatted output
- Examples

```
stat -c "mode=%a,nome=%n,last=%y,blocos=%b" SO
```

```
mode=755,nome=SO,last=2016-04-19 00:09:52.059043339 +0100,blocos=8
```

```
stat -c "blocos=%b,size=%s,mount=%m,type=%F" /tmp
```

```
blocos=8,size=4096,mount=/,type=directory
```

- See
 - `man stat`

The `strace` utility >>

Unix stat command (#3)

- Which system calls are used by the `stat` command?
 - Utility `strace`

Example

```
strace stat /tmp
```

```
execve("/usr/bin/stat", ["stat", "/tmp"], [/* 57 vars */]) = 0
brk(0)                                = 0x833b000
access("/etc/ld.so.nohwcap", F_OK)     = -1 ENOENT (No such file or directory)
mmap2(NULL, 8192, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_ANONYMOUS, -1, 0) = 0xb7728000
access("/etc/ld.so.preload", R_OK)     = -1 ENOENT (No such file or directory)
open("/etc/ld.so.cache", O_RDONLY|O_CLOEXEC) = 3
fstat64(3, {st_mode=S_IFREG|0644, st_size=70288, ...}) = 0
mmap2(NULL, 70288, PROT_READ, MAP_PRIVATE, 3, 0) = 0xb7716000
close(3)                               = 0
access("/etc/ld.so.nohwcap", F_OK)     = -1 ENOENT (No such file or directory)
open("/lib/i386-linux-gnu/libselinux.so.1", O_RDONLY|O_CLOEXEC) = 3
(...)
```


Directories (1)

- Directory (in Unix)
 - Contains a list of filenames
 - Each filename maps to an inode number
 - *inode* – element of the filesystem
- A directory can contain other directories
- All directories have a “parent” directory
 - Except the *root* directory, that is, /

```
.
├── bin
├── boot
│   └── grub
├── dev
│   ├── block
│   ├── bsg
│   ├── bus
│   ├── char
│   ├── cpu
│   ├── disk
│   ├── dri
│   ├── fd -> /proc/self/fd
│   ├── input
│   └── mapper
```

```
tree -d -L 2 /
```

Directories (2)

- Each directory contains two special entries
 - “.”: represents the current directory
 - “..”: represents the parent directory
- Example
 - `cd ..`
 - `ls -ld ..`
 - `cp /tmp/*.c .`

- The maximum length of a filename depends on the filesystem
 - ext2, ext3, ext4: name of a file can have up to 255 bytes
 - Some versions have an even longer limit
- Maximum pathname is 4096 bytes long
- Also defined in `<limits.h>`
 - `#define NAME_MAX 255` → # chars in a file name
 - `#define PATH_MAX 4096` → # chars in a path name including nul

Current directory

- Each process has a so-called *current directory*
- Getting the current directory
 - `char *getcwd(char *buf, size_t size);`
- Returns NULL on error and sets `errno`
- Returns `buf` on success, with the the full path for the current directory copied to the address pointed by `buf`
- Example

```
char cwd[256];
if (!getcwd (cwd, sizeof(cwd))) {
    fprintf(stderr, "Error getcwd(): %s", strerror(errno));
    exit (EXIT_FAILURE);
}
printf ("cwd = %s\n", cwd);
```

- Changing the current directory
 - `int chdir(const char *path);`
 - `int fchdir(int fd);`
 - `fd` is a descriptor of an opened directory
- Returns
 - 0 on success
 - -1 on error, with `errno` set appropriately

Creating directories

- Function `mkdir`
 - `int mkdir(const char *path, mode_t mode);`
- `path` represents the path of the directory to create
 - **relative** path or **absolute** path
 - **Relative**: relative to current directory
 - **Absolute**: relative to `/`
- `mode` represents the permission bits
- Returns
 - Failure: `mkdir()` returns `-1` and sets `errno`
 - Success: `mkdir()` returns `0`

Removing directories

- `int rmdir (const char *path);`
- The directory specified by `path` must be empty (except for `.` and `..`)
- Returns
 - 0 if delete was successful
 - -1 on error

Reading a directory (1)

- Content of a directory
 - List of files contained in the directory
- `DIR * opendir (const char *name);`
 - Returns a stream for the directory
- `struct dirent * readdir (DIR *dir);`
 - returns one *struct dirent* entry at the time
 - When the directory is all read, it returns NULL

`struct dirent >>`

Reading a directory (2)

- Content of the struct dirent (Linux)

```
struct dirent {  
    ino_t d_ino; → inode number  
    off_t d_off; → offset to the next dirent  
    unsigned short d_reclen; → len of this record  
    unsigned char d_type; → type of file  
    char d_name[256]; → filename  
}
```

- Note: in POSIX, the struct dirent has only the field **d_name**

Reading a directory (3)

- The function `readdir` is not reentrant
 - Not thread-safe
- `readdir_r` is a reentrant version of `readdir`
- ```
int readdir_r(DIR *dirp, struct dirent *entry,
 struct dirent **result);
```
- `entry` points to a struct `dirent` provided by the caller
- A pointer to the returned item is placed in `*result`
  - If `*result` is returned to NULL and the `readdir_r` returns 0, then the DIR streams is at the end
    - No more entries left in the directory

# Reading a directory (4)

- Finally...
- A DIR descriptor needs to be closed with closedir
  - `int closedir (DIR *dir);`
- Returns
  - 0 on success
  - -1 on error and sets errno appropriately

- Example – show current dir

```
#include ...

int mostra_dir(const char *nome_dir){
 DIR *dir_d;
 int finito, n_entradas;
 struct dirent *dir_entry;
 dir_d = opendir(nome_dir);
 if(dir_d == NULL){
 fprintf(stderr,
 "erro: impossível abrir DIR '%s' - %s\n",
 nome_dir, strerror(errno));
 return -1;
 }
 n_entradas = 0;
 finito = 0;
```

**Continue >>**

# Reading a directory (6)

- Example – show current dir

```
(...) do{
 dir_entry = readdir(dir_d);
 if(dir_entry == NULL){
 if(errno){
 fprintf(stderr,
"erro: readdir (entrada %d)\n", n_entradas);
 closedir(dir_d);
 return -1;
 }else{
 printf("Iteração de DIR '%s' "
 "terminada (%d entradas)\n",
 nome_dir, n_entradas);
 finito = 1;
 }
 }else{
 printf("entrada: '%s'\n", dir_entry->d_name);
 n_entradas++;
 }
 }while(finito==0);
```

- Example – show current dir

(...)

```
if(closedir(dir_d) == -1){
 fprintf(stderr, "erro: impossível fechar DIR '%s' - %s\n",
 nome_dir, strerror(errno));
 return -1;
}
```

```
printf("DIR '%s': %d entradas\n", nome_dir, n_entradas);
return 0;
}
```

}

```
int main(void){
 char *nome_diretorio = "."; /* current DIR */
 mostra_dir(nome_diretorio);
 return 0;
}
```

# Walking through directories (#1)

- Walking through directories
  - Visit each directory of a hierarchy of directories
- Example
  - How to visit each directory only once?

5 directories

\$tree

```
.
├── level01
│ ├── level01.txt
│ └── level02
│ ├── level02.txt
│ └── level03
│ └── level03.txt
└── localDir
 ├── subDir
 └── local.txt
```

5 directories, 4 files

# Walking through directories (#1)

- Walking through directories
  - Visit each directory of a hierarchy of directories
- Example
  - How to visit each directory only once?
    - Recursive approach

5 directories

\$tree

```
.
├── level01
│ ├── level01.txt
│ └── level02
│ ├── level02.txt
│ └── level03
│ └── level03.txt
├── localDir
│ └── subDir
└── local.txt
```

5 directories, 4 files



# Walking through directories (#2)

- Walking through directories
  - See
    - “Travessia de uma árvore de diretórios usando recursividade”, Revista Programar #51, Dezembro 2015
    - <http://www.revista-programar.info/edicoes/edicao-51-dezembro-2015/>
- Code for directory traversal
  - <http://pastebin.com/7YfUJE7L>



- *“File and Directory Management”, Chapter 8 - Linux System Programming, Robert Love, 2<sup>nd</sup> Edition, O’Reilly, 2013*
- Man pages for file and directory management
  - `man 1 stat (command)`
  - `man 2 stat`
  - `man 3 opendir`
  - `man 3 readdir`
  - ...