Operating Systems 2017/2018 | API Files & Directories

API Files & Directories

Ricardo Rocha

Department of Computer Science

Faculty of Sciences

University of Porto

For more information please consult

‘Advanced Programming in the UNIX® Environment, 3rd Edition,

W. Richard Stevens and Stephen A. Rago, Addison Wesley’

Sections 4.1–4.8 and 4.22

Getting Info About a File

```c
#include <sys/stat.h>

int stat(char *pathname, struct stat *buf);
int lstat(char *pathname, struct stat *buf);
int fstat(int filedes, struct stat *buf);
// * gets information about a file (struct stat) given a
//   pathname (stat()/lstat()) or a file descriptor (fstat())
// * lstat() avoids symbolic links returning information about
//   the symbolic link itself, not the file referenced by it
// * returns 0 if successful, -1 on error
```

Getting Info About a File

```
// * the definition of ‘struct stat’ can differ among systems,
//   but the most important fields could look like follows
struct stat {
  dev_t st_dev;         // device number (file system)
  ino_t st_ino;         // i-node number (serial number)
  mode_t st_mode;       // file type & mode (permissions)
  uid_t st_uid;         // user ID of owner
  gid_t st_gid;         // group ID of owner
  off_t st_size;        // size in bytes, for regular files
  time_t st_atime;      // time of last access
  time_t st_mtime;      // time of last modification
  blkcnt_t st_blocks;   // number of disk blocks allocated
  ... };
```

File Types

- We can determine the file type by using the macros:
  - **S_ISREG()** for regular files
  - **S_ISDIR()** for directory files
  - **S_ISCHR()** for character special files
  - **S_ISBLK()** for block special files
  - **S_ISFIFO()** for pipes or FIFOs
  - **S_ISLNK()** for symbolic links
  - **S_ISSOCK()** for sockets

```
struct stat buf;
if (lstat(pathname, &buf) < 0) { /* lstat error */ }
...
if (S_ISREG(buf.st_mode)) printf("regular file\n");
```

File Access Permissions

- We can determine the file access permissions bits by using the macros:
  - **S_IRUSR** owner has read permission
  - **S_IWUSR** owner has write permission
  - **S_IXUSR** owner has execution permission
  - **S_IRGRP** group has read permission
  - **S_IWGRP** group has write permission
  - **S_IXGRP** group has execution permission
  - **S_IROTH** others have read permission
  - **S_IWOTH** others have write permission
  - **S_IXOTH** others have execution permission

```
...
if (buf.st_mode & S_IRUSR)
  printf("owner has read permission\n");
```

File Mode Creation Mask

```
#include <sys/stat.h>

mode_t umask(mode_t cmask);
// * sets the file mode creation mask for the process and
//   returns the previous value (never returns an error)
// * receives as argument the bitwise OR of any of the nine
//   file access permissions bits
// * the file mode creation mask is used whenever the process
//   creates a new file/directory. Any bits that are on in the
//   file mode creation mask are turned off in the file's mode
```

File Mode Creation Mask: Example

```
#define RWRWRW (S_IRUSR|S_IWUSR|S_IRGRP|S_IWGRP|S_IROTH|S_IWOTH)

int main () {
  umask(0);
  if (open("new1.txt", O_CREAT | O_TRUNC, RWRWRW) < 0)
    { /* open error */ }
  umask(S_IRGRP | S_IWGRP | S_IROTH | S_IWOTH);
  if (open("new2.txt", O_CREAT | O_TRUNC, RWRWRW) < 0)
    { /* open error */ }
}      Running example: $ ./a.out
                        $ ls -l new1.txt new2.txt
                        -rw-rw-rw- 1 xpto 0 Dec 7 21:20 new1.txt
                        -rw------- 1 xpto 0 Dec 7 21:20 new2.txt
```

Reading Directories

```
#include <dirent.h>

DIR *opendir(char *pathname);
DIR *fdopendir(int fd);
// * opens a directory stream given a pathname (opendir()) or
//   a file descriptor (fdopendir())
// * returns a DIR pointer if successful, NULL on error
// * the DIR structure is an internal structure used to
//   maintain information about the directory stream being read

int closedir(DIR *dp);
// * closes an open directory stream
// * returns 0 if successful, -1 on error
```

Reading Directories

```
#include <dirent.h>

struct dirent *readdir(DIR *dp);
// * reads the next directory entry in a directory stream
// * returns a ‘struct dirent’ pointer if successful, NULL if
//   error or if at the end of the directory

// * the definition of ‘struct dirent’ can differ among
//   systems, but the most common fields look like follows
struct dirent {
  ino_t d_ino;                // i-node number
  char d_name[NAME_MAX + 1];  // null-terminated filename
}
```

Reading Directories: Example

```
int main () {
  DIR *dp;
  struct dirent *dirp;
  ...
  if ((dp = opendir(pathname)) = = NULL)
    { /* opendir error */ }
  while ((dirp = readdir(dp)) != NULL) {
    ...  // process entries
  }
  if (closedir(dp) < 0)
    { /* closedir error */ }
  ...
}
```