

CS241 File Systems

Files & Directories
... a practical primer for LMP1

Lawrence Angrave

Robbins Ch4.1-4.3, 4.6, 5.2

Stallings Ch12

?

API... open? stat?

Why so many options in open?

How do I make my code robust?

What concepts underpin the
POISX filesystem API?

What exactly is a file,
directory...?
I-node?

Unix concept of a FILE

Byte-orientated & sequential

So what?

Unicode file

Typed Fields and Records

Key indices

Unified. Get the contents of a file as bytes

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

```
ssize_t read(int fd, void *buf, size_t nbyte);
```

read

Don't need to read entire file

... Grab bytes into your buffer

... Next call to read will read next unread byte

Can even read the bytes of a 'directory'

read() boundary cases to trip you up

May not read anything

- signal interruption

- end of file

- asynchronous non-blocking mode

- hardware issue

Need to check # bytes actually read

It usually works as you might assume

Upon successful completion, `read()`, `readv()`, and `pread()` return the number of bytes actually read and placed in the buffer.

The system guarantees to read the number of bytes requested if the descriptor references a normal file that has that many bytes left before the end-of-file, but in no other case.

read() boundary cases

Practical Guide

Return 0: End of File

Return -1: Check errno and act accordingly

e.g. EINTR; restart

Return +N: # bytes placed into buffer

Sequential bytes != C string

Doesn't know or care about strings and NUL bytes

Ensure NUL termination before using printf debug statements contents!

A good alternative is to use write...

```
write(2, buf, nbytes)
```

What are we reading from?

Get and maintain a reference to a file

Integer file descriptor

POSIX `open()`

Let's dig deeper...

... Errors and options

... Directory organization & implementation

... How does O/S Manage file descriptors?

open – read – close

```
fd=open("/tmp/1.txt", options);
```

```
read(fd,buffer,sizeof(buffer));
```

```
close(fd);
```

open –... – read – ... – close

```
fd=open("/tmp/1.txt", options);
```

```
if(fd <1) error (e.g. File doesn't exist)
```

```
r=read(fd,buffer,sizeof(buffer))
```

```
handle special cases (eof,restart, media errors)
```

```
close(fd);
```

```
free up resources
```

What can go wrong with open?

EACCES The requested access to the file is not allowed, or search permission is denied for one of the directories in the path prefix of pathname, or the path does not exist yet and write access to the parent directory is not allowed. (See also `path_resolution(2)`.)

EEXIST pathname already exists and `O_CREAT` and `O_EXCL` were used.

EFAULT pathname points outside your accessible address space.

EISDIR pathname refers to a directory and the access requested involved writing (that is, `O_WRONLY` or `O_RDWR` is set).

ELOOP Too many symbolic links were encountered in resolving pathname, or `O_NOFOLLOW` was specified but pathname was a symbolic link.

EMFILE The process already has the maximum number of files open.

ENAMETOOLONG pathname was too long.

ENFILE The system limit on the total number of open files has been reached.

ENODEV pathname refers to a device special file and no corresponding device exists. (This is a Linux kernel bug; in this situation `ENXIO` must be returned.)

ENOENT `O_CREAT` is not set and the named file does not exist. Or, a directory component in pathname does not exist or is a dangling symbolic link.

ENOMEM Insufficient kernel memory was available.

ENOSPC pathname was to be created but the device containing pathname has no room for the new file.

ENOTDIR A component used as a directory in pathname is not, in fact, a directory, or `O_DIRECTORY` was specified and pathname was not a directory.

ENXIO `O_NONBLOCK` | `O_WRONLY` is set, the named file is a FIFO and no process has the file open for reading. Or, the file is a device special file and no corresponding device exists.

EOVERFLOW pathname refers to a regular file, too large to be opened; see `O_LARGEFILE` above.

EPERM The `O_NOATIME` flag was specified, but the effective user ID of the caller did not match the owner of the file and the caller was not privileged (i.e., did not have `CAP_FOWNER`).

EROFS pathname refers to a file on a read-only filesystem and write access was requested.

ETXTBSY pathname refers to an executable image which is currently being executed and write access was requested.

EWOULDBLOCK The `O_NONBLOCK` flag was specified, and an incompatible lease was held on the file (see `fcntl(2)`).

Just open the file. Please

EACCES

EEXIST

EFAULT

EISDIR

ELOOP

EMFILE

ENAMETOOLONG

ENFILE

ENODEV

ENOENT

ENOMEM

ENOSPC

ENOTDIR

ENXIO

E_OVERFLOW

EPERM

EROFS

ETXTBSY

EWOULDBLOCK

EACCES

EEXIST

EFAULT

EISDIR

ELOOP

EMFILE

ENAMETOOLONG

ENFILE

ENODEV

ENOENT

ENOENT :

O_CREAT is not set and the named file does not exist. Or, a directory component in pathname does not exist or is a dangling symbolic link.

Options and modes

R vs. R/W vs. Write only

Truncate an existing file before writing

Read-Write-Execute Permissions

```
open("/tmp/1.txt", ...)
```

Pathnames -> traverse directories

Imposes a hierarchy on files

Files can be referenced from more than one directory

Organization of files useful for security

stat

Meta-information about a file
modification and access time
Kind of file (e.g. Directory | regular file?)
Support for symbolic links

- **Three flavors**

```
int stat(const char *path, struct stat *buf);
```

```
int fstat(int filedes, struct stat *buf);
```

- *Info about link (more on this later)*

```
int lstat(const char *path, struct stat *buf);
```

```
struct stat {
    dev_t    st_dev;    /* ID of device containing file */
    ino_t    st_ino;    /* inode number */
    mode_t   st_mode;   /* protection */
    nlink_t  st_nlink; /* number of hard links */
    uid_t    st_uid;   /* user ID 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 filesystem I/O */
    blkcnt_t st_blocks; /* number of blocks allocated */
    time_t   st_atime; /* time of last access */
    time_t   st_mtime; /* time of last modification */
    time_t   st_ctime; /* time of last status change */};
```

Stat macros (`st_mode`)

`S_ISREG(m)` is it a regular file?

`S_ISDIR(m)` directory?

`S_ISCHR(m)` character device?

`S_ISBLK(m)` block device?

`S_ISFIFO(m)` FIFO (named pipe)?

`S_ISLNK(m)` symbolic link?*

`S_ISSOCK(m)` socket?*

*(Not in POSIX.1-1996.)

open –... – stat – ... – close

```
fd=open("/tmp/1.txt", options);
```

```
if(fd <1) error (e.g. File doesn't exist)
```

```
r=fstat(fd,&buffer)
```

```
handle special cases (eof,restart, media  
errors)S_ISDIR(buffer.st_mode)
```

```
close(fd);
```

```
free up resources
```

Directories Robbins Ch.5.2

```
struct dirent *entry;
    // add error handling!
DIR *dirp = opendir(".");

while((entry = readdir(dirp)) != NULL)
    printf("%s\n", direntp->name);

closedir(dirp);
```

Directories

readdir will return "." and ".."

readdir returns a pointer to a static structure

i.e. not threadsafe, not recursive-safe

Can't call opendir recursively!?!?

Don't forget to closedir!

System Perspective

File Descriptors

File Descriptors

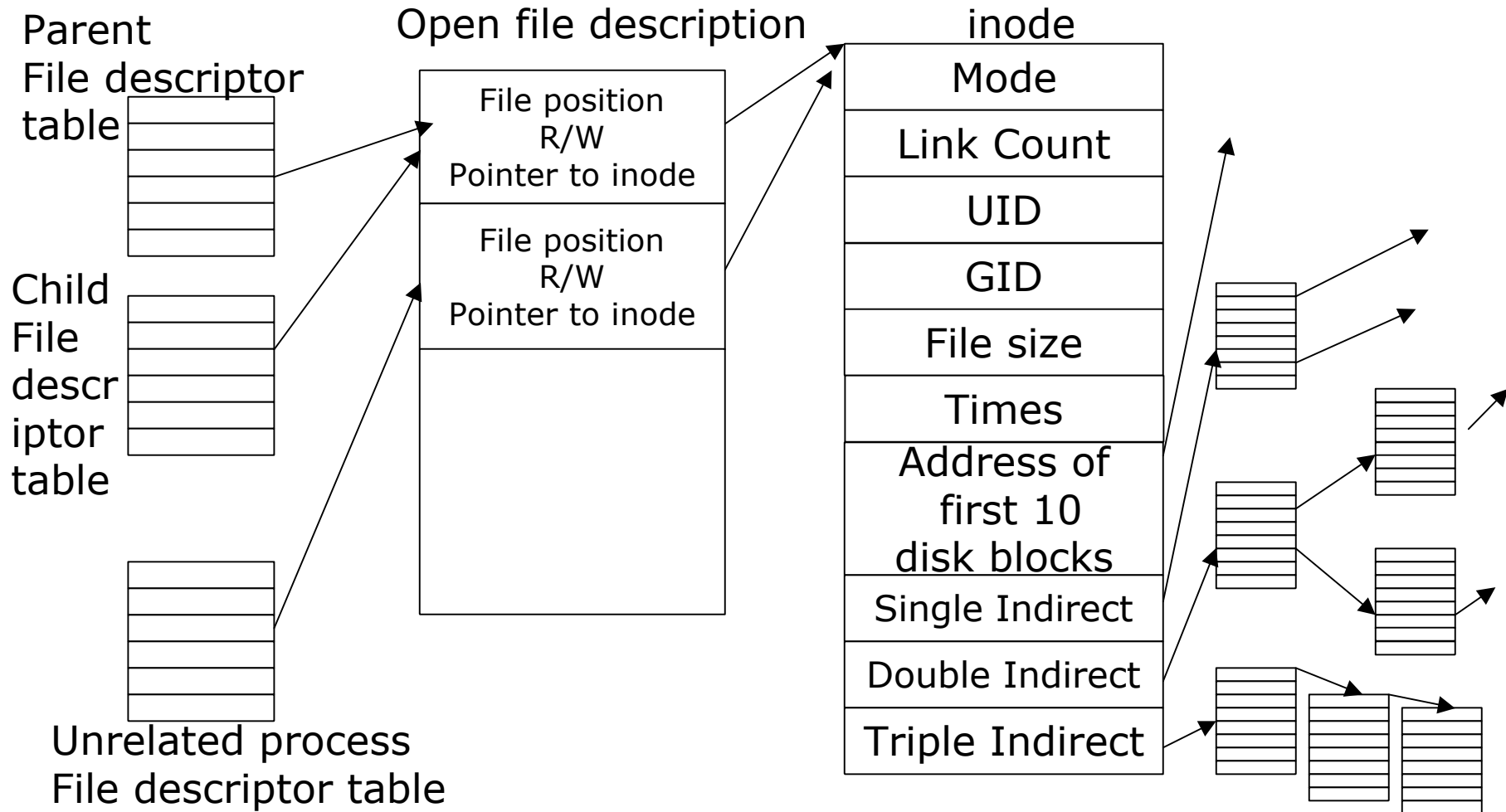
Specific to each process

Stored

fork() child inherits a copy of the parent's file descriptors

File descriptors to I-Nodes

UNIX file structure implementation



LMP1

Be lazy ... Read & use restart library
Read man pages for error codes
Useful code examples in Robbins Ch4,5

Understand Bonnie

Restart library makes programs simpler!

```
#include <unistd.h>
#include "restart.h"
#define BLKSIZE 1024
int copyfile(int fromfd, int tofd) {
    char buf[BLKSIZE];
    int bytesread, byteswritten;
    int totalbytes = 0;
    for ( ; ; ) {
        if ((bytesread = r_read(fromfd, buf, BLKSIZE)) <= 0)
            break;
        if ((byteswritten = r_write(tofd, buf, bytesread)) == -1)
            break;
        totalbytes += byteswritten;
    }
    return totalbytes;
}
```

Use the restart library for read/write –

e.g. r_write

```
ssize_t r_write(int fd, void *buf, size_t size) {
    char *bufp;
    size_t bytestowrite;
    ssize_t byteswritten;
    size_t totalbytes;
    for (bufp = buf, bytestowrite = size, totalbytes = 0;
        bytestowrite > 0;
        bufp += byteswritten, bytestowrite -= byteswritten) {
        byteswritten = write(fd, bufp, bytestowrite);
        if ((byteswritten) == -1 && (errno != EINTR))
            return -1;
        if (byteswritten == -1)
            byteswritten = 0;
        totalbytes += byteswritten;
    }
    return totalbytes;
}
```

LMP1

Staged

Test-driven development

Future LMPs build on LMP1

Bonnie

read,write,directory

benchmarks

Light reading for spring break?

The Diamond Age (Young Lady's Illustrated Primer)
, Neal Stephenson

Cryptonomicon, Neal Stephenson

Victorian Internet, Tom Standage

The man who mistook his wife for a hat (and other
clinical tales) , Oliver Sacks