UNIX File System

- Boot block: contains bootstrap code to boot OS
- Super block: describes state of file system size, how many files, location of free space
- Inode list: list of inodes
- Data blocks: contains file data

Inodes consist of

- File owner
- File type
- Access permissions (rw-r–r–)
- Size / # of blocks
- Access, Modify, and Status Time Change
- Access to data blocks (n data block pointers)

stat(2)

Obtains information about the file pointed to by *path* (or in the case of fstat(2), fd). If *path* is a symlink, returns info about the link itself.

```
struct stat {
    mode_t st_mode; /* file type & mode (permissions)*/
    ino_t st_ino; /* i-node number (serial number) */
    dev_t st_dev; /* device number (file system) */
    dev_t st_rdev; /* device number for special files*/
    nlink_t st_nlink; /* number of links */
    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 */
    struct timespec st_atim; /* last access time */
    struct timespec st_mtim; /* last modification time */
    struct timespec st_ctim; /* file status change time*/
    blksize_t st_blksize; /* best I/O block size */
    blkcnt_t st_blocks; /* # disk blocks allocated */
}
```

```
#include <sys/stat.h>
int stat(const char *path, struct stat *sb);
int lstat(const char *path, struct stat *sb);
int fstat(int fd, struct stat *sb);
```

st_mode

regular - most comon, interpretation is up to app

directory - contains names of other files and pointer to said info

char special - used for certain types of devices (eg terminal)

block special - used for disk devices

FIFO - used for interprocess communication

socket - used for network and non-network communication

symbolic link - points to another file

st_uid, st_guid

Every process has six or more IDs associated with it. st_uid and st_gid always specify the user owner and group owner of a file

real user/group ID	who we really are
effective user/group ID / extra group IDs	used for file access perms
saved set-user-ID / saved set-group-ID	saved by exec funcs

Access Tests Performed by Kernel:

1. e-uid == 0 (root), access granted
2. e-uid == st_uid, access granted if user permission set
3. e-gid == st_gid, access granted if group permission set
4. other permission bit set, access granted
5. access denied

chmod(2), lchmod(2), fchmod(2)

Changes the permission bits on the file.

```
#include <sys/stat.h>
#include <fcntl.h>
int chmod(const char *path, mode_t mode);
int lchmod(const char *path, mode_t mode);
int fchmod(const char *path, mode_t mode);
int fchmodat(int fd, const char *path, mode_t mode, int flag);
returns 0 if OK, -1 on error
```

Standard I/O

- 3 streams automatically created
  - STDIN_FILENO
  - STDOUT_FILENO
  - STDERR_FILENO
- These three are referenced as **stdin**, **stdout** , and **stderr**.
- Defined in stdio.h

Streams and FILEs

- fopen() returns pointed to FILE object
  - File descripter
  - Pointer to stream buffer
  - Buffer size
  - Number of chars in buffer
  - Error flag
- Applications need not examine FILE object
- Pass FILE pointer as argument to standard I/O funcs
- FILE * is called file pointer

Six ways to open standard I/O stream

Restriction	r	w	a	r+	w+	a+
File must already exist	•			•		
Previous contents discarded		•			•	
Readable	•			•	•	•
Writable		•	•	•	•	•
Readable at end of file			•			•

type	Description	open(2) Flags
r or **rb**	Reading	O_RDONLY
w or **wb**	Writing	O_WRONLY|O_CREAT|O_TRUNC
a or **ab**	Create for append	O_WRONLY|O_CREAT|O_APPEND
r+ or **rb+**	Update (reading and writing)	O_RDWR
w+ or **wb+**	Create/Erase and append	O_RDWR|O_CREAT|O_TRUNC
a+ or **ab+**	Open/Create for reading and append	O_RDWR|O_CREAT|O_APPEND

Buffering

- The goal of buffering is to use minimum # of read/write calls.
- Three types:
  - Fully Buffered: I/O performed when buffer is full
  - Line Buffered: I/O performed when newline is encountered
  - Unbuffered: I/O performed immediately

setbuf(): turns buffering on/off

```
void setbuf(FILE *fp, char *buf);
- a buffer of length BUFSIZ (stdio.h)
- NULL (disable buffering)
```

fclose: close an open stream

```
int fclose(FILE *fp);
- Output buffer data is flushed
- Input buffer data is discarded
- Automatically allocated buffer is released
- returns 0 if OK, EOF on error
```

3 types of Unformatted I/O

Character-at-a-time : getc, fgetc, getchar

Line-at-a-time : fgets, fputs

Direct : fread, fwrite