File I/O

Advanced Systems Programming in C/C++/Rust
(SoSe 21)

Harshavardhan Unnibhavi

Topics

- What is a filesystem?
- Filesystem abstractions
- Filesystem interface
- A very simple file system
- Read/Write on the very simple file system
- Speeding up filesytem operations

What is a filesystem?

- UNIX's defining feature: "Everything is a file"
- Network sockets, devices, data on disk...
- Filesystem is an implementation of the file interface
    - Decide how data is stored on disk
    - Performance, scalability, failures etc
    - Common operations include open, read, write
- Popular examples
    - ext4
    - procfs, sysfs
    - tmpfs

Filesystem abstractions: File

- 10,000 foot view: Linear array of bytes
- Each file has a low-level name called the **inode**.
    - Contains metadata about file
        - Size, permissions, creation time, last access time, file type
        - Locations of data via direct or indirect pointers.
- This metadata can be accessed via the **stat** system call.
- Files are created using the **open** system call.

Filesystem abstractions: Directory

- Each directory has an inode number.
- Contents are specific
  - Mapping between inodes and names
  - Eg: (foo, 10)
- Directory hierarchy.
  - Root directory (/)
  - "/" used to separate subsequent subdirectories and files
  - Eg: /foo/bar/hello.txt

Filesystem interface

- Create files or directories
- Access files or directories
    - Read or write
- Delete files or directories

Filesystem interface: Detour

- strace
  - Awesome tool
  - Refer to the previous lecture for more detail.

```
prompt> strace cat foo
…
openat(AT_FDCWD, "foo", O_RDONLY)       = 3
fstat(3, {st_mode=S_IFREG|0664, st_size=6, ...}) = 0
fadvise64(3, 0, 0, POSIX_FADV_SEQUENTIAL)  = 0
mmap(NULL, 139264, PROT_READ|PROT_WRITE,
MAP_PRIVATE|MAP_ANONYMOUS, -1, 0) = 0x7fad4e7b9000
read(3, "hello\n", 131072)              = 6
write(1, "hello\n", 6hello)             = 6
read(3, "", 131072)
...
```

Filesystem interface: Creating files

- **int open(const char ****pathname***, int** *flags***, mode_t** *mode***)**
  - For example to create a file use the **O_CREAT** flag.
  - More flags and modes of operations in man pages.
- Open() returns a file descriptor (fd)
  - Allows you to perform certain operations on the file.
  - Managed by the OS on a per process basis.
    - fd is an index into a global, open file table.
    - The open file table has one entry for each open call.
  - Every process has 3 file descriptors: stdin(0), stdout(1), stderr(2).

Filesystem interface: Reading and writing to files

- **ssize_t read(int** *fd*, **void ****buf*, **size_t** *count***)**
  - Return number of bytes read
- **ssize_t write(int** *fd*, **const void ****buf*, **size_t** *count***)**
  - Return number of bytes actually written
- **off_t lseek(int** *fd*, **off_t** *offset*, **int** *whence***)**
  - Change read or write offset into the file
  - Tracked by the OS for each open file.
- **int fsync**(**int** fd)
  - Force write all dirty data to disk
  - Useless sometimes: https://lwn.net/Articles/752063/

Filesystem interface: More calls to consider

- int rename(const char *oldpath, const char *newpath)
- int stat(const char *path, struct stat *buf)
- int mkdir(const char *pathname, mode_t mode)
- int readdir(unsigned int fd, struct old_linux_dirent *dirp,
- unsigned int count)
- int unlink(const char *pathname)
- int rmdir(const char *pathname)

Filesystem interface: Hard and soft links

- Hard links
  - ln [TARGET] [LINK NAME]
  - Another name for the same inode
  - Links incremented by one
  - File data deleted only if number of links is 0.
  - Cannot hard link to directory or files in other disk partitions.
- Symbolic or soft links
  - ln -s [TARGET] [LINK NAME]
  - Symbolic link is a file itself containing pathname of the target file as its data

Filesystem interface: Make and mount

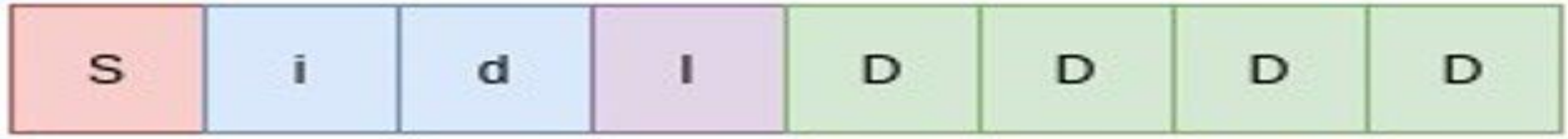
- **mkfs**
  - Takes as input a device and a filesystem type
  - Write an empty filesystem onto that partition
- **mount**
  - Make the filesystem available within the filesystem tree at an existing directory
  - Directory called mount point.

A very simple file system: Assumptions

- Assumptions
  - The storage device presents itself as a set of blocks that are logically addressable from 0 to N-1, where N is the total number of blocks.
  - Each block size is 4KiB.
  - Total number of blocks is 8.
  - Inode is 256 bytes in size.

A very simple file system: Organization

- 4 data blocks
- 1 inode block
- 1 block to store inode bitmaps
- 1 block to store data bitmaps
- 1 block to store the superblock of the filesystem
  - Contains filesystem metadata
  - Location is known
  - OS first reads this block before mounting the file system.



A very simple file system: Inode

- Contains file metadata
  - Type of file
  - Size of file
  - Blocks
  - Protection information
  - Time information
- Data blocks information
  - Direct pointers
  - Indirect pointers

A very simple file system: Directory

- Inode whose type is "directory".
- Inode contains pointers to data blocks
- Directory data blocks contain mapping between
  - File inode number
  - File name

A very simple file system: Free space management

- Bitmaps are used to track free space
  - Set bit indicates that the inode or data block is used up.
- Inode bitmap tracks free inodes
- Data bitmap tracks free data blocks.

A very simple file system: Read operation

- Open /foo/bar
- Read from the file
- Close the file
- Assume file is of 8KiB I.e 2 data blocks

A very simple file system: Read operation

	Data bitmap	Inode bitmap	Root inode	Foo inode	Bar inode	Root data	Foo data	Bar data[0]	Bar data[1]
open(bar)			read	read	read	read	read		
read()					read write			read	
read()					read write				read

A very simple file system: Write operation

- Create /foo/bar
- Write to the file
- Close the file
- Assume 4KiB of data, i.e 1 block, is written to the file.

A very simple file system: Write operation

	Data bitmap	Inode bitmap	Root inode	Foo inode	Bar inode	Root data	Foo data	Bar data[0]
create(/foo/bar)		read write	read	read write	read write	read	read write	
write()	read write				read write			write

A very simple file system: Caching

- Poor performance
  - Request same data blocks from disk repeatedly
  - Slow operation compared to reading from memory
- Solution
  - Cache blocks in main memory
  - Strategies to decide which blocks to keep and which to evict from memory
    - LRU
  - Modern OSes integrate virtual memory pages and file system pages into a unified page cache.

A very simple file system: Caching

- Write buffering
  - Buffer writes in main memory instead of immediately persisting them.
  - Advantages:
    - Batch updates
    - Better scheduling of I/O
      - Sequential faster than random
    - Possibility of avoiding writes

See you at the Q&A