Files

{ File Ops
  Meta-Data
  Directories
  Links

Understand disks:
  - SCSI
  - remember interface: blocks, read/write
  - why not just let users
  - use disk (read/write blocks)
  - protection, convenience, performance, 

File

User: “named collection of bytes”
  => array of bytes (ordering)
  => randomly accessible
  => permanent (lives across reboots, etc.)
  => convenient

OS: must convert disk/type blocks => file abstraction
  => must manage disk effectively

Role of FS

Naming
  how to select file
  find blocks

Protection
  only certain users to do certain things

Reliable/Available
  don’t lose info

Efficient
  space
  time

File Ops

Create
  find space, name

Access
  read, write, seek

Delete
  free space

Truncate
  set size to 0 (or some lesser value)

Rename

Unix:
  create(), open(), dopen()
  read(), write(), lseek()
  close()
  unlink()
  open("file", O_TRUNC)
  rename()
  chmod()
**Meta-Data**: FS keeps info about each file

- Name of file
- Type?
- Pointer(s) to data blocks
- Size
- Access/Mod. Time
- Owner, group
- Protection info
- Other: Special file (directory/sym link)

**Unix** e.g.

```
ls -li <file>
```

```
<inode #>  <protection bits>  <numLinks>  <owner>  <uid>  <gid>  <mode>  cpe
```

```
1000 -rw-rw--- 1
```

FS must track meta info/file

inode: in **Unix**

**FS**: on-disk data structure + access methods

**Disk**

index # (inode number)

```
index
# 1 3 4 5
```

file blocks

inode

```
10 20 30
```

```
```

```
```

```
```
Directories

> Organizing files
  Logical grouping

> Single-level directories
  Single directory for whole disk
  each file unique
  $\Rightarrow$ dedicate part of disk for directory
  (name, inode) pairs

> Two-level
  Directory per user
  /rem/ file1.c
  Somewhat more flexible

> Arbitrary Tree e.g. /a/b/c/file.c
  Directory: just a special file
  data: <file name, inode #> pairs
  e.g. directory "a"
  $\Rightarrow$ b/c/x.c
  name not in inode
  but in dir (2)
  index number

> How to distinguish "file" from "dir"?
  bit in meta-data

> Special separator used to parse path "/

> Special directories
  Root: top of tree, known
  .: this dir
  ..: parent dir

> E.g. mkdir /a/b/c
  root 2: find "a" $\Rightarrow$ 5
  5: find "b" $\Rightarrow$ 9
  ensure "c" doesn't exist
  allocate new inode (12), add "c" to "b"'s data
Opening Files

```
fd = open("file", o_RDONLY);
read/write(fd, buffer, size) or
close(fd);
```

Solution: open file first
- search directories, find file
- make entry in "open file" table
- return index into table (file descriptor/handle)
- use index to access file

reads writes = fast

Per-process open file table
- current "position" in file
- index into system open file table

Global "open file" table
- sized
- keeps count: how many processes have file open?

```
read("/a/b/c", buffer, size);
```

expensive
Links

> Might want 2+ names to refer to same file

( could copy ... )

⇒ links

Hard link:

c. `ln srcFile newLinkFile`

⇒ creates another directory entry w/ same index #

⇒ remove a file (unlink)

⇒ decrement ref count
deallocate when ref ≡ 0

Can't refer to directories (avoid cycles in tree)

Symbolic Link

`ln -s srcFile newLinkFile`

special file (bit in meta-data)

contents of file:

name of other file

What happens if original is removed?
Path Names

Absolute Path: start at root/full path
\textbackslash x \textbackslash y \textbackslash z \Rightarrow \text{cumbersome (shell)}

Relative:
- notion: "current" directory/ process
  look for files in current dir
  \texttt{\textbackslash cd \textbackslash x \textbackslash y \textbackslash z} \Rightarrow \text{full path?}
  \texttt{\textbackslash open(\textquotesingle\textbackslash foo\textquotesingle\textbackslash)} \Rightarrow \text{?}
  \texttt{\textbackslash open(\textquotesingle\textbackslash b/\textbackslash foo.c\textquotesingle\textbackslash)} \Rightarrow ?
  \texttt{\textbackslash open(\textquotesingle\textbackslash foo\textquotesingle\textbackslash)}

Short Cuts
- \texttt{\textbackslash \textasciitilde \textbackslash remai} (shell \Rightarrow \textbackslash home \textbackslash remai)
  etc.

Protection

> Types of access
  - read, write, execute, append, delete, list...

> Access Control List (ACL)
  - List of users w/ rights per file
  - very flexible (AFS)
  - cumbersome, hard to manage, does not scale

> Simpler: Bits
  - owner, group, everybody
  - read, write, execute
  - easy to implement
  - not as general