

File System, Part 1: Introduction

goldcase edited this page 12 days ago · 42 revisions

Design a file system! What are your design goals?

The design of a file system is difficult problem because there many high-level design goals that we'd like to satisfy. An incomplete list of ideal goals includes -

- Reliable and robust (even with hardware failures or incomplete writes dues to power loss)
- Access (security) controls
- Accounting and quotas
- Indexing and search
- Versioning and backup capabilities
- Encryption
- Automatic compression
- High performance (e.g. Caching in-memory)
- Efficient use of storage De-duplication

Not all filesystems natively support all of these goals. For example, many filesystems do not automatically compress rarely-used files

What are `.`, `..`, and `...` ?

- `.` represents the current directory
- `..` represents the parent directory
- `...` is NOT a valid representation of any directory (this not the grandparent directory)

What are absolute and relative paths?

Absolute paths are paths that start from the 'root node' of your directory tree. Relative paths are paths that start from your current position in the tree.

What are some examples of relative and absolute paths?

If you start in your home directory ("`~`" for short), then `Desktop/cs241` would be a relative path. Its absolute path counterpart might be something like `/Users/[yourname]/Desktop/cs241` .

Edit

New Page

▼ Pages 51

Home

#Example Markdown

#Informal Glossary

#Piazza: When And How to Ask For Help

C Programming, Part 1: Introduction

C Programming, Part 2: Text Input And Output

C Programming, Part 3: Common Gotchas

C Programming, Part 4: Debugging

Deadlock, Part 1: Resource Allocation Graph

Deadlock, Part 2: Deadlock Conditions

File System, Part 1: Introduction

File System, Part 2: Files are inodes (everything else is just data...)


File System, Part 3: Permissions

File System, Part 4: Working with directories

File System, Part 5: Virtual file systems

Show 36 more pages...

Clone this wiki locally

 Clone in Desktop

How do I simplify a/b/ ../c/ ./ ?

Remember that .. means 'parent folder' and that . means 'current folder'.

Example: a/b/ ../c/ ./

- Step 1: cd a (in a)
- Step 2: cd b (in a/b)
- Step 3: cd .. (in a, because .. represents 'parent folder')
- Step 4: cd c (in a/c)
- Step 5: cd . (in a/c, because . represents 'current folder')

Thus, this path can be simplified to a/c

Why make disk blocks the same size as memory pages?

To support virtual memory, so we can page stuff in and out of memory.

What information do we want to store for each file?

- Filename
- File size
- Time created, last modified, last accessed
- Permissions
- Filepath
- Checksum
- File data (inode)

What are the traditional permissions: user – group – other permissions for a file?

Some common file permissions include:

- 755: rwx r-x r-x

user: rwx , group: r-x , others: r-x

User can read, write and execute. Group and others can only read and execute.

- 644: rw- r-- r--

user: rw- , group: r-- , others: r--

User can read and write. Group and others can only read.

What are the the 3 permission bits for a regular file for each role?

- Read (most significant bit)
- Write (2nd bit)
- Execute (least significant bit)

What do "644" "755" mean?

These are examples of permissions in octal format (base 8). Each octal digit corresponds to a different role (user, group, world).

We can read permissions in octal format as follows:

- 644 - R/W user permissions, R group permissions, R world permissions
- 755 - R/W/X user permissions, R/X group permissions, R/X world permissions

What is an inode? Which of the above items is stored in the inode?

From [Wikipedia](#):

In a Unix-style file system, an index node, informally referred to as an inode, is a data structure used to represent a filesystem object, which can be one of various things including a file or a directory. Each inode stores the attributes and disk block location(s) of the filesystem object's data. Filesystem object attributes may include manipulation metadata (e.g. change, access, modify time), as well as owner and permission data (e.g. group-id, user-id, permissions).

How does inode store the file contents?

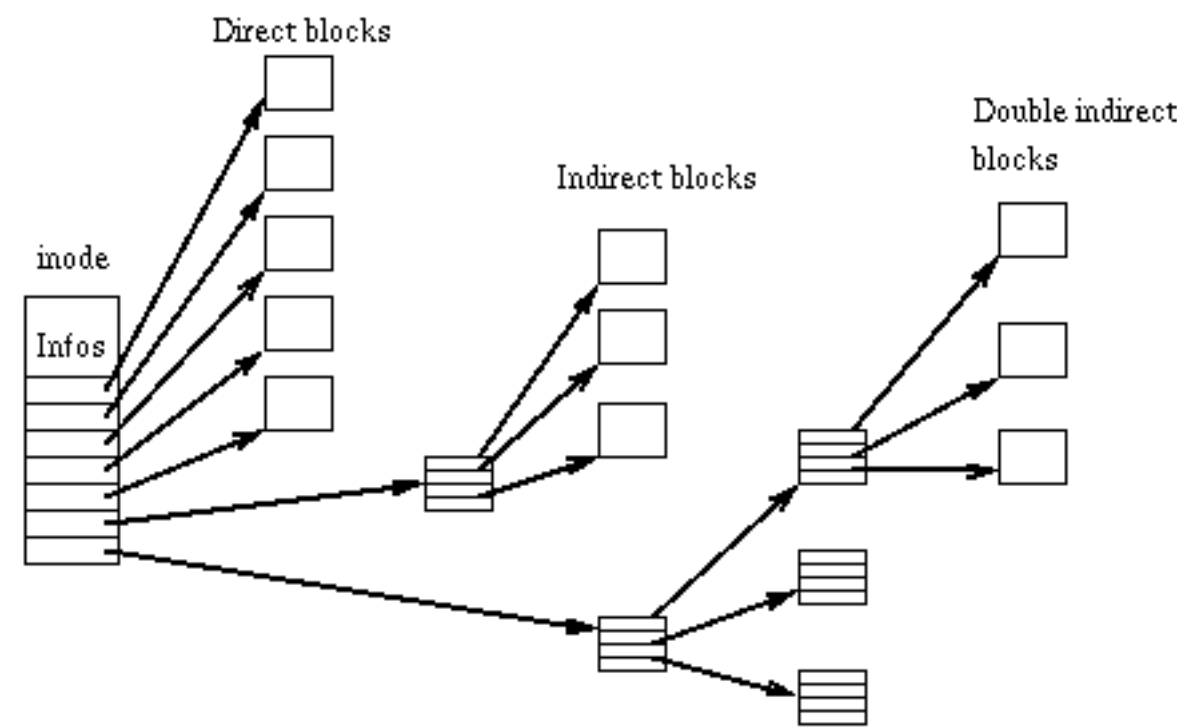


Image source: <http://en.wikipedia.org/wiki/Ext2>

"All problems in computer science can be solved by another level of indirection" - David Wheeler

How many pointers can you store in each indirection table?

As a worked example, suppose we divide the disk into 4KB blocks and we want to

address up to 2^32 blocks.

The maximum disk size is 4KB *2^32 = 16TB (remember 2^10 = 1024)

A disk block can store 4KB / 4B (each pointer needs to be 32 bits) = 1024 pointers. Each pointer refers to a 4KB disk block - so you can refer up to 1024*4KB = 4MB of data

For the same disk configuration, a double indirect block stores 1024 pointers to 1024 indirection tables. Thus a double-indirect block can refer up to 1024 * 4MB = 4GB of data.

Similarly a triple indirect block can refer up to 4TB of data.

[Go to File System: Part 2](#)

Legal and Licensing information: Unless otherwise specified, submitted content to the wiki must be original work (including text, java code, and media) and you provide this material under a [Creative Commons License](#). If you are not the copyright holder, please give proper attribution and credit to existing content and ensure that you have license to include the materials.

