Concurrent Systems

3D4 ⟷ CS2016

Operating Systems

Andrew Butterfield
ORI.G39, Andrew.Butterfield@scss.tcd.ie

Trinity College Dublin
Coláiste na Tríonóide, Baile Átha Cliath
The University of Dublin

with thanks to Mike Brady

File Systems

- A [computer] file originally referred to information stored on punched cards – e.g. a program or a data set was stored on 'files' of punched cards.
- Nowadays, a file is normally considered to be a one-dimensional array of bytes. (BTW, in the original Mac OS, a file consisted of two one-dimensional arrays, the "data fork' and the "resource fork".)
- Files have *unique IDs*.
- Files are normally associated with *directories* and with *path names*.

Files

- Files were originally used to organise storage facilities.
  - Also used for representing and handling other tree-like structures,
    - e.g. device 'trees', like "one-wire filesystem" (OWFS), etc.
- Thus, a file would be represented as a data structure that mapped a unique file ID to an allocation of space on the storage device.
  - Nowadays, that data structure is almost always called an *inode*.
- Thus, all the files in a file system would be represented as a collection of unique inodes.

the *inode*

- An *inode* represents a file, and contains whatever information is necessary to represent it.
  - e.g. information about which blocks of storage on the disk are allocated to store the file.
  - the *filesystem* of which the inode is a member.
  - but *not* a file name.
    - A file can have any number of file names.

Directory Entries

- A directory entry links two items:
  - A file name
  - An *inode* reference
- Many OSes will have a *link* command to enable the creation of directory entries, sometimes called *hard links*.

Two kinds of files…

- *Plain Files*
  - These are files that are used to contain data, formatted and encoded as appropriate
- *Directory Files*
  - These are special files that just contain directory entries, i.e. links between names and inodes.
  - Another way of saying this is that directory files contain lists of zero or more files:
    - They can be plain files or directory files.

inodes & link counts

- A link count is maintained for each inode, i.e. the number of directory entries referring to it.
- When a file is created, it is normally given one directory entry, giving it an initial link count of 1.
- Each hard link increments the link count.
- When a file is ‘deleted’, it is deleted by name, and what actually happens is the directory entry containing the file’s name and inode is deleted, and the inode’s link count is decremented.
- If an inode’s link count is 0, it effectively has no name, thus may be eligible for deletion.

file names

- Most frequently, a file will have a symbolic name – a *filename* – associated with it – i.e. there exists at least one mapping between a filename and the file's inode. This mapping is called a *directory entry*.

- A file can have more than one directory entry. That is, there can be many symbolic file names all pointing to the same inode.

- Extra directory entries are called hard links in unix.
  A file with no directory entries is inaccessible via the file system, so (perhaps) can be deleted and its resources recovered.

pathnames

- Imagine a file ‘foo.txt’ has just been created.
  - A new file is created and its inode and the name ‘foo.txt’ are formed into a directory entry and stored in a directory file, called, say, ‘samples’.
    - In turn, that file has a directory entry in another directory file called, say, ‘Documents’, which in turn … has an entry in special directory file called the master directory file.
- The file’s inode can be found by traversing the directory files from the master file:
- <root>—<“Documents”>—<“samples”>—<“foo.txt”>

pathname

- More compactly, it can be represented by the "path" of directory files to the inode:
- /Documents/samples/foo.txt
- with "/" being the directory name separator -- the 'path separator'.

deleting a file

- Deleting a file simply
  - deletes the associated directory entry
  - decrements the link count, which may reach 0.

Working with a file

- To work with a file, it must be *open*.
  - Opening a file associates a data structure with the file, containing, e.g.
    - File's Inode,
    - Current position in the file,
    - Mode of access (read/write/append, etc.)
    - Process or task that has opened it

Working with a file

- This data structure is deleted when the file is closed
- A file could be open many times simultaneously
- A file's inode keeps track of the number of times it's open (the open count)

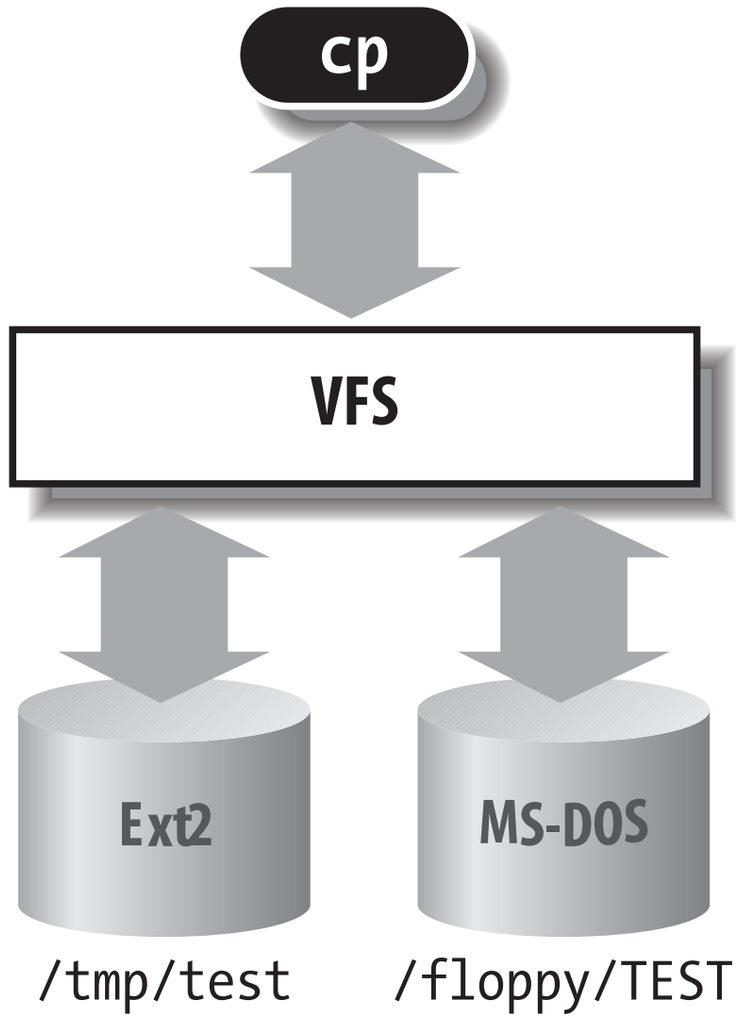
The Filesystem

- Files exist within a filesystem, a set of parameters and programs used to represent and manage
  - inodes,
  - aspects of the underlying I/O driver
  - disk quota,
  - etc.
- The filesystem is typically represented by a **Filesystem Control Block** (FCB). In the Virtual File System of Linux—the Linux VFS—the generalised data structure is called a *superblock*.

The Linux Virtual File System

- The Linux VFS represents underlying file systems in a uniform way – the UNIX way – to client programs.

- The VFS hands off appropriate commands and data to the underlying file systems, while presenting a generic UNIX file-handling API to the client.
  (Not always completely successfully!)

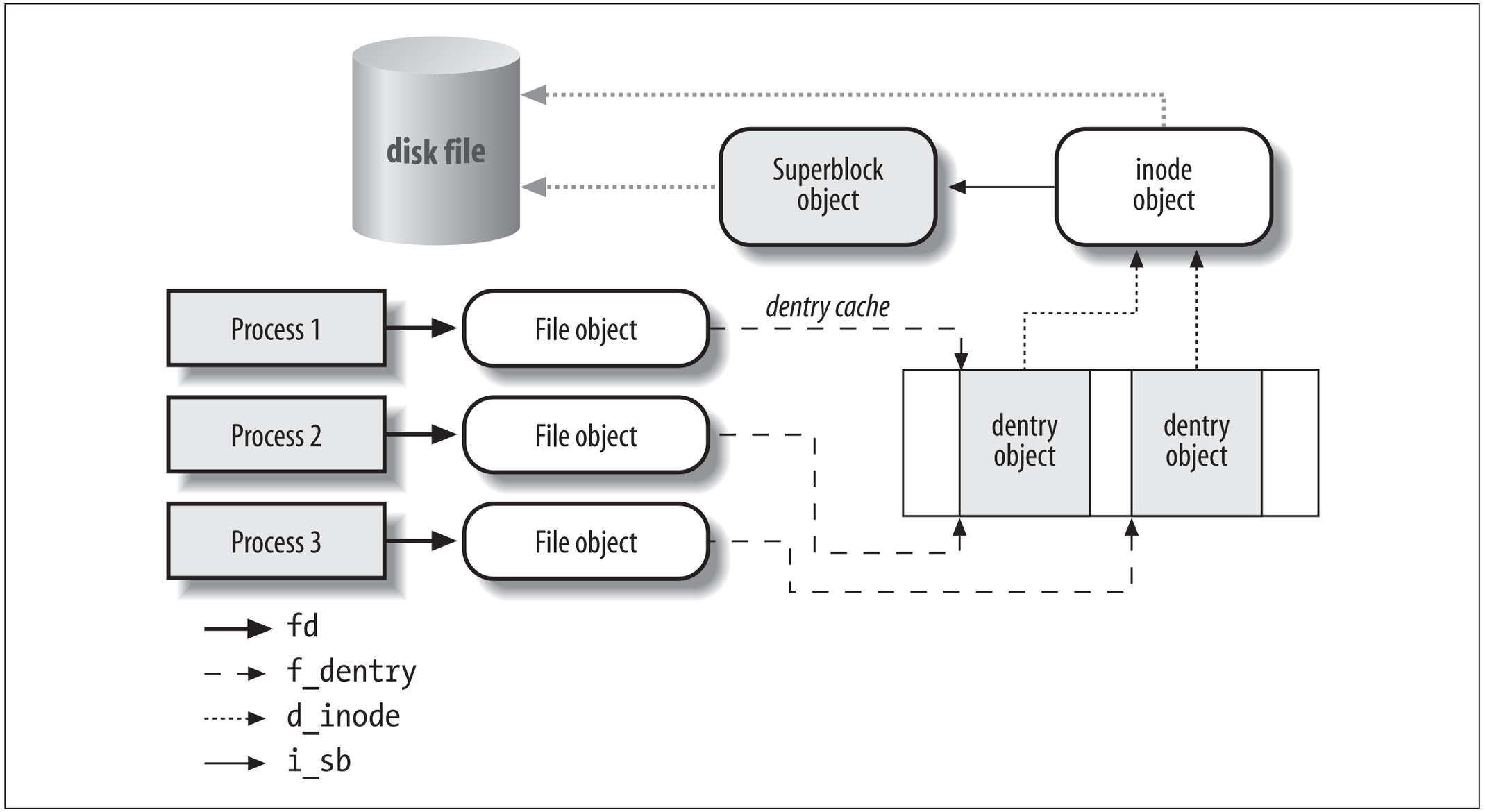


From: Understanding the Linux Kernel, 3rd Ed.

Some supported filesystems

- Second, Third and Fourth Extended Filesystems (EXT2, EXT3 and EXT4), 'native' to Linux
- UFS ("Unix File System")
- ISO9660 & UDF — CDROM & DVD Standard File systems
- MS-DOS, VFAT, NTFS — Windows
- HFS, HFS+, APFS — Macintosh
- NFS, SMB/CIFS, AFS, AFP — Various Networked File Systems
- 'Special' file systems that don't represent storage, e.g. /proc

Some components in the Linux VFS



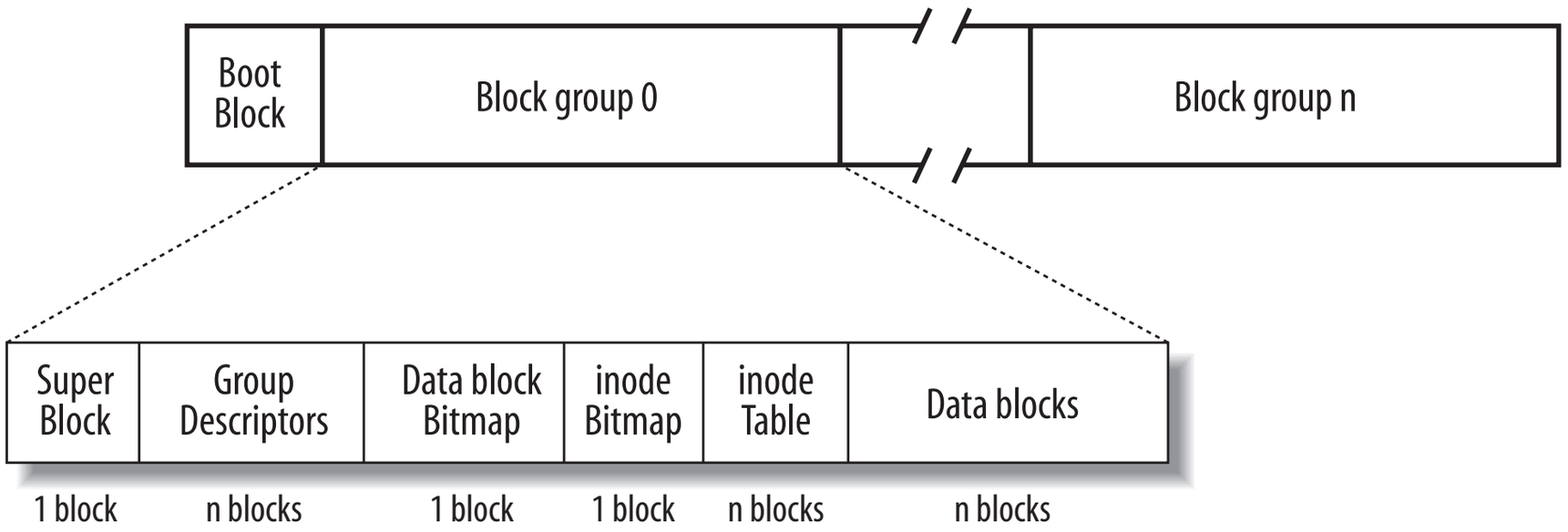
From: Understanding the Linux Kernel, 3rd Ed.

Quick Look at the Ext2 File System

- Very widely used in the Linux world.
- Ext3 is an evolution of Ext2 that has *journalling* — a two-step approach to applying changes to the file system — later!

Layout of an EXT2 Volume Partition

- A boot block and **n** *block groups*
- **n** depends on block size and partition size.
- Block size may be chosen to be from 1,024 to 4,096 bytes.
- Data block bitmap (free/occupied) must fit in one block.



From: Understanding the Linux Kernel, 3rd Ed.

Block Groups

- Block Group advantages
  - File data is closer to related metadata
  - The superblock is replicated for reliability

Block Groups

- The superblock contains key file system metadata
  - size of the file system
  - number of blocks
  - (partial) list of free blocks
  - index of next free blocks
  - size of the inode list
  - number of free i-nodes
  - list of free i-nodes
  - index of next free i-node
  - locks for free i-node and free block lists
  - modified flag

EXT2

- The block size can be selected when the partition is being initialised.
  - Small give a better match between file size and file allocation
  - Big gives less file fragmentation
- The number of inodes to be defined in a partition is specified at initialisation time.
- Fast soft links (aka aliases in the Mac) are supported.

Block Group

- Contains a copy of the superblock, which could be used by file-system repair utilities.
- The *block bitmap* indicates which blocks in the block group are free. The bitmap must fit into a block.
  - Max block size = 4096 bytes, thus 32768 blocks, thus
  - Max block group size = 128 MB
- The *inode bitmap* indicates which inodes are free or occupied.
- The group descriptor lays out this information.

Group Descriptor Fields

Type	Field	Description
__le32	bg_block_bitmap	Block number of block bitmap
__le32	bg_inode_bitmap	Block number of inode bitmap
__le32	bg_inode_table	Block number of first inode table block
__le16	bg_free_blocks_count	Number of free blocks in the group
__le16	bg_free_inodes_count	Number of free inodes in the group
__le16	bg_used_dirs_count	Number of directories in the group
__le16	bg_pad	Alignment to word
__le32 [3]	bg_reserved	Nulls to pad out 24 bytes

From: Understanding the Linux Kernel, 3rd Ed.

Inodes

- Recall that the inode is what really represents a file.
- In EXT, an inode occupies 128 bytes, thus inodes fit neatly into blocks.

EXT2 inode structure (on disk)

Type	Field	Description
__le16	i_mode	File type and access rights
__le16	i_uid	Owner identifier
__le32	i_size	File length in bytes
__le32	i_atime	Time of last file access
__le32	i_ctime	Time that inode last changed
__le32	i_mtime	Time that file contents last changed
__le32	i_dtime	Time of file deletion
__le16	i_gid	User group identifier
__le16	i_links_count	Hard links counter
__le32	i_blocks	Number of data blocks of the file
__le32	i_flags	File flags
union	osd1	Specific operating system information
__le32 [EXT2_N_BLOCKS]	i_block	Pointers to data blocks
__le32	i_generation	File version (used when the file is accessed by a network filesystem)
__le32	i_file_acl	File access control list
__le32	i_dir_acl	Directory access control list
__le32	i_faddr	Fragment address
union	osd2	Specific operating system information

A File's Block Allocation

- An Ext2 inode has an array of, typically, 15 block pointers.
- Blocks 0–11 point to the logical block numbers of the first 12 blocks of the file.
- Block 12 points to a logical block containing space for a further b/4 logical block pointers, if necessary. ('b' = blocksize.)
- Block 13 adds a second level of indirection.
- Finally, block 14 adds a third level of indirection.

Free Blocks

- How do we know which file system blocks are free?
  - When we want to allocate blocks to a file, how we know which blocks are already allocated to existing files and which are free?
- Traditional UNIX file systems used a linked-list approach to free block management
  - When the file system is initialized (formatted), every data block address is placed in a conceptual array of free blocks

Free Blocks

- Some disk data blocks are used to store the array
- The blocks are linked together in a linked list
- A small number of free disk blocks is maintained in the superblock
  - Efficient assignment of blocks to files when required
  - Allows the start of the linked list to be located (since the location and format of the superblock is well defined)
  - Blocks are allocated from the superblock free block list
  - When blocks are freed (by truncating or deleting a file), the free blocks are added to the end of the superblock free block list,

EXT2

- Very successful, mature, stable.
- Shortcomings:
  - Block fragmentation.
  - Compressed & encrypted files not handled ‘transparently’.
  - Journaling (sp!).