Linux File Systems

Rémy Card
Université Pierre et Marie Curie (Paris VI)
Remy.Card@linux.org

Outline

- Introduction
- Well Known File Systems
- File Systems in Linux
- The Ext2 File System Internals
- The Ext2 File System Tools
- Performances
- Conclusion

Introduction

- File Concept
- File Operations
- Directories
- File System Organization
- Space Management
- Directory Structure

File Concept

- File = logical storage unit
- Collection of related informations
- Files are mapped onto physical device
- Files refered to by names
- Files attributes:
  - type
  - location
  - size
  - protection
  - time, date, user identification

File Operations

- The OS provides operations to manipulate files
- Fundamental operations:
  - create
  - write
  - read
  - positionning
  - delete
  - truncate

Directories

- Files are organized in directories
- Directories operation:
  - search for a file
  - create a file
  - delete a file
  - list files
  - rename a file
  - traverse file system

File System Organization

- Devices are subdivided into physical blocks
- Files are made up of logical blocks
- The FS translates files logical blocks to physical blocks
- The FS uses the routines provided by the device driver to read and write physical blocks

Application Programs
↓ op(file, size)
Logical File System
↓ op(file, log. block)
File Organization Module
↓ op(device, block)
I/O Control
↓ request to controler
Devices

Space Management

- Block allocation:
  - Contiguous allocation
  - Linked allocation
  - Indexed allocation
- The FS has to keep track of allocated and free blocks
- Free list implementation:
  - Bits vector
  - Linked list
  - Grouping
  - Counting

Directory Structure

- Organization:
  - Single-level directory
  - Two-level directory
  - Tree-Structured directories
  - Acyclic-Graph directories
  - General-Graph directories
- Implementation:
  - Linear directories
  - Hash table

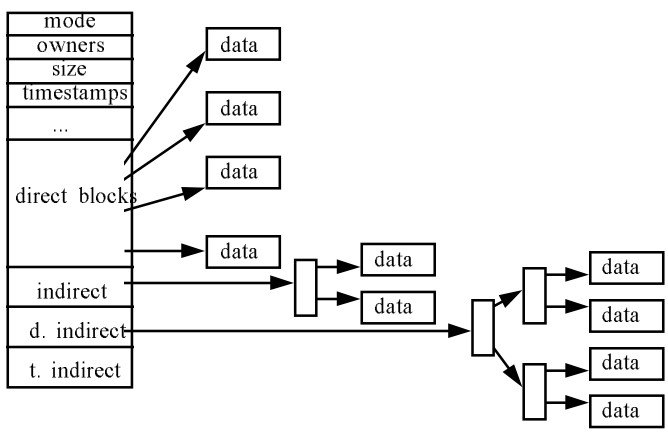
Well Know File Systems

- Fundamental Unix File Systems Concepts
- Unix V7 File System
- BSD Fast File System
- Log File Systems (Sprite, 4.4BSD)
- Viva File System

Inodes in Unix

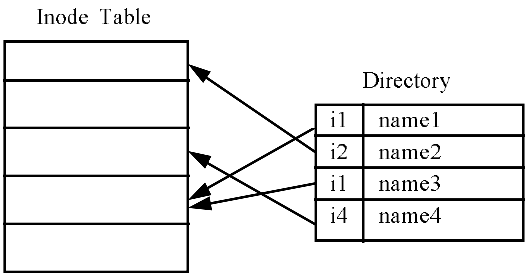
- Each file is represented by an “inode”
- An inode contains the description of the file:
  - mode (file type, access rights)
  - owners (uid, gid)
  - times (access, data modification, inode modification)
  - size
  - pointers to data blocks
  - ...
- Data blocks addresses are kept in the inode
- Inodes are numbered and grouped into an inode table

Structure of a Unix Inode



Directories

- Directories are another type of files
- A directory contains a list of entries:
  - inode number
  - file name



Device Special Files

- Character or block special files
- No disk space associated with special files
- I/O requests are forwarded to a (pseudo) device driver
- Referenced in the kernel by:
  - a major number (identifies the driver)
  - a minor number (identifies the unit)

Unix V7 File System (1)

- "Traditional" file system
- One file system per logical disk
- Layout of the file system:

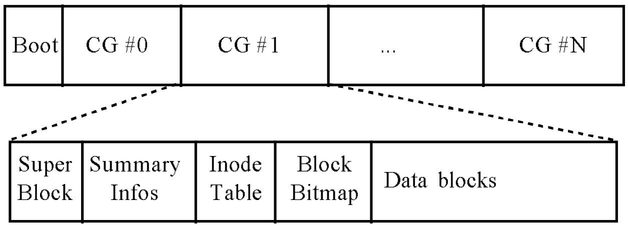
Boot	Super Block	Inode Table	Data Blocks

Unix V7 File System (2)

- Small block size (512 bytes):
  - Many indirect blocks
  - Many I/O requests to read data from a file
- Free blocks managed as a list:
  - FS tends to become fragmented after some use
  - Lots of heads seeks
- Directories:
  - Arrays of fixed size entries (14 characters)

BSD Fast File System

- New organization:
  - Bigger blocks
  - Cylinder groups
  - Variable length directory entries
- Layout of the file system:



Improvements in BSD FFS

- Bigger blocks:
  - Faster I/O
  - Less indirect blocks
- Fragments:
  - fragment = smaller block allocated at the end of a file
  - up to 8 fragments per block
  - reduce the disk space waste (50% of the last block)
  - but: as files grow, fragments have to be relocated and recopied into bigger fragments
- Cylinder groups:
  - control structures (e.g. superblock) are replicated

New policies in BSD FFS

- Synchronous writes for metadata (inodes, directories, indirect blocks):
  - Maintain FS metadata consistency
  - But: may result in file data corruption
- Cylinder groups:
  - Data blocks are allocated “near” their inode
  - File inodes are allocated “near” their parent directory’s inode
- Uses physical geometry of the disks:
  - useful on “old” hardware
  - suboptimal with modern drives

Log File Systems

- Implemented in the Sprite OS and 4.4 BSD
- Suitable for large servers with lots of memory (assume that most blocks are present in the buffer cache) - Sprite LFS needs 37 MB of RAM for 10 FS!
- LFS are optimized to make their writes sequential
- The FS is updated as a circular log: data are written at the tail of the log
- The knowledge of the log status and of the blocks writes order allows perfect recovery after a crash

VIVA File System (VIFS)

- Based on BSD FFS
- Not restricted to 8 fragments per block
- VIFS uses a sequential allocation mechanism:
  - separate bitmaps for free single blocks, double blocks, quadruple blocks, and so on...
  - location of contiguous free blocks is easy to find
- Indirect blocks are stored in compressed form:
  - Sequential allocations of blocks are stored in the indirect block
  - 20 MB of data may be stored in a 16 KB indirect block
  - But: holes are difficult to implement
- 128 bytes of data can be stored in the inode itself

File Systems in Linux

- History of Linux File Systems
- Description of File Systems
- The VFS Layer

History of Linux File Systems

- First Linux file system: Minix File System:
  - 14 characters file names
  - Maximal size: 64 MB
- April 1992: Extended File System:
  - Variable length file names (up to 255 characters)
  - Maximal size: 2 GB (theorical: 4 TB)
- January 1993: New Extended File System (renamed as Second Extended File System)
- January 1993: XIA File System

The Minix File System

- First Linux file system
- Implemented by Linus Torvalds when Linux was being cross-developed under Minix
- Simple file system:
  - strong limitations:
    - » 14 characters file names
    - » Maximal size: 64 MB
  - very similar to the V7 FS (but uses bitmaps instead of linked lists)
- New extension: supports 30 characters file names

The Extended File System

- Second Linux file system (based on the Minix FS kernel code)
- Implemented by Rémy Card (personal need) and distributed to the Linux community
- Less limitations than the Minix FS
- But: poorly designed free space management:
  - Free blocks are stored in a list
  - The FS tends to become fragmented after some use
  - Slower than the Minix FS
- FS defragmenter written by Stephen Tweedie
- Now obsolete

XIA FS

- Designed and implemented by Frank Xia as a simple and efficient file system (based on the Minix FS kernel code)
- New algorithms:
  - Variable length directory entries
  - Bigger control structures (disk addresses, inodes)
  - optimized allocation routines
- Features very similar to the Extended File System's ones
- But better performances than extfs (uses bitmaps instead of linked lists to keep track of free blocks)

The Second Extended File System

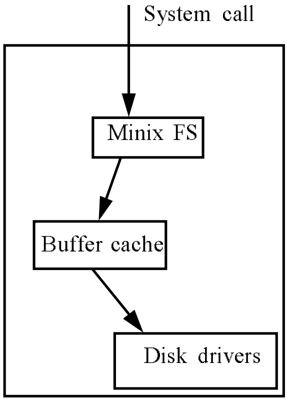
- Designed by Rémy Card and Wayne Davison after a "Request for Ideas" on the KERNEL channel
- Implemented by Rémy Card (based on the Extended FS kernel code with so many changes...)
- Enhanced by several other programmers (especially Stephen Tweedie for the kernel code and Theodore Ts'o for the utilities)
- Used by most Linux distributions
- Described later in this talk

The Virtual File System (1)

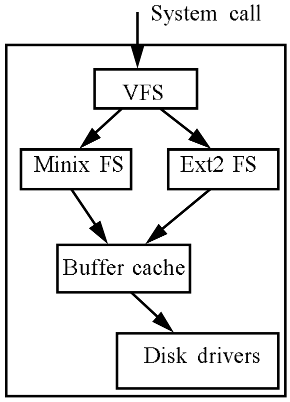
- The Linux kernel contains a Virtual File System layer
- File oriented system calls are directed to the VFS
- These calls are redirected to the appropriate FS modules
- Generic structures:
  - superblocks
  - inodes
  - files
- Function pointers:
  - superblock operations
  - inode operations
  - open file operations

The Virtual File System (2)

- Before the VFS



- With the VFS



The Ext2 FS Internals

- Motivations of Ext2fs
- Features
- Physical layout

Motivations of Ext2fs

- Make the best possible Linux file system
- Design a powerful file system:
  - Unix features
  - Advanced features
  - Excellent performances!
  - Stability
    - » No inconsistency in files
    - » No data loss
    - » No change in the FS structure
- Design an evolutive file system:
  - Features planned during the initial design
  - New extensions

"Standard" Ext2fs Features

- Unix files:
  - Regular files
  - Directories
  - Device special files
  - Symbolic links
- Big partitions: up to 2 TB
- File size: up to 2 GB
- Long file names (up to 255 characters)
- Reserved blocks for `root` (defaults to 5%)

Advanced Ext2fs Features (1)

- File attributes :
  - secure deletion
  - append-only
  - immutable
  - no-dump
- Fast symbolic links
- Selectable block size (at the FS creation)
- FS state and mount count used by fsck:
  - clean
  - not clean
  - errors detected

Advanced Ext2fs Features (2)

- BSD or System V Release 4 semantics (mount option):
  - BSD: gid is inherited from the parent directory
  - SVR4: gid is the process primary gid, or the parent directory's gid if the directory is setgid
- BSD like synchronous updates (mount option)
- Tunable parameters:
  - mount count
  - check interval
  - reserved blocks
- Tunable error behavior (tune2fs or mount option):
  - continue after error
  - remount FS in read-only mode
  - panic

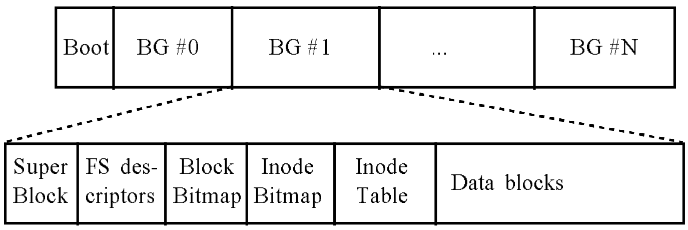
Planned Ext2fs Extensions

- File undeletion
- Access Control Lists:
  - File protection per user and/or per group
  - Will conform to Posix 1003.6
- Automatic file compression:
  - Files stored in gziped format
  - Decompression on the fly during reads
- Fragments?

Physical Layout (1)

- Influenced by BSD FFS
- A FS is made up of "block groups"
- Each block group contains:
  - a copy of the superblock
  - the FS descriptors
  - a block bitmap
  - an inode bitmap
  - a part of the inode table
  - data blocks

Physical Layout (2)



The Ext2fs Superblock

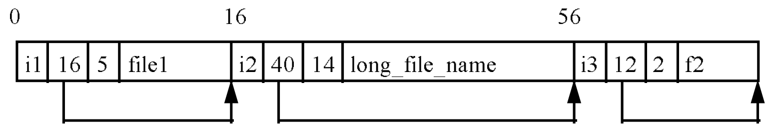
- Replicated in each group to ease recovery after a crash
- Describes the file system:
  - Number of blocks (total and reserved)
  - Number of inodes
  - Number of free blocks and inodes
  - Block and fragment size
  - Number of blocks and inodes per group
  - Last mount and write times
  - FS state (clean, errors detected)
  - current and maximal mount count
  - Last check time and check interval
  - Mount option default values
  - The revision level

The Ext2fs Descriptors

- Replicated in each group
- Each descriptor describes a block group:
  - Block bitmap location
  - Inode bitmap location
  - Inode table location
  - Number of free blocks
  - Number of free inodes
  - Number of allocated directories (used by the allocation routines)

Directories in Ext2fs

- Directory: list of variable length entries
- Each entry contains:
  - the inode number
  - the entry length
  - the file name length
  - the file name
- Example



The Ext2fs Tools

- Goals
- The Ext2fs Library
- The Ext2fs Checker
- The Ext2fs Debugger
- Possible New Tools

The Ext2fs Tools

- Suite of powerful utilities for the Ext2fs
- Several tools available for:
  - creating a FS: mke2fs
  - checking a FS: e2fsck
  - manipulating a FS: debugfs
  - changing FS parameters: tune2fs
  - dumping FS state: dumpe2fs
  - acting on file attributes: chattr, lsattr
  - ...
- Most of the tools are built on top of the Ext2fs Library

The Ext2fs Library (1)

- The library allows a program to manipulate control structures of a FS:
  - examination of data
  - change of data
- It interprets the structure of a FS on a device
- It is similar to the Ext2fs kernel code but runs in user mode
- The next release will contain a shared library:
  - Space saved on disk and in memory
  - No need to recompile tools when upgrading the library

The Ext2fs Library (2)

- FS operations:
  - open
  - close
  - create
  - read/write bitmaps
- Bad blocks list management operations
- Directory operations:
  - create
  - delete
  - expand
  - pathname to inode conversion
  - create/delete a link

The Ext2fs Library (3)

- Inodes operations:
  - scan the inode table
  - read inode
  - write inode
  - iterate on every block of an inode
- Allocation operations:
  - allocate a block
  - allocate an inode

The Ext2fs Checker (1)

- Goals:
  - repair any FS inconsistency
  - fast execution
- New version rewritten by Theodore Ts'o:
  - handles more cases than the previous version (and the other Linux FS checkers)
  - runs much faster
  - structured in 5 passes

The Ext2fs Checker (2)

- Pass 1 - iterates on all inodes:
  - checks the mode
  - checks the size and the number of allocated blocks
  - checks that data blocks are not shared between inodes
  - collects informations for next passes
- Pass 1B-D - handles data blocks shared between inodes:
  - either deallocates files
  - or copy the shared blocks
- Pass 2 - checks directories:
  - checks that entries are correct
  - checks that inode numbers are valid
  - checks '.' and '..'

The Ext2fs Checker (3)

- Pass 3 - checks directory connectivity:
  - moves unconnected inodes to **`/lost+found`**
  - removes links on directories
- Pass 4 - checks reference counts:
  - corrects link counts
  - moves unconnected files to **`/lost+found`**
- Pass 5 - checks:
  - bitmaps
  - summary informations contained in the FS descriptors

The Ext2fs Debugger

- Powerful tool able to examine and change the state of a FS
- Useful for:
  - examining the internal structures of the FS
  - manually repairing a corrupted FS
  - creating test cases for e2fsck
  - manually corrupting the FS :-)
- Operation mode:
  - interactive commands
  - debugfs translates commands into calls to the debugfs library

Possible New Tools

- The Ext2fs Library can be used as a basis for other powerful tools
- Dump/restore backup programs (Port of the 4.4BSD dump and restore utilities is in ALPHA test)
- MS/DOS driver able to read files on Ext2fs
- Program to read Ext2fs files under other Unix (e.g. BSD)
- New bootloader able to interpret an Ext2fs and to boot from any kernel image

Performances

- Benchmarks
- Description of the Tests
- Results

The Bonnie Benchmark

- Writes data to a file (using character I/O)
- Rewrites the file
- Writes data to the file (using block I/O)
- Reads the file (using character I/O)
- Reads the file (using block I/O)
- Seeks into the file
- Tests made on a 60 MB file (around maximal file size on the Minix and XIA FS)

The Andrew Benchmark

- Developed at CMU
- Used at Berkeley to benchmark FFS and LFS
- Five phases:
  - Create a directory hierarchy
  - Make a copy of the data
  - Recursively examine the status of every file
  - Examine every byte of every file
  - Compile several of the files
- Benchmark run 10 times on each FS

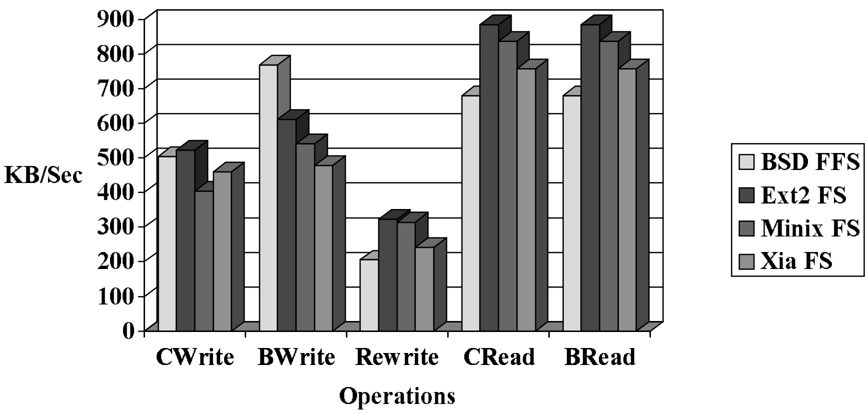
Description of the Tests

- Made on a standalone PC (i486DX 33, 16 MB RAM, Western Digital Caviar IDE drives)
- BSD FFS:
  - FreeBSD 1.1.5.1
  - Block size: 4096 bytes, fragment size: 1024 bytes
- Linux FS:
  - Linux 1.1.20 (includes cluster patches) + IDE Performance patch
  - Minix FS
  - XIA FS
  - Ext2 FS
  - Block size: 1024 bytes

Bonnie Results

	Char Write (KB/Sec)	Block Write (KB/Sec)	Rewrite (KB/Sec)	Char Read (KB/Sec)	Block Read (KB/Sec)	Seeks (/Sec)
BSD FFS	505	768	206	436	680	25.4
Ext2 FS	523	611	323	383	885	46.0
Minix FS	404	541	315	361	837	47.3
Xia FS	459	478	241	298	758	31.3

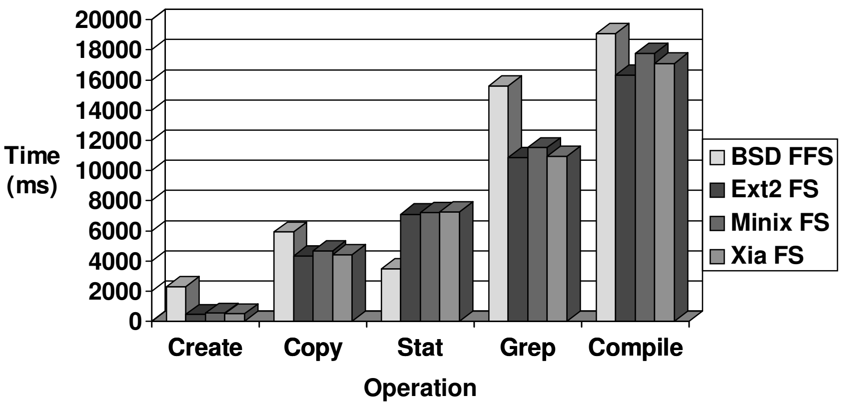
Bonnie Results



Andrew Results

	Phase 1 Create (ms)	Phase 2 Copy (ms)	Phase 3 Stat (ms)	Phase 4 Grep (ms)	Phase 5 Compile (ms)
BSD FFS	2296	5919	3482	15607	76333
Ext2 FS	450	4330	7104	10857	65382
Minix FS	536	4665	7209	11524	71038
Xia FS	492	4398	7241	10953	68430

Andrew Results



Conclusion

- Current Status of Ext2fs
- Future Directions

Current Status of Ext2fs

- The Ext2fs is probably the most widely used file system in the Linux community
- It offers:
  - advanced features
  - excellent performances
- It has provision for future extensions
- Advanced utilities:
  - Excellent FS checker
  - Defragmenter (defrag 0.6)
  - Backup suite (dump and restore)

Future Directions

- New kernel features:
  - Access Control Lists
  - Undelete
  - File Compression
- Port to other OS:
  - OS/2
  - GNU Hurd
  - FreeBSD ?
  - Other OS ?