Design and Implementation of the Second Extended File System

Rémy Card (card@masi.ibp.fr)

Theodore Ts'o (tytso@mit.edu)

Stephen Tweedie (sct@dcs.ed.ac.uk)

Outline

- History of Linux File Systems
- Basic File System Concepts
- The Virtual File System
- The Ext2 File System
- The Ext2fs Library
- The Ext2fs Tools
- Performances
- Conclusion

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 Unix 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

The Xia File System

- 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

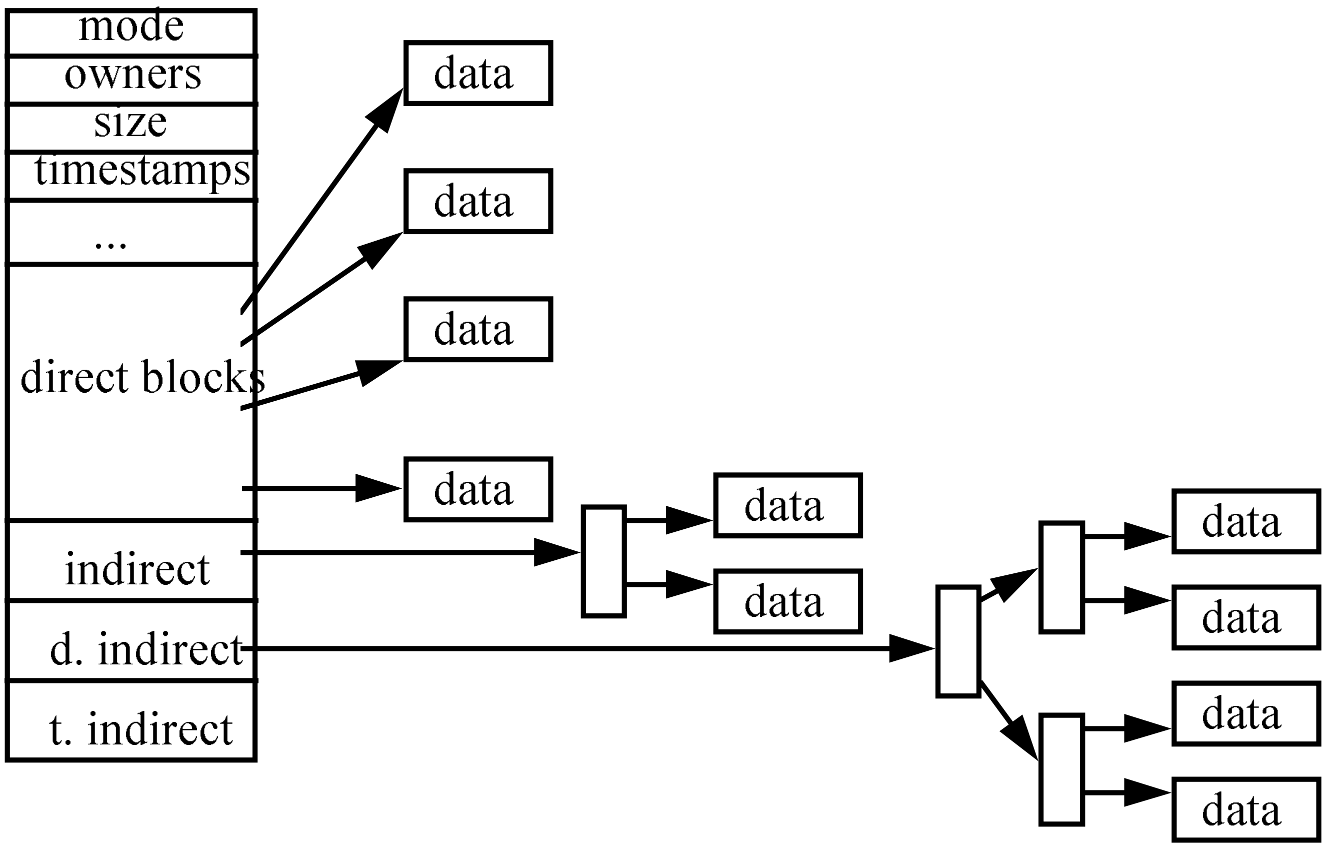
Summary

	Minix FS	Ext FS	Ext2 FS	Xia FS
Max FS size	64 MB	2 GB	4 TB	2 GB
Max file size	64 MB	2 GB	2 GB	64 MB
Max file name	14/30 c	255 c	255 c	248 c
3 times support	No	No	Yes	Yes
Extensible	No	No	Yes	No
Var. block size	No	No	Yes	No
Maintained	Yes	No	Yes	??

Inodes

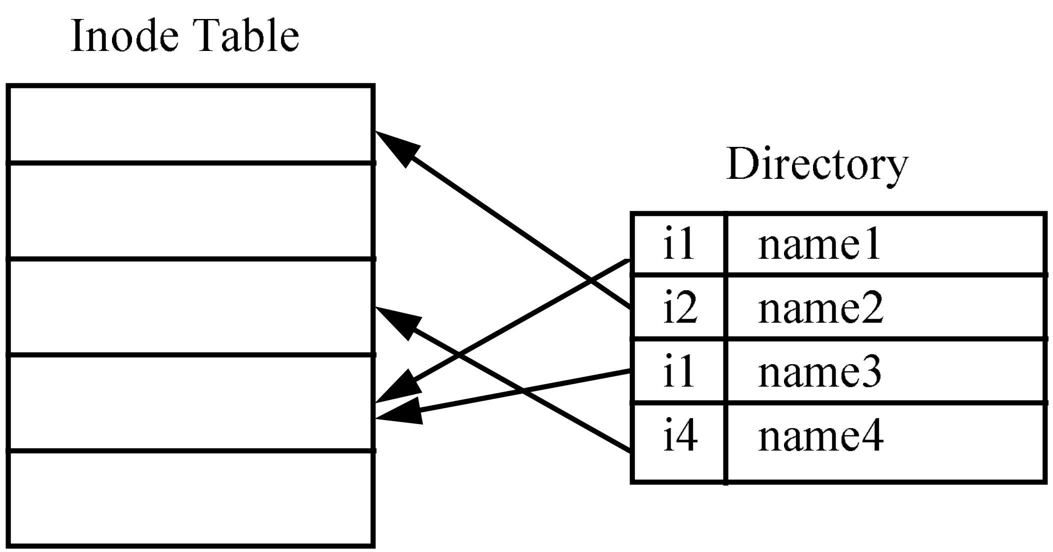
- Each file is represented by an "inode"
- An inode contains the description of the file:
  - mode (file type, access rights)
  - owners (uid, gid)
  - times (access, data modification, inode modification)
  - size
  - pointers to data blocks
  - ...
- Data blocks addresses are kept in the inode
- Inodes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

Structure of an inode



Directories

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



Links

- Two types of links: hard links and symbolic links
- Hard link:
  - additional directory entry associated with an inode
  - each inode contains a links count
  - some restrictions: no cross-filesystems hard links, no directory hard link
- Symbolic link:
  - file which contain a filename
  - when a symbolic link is encountered during a pathname to inode translation, the kernel restarts the interpretation with the link contents
  - no restrictions
  - but: uses more space (inode + data blocks) and slows down pathname resolutions

Device Special Files

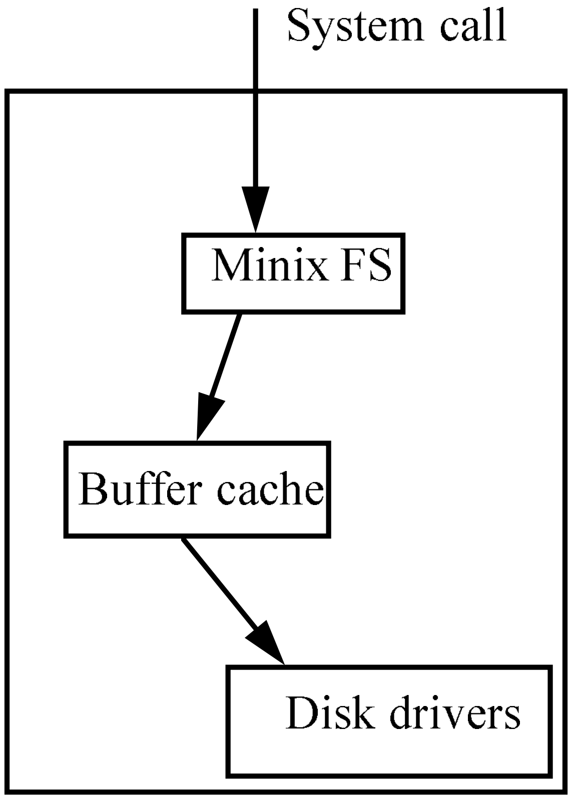
- Character or block special files
- No disk space associated with special files
- I/O requests are forwarded to a (pseudo) device driver
- Referenced in the kernel by:
  - a major number (identifies the driver)
  - a minor number (identifies the unit)

Principle of the Virtual File System (1)

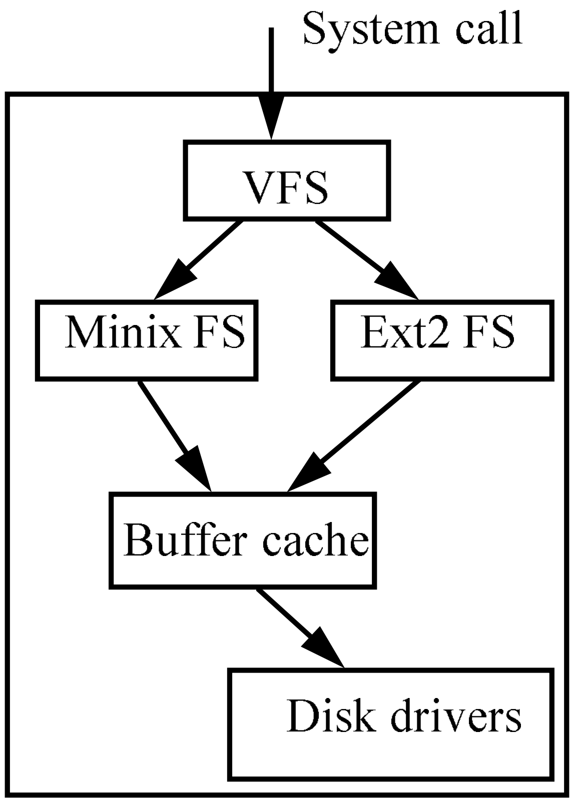
- The Linux kernel contains a Virtual File System layer
- File oriented system calls are directed to the VFS
- These calls are redirected to the appropriate FS modules
- Generic structures:
  - superblocks
  - inodes
  - files
- Function pointers:
  - superblock operations
  - inode operations
  - open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

Principle of the Virtual File System (2)

- Before the VFS



- With the VFS



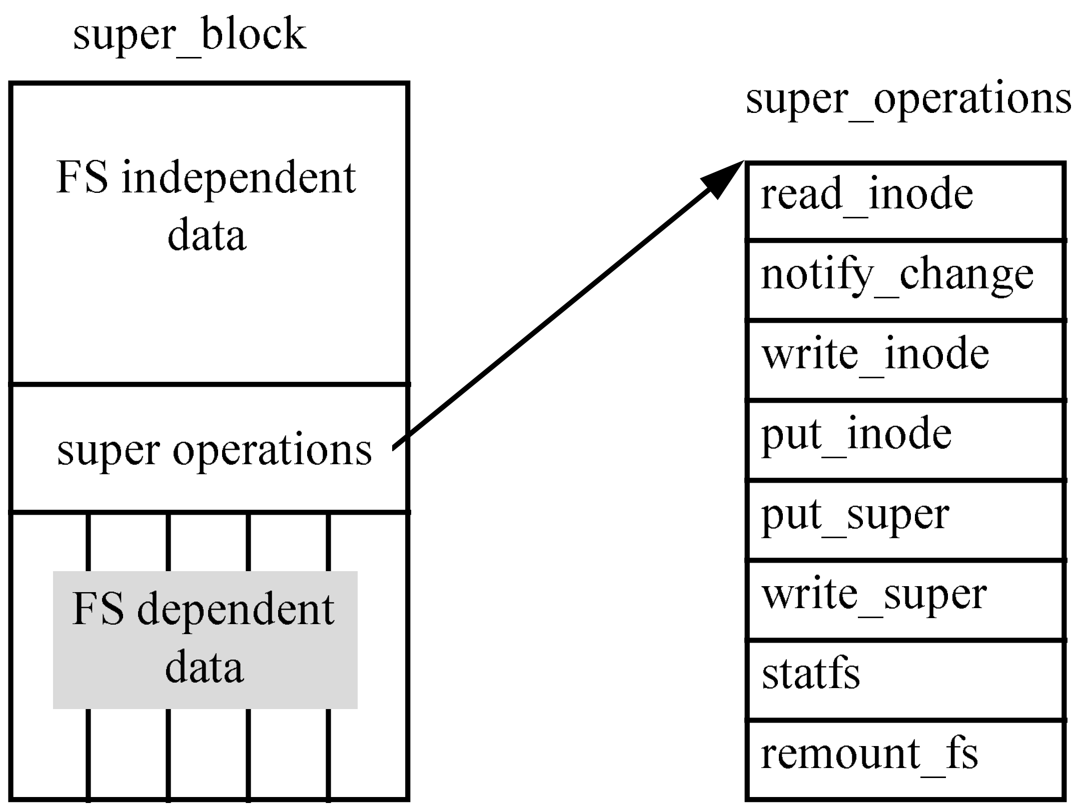
File System Types

- Each FS type is described by a structure:
  - name of the FS type
  - mount function
- When a mount request is made, the VFS calls the FS mount function (**`read_super (super, data, silent)`**)
- The FS mount function is responsible for:
  - initializing the FS internal structures
  - reading the FS metadata
  - initializing the FS operation table

Mounted File Systems (1)

- The VFS maintains a table of mounted FS
- Mounted FS are referenced by the structure **`super_block`**:
  - FS independent data
  - pointer to inode covered by this FS
  - pointer to the root inode of the FS
  - pointer to FS dependent operations
  - FS dependent data
- The contents of this structure is initialized at mount time by the FS dependent mount function

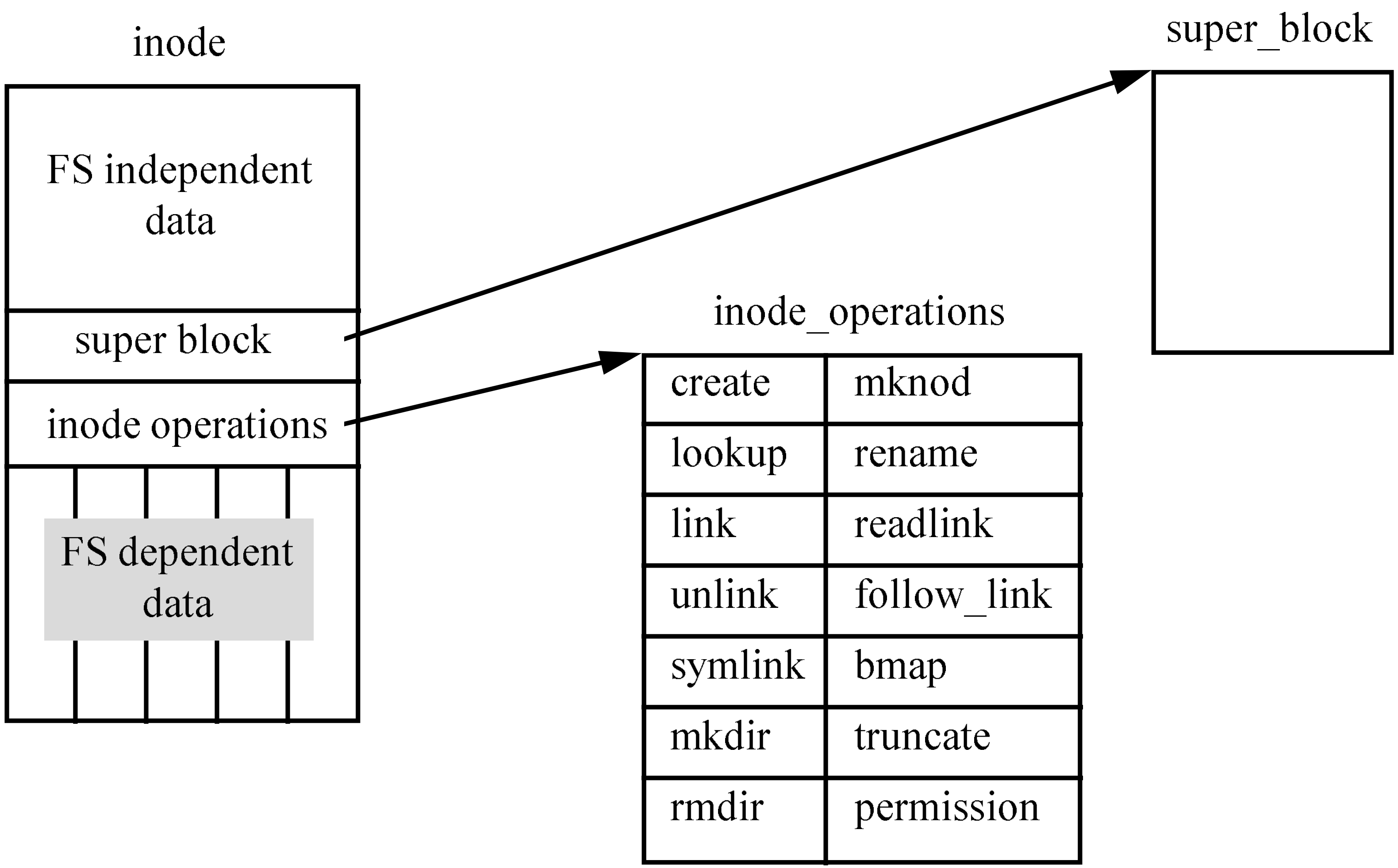
Mounted File Systems (2)



Inodes in Use (1)

- Every file operation is made on an inode
- The kernel translates file pathnames into inode numbers
- The VFS maintains a table of inodes in use
- Inodes are referenced by the structure **`inode`**:
  - FS independent data
  - pointer to FS dependent operations
  - FS dependent data
- Usually one operation table per inode type (regular file, directory, symbolic link, ...)

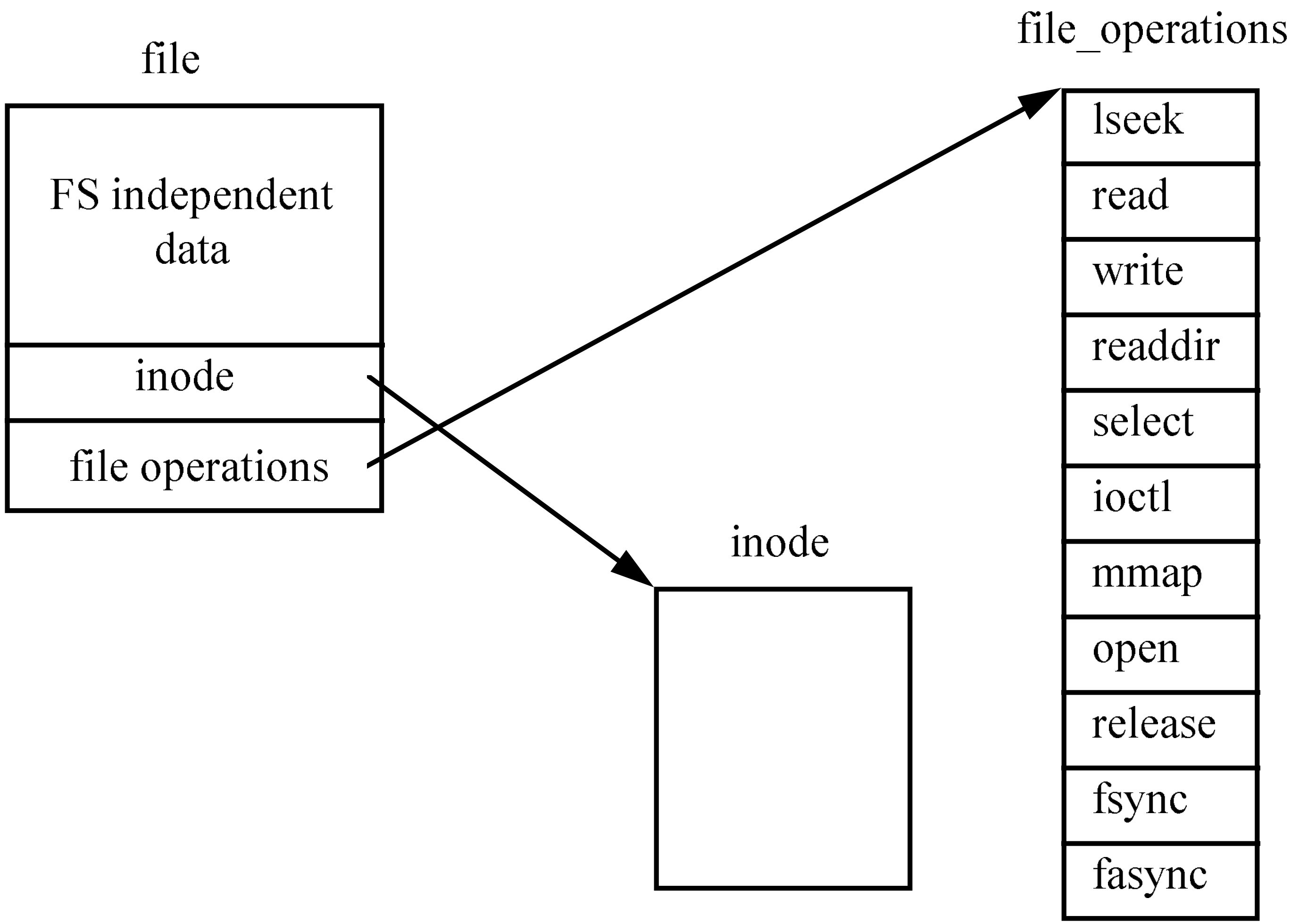
Inodes in Use (2)



Open Files (1)

- Some system calls act on open files
- Each open file is associated to an inode
- Open files are referenced by the structure `file`:
  - FS independent data
  - pointer to the associated inode
  - pointer to FS dependent operations

Open Files (2)



file
FS independent data
inode
file operations
inode
file_operations
lseek
read
write
readdir
select
ioctl
mmap
open
release
fsync
fasync

The Second Extended File System

- Motivations of Ext2fs
- Standard Features
- Advanced Features
- Physical Structure
- Performance Optimizations

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 4 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
- 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):
  - metadata are written synchronously to the disk
  - poor performances
  - possible corruption in user data not flagged by fsck

Advanced Ext2fs Features (2)

- Selectable block size (at the FS creation):
  - big blocks => faster I/O (fewer I/O requests)
  - but: more disk space wasted
  - preallocation techniques produce the same advantages
- Fast symbolic links:
  - target file name contained in the inode
  - saves disk space and speeds up name interpretation
  - this scheme can be extended to small files
- FS state and mount count used by fsck:
  - clean
  - not clean
  - errors detected

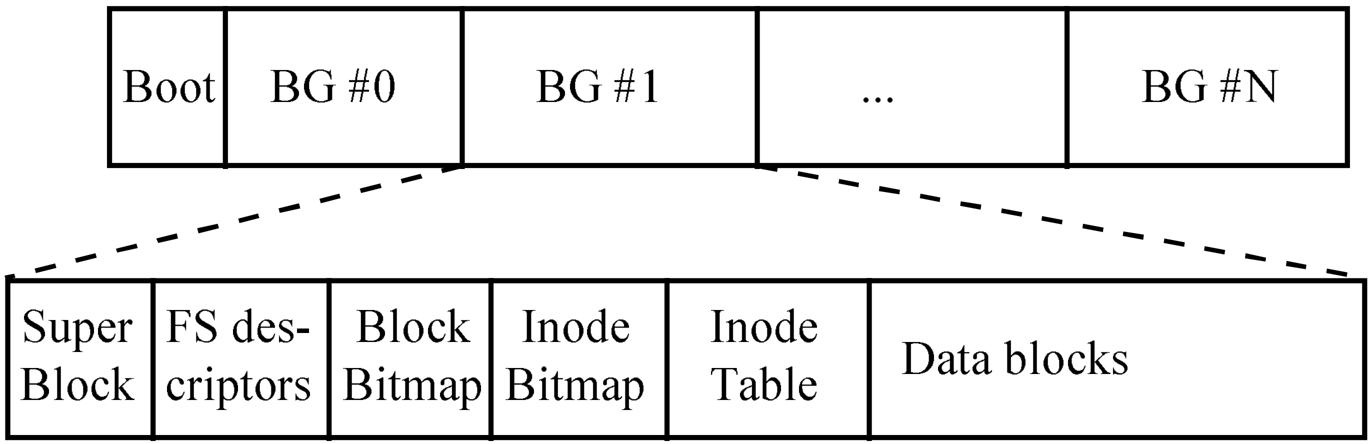
Advanced Ext2fs Features (3)

- 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
- 4.4BSD like immutable and append-only files

Physical Structure (1)

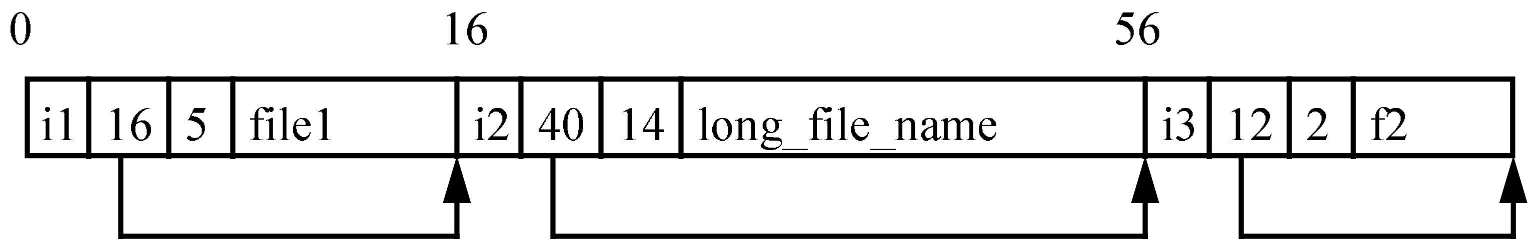
- Influenced by BSD FFS
- A FS is made up of “block groups”
- Each block group contains:
  - a copy of the superblock
  - the FS descriptors
  - a block bitmap
  - an inode bitmap
  - a part of the inode table
  - data blocks

Physical Structure (2)



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



Performance Optimizations (1)

- During reads:
  - performs readaheads for sequential reads
  - extends readahead to explicit (readdir(2)) and implicit (namei kernel directory lookup) directory reads
  - number of blocks to read-ahead is device specific
- Allocations:
  - Block groups are used to cluster together related inodes and data (not necessarily related to the physical geometry of the disk)
  - Preallocates up to 8 adjacent blocks when allocating a new block:
    - deallocates extra block on file close or if next allocation is not sequential
    - preallocation achieves good performances under heavy load
    - preallocation hit rates are around 75% even on very full file systems

Performance Optimizations (2)

- Good locality:
  - of related files through block groups
  - of related blocks through 8-bits clustering of block allocations
- Block preallocation reduces allocation overhead by a factor of 4
- Rapidly detects and deals with sequential allocations

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
- Most of the Ext2fs tools are built on top of this library:
  - Any change to reflect new features need only be made in the library
  - Smaller binaries when built as a shared library
- Abstract and general functions

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, inode to pathname 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 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 Checker (1)

- Goals:
  - repair FS inconsistencies
  - fast execution
- Original version based on Linus Torvalds' fsck program for the Minix filesystem
- New version rewritten from scratch 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 inodes as unconnected objects (checks the mode, the size, the number of allocated blocks and the validity of block addresses)
  - checks that data blocks are not shared between inodes
  - builds working copies of block and inode bitmaps
- Pass 1B-D - handles data blocks shared between inodes:
  - invoked if shared blocks have been detected by pass 1
  - either deallocates files
  - or clone the shared blocks
- Pass 1 is the longest pass to execute:
  - all of the inodes have to been read into memory
  - caches information to readuce the I/O time necessary in future passes

The Ext2fs Checker (3)

- Pass 2 - checks directories as unconnected objects
  - checks that entries are correct
  - checks that inode numbers are valid
  - checks '.' and '..'
  - caches information (parent directory)
- At the end of pass 2, nearly all disk I/O is complete, most informations are cached in memory
- Pass 3 - checks directory connectivity:
  - traces the path of each directory back to the root
  - checks that '..' is valid
  - moves unconnected directories to **`/lost+found`**
  - removes links on directories

The Ext2fs Checker (4)

- Pass 4 - checks reference counts:
  - iterates over all the inodes
  - compare link counts against counters computed during passes 2 and 3
  - moves unconnected files to **`/lost+found`**
- Pass 5 - checks the file system summary informations:
  - 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 Ext2fs library

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

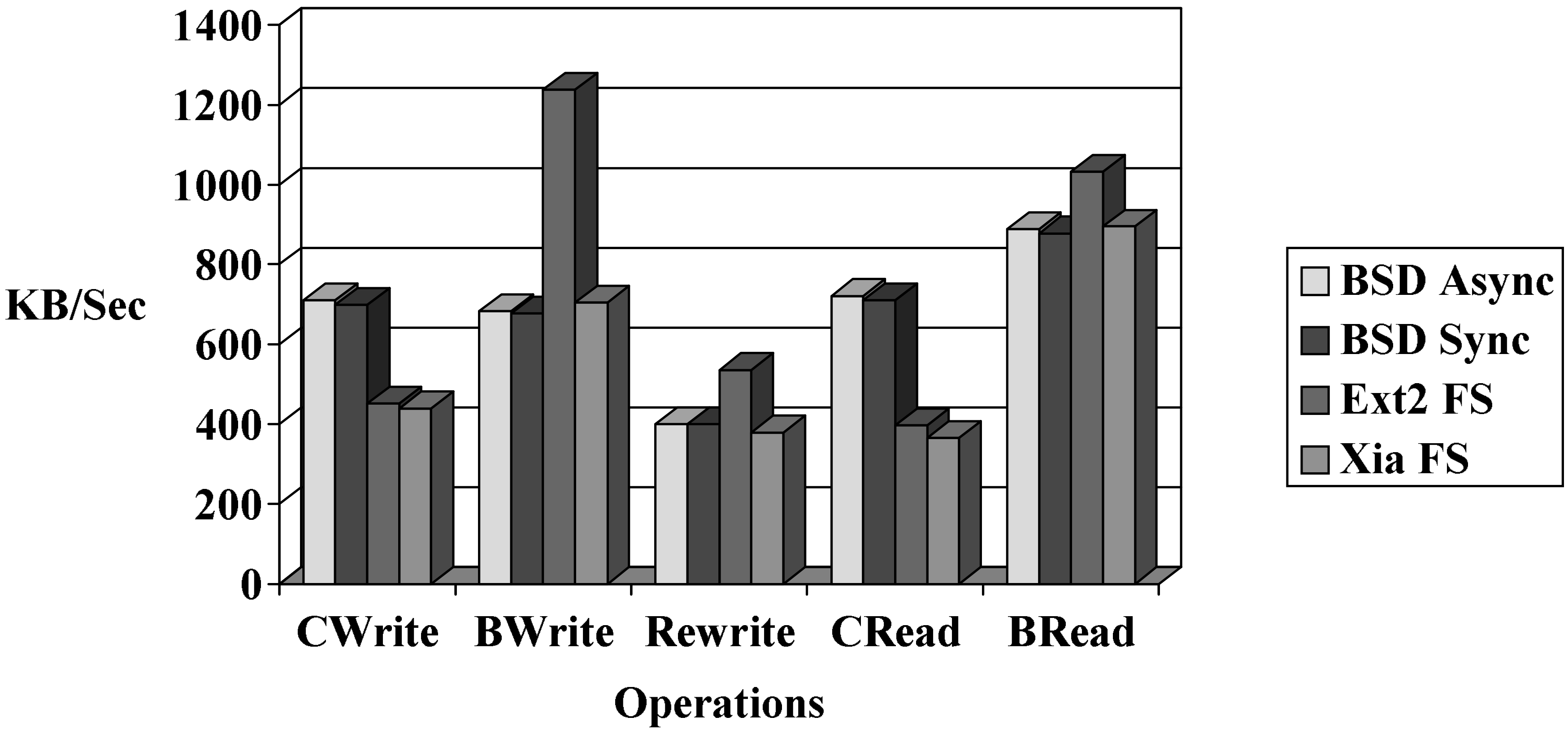
Description of the Tests

- Made on a standalone PC (i486DX2, 16 MB RAM, Western Digital Caviar IDE drives)
- BSD FFS:
  - FreeBSD 2.0 Alpha, based on the 4.4BSD Lite distribution
  - Block size: 4096 bytes, fragment size: 1024 bytes
  - Tested in asynchronous and in synchronous modes
- Linux FS:
  - Linux 1.1.62
  - 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 Async	710	684	401	721	888	27.6
BSD Sync	699	677	400	710	878	27.4
Ext2 FS	452	1237	536	397	1033	46.6
Xia FS	440	704	380	366	895	36.0

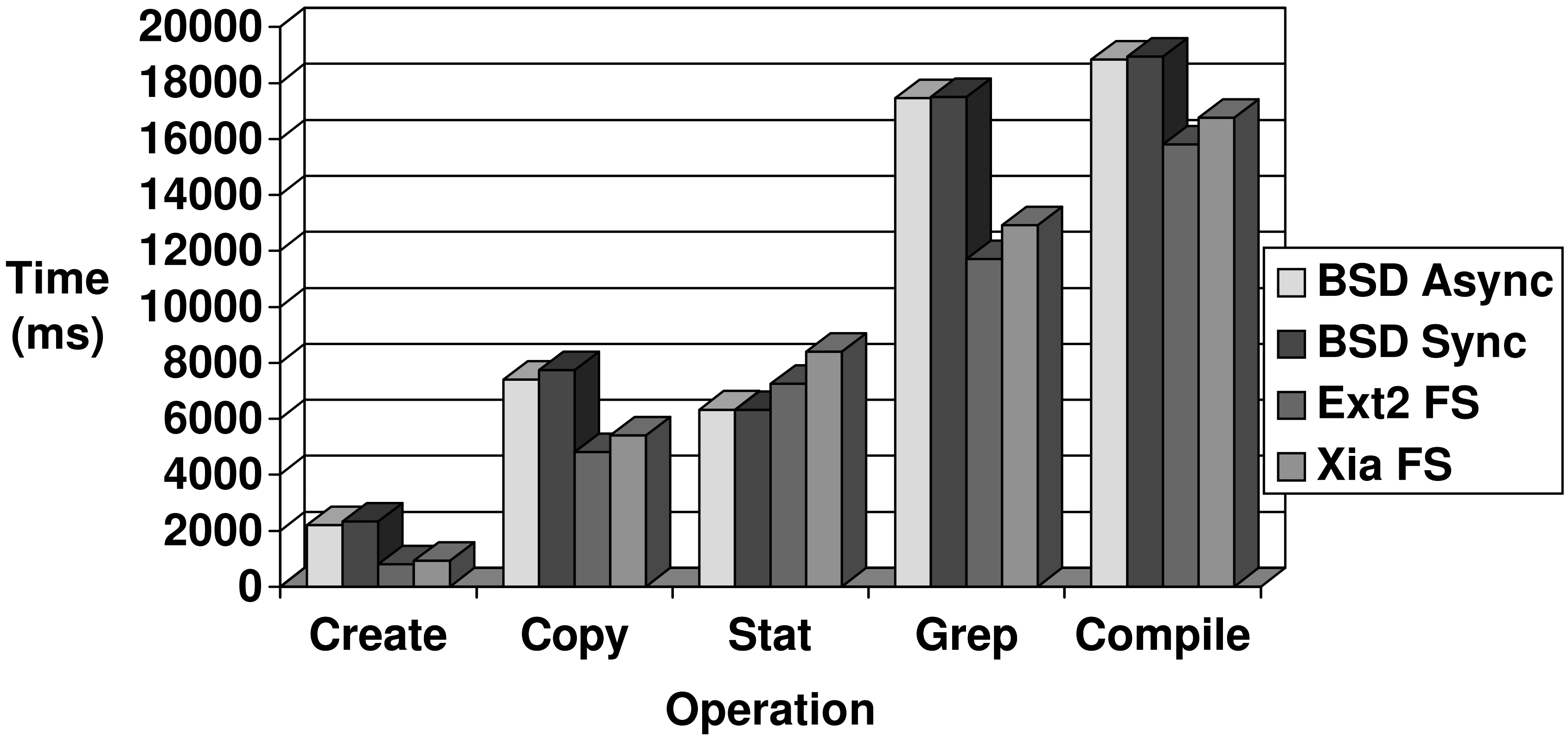
Bonnie Results



Andrew Results

	Phase 1 Create (ms)	Phase 2 Copy (ms)	Phase 3 Stat (ms)	Phase 4 Grep (ms)	Phase 5 Compile (ms)
BSD Async	2203	7391	6319	17466	75314
BSD Sync	2330	7732	6317	17499	75681
Ext2 FS	790	4791	7235	11685	63210
Xia FS	934	5402	8400	12912	66997

Andrew Results



Current Status of Ext2fs

- The Ext2fs is probably the most widely used file system in the Linux community
- It provides:
  - 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
  - LITES
  - VSTa
  - FreeBSD ?