File Systems

(w

axgopala@imperial.ac.uk

Imperial College London

- Long term non-volatile, online storage → e.g. programs, data, text, photos, music, ...

- Shar
  appli

- Conc
  , ...

- Organisation and management of data
  directories, symbolic names, backups, snapsh

- **File**: Named collection of data of arbitrary size

File Type	Usual Extension	Function
executable	exe, com, bin	read to run machine-language pro-
obj	[illegible]	[illegible] ot
source code	c, cc, java, py, hs	sour
batch	bat, sh	com [illegible] prete
...	...	...
...	...	...

What is a file?

File User Functions

Create	Create empty file. Allocate space and add to directory
Delete	Deallocate space. Invalidate or remove directory entry
Open	Search directory for file name. Check access validity and set pointers to file
Clo	
Rea	
Write	Access file, update pointers
Reposition/seek	Set current position in file to
Truncate	Erase contents but keep all
Rename	Change file name
Read attributes	e.g. creation date, size, archive flag, . . .
Write attributes	e.g. protection, immutable flag, . . .

System Call	Description
open (file, how, ...)	Open a file for reading/writing
cl	
re	
write (fd, buf, nbytes)	Write
lseek (fd, offset, ...)	Move
stat (name, &buf)	Get file
fnctl (fd, cmd, ...)	File locking and other operations

File System
Support Functions

Logical name to physical disk address translation

- i.e. /homes/axgopala/.vimrc $\rightarrow$ disk 2, block 399

Management of disk space

-

File lo

Perf

- Caching and buffering

Protection against system failure

- Back-up and restore

Security

- Protection against unauthorised access

Basic information

file name	symbolic name; unique within directory
file type	text, binary, executable, directory, . . .
file o	
file c	

Address information

volume	disk drive, partition
start addresses	cyl, head, sect, LBA
size used	
size allocated	

Access control information

owner	person who controls file (often creator)
authentication	password
per	

Usag

creation timestamp	date and time
last modified	
last read	
access activity counts	

File attributes can be accessed using system call `stat(2)` (`man 2 stat`)

- Return information about specified file in `struct stat`


```
struct stat {
  de          st               /* ID o
  in          st               /* in
  mo
  nl
  uid_t       st_uid;          /* user ID of owner */
  gid_t       st_gid;          /* group ID of owner */
  ...
  off_t       st_size;         /* total size, in bytes */
  struct timespec st_atim; /* time of last access */
  struct timespec st_mtim; /* time of last modification */
  struct timespec st_ctim; /* time of last status change */
};
```

File size naturally variable

Space allocated in blocks (typically 512 – 8192 bytes)

Choosing block size

-
  - More memory needed for buffer space
- Block size too small → wastes spa

  - High overhead in terms of management
  - High file transfer time: seek time greater t

Which allocation works the best?

Contiguous file allocation

Bloc

File al

Index blocks

Place file data at contiguous addresses on storage device

Advantages

- Successive logical records typically physically adjacent

Disa

-
- Poor performance if files grow and shrink ov
- File grows beyond size originally specified a free blocks available
  - Must be transferred to new area of adequate size
  - Leads to additional I/O operations

(a) Contiguous allocation of disk space for seven files

(b) The state of the disk after files *D* and *F* have been removed

Place file data by linking them together ⇒ insertion/deletion by modifying pointer in previous block

Need to search list to find data block

- Block-to-block seeks occur

Wastes pointer space in each block

Block Allocation Table I

Store pointers to file blocks

- Directory entries indicate first block of file
- Block number as index into block allocation table
  - Determines location of next block
  -

File Al → ak n
to Blo

- Stored on disk but cached in memory for perf

Reduces number of lengthy seeks to access given re

- But files become fragmented → periodic defragmentation
- Table can get very large

Block Linkage vs. FAT

Consider a disk with a block size of 1024 bytes. Each disk address can be s
each b

1. nd data byte and the 510100^{th} data byte?
   Hint: 500 x 1020 = 510000 and 498 x 1024 = 509

2. How does this change if a file allocation table (

Answer: Block Linkage

There are 1020 data bytes per block. The 1022^{nd} byte is resident on the 2^{nd} disk block $\rightarrow$ 2 reads are required

The 510100^{th} data byte is resident in the 501 01 reads are required

Example Problem

Answer: FAT

There are 1024 data bytes per block. Each block of the FAT can represent $\frac{1024}{4} = 256$ data blocks

1. The 1020^{th} byte is on the 1^{st} block and requires 1 read for the

2.
    - At best, all of the first 499 blocks of the file can be represented in 2 FAT blocks
    - At worst, 499 reads could be performed fo

Either case requires 1 extra read for the data. H

- Best case requires 3 reads
- Worst case requires 500 reads

Example Problem

Answer: FAT

Answer: FAT

How can we improve?

Index Blocks I

Each file has one (or more) index blocks

- Contain list of pointers that point to file data blocks
- File's directory entry points to its index block
-

Adv

- Searching may take place in index blocks the
- Place index blocks near corresponding da k access to data
- Can cache index blocks in memory for faster access

Index blocks called inodes (index nodes) in UNIX/Linux

On file open, OS opens inode table → inode entry created in memory

Str
clu

1. Disk device number
2. Inode number (for re-write)
3. Num of processes with opened file
4. Major/minor device number

inode

Type and access control
Number of links
User ID
Group ID
Direct pointer
Indirect pointer
Double indirect pointer
Triple indirect pointer

Inodes I

In a particular OS, an inode contains 6 direct pointers, 1 pointer to a (sin

Each

byte

1. What is the maximum file size for this file system?

2. What is the maximum file size if the OS would u indirect pointers?

Answer: Inodes I

1. The maximum file size is:

   6 x 1024 (data directly indexed)
   $+$ 128 x 1024 (data referenced by single indirect)

2.

   6 x 1024 (data directly indexed)
   $+$ 128 x 1024 (data referenced by single indirec
   $+$ 128^2 x 1024 (data referenced by double i
   $+$ 128^3 x 1024 (data referenced by triple indirect)
   $=$ 2.02 GB

Example Problem

Inodes II

In a particular OS, an inode contains 6 direct pointers, 1 pointer to a (single) indirect block and 1 pointer to a doubly indirect block.

Each

byte

How

1. the 1020^{th} data byte?
2. the 510100^{th} data byte?

Answer: Inodes II

	Block	FAT	Inodes
Byt			ode ory)
Byt		worst case	ode

How do we manage a storage device's free space?

Need quick access to free blocks for allocation

Use fr

-
-
- Newly-freed blocks appended to end of list

Low overhead to perform free list maintenance o

Files likely to be allocated in noncontiguous blocks → increases file access time

Free List

Block size: 1 KB

Disk b

Num
requi

Hard drive size: 500 GB

Number of blocks: 488 million

Number of blocks required to store all addresses: 1. $\frac{88}{255})$

Bitmap contains one bit (in memory) for each disk block

- Indicates whether block in use
- i^{th} bit corresponds to i^{th} block on disk

Adv

- in locations on secondary storage

Disadvantage

- May need to search entire bitmap to find free b in execution overhead

Bitmap

Block size: 1 KB

Hard

Num

Number of bits required: 488 million

Number of blocks required to store the bitmap: 60, A possible file system layout

File System Layout II

Fixed disk layout (with inodes)

- boot block
- superblock
- free inode bitmap
-
-

Supe

- no of inodes
- no of data blocks
- start of inode & free space bitmap
- first data block
- block size
- maximum file size, . . .

Directory

- Maps symbolic file names to logical disk locations (e.g. blah.txt → disk 0, block 2 (LBA))
  - Helps with file organisation
  - Ensures uniqueness of names

Single-level (or flat) file system

- Simplest file system organisation
-
-

FS often performs linear search of directory to locate file

- Leads to poor performance

Little flexibility in terms of file organisation

MultiLevel (Tree) Directory Structure

Hierarchical file system

- UNIX, Linux, Windows, Mac, ...
- **Root** indicates where on disk root directory begins
- **Root directory** points to various directories
  - Each of which contains entries for its files
  -

Pathnames

- File names usually given as path from root directory to file

Absolute pathnames

Rela

- Relative to working (or current) directory
- Can be changed using cd comma

- Displayed with `pwd`
- Current directory: .
- Parent directory: ..

Directory Operations

open/close	Open or close a directory
search	Find file in directory system using pattern matching on string, wildcard characters
create/delete	Create or delete files/directories
link	
unli	
cha	
list	Lists or displays files in director multiple read entry operatio
read attributes	Read attributes of file
write attributes	Change attributes of file, e.g. p or name
mount	Creates link in directory to directory in different file system, e.g. on another disk or remote server

Unix/Linux: Directory System Calls

System Call	Description
`s = mkdir (path, mode)`	Create a new directory
`s = rmdir (path)`	Remove directory
`s = li`	
`s = un`	
`s = chdir (path)`	
`dir = opendir (path)`	
`s = closedir (dir)`	
`dirent = readdir (dir)`	Read one entry from directory
`rewinddir (dir)`	Rewind directory to re-read

```
stru
{
  long d_ino;                  /* inode number */
  off_t d_off;                 /* offset to next dirent */
  unsigned short d_reclen;     /* length of this d_
  unsigned char d_type;        /* file type; not supporte
                               /* by all file system types */
  char d_name [NAME_MAX+1];/* file name (null-terminated) */
};
```

Steps in looking up **/usr/ast/mbox**

Links

Link: Reference to directory/file in another part of FS

- Allows alternative names (and different locations in tree)

Hard link: Reference address of file

- Only supported for files in Unix

Sym

-

Problems

- File deletion: search for links and remove the
  - Leave links and cause exception when used (symbolic links)
  - Keep link count with file → delete file when count = 0 (hard links)
- Looping: directory traversal algorithms may loop

Mount operation

- Combines multiple FSs into one namespace
- Allows reference from single root directory
- Support for soft-links to files in mounted FSs but not

Mou

- Directory in native FS assigned to root of mou

FSs manage mounted directories with ________

- Information about location of mount poin
- When native FS encounters mount point, use mount table to determine device and type of mounted FS

Second extended file system (1993)

Goal: high-performance, robust FS with support for advanced featu

Typi

Safety mechanism: 5% of blocks reserved for root

- Allow root processes to continue to run after user process consumes all FS disk space

ext2fs Inode

Represents files and directories in ext2 FS

Stores information relevant to single file/directory → e.g. time stamps, permissions, owner, pointers to data blocks

ext2 inode pointers

-
-
    -
- 14^{th} pointer is a doubly-indirect pointer
    - Locates block of indirect pointers

- 15^{th} pointer is triply-indirect pointer
    - Locates block of doubly indirect pointers

Provides fast access to small files, while supporting very large files

Block groups

- Clusters of contiguous blocks
- FS attempts to store related data in same block group
- Reduces seek time for accessing groups of related data

Bloc

-
    - FS was mounted, . . .
    - Redundant copies of superblock in som

Inode table: Contains entry for each inode in block group

Inode allocation bitmap: Inodes used within block group

Block allocation bitmaps: Blocks used within group

Group descriptor: block numbers for location of:

-
-
- block allocation bitmap
- accounting information

Data blocks: Remaining blocks store file/dir

- Directory information stored in directory entries
- Each directory entry is composed of: inode number, directory entry length, file name length, file type, file name

Feature	ext2	ext3	ext4
Year	1993	2001	2008
			2.6.19
			Y
Max file size	16 GB – 2TB	1	
File system size	2 GB – 32 GB	2 G	te)