Filesystems

Requirements

There are a lot of things we want from files

Filesystems

Requirements

There are a lot of things we want from files

- create a new file

Filesystems

Requirements

There are a lot of things we want from files

- create a new file
- delete a file

Filesystems

Requirements

There are a lot of things we want from files

- create a new file
- delete a file
- open a file to access it

Filesystems

Requirements

There are a lot of things we want from files

- create a new file
- delete a file
- open a file to access it
- read data from a file

Filesystems

Requirements

There are a lot of things we want from files

- create a new file
- delete a file
- open a file to access it
- read data from a file
- write data to a file

Filesystems

Requirements

There are a lot of things we want from files

- create a new file
- delete a file
- open a file to access it
- read data from a file
- write data to a file
- close a file when we are done

Filesystems

Requirements

There are a lot of things we want from files

- create a new file
- delete a file
- open a file to access it
- read data from a file
- write data to a file
- close a file when we are done
- rename a file

Filesystems

Requirements

There are a lot of things we want from files

- create a new file
- delete a file
- open a file to access it
- read data from a file
- write data to a file
- close a file when we are done
- (rename a file)

That last one is actually a directory operation as we shall see in a moment

Filesystems

Requirements

And directories

Filesystems

Requirements

And directories

- create a new directory

Filesystems

Requirements

And directories

- create a new directory
- delete a directory

Filesystems

Requirements

And directories

- create a new directory
- delete a directory
- scan a directory for a filename or directory name

Filesystems

Requirements

And directories

- create a new directory
- delete a directory
- scan a directory for a filename or directory name
- add a file to a directory

Filesystems
Requirements

And directories

- create a new directory
- delete a directory
- scan a directory for a filename or directory name
- add a file to a directory
- remove a file from a directory

Filesystems

Requirements

And directories

- create a new directory
- delete a directory
- scan a directory for a filename or directory name
- add a file to a directory
- remove a file from a directory
- rename a file

Filesystems

Requirements

And directories

- create a new directory
- delete a directory
- scan a directory for a filename or directory name
- add a file to a directory
- remove a file from a directory
- rename a file

The last three are intertwined

Filesystems

Requirements

This all is before we come to things like

Filesystems

Requirements

This all is before we come to things like

- speed of access

Filesystems

Requirements

This all is before we come to things like

- speed of access
- speed of update

Filesystems

Requirements

This all is before we come to things like

- speed of access
- speed of update
- scalability to large numbers of files

Filesystems

Requirements

This all is before we come to things like

- speed of access
- speed of update
- scalability to large numbers of files
- efficient use of disk space

Filesystems

Requirements

This all is before we come to things like

- speed of access
- speed of update
- scalability to large numbers of files
- efficient use of disk space
- reliability

Filesystems

Requirements

This all is before we come to things like

- speed of access
- speed of update
- scalability to large numbers of files
- efficient use of disk space
- reliability
- protection/security

Filesystems

Requirements

This all is before we come to things like

- speed of access
- speed of update
- scalability to large numbers of files
- efficient use of disk space
- reliability
- protection/security
- simple backup and recovery

Filesystems

We shall be looking at the classical Unix filesystem as an example

Filesystems

We shall be looking at the classical Unix filesystem as an example

Other filesystems are similar in their principles, though modern filesystems are immensely tweaked and tuned

Filesystems

We shall be looking at the classical Unix filesystem as an example

Other filesystems are similar in their principles, though modern filesystems are immensely tweaked and tuned

They vary in their choice of datastructures and algorithms to implement the hierarchy for efficiency or other reasons

Filesystems

Records

Modern files tend to be essentially long arrays of bytes with no further structure

Filesystems

Records

Modern files tend to be essentially long arrays of bytes with no further structure

Early files had structure, namely *records*

Filesystems

Records

Modern files tend to be essentially long arrays of bytes with no further structure

Early files had structure, namely *records*

This was a hangover from early systems using things like punched cards

Filesystems

Records

Modern files tend to be essentially long arrays of bytes with no further structure

Early files had structure, namely *records*

This was a hangover from early systems using things like punched cards

A record is a fixed-size block of data, say 80 bytes

Filesystems

Records

Modern files tend to be essentially long arrays of bytes with no further structure

Early files had structure, namely *records*

This was a hangover from early systems using things like punched cards

A record is a fixed-size block of data, say 80 bytes

Records could only be read or written as a whole: this meant implementation on the hardware of the time was easy

Filesystems

Records

It also aligned with the way data was regarded at the time: records of peoples names, job classification, salary and so on (*fields*)

Filesystems

Records

It also aligned with the way data was regarded at the time: records of peoples names, job classification, salary and so on (*fields*)

They would expect an entire record to be read or written at once

Filesystems

Records

It also aligned with the way data was regarded at the time: records of peoples names, job classification, salary and so on (*fields*)

They would expect an entire record to be read or written at once

Modern filesystems are *byte oriented* and you can access them however you please

Filesystems

Inodes

The design of the traditional Unix filesystem is based on the *inode*

Filesystems

Inodes

The design of the traditional Unix filesystem is based on the *inode*

Each file has its own inode

Filesystems

Inodes

The design of the traditional Unix filesystem is based on the *inode*

Each file has its own inode

The inode is a fixed size structure (stored on disk) that contains all the information about a file, its *metadata*

Filesystems

Inodes

Information in the inode includes

Filesystems

Inodes

Information in the inode includes

- Timestamps. Dates and times this file was last accessed and last modified

Filesystems

Inodes

Information in the inode includes

- Timestamps. Dates and times this file was last accessed and last modified
- Ownership. The userid of the owner of this file, for protection purposes

Filesystems

Inodes

Information in the inode includes

- Timestamps. Dates and times this file was last accessed and last modified
- Ownership. The userid of the owner of this file, for protection purposes
- Size. How big the file is currently

Filesystems

Inodes

Information in the inode includes

- Timestamps. Dates and times this file was last accessed and last modified
- Ownership. The userid of the owner of this file, for protection purposes
- Size. How big the file is currently
- Type. Whether this is a *plain file*, or a directory, or some other kind of *special file*

Filesystems

Inodes

Information in the inode includes

- Timestamps. Dates and times this file was last accessed and last modified
- Ownership. The userid of the owner of this file, for protection purposes
- Size. How big the file is currently
- Type. Whether this is a *plain file*, or a directory, or some other kind of *special file*
- Access permissions. Who can read or write or run this file (if it is a program)

Filesystems

Inodes

Information in the inode includes

- Timestamps. Dates and times this file was last accessed and last modified
- Ownership. The userid of the owner of this file, for protection purposes
- Size. How big the file is currently
- Type. Whether this is a *plain file*, or a directory, or some other kind of *special file*
- Access permissions. Who can read or write or run this file (if it is a program)
- Reference count. The number of names this file has

Filesystems

Inodes

Information in the inode includes

- Timestamps. Dates and times this file was last accessed and last modified
- Ownership. The userid of the owner of this file, for protection purposes
- Size. How big the file is currently
- Type. Whether this is a *plain file*, or a directory, or some other kind of *special file*
- Access permissions. Who can read or write or run this file (if it is a program)
- Reference count. The number of names this file has
- Pointers to areas on the disk where the actual data lives

Filesystems

Inodes

Notice that filenames are *not* in the inode

Filesystems

Inodes

Notice that filenames are *not* in the inode

Filenames are stored in directories

Filesystems

Inodes

Notice that filenames are *not* in the inode

Filenames are stored in directories

A directory is essentially just a list of names of files and subdirectories, together with their inode numbers

Filesystems

Inodes

Notice that filenames are *not* in the inode

Filenames are stored in directories

A directory is essentially just a list of names of files and subdirectories, together with their inode numbers

Name	Inode
`foo.c`	23
`ff.html`	42
`mydata`	7

Filesystems

Inodes

Notice that filenames are *not* in the inode

Filenames are stored in directories

A directory is essentially just a list of names of files and subdirectories, together with their inode numbers

Name	Inode
`foo.c`	23
`ff.html`	42
`mydata`	7

Originally just a table, these days clever datastructures are used to manage the large numbers of names we use

Filesystems

Inodes

This is how a file can have many names: multiple directory entries referring to the same inode

Filesystems

Inodes

This is how a file can have many names: multiple directory entries referring to the same inode

As a consequence a file cannot know its own name(s) as the names are independent of the file

Filesystems

Inodes

This is how a file can have many names: multiple directory entries referring to the same inode

As a consequence a file cannot know its own name(s) as the names are independent of the file

In some sense, the inode number is the true name of the file

Filesystems

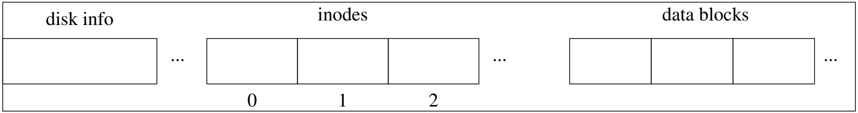
Inodes

As inodes are a fixed size, it is easy to put them in a simple array on disk and just refer to them by their index in the array: the *inode number*

Filesystems

Inodes

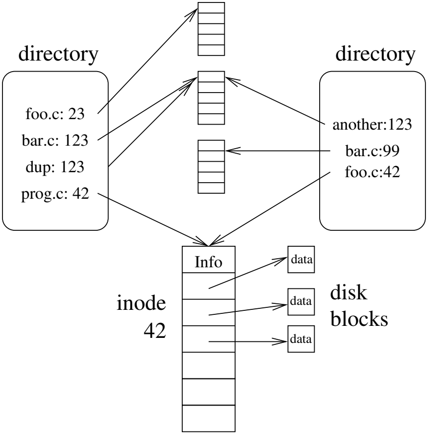
As inodes are a fixed size, it is easy to put them in a simple array on disk and just refer to them by their index in the array: the *inode number*



Disk blocks

Filesystems

Inodes



Filesystems

Inodes

The inode contains a *reference count*: the number of names the file/inode has

Filesystems

Inodes

The inode contains a *reference count*: the number of names the file/inode has

If the count drops to zero, the OS can remove the file

Filesystems

Inodes

The inode contains a *reference count*: the number of names the file/inode has

If the count drops to zero, the OS can remove the file

Deleting a file is a matter of

Filesystems

Inodes

The inode contains a *reference count*: the number of names the file/inode has

If the count drops to zero, the OS can remove the file

Deleting a file is a matter of

- Removing the name reference in the relevant directory

Filesystems

Inodes

The inode contains a *reference count*: the number of names the file/inode has

If the count drops to zero, the OS can remove the file

Deleting a file is a matter of

- Removing the name reference in the relevant directory
- Decrementing the reference count in the inode

Filesystems

Inodes

The inode contains a *reference count*: the number of names the file/inode has

If the count drops to zero, the OS can remove the file

Deleting a file is a matter of

- Removing the name reference in the relevant directory
- Decrementing the reference count in the inode
- If the count reaches 0, the OS can free the inode and the disk blocks it refers to

Filesystems

Inodes

In fact, each time a program opens a file the OS increments the count; and decrements it when the program closes the file (possibly when the program exits)

Filesystems

Inodes

In fact, each time a program opens a file the OS increments the count; and decrements it when the program closes the file (possibly when the program exits)

So it is possible for a program to create a new file (inc); open it (inc); delete it (dec); and still be able to read and write to it

Filesystems

Inodes

In fact, each time a program opens a file the OS increments the count; and decrements it when the program closes the file (possibly when the program exits)

So it is possible for a program to create a new file (inc); open it (inc); delete it (dec); and still be able to read and write to it

The file will only disappear when the program ends (dec)

Filesystems

Inodes

In fact, each time a program opens a file the OS increments the count; and decrements it when the program closes the file (possibly when the program exits)

So it is possible for a program to create a new file (inc); open it (inc); delete it (dec); and still be able to read and write to it

The file will only disappear when the program ends (dec)

No other process can see this file: there is no name in any directory

Filesystems

Inodes

The data on the disk is in *disk blocks*, fixed size areas on the disk, e.g., 512 or 1024 bytes

Filesystems

Inodes

The data on the disk is in *disk blocks*, fixed size areas on the disk, e.g., 512 or 1024 bytes

An inode contains *block pointers*, namely a list of the address of the blocks for that file

Filesystems

Inodes

The data on the disk is in *disk blocks*, fixed size areas on the disk, e.g., 512 or 1024 bytes

An inode contains *block pointers*, namely a list of the address of the blocks for that file

Having a fixed size allows for easy and fast allocation and deallocation

Filesystems

Inodes

The data on the disk is in *disk blocks*, fixed size areas on the disk, e.g., 512 or 1024 bytes

An inode contains *block pointers*, namely a list of the address of the blocks for that file

Having a fixed size allows for easy and fast allocation and deallocation

This is similar to pages in memory; but now physical location of blocks *is* important as discs are mechanical devices

Filesystems

Inodes

The data on the disk is in *disk blocks*, fixed size areas on the disk, e.g., 512 or 1024 bytes

An inode contains *block pointers*, namely a list of the address of the blocks for that file

Having a fixed size allows for easy and fast allocation and deallocation

This is similar to pages in memory; but now physical location of blocks *is* important as discs are mechanical devices

Whole numbers of blocks are always allocated to files

Filesystems

Inodes

The data on the disk is in *disk blocks*, fixed size areas on the disk, e.g., 512 or 1024 bytes

An inode contains *block pointers*, namely a list of the address of the blocks for that file

Having a fixed size allows for easy and fast allocation and deallocation

This is similar to pages in memory; but now physical location of blocks *is* important as discs are mechanical devices

Whole numbers of blocks are always allocated to files

This can lead to wastage, e.g., a 1025 byte file might need two blocks, but uses just over half of the space. Though there are lot of tricks in real filesystems to avoid the worst of this

Filesystems

Inodes

Exercise Being of a fixed size an inode will only have room for a fixed number of block pointers. This will limit the size of a file. Read about *indirect blocks*

Exercise Read about *soft links* (similar to a Windows *shortcut*) that allows an inode to refer to a name (not a file, but a name of a file), and about the problems they solve

Filesystems

Inodes

When a program opens a file, the OS must find where on disk the file lives

Filesystems

Inodes

When a program opens a file, the OS must find where on disk the file lives

Say we are looking for the file `prog.c` with a cwd of `/home/rjb`

Filesystems

Inodes

When a program opens a file, the OS must find where on disk the file lives

Say we are looking for the file `prog.c` with a cwd of `/home/rjb`

- The name is incomplete, so the OS prepends the cwd giving `/home/rjb/prog.c`

Filesystems

Inodes

When a program opens a file, the OS must find where on disk the file lives

Say we are looking for the file `prog.c` with a cwd of `/home/rjb`

- The name is incomplete, so the OS prepends the cwd giving `/home/rjb/prog.c`
- The OS reads the block containing the root directory off disk and scans through it for the name `home`

Filesystems

Inodes

When a program opens a file, the OS must find where on disk the file lives

Say we are looking for the file `prog.c` with a cwd of `/home/rjb`

- The name is incomplete, so the OS prepends the cwd giving `/home/rjb/prog.c`
- The OS reads the block containing the root directory off disk and scans through it for the name `home`
- It finds it and gets the inode number for `home`

Filesystems

Inodes

When a program opens a file, the OS must find where on disk the file lives

Say we are looking for the file `prog.c` with a cwd of `/home/rjb`

- The name is incomplete, so the OS prepends the cwd giving `/home/rjb/prog.c`
- The OS reads the block containing the root directory off disk and scans through it for the name `home`
- It finds it and gets the inode number for `home`
- It reads that inode off disk and finds it refers to a directory

Filesystems

Inodes

When a program opens a file, the OS must find where on disk the file lives

Say we are looking for the file `prog.c` with a cwd of `/home/rjb`

- The name is incomplete, so the OS prepends the cwd giving `/home/rjb/prog.c`
- The OS reads the block containing the root directory off disk and scans through it for the name `home`
- It finds it and gets the inode number for `home`
- It reads that inode off disk and finds it refers to a directory
- It reads the block containing the directory off disk

Filesystems

Inodes

When a program opens a file, the OS must find where on disk the file lives

Say we are looking for the file `prog.c` with a cwd of `/home/rjb`

- The name is incomplete, so the OS prepends the cwd giving `/home/rjb/prog.c`
- The OS reads the block containing the root directory off disk and scans through it for the name `home`
- It finds it and gets the inode number for `home`
- It reads that inode off disk and finds it refers to a directory
- It reads the block containing the directory off disk
- It scans the directory for the name `rjb`

Filesystems

Inodes

- It finds it and gets the inode number for `rjb`

Filesystems

Inodes

- It finds it and gets the inode number for `rjb`
- It reads that inode off disk and finds it refers to a directory

Filesystems

Inodes

- It finds it and gets the inode number for `rjb`
- It reads that inode off disk and finds it refers to a directory
- It reads the block containing the directory off disk

Filesystems

Inodes

- It finds it and gets the inode number for `rjb`
- It reads that inode off disk and finds it refers to a directory
- It reads the block containing the directory off disk
- It scans the directory for the name `prog.c`

Filesystems

Inodes

- It finds it and gets the inode number for `rjb`
- It reads that inode off disk and finds it refers to a directory
- It reads the block containing the directory off disk
- It scans the directory for the name `prog.c`
- It finds it and gets the inode number for `prog.c`

Filesystems

Inodes

- It finds it and gets the inode number for `rjb`
- It reads that inode off disk and finds it refers to a directory
- It reads the block containing the directory off disk
- It scans the directory for the name `prog.c`
- It finds it and gets the inode number for `prog.c`
- It reads that inode off disk and finds it refers to a file

Filesystems

Inodes

- It finds it and gets the inode number for `rjb`
- It reads that inode off disk and finds it refers to a directory
- It reads the block containing the directory off disk
- It scans the directory for the name `prog.c`
- It finds it and gets the inode number for `prog.c`
- It reads that inode off disk and finds it refers to a file
- It reads the blocks containing the file off disk

Filesystems

Inodes

- It finds it and gets the inode number for `rjb`
- It reads that inode off disk and finds it refers to a directory
- It reads the block containing the directory off disk
- It scans the directory for the name `prog.c`
- It finds it and gets the inode number for `prog.c`
- It reads that inode off disk and finds it refers to a file
- It reads the blocks containing the file off disk

This must be done for every file opened

Filesystems

Inodes

- It finds it and gets the inode number for `rjb`
- It reads that inode off disk and finds it refers to a directory
- It reads the block containing the directory off disk
- It scans the directory for the name `prog.c`
- It finds it and gets the inode number for `prog.c`
- It reads that inode off disk and finds it refers to a file
- It reads the blocks containing the file off disk

This must be done for every file opened

Caching can be used to great effect here: the kernel keeps copies of the inodes and directories in memory, rather than re-reading them every time

There's a great deal we haven't covered here!

Exercise Have a look at a modern filesystem

Exercise Solid state disks (SSDs) are common these days. What differences do they bring to the way filesystems should be implemented?

Filesystems

Other filesystems you might like to look at

- btrfs
- ext4
- FAT, VFAT
- FUSE
- GFS (Global File System)
- Google File System
- HFS+
- ISO 9660
- JFFS2
- Lustre
- NFS
- NTFS
- OCFS2
- procfs
- Reiser
- ReFS (Resilient File System)
- UnionFS
- ZFS

Also see “List of file systems” on Wikipedia