Filesystems

We now turn to files and filesystems

Filesystems

We now turn to files and filesystems

Current technology has main memory being limited in size (a few gigabytes) and *volatile*: the values disappear when you remove the power

Filesystems

We now turn to files and filesystems

Current technology has main memory being limited in size (a few gigabytes) and *volatile*: the values disappear when you remove the power

To be able to manipulate more data and to make it *persistent* we turn to larger, but slower, devices like disks

Filesystems

We now turn to files and filesystems

Current technology has main memory being limited in size (a few gigabytes) and *volatile*: the values disappear when you remove the power

To be able to manipulate more data and to make it *persistent* we turn to larger, but slower, devices like disks

And to organise everything we need *filesystems*

Filesystems

Note: not all applications want to use filesystems, in particular enterprise databases like to have direct access to disks themselves in order to optimise access for their very specific needs

Filesystems

Note: not all applications want to use filesystems, in particular enterprise databases like to have direct access to disks themselves in order to optimise access for their very specific needs

Some people have experimented with making ordinary applications use DB-like access, mostly to a resounding failure

Filesystems

Note: not all applications want to use filesystems, in particular enterprise databases like to have direct access to disks themselves in order to optimise access for their very specific needs

Some people have experimented with making ordinary applications use DB-like access, mostly to a resounding failure

In general, a filesystem is what people want: a simple, efficient way of accessing their data

Filesystems

Another note: a filesystem is just an organisation of data, and doesn't need to be associated with *disks*

Filesystems

Another note: a filesystem is just an organisation of data, and doesn't need to be associated with *disks*

Filesystems can be found whenever we have large amounts of data that needs organising

Filesystems

Another note: a filesystem is just an organisation of data, and doesn't need to be associated with *disks*

Filesystems can be found whenever we have large amounts of data that needs organising

USB keys, iPods, phones, ...

Filesystems

Another note: a filesystem is just an organisation of data, and doesn't need to be associated with *disks*

Filesystems can be found whenever we have large amounts of data that needs organising

USB keys, iPods, phones, ...

It's even occasionally useful to have a filesystem *in memory*, again as an organisational mechanism

Filesystems

Yet another note: and it's not necessary that the object or objects behind the filesystem store data

Filesystems

Yet another note: and it's not necessary that the object or objects behind the filesystem store data

We can have it so that reading from one particular file actually returns keystrokes from the keyboard

Filesystems

Yet another note: and it's not necessary that the object or objects behind the filesystem store data

We can have it so that reading from one particular file actually returns keystrokes from the keyboard

Or writing to another file is actually sending sound to a soundcard

Filesystems

Yet another note: and it's not necessary that the object or objects behind the filesystem store data

We can have it so that reading from one particular file actually returns keystrokes from the keyboard

Or writing to another file is actually sending sound to a soundcard

In fact, a Unix philosophy is "all devices are files"

Filesystems

Yet another note: and it's not necessary that the object or objects behind the filesystem store data

We can have it so that reading from one particular file actually returns keystrokes from the keyboard

Or writing to another file is actually sending sound to a soundcard

In fact, a Unix philosophy is "all devices are files"

This makes accessing devices incredibly easy for the programmer: just read and write

Filesystems

Yet another note: and it's not necessary that the object or objects behind the filesystem store data

We can have it so that reading from one particular file actually returns keystrokes from the keyboard

Or writing to another file is actually sending sound to a soundcard

In fact, a Unix philosophy is "all devices are files"

This makes accessing devices incredibly easy for the programmer: just read and write

Exercise. Compare with using virtual memory to do the same

Filesystems

But, for now, we shall think of files in the traditional sense

Filesystems

But, for now, we shall think of files in the traditional sense

A *file* is simply a named chunk of data stored somehow on a disk

Filesystems

But, for now, we shall think of files in the traditional sense

A *file* is simply a named chunk of data stored somehow on a disk

Humans like easy names like `prog.c`, so there needs to be a mechanism to convert names to the place on disk where the data is stored

Filesystems

But, for now, we shall think of files in the traditional sense

A *file* is simply a named chunk of data stored somehow on a disk

Humans like easy names like `prog.c`, so there needs to be a mechanism to convert names to the place on disk where the data is stored

And when we have thousands or millions of files, meaning thousands or millions of names, we need some way of organising the names (even before we have thought of organising the data itself!)

Filesystems

Names

Notice the distinction between the *name* and the *data*

Filesystems

Names

Notice the distinction between the *name* and the *data*

This is *very* important and the distinction runs throughout computer science

Filesystems

Names

Notice the distinction between the *name* and the *data*

This is *very* important and the distinction runs throughout computer science

The same name can refer to different data (otherwise the whole thing would be useless, we could never fix bugs in `prog.c`)

Filesystems

Names

Notice the distinction between the *name* and the *data*

This is *very* important and the distinction runs throughout computer science

The same name can refer to different data (otherwise the whole thing would be useless, we could never fix bugs in `prog.c`)

Different names can refer to the same data. We tend to forget that, in real life, we can use different names to refer to the same thing: “Lewis Carroll” and “Charles Dodgson”

Filesystems

Names

Notice the distinction between the *name* and the *data*

This is *very* important and the distinction runs throughout computer science

The same name can refer to different data (otherwise the whole thing would be useless, we could never fix bugs in `prog.c`)

Different names can refer to the same data. We tend to forget that, in real life, we can use different names to refer to the same thing: “Lewis Carroll” and “Charles Dodgson”

All but the simplest filesystems allow the same data to have multiple filenames

Filesystems

Names

For the philosophers:

It is possible to have a thing without a name (so how can we refer to it?)

It is possible to have a name without a thing it refers to

It is possible for names to have names

Filesystems

Names

For the philosophers:

It is possible to have a thing without a name (so how can we refer to it?)

It is possible to have a name without a thing it refers to

It is possible for names to have names

Exercise: read the introduction to the poem “Haddocks’ Eyes”, in “Through the Looking-Glass” by Lewis Carroll and explain the relevance

Filesystems

Names

For the philosophers:

It is possible to have a thing without a name (so how can we refer to it?)

It is possible to have a name without a thing it refers to

It is possible for names to have names

Exercise: read the introduction to the poem “Haddocks’ Eyes”, in “Through the Looking-Glass” by Lewis Carroll and explain the relevance

And explain the use of quotes ’“’ in the above