Filesystems

Names

So names need to be organised; this is usually (but not always) done as a simple *hierarchy*

Filesystems

Names

So names need to be organised; this is usually (but not always) done as a simple *hierarchy*

Rather than simply presenting all filenames to the user (a *flat* filesystem), we gather together related files and put them into a *directory*. Also called a *folder*

Filesystems

Names

So names need to be organised; this is usually (but not always) done as a simple *hierarchy*

Rather than simply presenting all filenames to the user (a *flat* filesystem), we gather together related files and put them into a *directory*. Also called a *folder*

A directory is just a collection of (names of) files, but it allows us to simplify our thought processes

Filesystems

Names

So names need to be organised; this is usually (but not always) done as a simple *hierarchy*

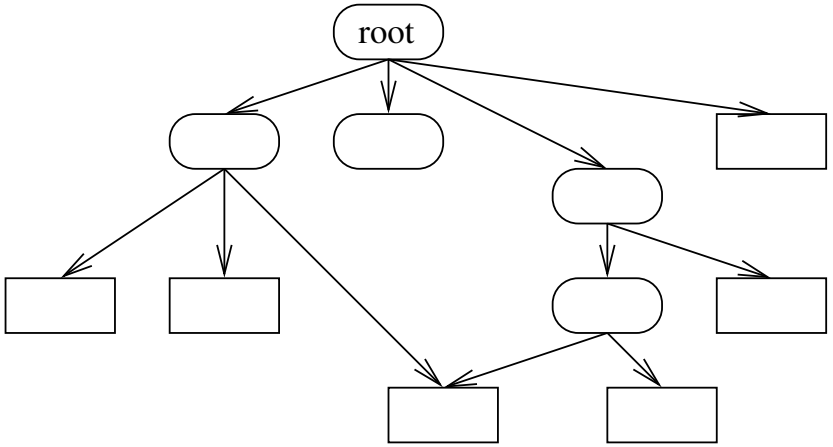
Rather than simply presenting all filenames to the user (a *flat* filesystem), we gather together related files and put them into a *directory*. Also called a *folder*

A directory is just a collection of (names of) files, but it allows us to simplify our thought processes

And (names of) directories can be collected in other directories and so on until we get to the top of the hierarchy, the *root*

Filesystems

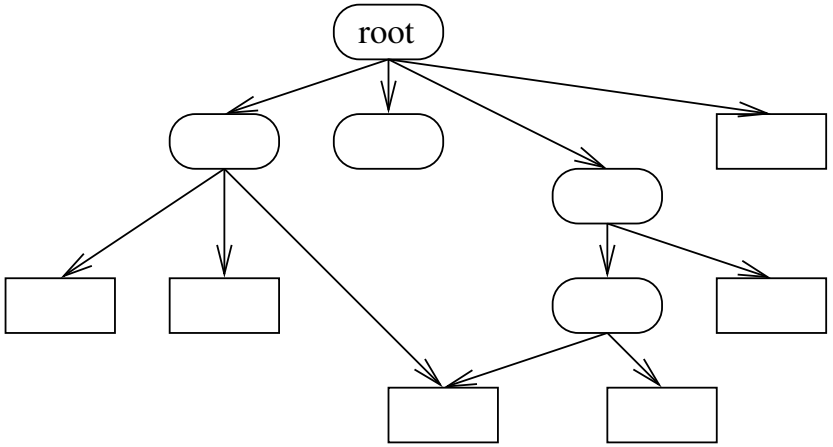
Names



Files can appear at all levels

Filesystems

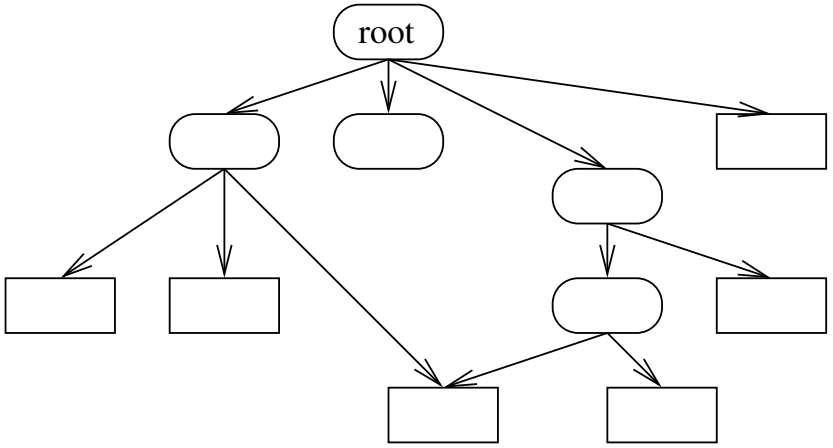
Names



But always within a directory

Filesystems

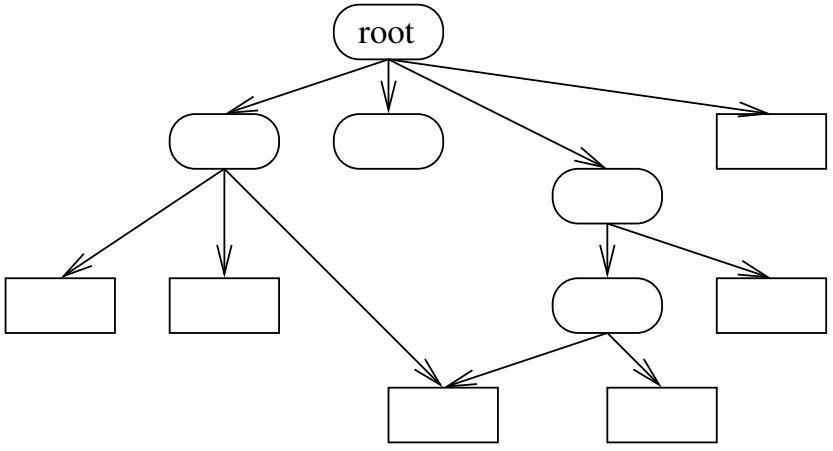
Names



In some systems, a file can be in more than one directory

Filesystems

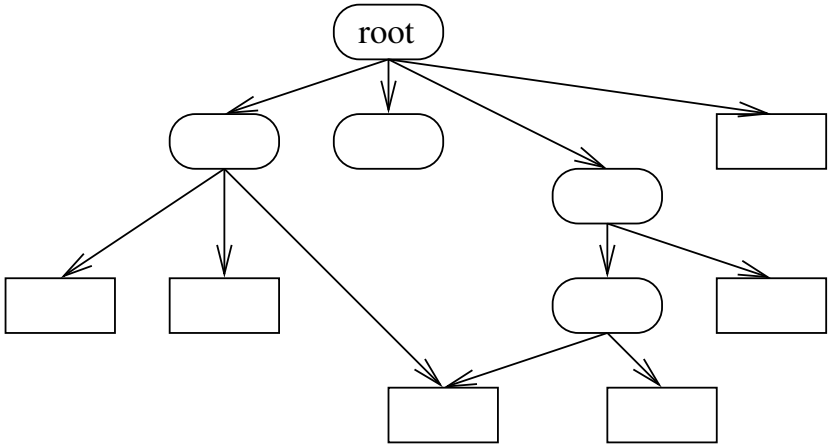
Names



Generally, directories can only be within exactly *one* directory, for implementation reasons

Filesystems

Names



Directories can be empty

Filesystems

Names

The namespace hierarchy makes referring to a file easy

Filesystems

Names

The namespace hierarchy makes referring to a file easy

A Unix example: `/usr/bin/ls` refers to a file named `ls` inside a directory named `bin` inside a directory named `usr` which is in the root directory

Filesystems

Names

The namespace hierarchy makes referring to a file easy

A Unix example: `/usr/bin/ls` refers to a file named `ls` inside a directory named `bin` inside a directory named `usr` which is in the root directory

The `/`s separate the names

Filesystems

Names

The namespace hierarchy makes referring to a file easy

A Unix example: `/usr/bin/ls` refers to a file named `ls` inside a directory named `bin` inside a directory named `usr` which is in the root directory

The `/`s separate the names

The root directory is referred to as `/`, though its actual name is empty

Filesystems

Names

The namespace hierarchy makes referring to a file easy

A Unix example: `/usr/bin/ls` refers to a file named `ls` inside a directory named `bin` inside a directory named `usr` which is in the root directory

The `/`s separate the names

The root directory is referred to as `/`, though its actual name is empty

Other OSs have similar ideas, but use different separators

Filesystems

Names

The namespace hierarchy makes referring to a file easy

A Unix example: `/usr/bin/ls` refers to a file named `ls` inside a directory named `bin` inside a directory named `usr` which is in the root directory

The `/`s separate the names

The root directory is referred to as `/`, though its actual name is empty

Other OSs have similar ideas, but use different separators

Files can have multiple names: we might find that `/usr/local/bin/dir` refers to the same file as `/usr/bin/ls`

Filesystems

Names

The directory hierarchy forms a *directed acyclic graphs* (DAG)

Filesystems

Names

The directory hierarchy forms a *directed acyclic graphs* (DAG)

This means: no loops

Filesystems

Names

The directory hierarchy forms a *directed acyclic graphs* (DAG)

This means: no loops

No loops means we can simply traverse the whole hierarchy and never get stuck in a loop and no unconnected loops if we delete a directory

Filesystems

Names

The directory hierarchy forms a *directed acyclic graphs* (DAG)

This means: no loops

No loops means we can simply traverse the whole hierarchy and never get stuck in a loop and no unconnected loops if we delete a directory

We might find the same file twice, though

Filesystems

Names

The directory hierarchy forms a *directed acyclic graphs* (DAG)

This means: no loops

No loops means we can simply traverse the whole hierarchy and never get stuck in a loop and no unconnected loops if we delete a directory

We might find the same file twice, though

This is a tradeoff of flexibility vs. ease of system implementation

Filesystems

Names

To make referring to files even easier, each process has a *current working directory* (cwd)

Filesystems

Names

To make referring to files even easier, each process has a *current working directory* (cwd)

This is just a prefix, stored in the PCB for each process, so that whenever the process asks for a file by an incomplete filename (not starting with a /), the kernel glues the cwd prefix on to the given name and uses that full name instead

Filesystems

Names

To make referring to files even easier, each process has a *current working directory* (cwd)

This is just a prefix, stored in the PCB for each process, so that whenever the process asks for a file by an incomplete filename (not starting with a `/`), the kernel glues the cwd prefix on to the given name and uses that full name instead

So, with a cwd of `/u/cs/1/cs1abc` a process that asks for file `prog.c` gets file `/u/cs/1/cs1abc/prog.c`

Filesystems

Names

To make referring to files even easier, each process has a *current working directory* (cwd)

This is just a prefix, stored in the PCB for each process, so that whenever the process asks for a file by an incomplete filename (not starting with a `/`), the kernel glues the cwd prefix on to the given name and uses that full name instead

So, with a cwd of `/u/cs/1/cs1abc` a process that asks for file `prog.c` gets file `/u/cs/1/cs1abc/prog.c`

With a cwd of `/u/cs/1/cs1def` a process that asks for file `prog.c` gets file `/u/cs/1/cs1def/prog.c`

Filesystems

Names

This is how different processes can refer to the same name `prog.c` but get different files

Filesystems

Names

This is how different processes can refer to the same name `prog.c` but get different files

The cwd is a convenience for the programmer and may be changed as often as you like (`cd`, `chdir`)