NAME

mount_nullfs - mount a loopback file system sub-tree; demonstrate the use of a null file system layer

SYNOPSIS

mount_nullfs [**-o** *options*] *target mount-point*

DESCRIPTION

The **mount_nullfs** utility creates a nullfs(5) layer, duplicating a sub-tree of the file system name space under another part of the global file system namespace. This allows existing files and directories to be accessed using a different pathname.

The primary differences between a virtual copy of the file system and a symbolic link are that the getcwd(3) functions work correctly in the virtual copy, and that other file systems may be mounted on the virtual copy without affecting the original. A different device number for the virtual copy is returned by stat(2), but in other respects it is indistinguishable from the original.

The **mount_nullfs** utility supports mounting both directories and single files. Both *target* and *mount_point* must be the same type. Mounting directories to files or files to directories is not supported.

The **mount_nullfs** file system differs from a traditional loopback file system in two respects: it is implemented using a stackable layers techniques, and its "null-node"s stack above all lower-layer vnodes, not just over directory vnodes.

The options are as follows:

-o Options are specified with a **-o** flag followed by a comma separated string of options. See the mount(8) man page for possible options and their meanings. Additionally the following option is supported:

nocache

Disable metadata caching in the null layer. Some lower-layer file systems may force this option. Depending on the access pattern, this may result in increased lock contention.

The null layer has two purposes. First, it serves as a demonstration of layering by providing a layer which does nothing. (It actually does everything the loopback file system does, which is slightly more than nothing.) Second, the null layer can serve as a prototype layer. Since it provides all necessary layer framework, new file system layers can be created very easily by starting with a null layer.

The remainder of this man page examines the null layer as a basis for constructing new layers.

INSTANTIATING NEW NULL LAYERS

New null layers are created with **mount_nullfs**. The **mount_nullfs** utility takes two arguments, the pathname of the lower vfs (target-pn) and the pathname where the null layer will appear in the namespace (mount-point-pn). After the null layer is put into place, the contents of target-pn subtree will be aliased under mount-point-pn.

OPERATION OF A NULL LAYER

The null layer is the minimum file system layer, simply bypassing all possible operations to the lower layer for processing there. The majority of its activity centers on the bypass routine, through which nearly all vnode operations pass.

The bypass routine accepts arbitrary vnode operations for handling by the lower layer. It begins by examining vnode operation arguments and replacing any null-nodes by their lower-layer equivalents. It then invokes the operation on the lower layer. Finally, it replaces the null-nodes in the arguments and, if a vnode is returned by the operation, stacks a null-node on top of the returned vnode.

Although bypass handles most operations, *vop_getattr*, *vop_inactive*, *vop_reclaim*, and *vop_print* are not bypassed. *Vop_getattr* must change the fsid being returned. *Vop_inactive* and *vop_reclaim* are not bypassed so that they can handle freeing null-layer specific data. *Vop_print* is not bypassed to avoid excessive debugging information.

INSTANTIATING VNODE STACKS

Mounting associates the null layer with a lower layer, in effect stacking two VFSes. Vnode stacks are instead created on demand as files are accessed.

The initial mount creates a single vnode stack for the root of the new null layer. All other vnode stacks are created as a result of vnode operations on this or other null vnode stacks.

New vnode stacks come into existence as a result of an operation which returns a vnode. The bypass routine stacks a null-node above the new vnode before returning it to the caller.

For example, imagine mounting a null layer with

```
mount_nullfs /usr/include /dev/layer/null
```

Changing directory to */dev/layer/null* will assign the root null-node (which was created when the null layer was mounted). Now consider opening *sys*. A vop_lookup would be done on the root null-node. This operation would bypass through to the lower layer which would return a vnode representing the UFS *sys*. Null_bypass then builds a null-node aliasing the UFS *sys* and returns this to the caller. Later operations on the null-node *sys* will repeat this process when constructing other vnode stacks.

CREATING OTHER FILE SYSTEM LAYERS

One of the easiest ways to construct new file system layers is to make a copy of the null layer, rename all files and variables, and then begin modifying the copy. The sed(1) utility can be used to easily rename all variables.

The umap layer is an example of a layer descended from the null layer.

INVOKING OPERATIONS ON LOWER LAYERS

There are two techniques to invoke operations on a lower layer when the operation cannot be completely bypassed. Each method is appropriate in different situations. In both cases, it is the responsibility of the aliasing layer to make the operation arguments "correct" for the lower layer by mapping a vnode argument to the lower layer.

The first approach is to call the aliasing layer's bypass routine. This method is most suitable when you wish to invoke the operation currently being handled on the lower layer. It has the advantage that the bypass routine already must do argument mapping. An example of this is *null_getattrs* in the null layer.

A second approach is to directly invoke vnode operations on the lower layer with the *VOP_OPERATIONNAME* interface. The advantage of this method is that it is easy to invoke arbitrary operations on the lower layer. The disadvantage is that vnode arguments must be manually mapped.

SEE ALSO

nullfs(5), mount(8)

UCLA Technical Report CSD-910056, *Stackable Layers: an Architecture for File System Development*.

HISTORY

The **mount_null** utility first appeared in 4.4BSD. It was renamed to **mount_nullfs** in FreeBSD 5.0.