From "Hello World\n" to the VFS Layer

Building a HAMMER2 `beadm(1)` in C

newnix

Exile Heavy Industries

September 22, 2018

Introductions

Who am I?

- College dropout
- Network, VM, and Storage Administrator/Engineer
- General UNIX-like systems admin
- General computing and hacking enthusiast
- Guy that loves working with C and other "low-level" languages
- Minimalism and Simplicity Enthusiast

I'm not a professional (Developer)

My first point of pride was this simple difference in executable size:

```
 % stat -x `which env` `which nxenv`
File:  ``/usr/bin/env''
Size:  77256 FileType:  Regular File
```

I'm not a professional (Developer)

My first point of pride was this simple difference in executable size:

```
 % stat -x `which env` `which nxenv`
File:  ``/usr/bin/env''
Size:  77256 FileType:  Regular File
File:  ``/home/newnix/bin/c/nxenv''
Size:  6168 FileType:  Regular File
```

Docs! Everyone's Favorite

`man(1)` is actually useful

- **libc** Everything's actually documented in section 3, so easy to get to work
- **syscalls** Syscalls are all under section 2, including everyone's favorite: `ioctl(2)`!
- **perl** I'm not the biggest fan, but I love that section 3p actually exists
- **examples** The single best thing when learning, is having example code or struct layouts embedded in the documentation

Brief intro to HAMMER2

Probably the best HAMMER yet

CoW One of the single coolest features available in filesystems today

LZ4 By default, everything gets compressed, if it can be

PFS Analagous to ZFS datasets, this lets you create separate filesystems for certain sections of your install

Clustering Not available yet, but HAMMER2 is designed to allow mounting over the network, so you can distribute your storage

Get the Filesystems

This turned out to be pretty simple after some experimenting with `statfs(2)` and `statvfs(2)`, as well as `getfsstat(2)`

Downside: Requires creating a buffer large enough to store results

Upside: Passing a NULL pointer returns number of mounted filesystems, actually returns usable data

Conclusion: I would not be surprised if there's a better method, but I haven't come across it yet

Example use of getfsstat

```
if ((fscount = getfsstat(NULL, 0, MNT_WAIT)) > 0) {
  if ((buf = calloc(sizeof(struct statfs*), fscount)) != NULL) {
    ret = getfsstat(buf, (sizeof(*buf) * fscount), MNT_WAIT);
  }
}
```

Determine if they're HAMMER2

statfs(2) `statfs->f_fstype` has the filysestem type string

Determine if they're HAMMER2

statfs(2) `statfs->f_fstype` has the filysestem type string

getfsent(3) `fstab->fs_vfstype` only available if the filesystem's listed in the fstab

Determine if they're HAMMER2

statfs(2) `statfs->f_fstype` has the filysestem type string

getfsent(3) `fstab->fs_vfstype` only available if the filesystem's listed in the fstab

ioctl(2) `HAMMER2IOC_PFS_LOOKUP` will return false positives for NULLFS mounts

My code is currently relying on a few assumptions I'd like to eliminate by finding a better means of identifying filesystems

And now everyone's favorite thing about C: STRINGS!

This has so far been the most difficult part of the project

fstab(5) 100% strings, fortunately, the `getfsent(3)` function will parse it into an `fstab` struct for us

And now everyone's favorite thing about C: STRINGS!

This has so far been the most difficult part of the project

fstab(5) 100% strings, fortunately, the `getfsent(3)` function will parse it into an `fstab` struct for us

ioctl(2) `HAMMER2IOC_PFS_GET` will provide a `pfs.name`, which has to be parsed and updated for the BE creation

And now everyone's favorite thing about C: STRINGS!

This has so far been the most difficult part of the project

fstab(5) 100% strings, fortunately, the `getfsent(3)` function will parse it into an `fstab` struct for us

ioctl(2) `HAMMER2IOC_PFS_GET` will provide a `pfs.name`, which has to be parsed and updated for the BE creation

statfs(2) `fstat->f_fstypename` is a string as well, and using `fstat->f_type` is unreliable

Atomicity

One thing that's been increasingly important in modern programs is the concept of atomic operations, I'd like to have some guaranteed level of atomicity in this project as well.

ROFS If root, and therefore `/etc` is read-only, then we can't install the new fstab

OOM Print an error message, alert the user, clean up and exit

SIGINT Present confirmation prompt to the user prior to cleanup

SIGTERM Cleanup and exit without confirmation

SIGKILL Trickier; ideally have a cleanup thread run in case of partial creation

Design and abstraction

All functions and variables are scope limited to the operations that need them

FS Functions other than `ish2()` should be filesystem-agnostic

Plugins I'd like to create a means of adding functionality at runtime, to say, add a hook for calling some backup utility

Functions Right now, I'm leaning pretty heavily on the best syscall ever, `ioctl`, but I'd like to get to a point where this functionality is abstracted into at least one library. So if desired, others can build a GUI or other front-end to this functionality.

Internal Struct

The primary struct in use is the `bootenv_data` struct, which gets passed to every function working on filesystems

```
struct bootenv_data {
        struct fstab fstab;
        char curlabel[NAME_MAX];
        struct hammer2_ioc_pfs snapshot;
        bool snap;
}
```

All You Need is `ioctl(2)`

As stated previously, `ioctl(2)` is the best syscall of all time
This project really only requires 3 of them, but it could be expanded to use others

`HAMMER2IOC_PFS_DELETE`

All You Need is `ioctl(2)`

As stated previously, `ioctl(2)` is the best syscall of all time
This project really only requires 3 of them, but it could be expanded to use others

`HAMMER2IOC_PFS_DELETE`
`HAMMER2IOC_PFS_GET`

All You Need is `ioctl(2)`

As stated previously, `ioctl(2)` is the best syscall of all time
This project really only requires 3 of them, but it could be expanded to use others

`HAMMER2IOC_PFS_DELETE`
`HAMMER2IOC_PFS_GET`
`HAMMER2IOC_PFS_SNAPSHOT`

Creation

Creating a new "boot environment" is simply creating a new PFS for every existing HAMMER2 mountpoint with a user-provided label, using a function that looks a bit like this:

```
strlcpy(bootenv_data.snapshot.name, label, NAME_MAX);
if ((fd = open(bootenv_data.fstab.fs_file, O_RDONLY)) < 0) {
     ioctl(fd, HAMMER2IOC_PFS_SNAPSHOT, &bootenv_data.snapshot);
}
```

Activation

Due to the nature of HAMMER2, "activation" requires replacing the current fstab with one generated using the new filesystem data. This is a simple loop of:

```
for ( i = 0; i < fscount; i++) {
     fprintf(efstab,"%s\n",bedata[i].fstab_entry);
}
```

What's left to do?

My work on this project is far from done, so in no particular order:

- Priviledge drops for functions that don't need root access

What's left to do?

My work on this project is far from done, so in no particular order:

- Priviledge drops for functions that don't need root access
- Better task isolation

What's left to do?

My work on this project is far from done, so in no particular order:

- Priviledge drops for functions that don't need root access
- Better task isolation
- Cleanup handler

What's left to do?

My work on this project is far from done, so in no particular order:

- Priviledge drops for functions that don't need root access
- Better task isolation
- Cleanup handler
- Better filesystem detection

What's left to do?

My work on this project is far from done, so in no particular order:

- Priviledge drops for functions that don't need root access
- Better task isolation
- Cleanup handler
- Better filesystem detection
- Plugin system to extend functionality

What's left to do?

My work on this project is far from done, so in no particular order:

- Priviledge drops for functions that don't need root access
- Better task isolation
- Cleanup handler
- Better filesystem detection
- Plugin system to extend functionality
- Debug interface and status reporting with verbosity options

What's left to do?

My work on this project is far from done, so in no particular order:

- Priviledge drops for functions that don't need root access
- Better task isolation
- Cleanup handler
- Better filesystem detection
- Plugin system to extend functionality
- Debug interface and status reporting with verbosity options
- Library holding FS functions

Sources and contact info

mastodon	https://linuxrocks.online/@architect
mastodon	https://bsd.network/@newnix
site	https://exile.digital
code	https://exile.digital/code/c/bsd/dfbsd/beadm
github	https://github.com/newnix/Forge
efnet	newnix
freenode	newnix
geekshed	architect
Email	find me later