Variant Symbolic Links for FreeBSD

Brooks Davis

Computer Systems Research Department
Computers and Software Division
The Aerospace Corporation

August 2008

Outline

Outline

What are Variant Symlinks?

Symbolic links that change targets based on variables

```
$ echo bar > bar; echo baz > baz
$ ln -s '${XXX}' foo
$ ls -l foo
lrwxr-xr-x  1 brooks  wheel ... foo -> ${XXX}
$ varsym XXX=bar cat foo
bar
$ varsym XXX=baz cat foo
baz
```

Prior Art

AFS @sys

AFS allows symlinks to contain the magic variable @sys which identifies the local system type.

Domain/OS

Apollo's Domain/OS allows arbitrary environment variables[a] in symlinks.

[a]Possible due to path lookup being done in userspace.

Outline

Overview

Derived from DragonFly BSD Implementation

- Matt Dillon did the DFBSD version
- Andrey Elsukov did an initial port to FreeBSD

`/bin/sh` style syntax

- `${VAR}` can appear anywhere in a symlink path
- Administrator may optionally enable `${VAR:default}` support.
- Variables are set with `varsym(1)`

Overview

Derived from DragonFly BSD Implementation

- Matt Dillon did the DFBSD version
- Andrey Elsukov did an initial port to FreeBSD

`/bin/sh` style syntax

- `${VAR}` can appear anywhere in a symlink path
- Administrator may optionally enable `${VAR:default}` support.
- Variables are set with `varsym(1)`

Namespaces

System Scope Variables

- Take precedence over process variables
- Settable by super user only
- No allocation limits
- Target for virtualization

Process Scope Variables

- Settable on the current process
- Variables follow fork
- Setting is a privileged operation by default
- Limited in number if unprivileged

syscalls

```
int varsym_set(int scope, id_t which,
    const char *name, const char *data)
```

Sets the variable `name` in the object specified by `scope` and `which` to the value pointed to by `data`.

```
int varsym_get(int scope, id_t which,
    const char *name, char *buf, size_t *size)
```

Retrieves the variable `name` in the object specified by `scope` and `which` and returns the value in `buf`. The amount written is returned in `size`.

syscalls

```
int varsym_list(int scope, id_t which,
    char *buf, size_t *size);
```

Retrieves all variables in the object specified by `scope` and `which` and writes them to `buf` as a 0 separated list. The amount written is returned in `size`.

General Notes

- The `which` variable is currently unused. To prevent applications from setting values that might someday be used, we require `which` to be set to 0.
- There is no easy way to size the buffer for `varsym_list()` so allocating something largish and looping until you don't get E2BIG is required.

Kernel Structures

varsym_t

```
struct varsym {
    u_int          vs_refs;
    int            vs_namelen;
    char           *vs_name;
    char           *vs_data;
};
typedef struct varsym   *varsym_t;
```

THE AEROSPACE CORPORATION

Kernel Structures

struct varsymset

```
struct varsyment {
    TAILQ_ENTRY(varsyment) ve_entry;
    varsym_t     ve_sym;
};

struct varsymset {
    TAILQ_HEAD(, varsyment) vx_queue;
    int          vx_setsize;
};
```

THE AEROSPACE CORPORATION

Outline

THE AEROSPACE CORPORATION

Other Things I'm Thinking About

- Should we use `/bin/sh`, AFS, or some other syntax like %%VAR%%?
- Should we limit varsyms when they can only be manipulated by privileged users?
- Should we have separate privileged and unprivileged per-process sets?
- Syscalls return ENOSYS when disabled, is that OK?
- Should we put this in GENERIC?