A Clash of Things

Steps towards unambiguous symbol resolution

Nick Alcock, Oracle Corporation

```
                 ...
ffffffff951d4540 t m_next
ffffffff951e9160 t m_next
ffffffff95341340 t m_next
ffffffff9539f3c0 t m_next
ffffffff951d4ea0 t m_show
ffffffff95340980 t m_show
                 ...
```

The problem

Symbol names in the kernel are not unique!

Identical symbols can and do recur at frequent intervals: some have static scope (often inlines); some are in modules that happen to be built-in; all are trouble if you’re doing symbol resolution

/proc/kallmodsyms

The /proc/kallmodsyms series I've been working on for a few years is a partial solution: function symbols are disambiguated by the module they *would* be in if the module they were in were not built into the kernel: so, unaffected by .config

```
ffffffff95341340 6b t m_next
ffffffff9539f3c0 46 t m_next
ffffffff951d4540 1a t m_next    [module]
ffffffff951e9160 18 t m_next    [user_namespace]
```

Efficient, but not perfect

```
ffffffff95341340 6b t m_next
ffffffff9539f3c0 46 t m_next
ffffffff951d4540 1a t m_next    [module]
ffffffff951e9160 18 t m_next    [user_namespace]
```

Some `m_next` symbols disambiguated, but some are in the core kernel repeatedly, so are still not disambiguated

We can fix this! (If we want to.)

Possible improvement

```
ffffffff95341340 6b t m_next {kernel/foo.o}
ffffffff9539f3c0 46 t m_next {mm/bar.o}
ffffffff951d4540 1a t m_next {kernel/module.o} [module]
ffffffff951e9160 18 t m_next {kernel/user_namespace.o} [user_namespace]
```

Decorating with object file names as well as modules is possible: it means we store more names, though (less space-efficient).

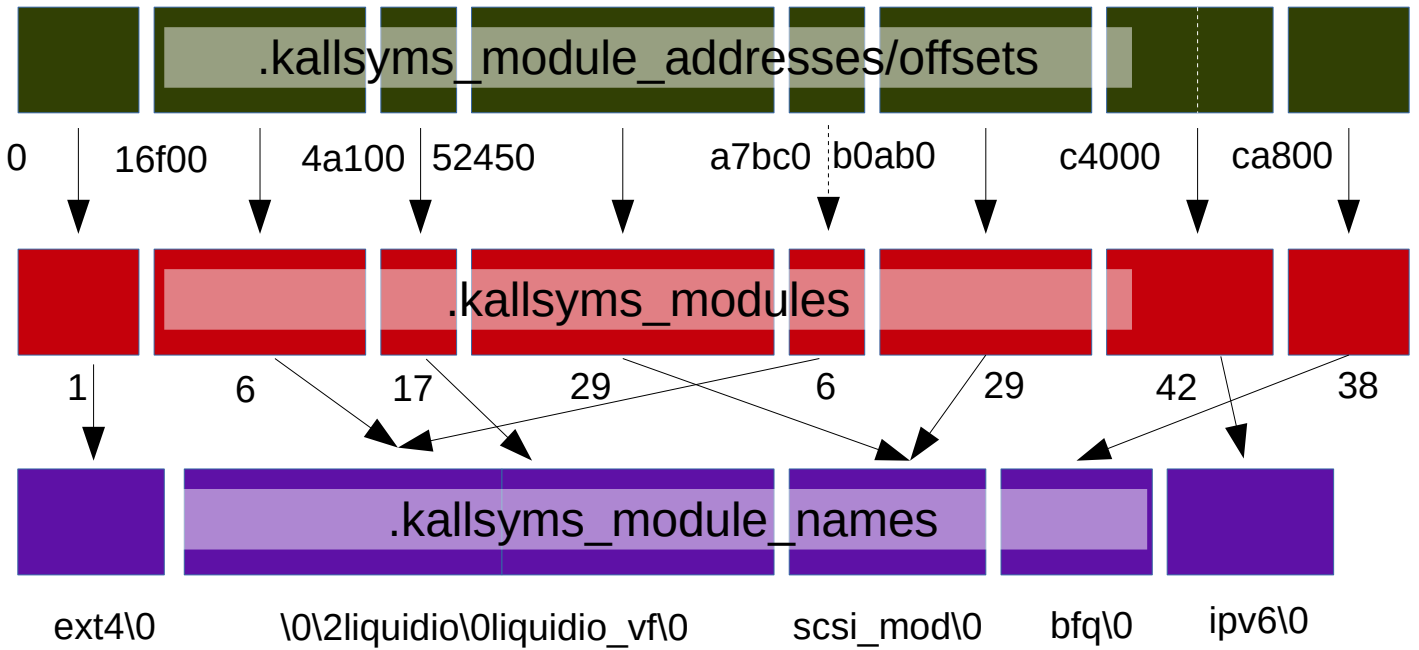
We can get that space back by storing only names for TUs containing symbols that need it for disambiguation.

Details: representation

Both old and new approach handled using linker maps; rather than tracking symbols (too many!) we track whole TUs at once, mapping from address range to TU to the modules (possibly multiple!) that make up that TU

Right now, we don’t store any names for the TUs, but adding that is trivial.

Data structure



(object files from linker mapfile)

Questions

DTrace at least found it repeatedly useful to know that ext4 symbols are in [ext4] even if they are built into the kernel: our CTF type info is organized that way, and it means users can probe ext4`foo without worrying about whether ext4 is built in or not (since users more or less never care and this changes frequently).

Does anyone else find this useful? Both code and data overheads are very small (under 10KiB). Tracking TU names would add more, since there are quite a lot.

Are there any objections to the representation, or the syntax used to show it? Is it actually so unobjectionable that it should go into `/proc/kallsyms` directly? (I suspect that would break too many parsers...)

Useful links

- Most recent kallmodsyms posting:

  https://lore.kernel.org/linux-modules/20220208184309.148192-1-nick.alcock@oracle.com/t/#u

- git tree (latest against 6.0-rc3):

  https://github.com/oracle/dtrace-linux-kernel kallmodsyms/latest