

Re: C++ in the kernel

Source: <http://unix.derkeiler.com/Mailing-Lists/FreeBSD/arch/2007-10/msg00139.html>

- *From:* "Poul-Henning Kamp" <phk@xxxxxxxxxxxxxxxx>
 - *Date:* Tue, 30 Oct 2007 17:14:38 +0000
-

In message <20071030163613.E70665B30@xxxxxxxxxxxxxxxxxxxx>, Bakul Shah writes:

Is there a more detailed writeup of Poul's ideas on K language beyond what is on the wiki.freebsd.org/K? Reading it brings to mind Brinch Hansen's extensions to Pascal.

Knowing that I may regret this, here is my private notes file on K syntax extensions.

Preprocessor shell escape

```
#! <command> { ident  
<stdin>  
} ident
```

Generate C/K source with external program. Can be used to generate tables or include firmware files directly from the binary.

Public/Private structs

```
public struct foo {  
...  
}
```

Has `sizeof() = -1`, cannot be assigned.

```
private struct foo {  
...  
}
```

Adds fields to the public struct foo, has `sizeof` the joint size. Acts like normal struct.

Bitmaps

```
bitmap foo { bar, barf, ...};
```

Like enum, but assigns bit values and allows logical operations.

Style/Syntax checks

Flag misindentation:

```
if (some_cond)
do_this(foo);
and_that(bar);
```

Style(9) warnings

Macros taking code as argument

```
#define LOCK(a, { })
do {
mtx_lock(&a);
__CODE__
mtx_unlock(&a);
} while (0);
```

XXX: what happens on return/break/etc ?

```
#define LOCK(a, { })
__CODE__(
entry { mtx_lock(&a); }
return { mtx_unlock(&a); }
)
```

XXX: not happy about syntax

Pointer colors

```
void * "userland" ptr;
```

Cannot be used as regular void *, but must be passed to functions which has same color prototype.

Integer ranges

```
int foo [21...45];
or
int foo range [21...45];
```

Integer endianness

```
uint32_t big_endian foo;
```

Atomic variables

```
uint32_t atomic foo;
```

Struct offsets and sizes

```
struct foo {  
    sizeof 128;  
    big_endian;  
    align 16;  
    uint8_t foo @0;  
    uint32_t little_endian bar @12;  
}
```

Multi-loop break

```
if (foo != NULL) {  
    TAILQ_FOREACH(foo, &bar, list) {  
        if (foo->idx == idx)  
            break(2);  
    }  
    panic("not found");  
}
```

Call identification

```
int foo(int barf, void *there, private void **id);
```

The "id" argument points to a call specific, globally unique, static instance of the pointed to type. This can be used for instance to cache a function pointer for lazy binding (KOBJ ?)

Alternatively, even faster, go the full length and make run-time resolved function pointers (trouble hunting them down at modunload ?)

Sensible jump prediction

```
TAILQ_FOREACH(foo, &bar, list) {{  
    do_this_alot(foo);
```

```
}}
```

```
#include pointlessness warnings
```

Warn about #includes that have no effect

Cross-Referencing

Generate cross-reference data on joint preproc/C/K level.

Function instantiation by prototype

```
typedef int foo_f(int arg, void *priv, struct *obj);
```

```
static  
foo_f thisfunc(.) {  
if (arg) {  
...  
}  
}
```

Argument struct/array building

```
int foo(struct timeval tv);
```

```
int  
bar(int someargs) {  
  
foo({.tv_sec = someargs});  
}
```

Compile time debugging aids

* Look out for 0xdead0de+/-256
Check any pointer dereference against the indicated interval

* Struct */void * typecheck
Give each struct a magic identifier, check after void* transport.

* Struct canary insertion
Insert canary elements in struct and check their magic values
whenever neighbouring elements are tweaked.

* Assignment code insertion
Insert code "foo" whenever variable "bar" is assigned or changed.
Typical values of "foo" would be "mtx_assert_locked(...)"

<sys/queue.h> replacement

list_head(struct foo, ...) foolist;
possible options:
single Single linked list
double Double linked list
tail Tail is accessible from head
head Head is accessible from elem
member <name>
Only allow use with member <name> in target struct.

```
struct foo {  
list_member list;  
list_member(opts...) list2;  
};
```

```
list_empty(head)  
list_next(elem)  
list_prev(elem)  
list_first(head)  
list_last(head)  
list_insert_head(elem, head)  
list_insert_tail(elem, tail)  
list_insert_before(elem, elem {,head ?})  
list_insert_after(elem, elem {,head ?})  
list_remove(elem {,head ?})  
list_init_head(head)  
list_take_first(head)  
list_take_last(head)  
list_foreach(elem, head {,member})  
list_foreach_safe(elem, head {,member})  
list_foreach_reverse(elem, head {,member})  
list_foreach_reverse_safe(elem, head {,member})
```

—
Poul-Henning Kamp | UNIX since Zilog Zeus 3.20
phk@xxxxxxxxxxxx | TCP/IP since RFC 956
FreeBSD committer | BSD since 4.3-tahoe
Never attribute to malice what can adequately be explained by incompetence.

freebsd-arch@xxxxxxxxxxxx mailing list
<http://lists.freebsd.org/mailman/listinfo/freebsd-arch>
To unsubscribe, send any mail to "freebsd-arch-unsubscribe@xxxxxxxxxxxx"