A Domain-Specific Language for Device Drivers

Christopher Conway

30 October 2002

What's wrong with device drivers?

- Drivers usually run in supervisor mode. They are critical code.

- Chou, et al.: Drivers account for 70-90% of bugs in the Linux kernel and have an error rate 7x that of the rest of the kernel.

- So more robust drivers would go a long way to making a more robust operating system. Just ask Microsoft: VAULT, SLAM.

VAULT

Deline and Fändrich

Type guards statically enforce use protocols.

```
void fclose(tracked(F) FILE f) [-F];

int main(...) {
  tracked(F) FILE f ; // there is a key F protecting FILE f
  ...
  flose(f) ; // the key is deleted by fclose
  /* any use of f from this point is invalid */
}
```

Problem: defining these use protocols is difficult and tricky.

GAL

Thibault, Marlet and Consel

Domain-specific language for X Windows video drivers.

```
mode HighRes := HTotal > 800;
enable HighRes sequence is
     Control[5] <= 1;
```

Pros: Small code (90% smaller than C), fast

But: Not a general solution

Devil

Mérillon, et al.

A device specification is transformed into C macros.

```
variable nicState = write CommandReg[1..0] :
{
  START   => '10',
  STOP    => '01',
  CURRENT => '00'
};
```

Called from the driver in C:

```
dev_ns8390 *ns8390 = new_ns8390(GFP_KERNEL, ioaddr);
set_nicState(ns8390, STOP);
```

Pros:

- Cleanly separates device interface from use.
- Reduces potential errors from misuse of device.

Cons:

- No type safety.
- No code simplification
- Doesn't attack the heart of the problem: operating system interaction

My Language

- Type safe: maps device registers to high-level types.
- Simple: directly supports the semantics of device drivers.
- Platform-independent: the compiler handles the details of the operating systems' device driver protocols.
- Useful: captures the behavior of a broad class of devices, makes programming drivers easier.