Towards safer Concurrent Device Drivers

Modeling RMoX Drivers in CSP

Martin Ellis

School of Computing
University of Kent

Communicating Process Architectures, 2011

Outline

Outline

The World So Far. . .

We want "safe" and "correct" concurrent device drivers.

- Device Driver / Kernel interface well understood.
- Device Driver / Hardware interface less so.

Outline

Existing Techniques

Most previous work done on kernel/driver interfaces.

- Slam.
  - Static analysis of Windows drivers.
  - Tried to help provent kernel crashes (BSoD).
- DDVERIFY
  - Static analysis of Linux drivers.
  - Handles concurrent Linux drivers.
- Fred Barne's work on modeling drivers is CSP.
  - Prove deadlock freedom of RMoX drivers.
  - Only considered the Driver/Kernel interface.
- Driver synthesis.
  - Chinook.
  - Mattias I'Nils' and Axel Jantsch's work with ProGram.

Device Driver Complexities.

- Memory mapped IO vs port mapped IO.
- Overloaded addresses.
- Bitfields.
- Concurrent access.

bitfield port

0 4 8 12 15

rdy	parity	data	reserved

Select and Access ports

0 4 8 12 15

Register Select	0x200
Access	0x216

bitfield port

0	4	8	12	15

rdy	parity	data	reserved

Select and Access ports

0	4	8	12	15

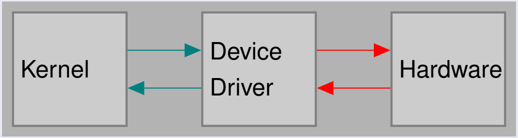
Register Select	} 0x200
Access	} 0x216

Concurrent Access

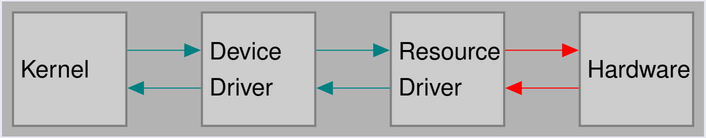
- Placed memory/channels
- Circumvents parallel usage checking
- All the usual issues with data aliasing.

Outline

- Kernel / Driver interface made of well understood occam channels.
- Hardware / Driver interface made of "magic".
- Abstract things into nice occam channels.



- Kernel / Driver interface made of well understood occam channels.
- Hardware / Driver interface made of "magic".
- Abstract things into nice occam channels.



Resource Driver

So what does the resource driver give us?

- Primitives for reading registers "correctly"
- Sanity checks (no use before decleration etc)
- These runtime checks are slow though. ☹

Resource Driver

So what does the resource driver give us?

- Primitives for reading registers "correctly"
- Sanity checks (no use before decleration etc)
- These runtime checks are slow though. ☹

Outline

Extending KRoC

KRoC's CSP model generation has been extended.

- Now includes details of variant channels.
- Number of parameters.
- Values known known at compile time.

Protocol → CSP

occam

```
PROTOCOL P.RES
 CASE
   a; INT
   b; INT; BYTE
:
```

CSP

$$
\begin{aligned}
U &= (-999) \\
NUMBER &= \{U\} \cup \{0..99\} \\
channel\ res &: a.NUMBER \mid \\
&\quad b.NUMBER.NUMBER
\end{aligned}
$$

Protocol → CSP

occam

```
PROTOCOL P.RES
 CASE
   a; INT
   b; INT; BYTE
:
```

CSP

$$
\begin{array}{lcl}
U & = & (-999) \\
NUMBER & = & \{U\} \cup \{0..99\} \\
channel res & : & a.NUMBER \mid \\
 & & b.NUMBER.NUMBER
\end{array}
$$

Communication → CSP

occam

```
PROC x (chan P.RES res!)
  SEQ
    res ! a; x
    res ! b; y; z
:
```

CSP

$$\begin{aligned} X(res) \quad = \quad & res.a!(U) \rightarrow \\ & res.b!(U).(U) \rightarrow \\ & SKIP \end{aligned}$$

Communication $\rightarrow$ CSP

occam

```
PROC x (chan P.RES res!)
  SEQ
    res !  a; x
    res !  b; y; z
:
```

CSP

$$
\begin{aligned}
X(res) \quad = \quad & res.a!(U) \rightarrow \\
& res.b!(U).(U) \rightarrow \\
& SKIP
\end{aligned}
$$

Constant Propagation

occam

```
PROC x (chan P.RES res!)
  SEQ
    res ! a; 42
    res ! b; y; z
:
```

CSP

$$
\begin{aligned}
X(res) \;=\; & res.a!(\mathbf{42}) \rightarrow \\
& res.b!(U).(U) \rightarrow \\
& SKIP
\end{aligned}
$$

Externalising internal choice.

Generated CSP

$$
\begin{aligned}
&PPORT\_HANDLER_g = \\
&\quad (srv.InPResResInDeclare?vv.pa \rightarrow \\
&\qquad PPORT\_HANDLER \sqcap STOP\ ) \ \Box \\
&\quad (srv.InPResPortInDeclare?vv.pa.pc \rightarrow \\
&\qquad PPORT\_HANDLER \sqcap STOP) \ \Box \\
&\quad (srv.other1 \ \Box \ srv.other2 \ \Box \ \ldots\ ) \ ; PPORT\_HANDLER
\end{aligned}
$$

Externalising internal choice.

"Tweeked" CSP

$$
\begin{array}{l}
PPORT\_HANDLER_t(RESS, PORTS) = \\
\quad (srv.InPResResInDeclare?vv.pa \rightarrow \\
\qquad PPORT\_HANDLER(RESS, \{pc\} \bigcup PORTS) \\
\qquad\quad \not\triangleleft (pa \notin RESS) \not\triangleright STOP) \, \square \\
\quad (srv.InPResPortInDeclare?vv.pa.pc \rightarrow \\
\qquad PPORT\_HANDLER(RESS, \{pc\} \bigcup PORTS) \\
\qquad\quad \not\triangleleft (pa \in RESS) \bigwedge (pc \notin PORTS) \not\triangleright STOP) \, \square \\
\quad (srv.other1 \, \square \, srv.other2 \, \square \\
\ldots) \, ; \, PPORT\_HANDLER(RESS, PORTS)
\end{array}
$$

Refinement

$$SYSTEM\_PRES\_DRIVER_t \sqsubseteq_T SYSTEM\_PRES\_DRIVER_g$$

$$(SYSTEM\_PRES\_DRIVER_t \parallel DEVICE\_DRIVER) \quad deadlocks?$$

Refinement

$$SYSTEM\_PRES\_DRIVER_t \sqsubseteq_T SYSTEM\_PRES\_DRIVER_g$$

$$(SYSTEM\_PRES\_DRIVER_t \parallel DEVICE\_DRIVER) \quad deadlocks?$$

And It Works...

This works..."ish"

- The state space is huge.
- The *NUMBER* type has to be narrow.

And It Works...

This works..."ish"

- The state space is huge.
- The *NUMBER* type has to be narrow.

And It Works...

This works..."ish"

- The state space is huge.
- The *NUMBER* type has to be narrow.

Can we write specification for drivers in CSP?

$$
\begin{aligned}
READ &= 0 \\
WRITE &= 1 \\
DEFINE_{DSP} &= (port.InPResPortInDefineRes!0.220 \\
&\quad \sqcap port.InPResPortInDefineRes!0.240) \rightarrow \\
&DECLARE.PORTS_{DSP} \rightarrow RESET_{DSP} \rightarrow SKIP \\
DECLARE.PORTS_{DSP} &= (port.InPResPortInDeclare!0.U.1.6.7.8.WRITE \mid\mid\mid \\
&port.InPResPortInDeclare!0.U.1.A.7.8.READ \mid\mid\mid \\
&port.InPResPortInDeclare!0.U.2.C.7.8.WRITE \mid\mid\mid \\
&port.InPResPortInDeclare!0.U.3.C.7.8.READ \mid\mid\mid \\
&port.InPResPortInDeclare!0.U.4.E.7.8.READ) \rightarrow \\
&SKIP \\
RESET_{DSP} &= port.write!0.1 \rightarrow \\
&port.setDelay!3 \rightarrow \\
&port.write!0.0 \rightarrow \\
&port.wait!1.\#AA.100.3 \rightarrow \\
&SKIP
\end{aligned}
$$

Summary

- We can produce nice models of device drivers.
- We have an issue with state space.
- How can we model drivers' expected behaviour?
- How do we deal with the state space issues in FDR?

For Further Reading I

Ball, T. and Cook, B. and Levin, V. and Rajamani, S.K.
SLAM and Static Driver Verifier: Technology transfer of formal methods inside Microsoft.
Springer, 2004.

Barnes, F.R.M. and Ritson, C.G.
Checking process-oriented operating system behaviour using CSP and refinement.
ACM SIGOPS Operating Systems Review, 2010