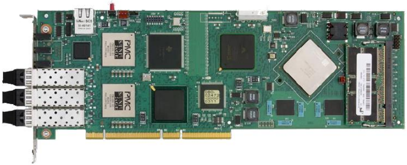
# High-Level Languages for Device Drivers

Stephen A. Edwards

Columbia University

Intel, Hillsboro, Oregon, March 16, 2012



# Between a Rock and a Hard Place

# Between a Rock and a Hard Place

Operating System

Device Driver

Hardware

# A Major Source of Bugs

"These graphs show that driver code is the most buggy, both in terms of absolute number of bugs (as we would suspect from its size) and in terms of error rate."

Chou et al. [SOSP 2001]

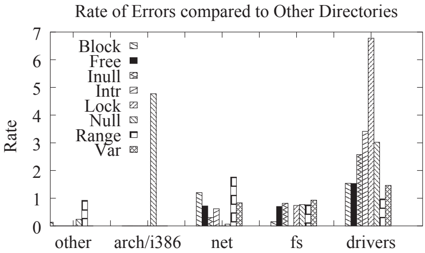
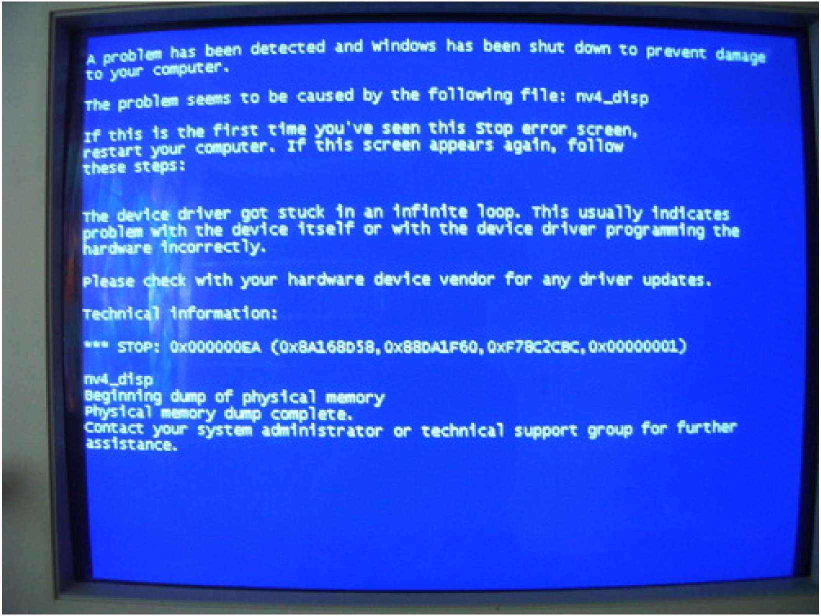


Figure 4: This graph shows drivers have an error rate up to 7 times higher than the rest of the kernel. The `arch/i386` directory has a high error rate for the `Null` checker because we found 3 identical errors in `arch/i386`, and `arch/i386` has relatively few notes.

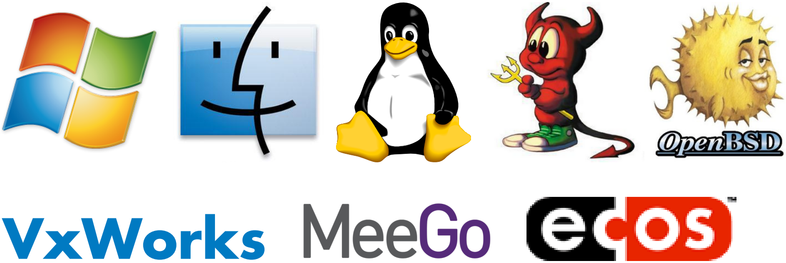
Fault with highest rate: "release acquired locks, do not double-acquire locks"

# Drivers Run in Kernel Mode...Unfortunately



```
A problem has been detected and windows has been shut down to prevent damage
to your computer.

The problem seems to be caused by the following file: nv4_disp

If this is the first time you've seen this Stop error screen,
restart your computer. If this screen appears again, follow
these steps:


The device driver got stuck in an infinite loop. This usually indicates
problem with the device itself or with the device driver programming the
hardware incorrectly.

Please check with your hardware device vendor for any driver updates.

Technical information:

*** STOP: 0x000000EA (0x8A168D58,0x88DA1F60,0xF78C2CBC,0x00000001)

nv4_disp
Beginning dump of physical memory
Physical memory dump complete.
Contact your system administrator or technical support group for further
assistance.
```

# 10s of OSes; Tens of Thousands of Devices

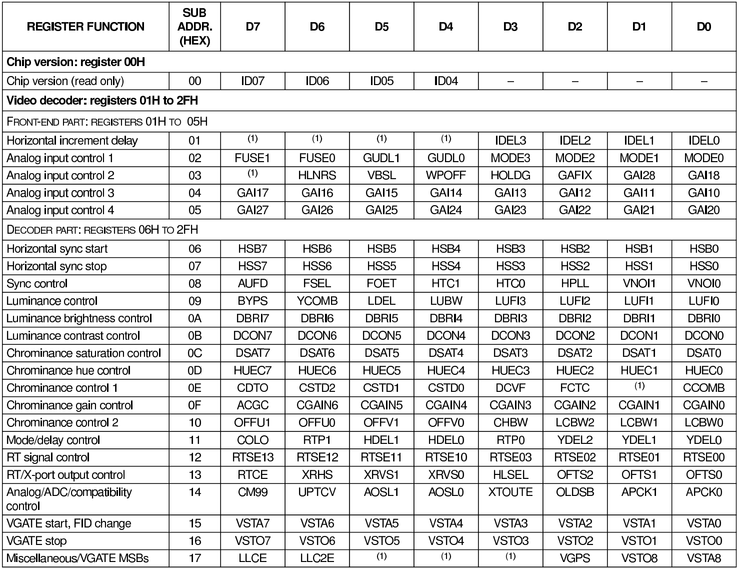


My Linux distribution recognizes 12000 USB devices and 16000 PCI devices

# OS Protocols Complex: WinHEC “Hello World”

```
#include "stddcls.h"
#include "driver.h"
#include "version.h"

DFWSTATUS EvtDriverDeviceAdd(DFWDRIVER hDriver, DFWDEVICE hDevice);
VOID EvtDriverUnload(DFWDRIVER hDriver);

#pragma PAGEDCODE
extern "C" NTSTATUS DriverEntry(PDRIVER_OBJECT DriverObject, PUNICODE_STRING RegistryPath)
{
  PAGED_CODE();
  DfwTraceDbgPrint(DRIVERNAME "␣−␣Version␣%d.%2.2d.%3.3d,␣%s␣%s\n", VERMAJOR, VERMINOR, BUILD, __DATE__, __TIME__);
  DFW_DRIVER_CONFIG config = {sizeof(DFW_DRIVER_CONFIG)};
  config.DeviceExtensionSize = 0;
  config.RequestContextSize = 0;
  config.Events.EvtDriverDeviceAdd = EvtDriverDeviceAdd;
  config.Events.EvtDriverUnload = EvtDriverUnload;
  config.DriverInitFlags = 0;
  config.LockingConfig = DfwLockingDevice;
  config.ThreadingConfig = DfwThreadingAsynchronous;
  config.SynchronizationConfig = DfwSynchronizationNone;
  DFWDRIVER Driver;
  DFWSTATUS status = DfwDriverCreate(DriverObject, RegistryPath, NULL, &config, &Driver);
  if (!NT_SUCCESS(status))
    DfwTraceError(DRIVERNAME "␣−␣DfwDriverCreate␣failed␣−␣%X\n", status);
  return status;
}

#pragma LOCKEDCODE
DFWSTATUS EvtDriverDeviceAdd(DFWDRIVER hDriver, DFWDEVICE hDevice)
{
  DfwTraceDbgPrint(DRIVERNAME "␣−␣EvtDriverDeviceAdd␣entered␣−␣IRQL␣is␣%d\n", KeGetCurrentIrql());
  DFWSTATUS status;
  status = DfwDeviceInitialize(hDevice);
  if (!NT_SUCCESS(status)) {
    DfwTraceError(DRIVERNAME "␣−␣DfwDeviceInitialize␣failed␣−␣%X\n", status);
    return status;
  }
  DFW_FDO_EVENT_CALLBACKS callbacks;
  DFW_FDO_EVENT_CALLBACKS_INIT(&callbacks);
  status = DfwDeviceRegisterFdoCallbacks(hDevice, &callbacks);
  if (!NT_SUCCESS(status)) {
    DfwTraceError(DRIVERNAME "␣−␣DfwDeviceRegisterFdoCallbacks␣failed␣−␣%X\n", status);
    return status;
  }
  return status;
}

#pragma PAGEDCODE
VOID EvtDriverUnload(DFWDRIVER Driver)
{
  PAGED_CODE();
  DfwTraceDbgPrint(DRIVERNAME "␣−␣Unloading␣driver␣−␣IRQL␣is␣%d\n", KeGetCurrentIrql());
}
```

# Hardware Interfaces are Complex

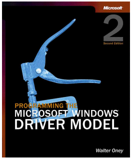


| REGISTER FUNCTION | SUB ADDR. (HEX) | D7 | D6 | D5 | D4 | D3 | D2 | D1 | D0 |
|---|---|---|---|---|---|---|---|---|---|
| **Chip version: register 00H** | | | | | | | | | |
| Chip version (read only) | 00 | ID07 | ID06 | ID05 | ID04 | – | – | – | – |
| **Video decoder: registers 01H to 2FH** | | | | | | | | | |
| FRONT-END PART: REGISTERS 01H TO 05H | | | | | | | | | |
| Horizontal increment delay | 01 | (1) | (1) | (1) | (1) | IDEL3 | IDEL2 | IDEL1 | IDEL0 |
| Analog input control 1 | 02 | FUSE1 | FUSE0 | GUDL1 | GUDL0 | MODE3 | MODE2 | MODE1 | MODE0 |
| Analog input control 2 | 03 | (1) | HLNRS | VBSL | WPOFF | HOLDG | GAFIX | GAI28 | GAI18 |
| Analog input control 3 | 04 | GAI17 | GAI16 | GAI15 | GAI14 | GAI13 | GAI12 | GAI11 | GAI10 |
| Analog input control 4 | 05 | GAI27 | GAI26 | GAI25 | GAI24 | GAI23 | GAI22 | GAI21 | GAI20 |
| DECODER PART: REGISTERS 06H TO 2FH | | | | | | | | | |
| Horizontal sync start | 06 | HSB7 | HSB6 | HSB5 | HSB4 | HSB3 | HSB2 | HSB1 | HSB0 |
| Horizontal sync stop | 07 | HSS7 | HSS6 | HSS5 | HSS4 | HSS3 | HSS2 | HSS1 | HSS0 |
| Sync control | 08 | AUFD | FSEL | FOET | HTC1 | HTC0 | HPLL | VNOI1 | VNOI0 |
| Luminance control | 09 | BYPS | YCOMB | LDEL | LUBW | LUFI3 | LUFI2 | LUFI1 | LUFI0 |
| Luminance brightness control | 0A | DBRI7 | DBRI6 | DBRI5 | DBRI4 | DBRI3 | DBRI2 | DBRI1 | DBRI0 |
| Luminance contrast control | 0B | DCON7 | DCON6 | DCON5 | DCON4 | DCON3 | DCON2 | DCON1 | DCON0 |
| Chrominance saturation control | 0C | DSAT7 | DSAT6 | DSAT5 | DSAT4 | DSAT3 | DSAT2 | DSAT1 | DSAT0 |
| Chrominance hue control | 0D | HUEC7 | HUEC6 | HUEC5 | HUEC4 | HUEC3 | HUEC2 | HUEC1 | HUEC0 |
| Chrominance control 1 | 0E | CDTO | CSTD2 | CSTD1 | CSTD0 | DCVF | FCTC | (1) | CCOMB |
| Chrominance gain control | 0F | ACGC | CGAIN6 | CGAIN5 | CGAIN4 | CGAIN3 | CGAIN2 | CGAIN1 | CGAIN0 |
| Chrominance control 2 | 10 | OFFU1 | OFFU0 | OFFV1 | OFFV0 | CHBW | LCBW2 | LCBW1 | LCBW0 |
| Mode/delay control | 11 | COLO | RTP1 | HDEL1 | HDEL0 | RTP0 | YDEL2 | YDEL1 | YDEL0 |
| RT signal control | 12 | RTSE13 | RTSE12 | RTSE11 | RTSE10 | RTSE03 | RTSE02 | RTSE01 | RTSE00 |
| RT/X-port output control | 13 | RTCE | XRHS | XRVS1 | XRVS0 | HLSEL | OFTS2 | OFTS1 | OFTS0 |
| Analog/ADC/compatibility control | 14 | CM99 | UPTCV | AOSL1 | AOSL0 | XTOUTE | OLDSB | APCK1 | APCK0 |
| VGATE start, FID change | 15 | VSTA7 | VSTA6 | VSTA5 | VSTA4 | VSTA3 | VSTA2 | VSTA1 | VSTA0 |
| VGATE stop | 16 | VSTO7 | VSTO6 | VSTO5 | VSTO4 | VSTO3 | VSTO2 | VSTO1 | VSTO0 |
| Miscellaneous/VGATE MSBs | 17 | LLCE | LLC2E | (1) | (1) | (1) | VGPS | VSTO8 | VSTA8 |

Philips SAA7114H Video Decoder Registers (Page 1 of 7)

> “Because of the complexity of driver programming, we tend, as an industry, to end up with lots of poorly implemented drivers and with confused, disgruntled users. We also don’t fulfill our potential for hardware innovation, because hardware manufacturers are easily stymied by the cost and delay associated with driver development.”

—Walter Oney



http://www.wd-3.com/archive/FrameworkIntro.htm

# Who Writes These Things?

| Author | Knows the Hardware | Knows the OS Interface |
|---|---|---|
| OS Developer | No<br>(a software person) | Yes |
| Hardware Manufacturer | Yes | No<br>(a hardware person) |
| Third Party | No<br>(it’s undocumented) | No<br>(too complex) |

# Wish List

- Model of hardware interface/behavior
- Model for OS interface
- Ways to statically check both models against driver code
- Way to dynamically verify hardware model faithful to real hardware
- Language support for concurrency and events (interrupts)

# We Need a Domain-Specific Language for Device Drivers

*Preventing* (unwanted) behavior the main objective

Libraries add functionality but can’t enforce rules.

# A Model of the Hardware

What can it do and how do you ask for it?

RT-level models far too detailed, proprietary, and provide no insight.

Instead, a model of user-visible states and actions.

# Validating the Hardware Model

Driver developer writes model; must validate against real hardware.

Formal comparison with RTL unrealistic (business & technical); need to validate it independently.

Two ideas:

1. Dynamic validation: maintain model state and check against hardware state as driver runs. Requires test cases.
2. Static validation: “model-check” actual hardware against the model. No test cases but may require guidance.

# A Model of the Operating System

OS developer may write the model (far fewer OSes than devices)

Formal comparison with OS code probably unrealistic

Again, two ideas:

1. Dynamic validation: check each OS/driver interaction for compliance with the model
2. Static validation: “model-check” the OS model against the OS itself. Use a “model checking” driver that can supply all sorts of different stimulus to the OS.

# Static and Dynamic Checks

Want to be able to check driver behavior for compliance against both models.

Again, combination of static and dynamic approaches viable.

Language semantics need to help as much as possible.

# NDL

A first attempt:

Christopher L. Conway and Stephen A. Edwards.
NDL: A Domain-Specific Language for Device Drivers.
In *Proceedings of the ACM Conference on Languages, Compilers, and Tools for Embedded Systems (LCTES)*, Washington, DC, June, 2004.

# NDL

NDL for starting a DMA transfer:

```
start = true;
dmaState = DISABLED;
remoteDmaByteCount = count;
```

The equivalent C:

```
outb(E8390_NODMA + E8390_PAGE0 + E8390_START,
      nic_base + NE_CMD);
outb(count & 0xff, nic_base + EN0_RCNTLO);
outb(count >> 8, nic_base + EN0_RCNTHI);
```

# NDL

Doing a DMA transfer:

```
remoteByteCount = count;
remoteStartAddr = start_page ∗ FRAME_LEN;

trans DMA_WRITING; // Transition to state

dataport =<16> buffer; // Write data to buffer

wait 20ms for remoteDmaIrq else {
   print("ne2k:␣Timeout␣waiting␣for␣Tx␣RDC.");
   soft_reset();
   start_dev();
}

remoteDmaIrq=ACK;
```

# Device Registers

- Device interface typically a block of memory-mapped I/O locations
- NDL provides a structured view of these
  - Fields laid out sequentially; no implicit padding
  - Compiler generates shifts, masks
  - Offset and range assertions checked for sanity
  - Fields can be as small as 1 bit
  - Support for predicated registers (only visible in certain states)
- Device registers appear like variables in NDL code

# Device Registers for NE2000 Compatibles

```
ioports {
  command = {
    0:  stop : trigger except 0,
    1:  start :  trigger except 0,
    2:  transmit : trigger except 0,
    3..5:
        dmaState : {
          READING = #001
          WRITING = #010
          SENDING = #011
          DISABLED = #1**
        } volatile,
    6..7:
        registerPage : int{0..2}
  },

0x01..0x0f:
  [
    ( PAGE(0) ) => { /* predicated regs. */
      write rxStartAddr,
      write rxStopAddr,
      boundaryPtr,
      [
        read txStatus = { /* overlay reg. */
          0: packetTransmitted,
          1: _,
          2: transmitCollided,
          3: transmitAborted,
          4: carrierLost,
          5: fifoUnderrun,
          6: heartbeatLost,
          7: lateCollision
        } volatile
      ||
        write txStartAddr
      ],

      /*  ...  eleven bytes elided ... */
    }
  ||
    ( PAGE(1) ) => { /* predicated regs. */
      physicalAddr : byte[6],
      currentPage : byte,
      multicastAddr : byte[8]
    }
  ||
    ( PAGE(2) ) => { /* predicated regs. */
      _ : byte[13],
      read dataConfig,
      read interruptMask
    }
  ],

0x10: dataport : fifo[1] trigger,
      _ : byte[14],
0x1f: reset : byte trigger
}
```

# States

```
  state STOPPED {
    goto DMA_DISABLED;
    stop = true;
  }
||
  STARTED { start = true; }
```

```
  state DMA_DISABLED {
    dmaState = DISABLED;
  }
||
  DMA_READING {
    goto STARTED;
    dmaState = READING;
  }
||
  DMA_WRITING {
    goto STARTED;
    dmaState = WRITING;
  }

  state PAGE(i : int{0..2}) {
    registerPage = i;
  }
```

# Interrupt Functions

@ indicates the interrupt condition that triggers this function

```
critical function @(countersIrq) {
   rxFrameErrors += frameAlignErrors;
   rxCrcErrors += crcErrors;
   rxMissedErrors += packetErrors;
   countersIrq = ACK;
}
```