Subclassing NEWBUS

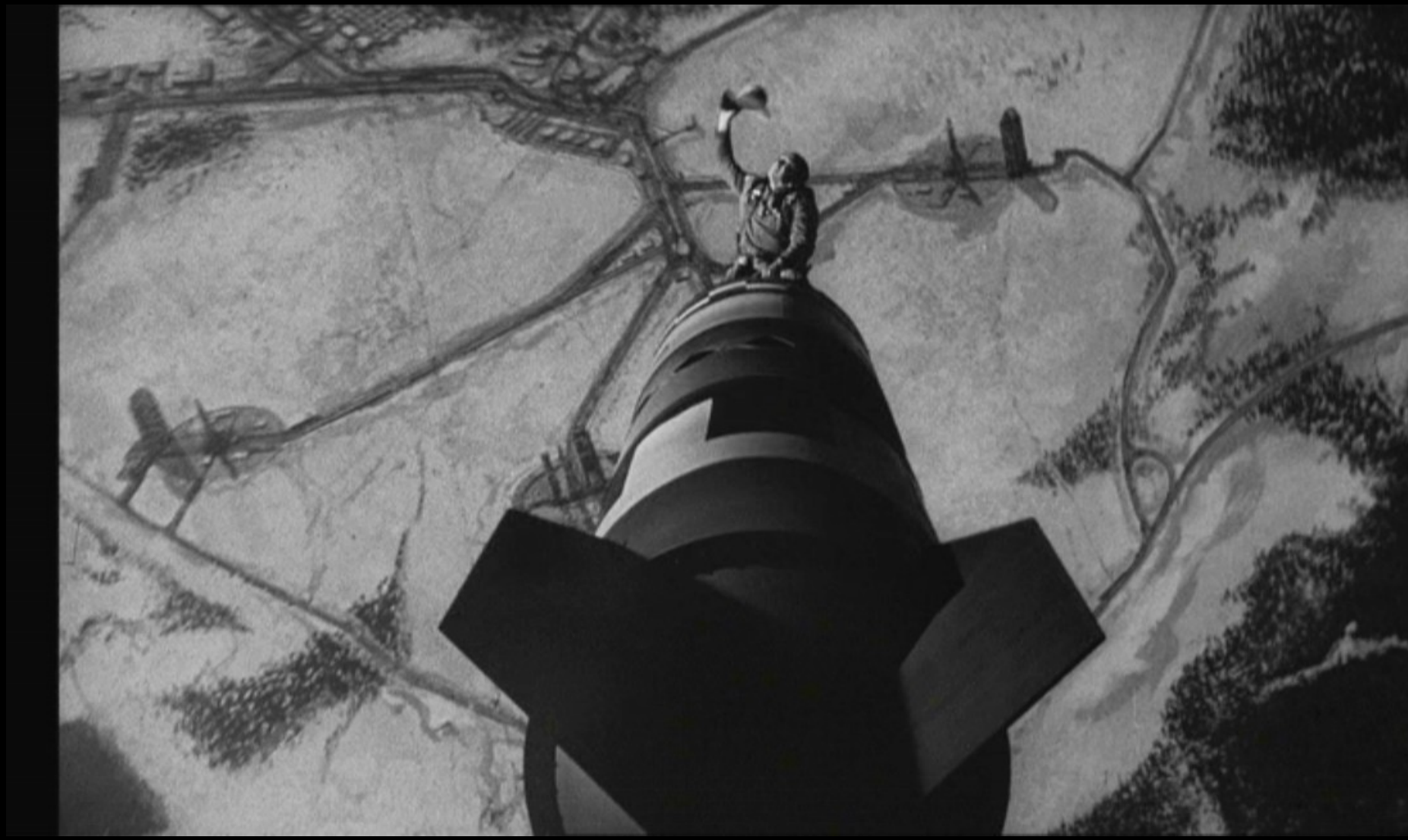
Unlocking NEWBUS' potential

Warner Losh
imp@FreeBSD.org

The FreeBSD Project

BSDCan 2013 — Ottawa, Canada
18 May 2013

http://people.freebsd.org/~imp/bsdcan2013-slides.pdf



How I Learned to Stop Worrying and Love NEWBUS

Warner Losh
imp@FreeBSD.org

[1] Still from *Dr. Srangelove*

Outline

Outline

Definitions

- OOP - Object Oriented Programming[2]
  - Dynamic Dispatch
  - Encapsulation
  - Subtype Polymorphism
  - Object Delegation / Inheritence
  - Open Recursion
- Optional Features
  - Classes of Objects
  - Instances of Classes
  - Methods which act on Objects
  - Message Passing
  - Abstraction
  - Type Safety

[2] https://en.wikipedia.org/wiki/Object-oriented_programming
From Pierce, Benjamin (2002). *Types and Programming Languages.* MIT Press. ISBN 0-262-16209-1., section 18.1 "What is Object-Oriented Programming"

Visual Approximation of NEWBUS

[3]

[3] http://failblog.cheezburger.com/thereifixedit

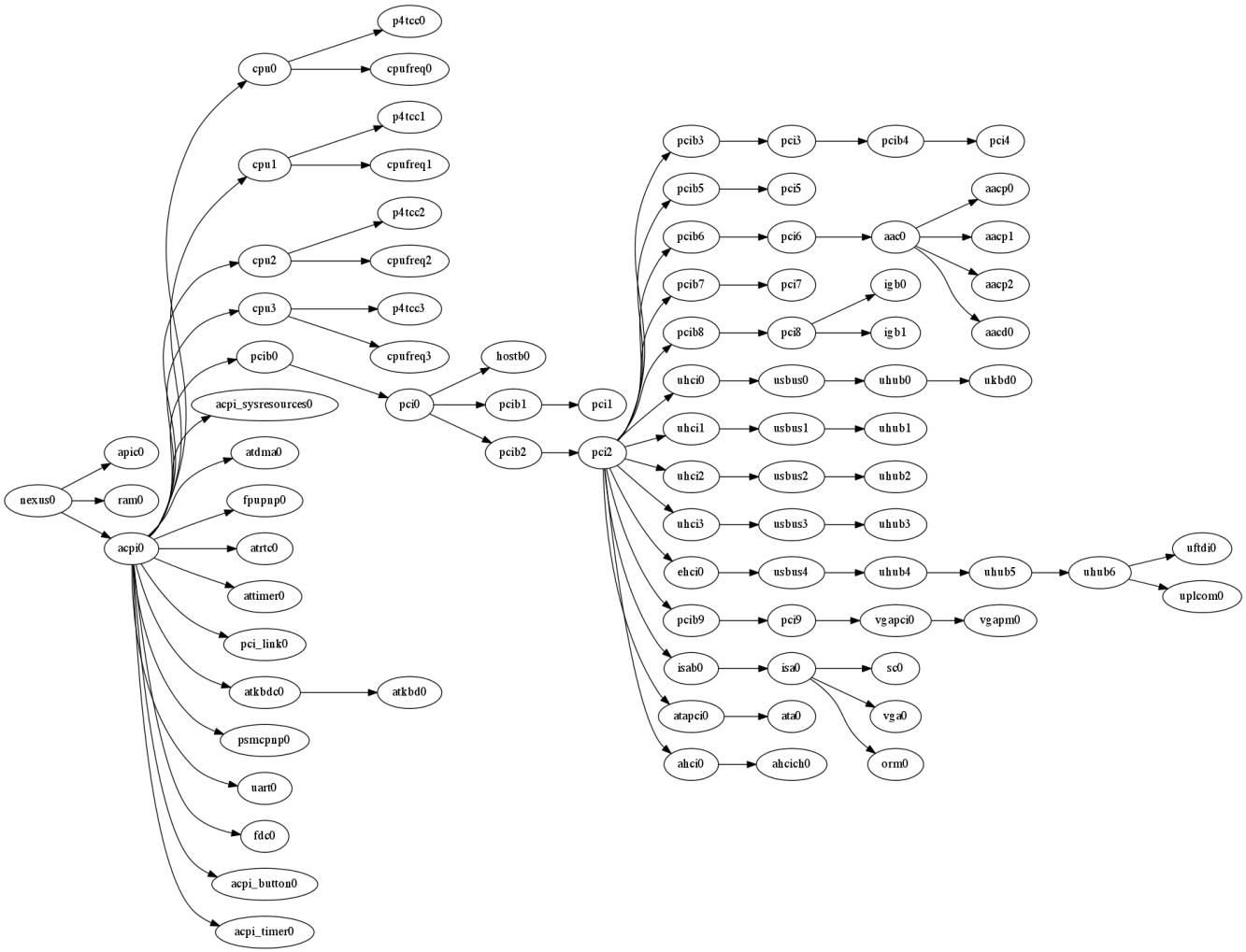
NEWBUS

- FreeBSD's Driver Configuration Mechanism
- Extensible Interface
- Named Function Dispatch (kobj)
- Name Space Management (devclass_t and device_t)
- Polymorphic Attachment (bus specialized drivers)
- Resource Management and Reporting (devinfo)
- bus_space(9) and busdma(9) integration
- Interrupt Integration
- Hierarchical Resource Allocation

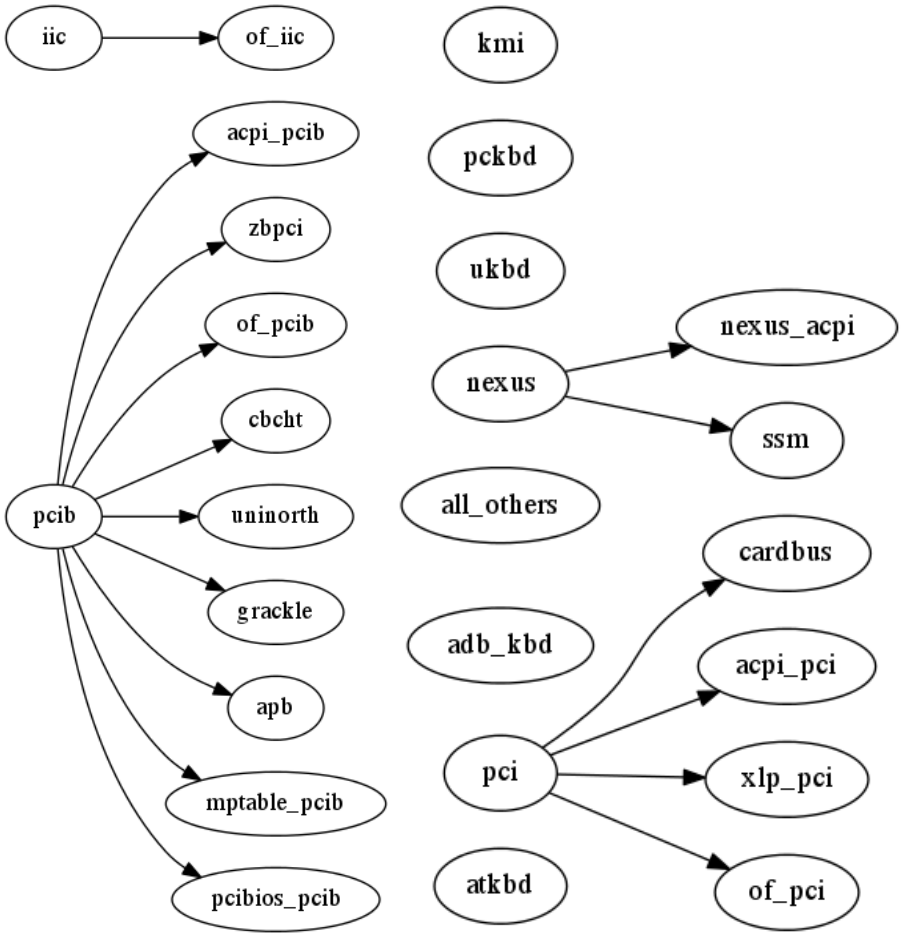
NEWBUS Summary

- Provides method calls (via foo_if.m)
- Method calls bind at runtime
- Design pattern for data (ivars)
- Method Inheritance (via code)
- Data Inheritance (via ivars design pattern)
- Hierarchy Traversal (design pattern)

Typical Device Tree



"Direct" Class Hierarchy



Type Safety

[4]

Unsafe:

```
int foo_probe(device_t, int);
device_method_t foo_methods[] = {
DEVMETHOD(device_probe, foo_probe),
```

Less Unsafe:

```
device_probe_t foo_probe;
device_method_t foo_methods[] = {
DEVMETHOD(device_probe, foo_probe),
```

[4] http://failblog.cheezburger.com/parenting

Type Safety

- Most functions declared w/o interface_method_t.
- DEVMETHOD doesn't cast for safety
- Easy to mix up function in DEVMETHOD
- Most safety happens because of naming conventions and simplicity
- Casting fixes possible, need testing

```
Index: sys/sys/kobj.h
===================================================================
--- sys/sys/kobj.h(revision 250733)
+++ sys/sys/kobj.h(working copy)
@@ -95,7 +95,7 @@
  * has a signature that is not compatible with kobj method signature.
  */
 #define KOBJMETHOD(NAME, FUNC) \
-{ &NAME##_desc, (kobjop_t) (1 ? FUNC : (NAME##_t *)NULL) }
+{ &NAME##_desc, (kobjop_t) (1 ? (NAME##_t *)FUNC : (NAME##_t *)NULL) }

 /*
  *
```

NEWBUS OO Matchup

- Dynamic Dispatch - Yes (_if.m)
- Encapsulation - Yes (softc hiding and ivars externalization)
- Subtype Polymorphism - Methods only
- Object Delegation / Inheritence - Yes
- Open Recursion - Not really
- Classes of Object - Possible
- Methods which act as Objects - No
- Message Passing - Yes
- Abstraction - Some
- Type Safety - No

Types of Subclassing / Inheritance

- Direct Inheritance
- Interface
- Data

Direct Inheritance

- Done via the DEFINE_CLASS_x macro
- Inherits all interfaces of parents
- Inherited busses allow subclass attachments
  - All PCI drivers could attach to CardBus bus.
- Rare in our tree: only pci, pcib, cardbus, iic and nexus
- Except for CardBus, used only to specialize quirks of bus

Interface Inheritance

- Defined in the foo_methods array
- Best thought of as an interface protocol
- Can implement all or part of an interface
- Up to device node to implement interface protocol properly
- Very common for nodes to implement many interfaces
- Can be hard to change protocols

Data Inheritance

- Mostly outside the scope of NEWBUS
- IVARS can be used for direct inheritance
  - Not really the same thing
  - More of "property" inheritance
- Fall back to structure nesting in C

Quick Example

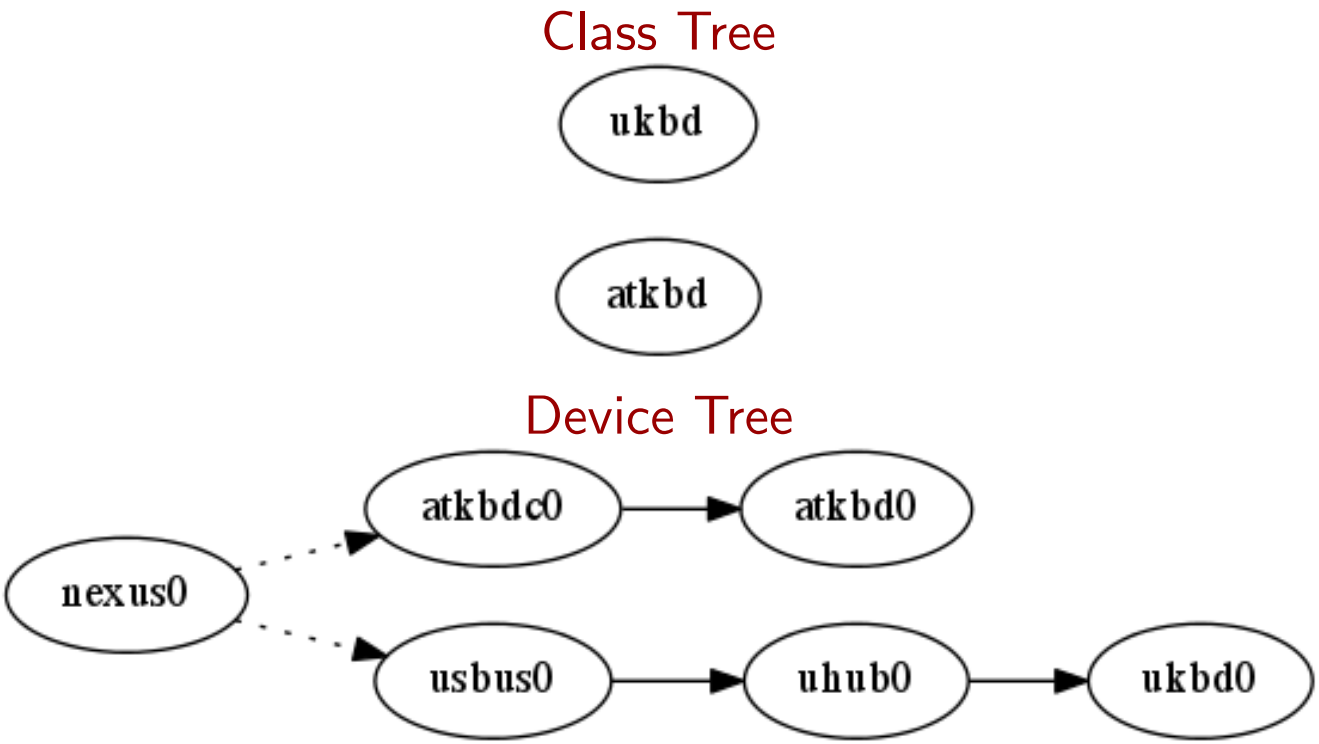
```
static device_method_t ep_pccard_methods[] = {
        /* Device interface */
        DEVMETHOD(device_probe, ep_pccard_probe),
        DEVMETHOD(device_attach, ep_pccard_attach),
        DEVMETHOD(device_detach, ep_detach),

        DEVMETHOD_END
};

static driver_t ep_pccard_driver = {
        "ep",
        ep_pccard_methods,
        sizeof(struct ep_softc),
};

extern devclass_t ep_devclass;

DRIVER_MODULE(ep, pccard, ep_pccard_driver, ep_devclass, 0, 0);
```

Final Points

- Multiple DRIVER_MODULESs can have the same name
- Interfaces define a protocol between bits
- Drivers can subclass other drivers, but only with coordination
- Sometimes the same name is used for attachment points on different archs
- Sometimes long lists are used
- Some of the long lists should be done with base classes

Outline

Keyboard Topology Today



Problems with Keyboards

- Which device provides kbd2?
- No information in devinfo
- Must grep dmesg - unreliable

```
% dmesg | grep kbd
kbd1 at kbdmux0
atkbdc0: <Keyboard controller (i8042)> port 0x60,0x64 irq 1 on acpi0
atkbd0: <AT Keyboard> irq 1 on atkbdc0
kbd0 at atkbd0
atkbd0: [GIANT-LOCKED]
ukbd0: <EP1 Interrupt> on usbus0
kbd2 at ukbd0
```

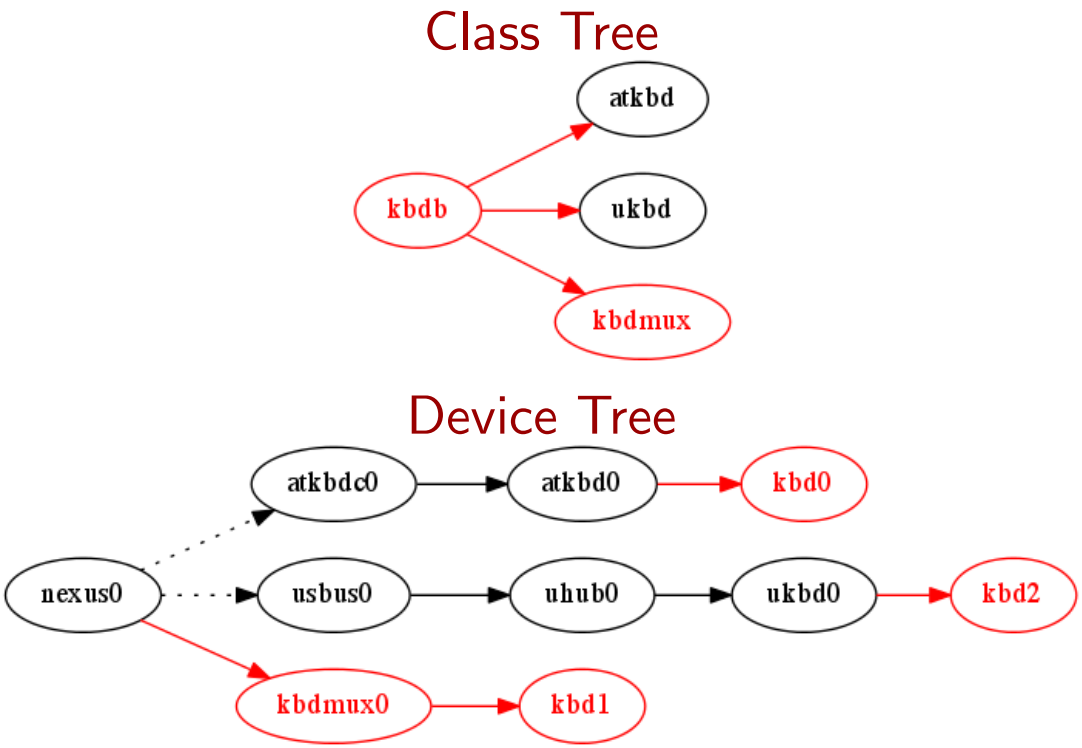
GRAVES

cause sometimes you gotta dig your own

Proposed Solution

- Create new kbdmux NEWBUS devclass_t
- Create new kbd NEWBUS devclass_t
- Create new kbdb base class
- Derive all keyboard classes in tree from kbdb: pckbd, kmi, atkbd, ukbd, adb_kbd
- Change kbdmux code to attach kbdmux0 to nexus0
- Change kbd attach code to also attach a NEWBUS kbd instance too

Proposed Solution Picture



Solution Changes

For each keyboard driver: pckbd, kmi, atkbd, ukbd, uart_ebus, and adb_kbd; make a change like:

```
Index: pc98/cbus/pckbd.c
===================================================================
--- pc98/cbus/pckbd.c (revision 249553)
+++ pc98/cbus/pckbd.c (working copy)
@@ -67,12 +67,7 @@
        { 0, 0 }
 };

-static driver_t pckbd_driver = {
-       DRIVER_NAME,
-       pckbd_methods,
-       1,
-};
-
+DEFINE_CLASS_1(pckbd, pckbd_driver, pckbd_methods, 1, kbdb_driver);
 DRIVER_MODULE(pckbd, isa, pckbd_driver, pckbd_devclass, 0, 0);

 static bus_addr_t pckbd_iat[] = {0, 2};
@@ -209,7 +204,7 @@

 #ifdef KBD_INSTALL_CDEV
        /* attach a virtual keyboard cdev */
-       error = kbd_attach(*kbd);
+       error = kbd_attach_device(*kbd, dev);
        if (error)
                return error;
 #endif /* KBD_INSTALL_CDEV */
```

Solution Changes

Create a kbdb base class to use for all devices that will be attaching a kbd to.

```
/*
 * device_t baseclass for all devices that wish to attach a keyboard
 * to them. This will allow us to track the connection between keyboard
 * unit numbers and their underlying device, if any.
 */

static int kbdb_print_child(device_t kbdc, device_t kbd)
{
    int retval = 0;

    retval += bus_print_child_header(kbdc, kbd);
    retval += bus_print_child_footer(kbdc, kbd);

    return (retval);
}

static device_method_t kbdb_methods[] =
{
        /* Bus interface */
        DEVMETHOD(bus_print_child, kbdb_print_child),

        DEVMETHOD_END
};

DEFINE_CLASS_0(kbdb, kbdb_driver, kbdb_methods, 1);
```

Solution Changes

Create a kbd driver.

```
/*
 * device_t for a simple keyboard node in the device tree.  This node
 * is attached to the real device (rather than trying to force the real
 * device to all have the same name). The unit number matches the unit
 * number in the dev tree.
 */

static device_method_t kbd_methods[] =
{
        /* Device Interface */
        DEVMETHOD(device_probe, bus_generic_probe),
        DEVMETHOD(device_attach, bus_generic_attach),
        DEVMETHOD(device_detach, bus_generic_detach),

        DEVMETHOD_END
};

DEFINE_CLASS_0(kbd, kbd_driver, kbd_methods, 1);
```

Solution Changes

Implement kdb_attach_device and kdb_attach_device.

```
int kbd_attach_device(keyboard_t *kbd, device_t dev)
{
        int retval;
        device_t child;

        retval = kbd_attach(kbd);
        if (retval != 0)
                return retval;
        child = device_add_child(dev, "kbd", kbd->kb_index);
        if (child == NULL)
                return ENOMEM;
        retval = device_probe_and_attach(child);
        if (retval != 0)
                return retval;
        kbd->kb_device = dev;

        return 0;
}

int kbd_detach_device(keyboard_t *kbd, device_t dev)
{
        int retval;

        retval = kbd_detach(kbd);
        if (retval != 0)
                return retval;
        device_detach(kbd->kb_device);
        kbd->kb_device = NULL;

        return 0;
}
```

TTY Mess

- No baseclass
- No common naming in /dev
- No way to trace back USB devices to ttyUx entries
- USB maps multiport cards to ttyUx.y, which has no representation in NEWBUS
- Way more drivers in tree (especially usb)

[5]

[5] http://failblog.cheezburger.com/thereifixedit

Proposed Solution

- Create new tty NEWBUS devclass_t
- Create new ttyb base class
- Derive all tty classes in tree from ttyb
- Connect N tty ports to classes derived from ttyb
- Add device=XXXX to location in ttyb bus for actual /dev/tty name
  - So devinfo -v can be used
- Add wrapper routines to make this easy, same as for kbd

Proposed Solution

Before

```
before% devinfo -v
uhub6 pnpinfo ... at ...
  uftdi0 pnpinfo ... at ...
  uplcom0 pnpinfo ... at ...
uart0 pnpinfo ... at ...
uart1 pnpinfo ... at ...
```

After

```
after% devinfo -v
uhub6 pnpinfo ... at ...
  uftdi0 pnpinfo ... at ...
    tty0 at device=ttyU0.0
    tty1 at device=ttyU0.1
  uplcom0 pnpinfo ... at ...
    tty2 at device=ttyU1
uart0 pnpinfo ... at ...
  tty3 at device=ttyu0
uart1 pnpinfo ... at ...
  tty4 at device=ttyu1
```

Alternate Solution

- Just use ucom as the name for all usb serial devices
- The unit number assignment will sort it out
- Breaks module to newbus name
- Hard to know what driver to use

Another Alternate Solution

- Change the tty name to /dev/newbus.tty and /dev/newbus.cua
- Use the newbus name everywhere
- Breaks users' programs assumptions about names
- Yet another name change

miibus problems

- No baseclass
- Every driver defines attachment
- With proper baseclassing, this problem goes away
- Every driver has an attachment line

Solution Changes

Derive all network drivers from miib, similar to below

```
Index: if_dc.c
===================================================================
--- if_dc.c(revision 250733)
+++ if_dc.c(working copy)
@@ -346,17 +346,10 @@
 DEVMETHOD_END
 };

-static driver_t dc_driver = {
-       "dc",
-       dc_methods,
-       sizeof(struct dc_softc)
-};
-
+DEFINE_CLASS_1(dc, dc_driver, dc_methods, sizeof(struct dc_softc),
+    miib_driver);
 static devclass_t dc_devclass;
-
 DRIVER_MODULE_ORDERED(dc, pci, dc_driver, dc_devclass, NULL, NULL,
     SI_ORDER_ANY);
-DRIVER_MODULE(miibus, dc, miibus_driver, miibus_devclass, NULL, NULL);
```

miibus Solution

- Shorter
- Intent better understood
- Not much gain over what we have today

mmc bus problems

- mmcbus uses a list of doom for what bridges to use.
- With proper baseclassing, this problem goes away.

MULTICLASSING

Everyones Doing it

mmc bus problems

```
/* Today's list in mmc.c */

DRIVER_MODULE(mmc, ti_mmchs, mmc_driver, mmc_devclass, NULL, NULL);
DRIVER_MODULE(mmc, at91_mci, mmc_driver, mmc_devclass, NULL, NULL);
DRIVER_MODULE(mmc, sdhci_pci, mmc_driver, mmc_devclass, NULL, NULL);
DRIVER_MODULE(mmc, sdhci_bcm, mmc_driver, mmc_devclass, NULL, NULL);
DRIVER_MODULE(mmc, sdhci_fdt, mmc_driver, mmc_devclass, NULL, NULL);

/* Becomes */

DRIVER_MODULE(mmc, mmcb, mmc_driver, mmc_devclass, NULL, NULL);
```

sdhci busspace

- Uses a kobj interface for something that looks like a bus space
- Should just move to bus space.
- PCI uses bus_barrier, but no others don't
- bcm attachment does endian hacking, which is the job of bus space

sdhci problems

```
static device_method_t sdhci_methods[] = {
...
        /* SDHCI registers accessors */
        DEVMETHOD(sdhci_read_1,         sdhci_pci_read_1),
        DEVMETHOD(sdhci_read_2,         sdhci_pci_read_2),
        DEVMETHOD(sdhci_read_4,         sdhci_pci_read_4),
        DEVMETHOD(sdhci_read_multi_4,   sdhci_pci_read_multi_4),
        DEVMETHOD(sdhci_write_1,        sdhci_pci_write_1),
        DEVMETHOD(sdhci_write_2,        sdhci_pci_write_2),
        DEVMETHOD(sdhci_write_4,        sdhci_pci_write_4),
        DEVMETHOD(sdhci_write_multi_4,  sdhci_pci_write_multi_4),
...
};
```

sdhci problems

```
static uint8_t
sdhci_pci_read_1(device_t dev, struct sdhci_slot *slot, bus_size_t off)
{
        struct sdhci_pci_softc *sc = device_get_softc(dev);
        bus_barrier(sc->mem_res[slot->num], 0, 0xFF,
            BUS_SPACE_BARRIER_READ | BUS_SPACE_BARRIER_WRITE);
        return bus_read_1(sc->mem_res[slot->num], off);
}
static uint8_t
sdhci_fdt_read_1(device_t dev, struct sdhci_slot *slot, bus_size_t off)
{
        struct sdhci_fdt_softc *sc = device_get_softc(dev);
        return (bus_read_1(sc->mem_res[slot->num], off));
}
static uint8_t
bcm_sdhci_read_1(device_t dev, struct sdhci_slot *slot, bus_size_t off)
{
        struct bcm_sdhci_softc *sc = device_get_softc(dev);
        uint32_t val = RD4(sc, off & ~3);
        return ((val >> (off & 3)*8) & 0xff);
}
```


WHAT'S WRONG
WITH
THIS PICTURE?

Head scratching

- Bus barriers are needed when you write descriptor rings into device's resource
- Typically used just before write that turns over dtor to hardware
- Why does every access need a barrier (and not just the last one?)
- Why does PCI need it and not FDT or BCM?
- Why isn't BCM doing right endian swapping in its bus space?
- Why do the stream methods need a barrier?

Questions? Comments?

Warner Losh
imp@FreeBSD.org
http://people.freebsd.org/~imp/bsdcan2013-slides.pdf