Device Tree Plumbers 2015 Dynamic DT and tools Pantelis Antoniou < pantelis.antoniou@konsulko.com >

Device Tree Overlays overview and use cases

- + Device Tree Overlays are now in the mainline kernel. This session will cover what they are, how they are used.
- + Device tree overlays
- + Device tree changeset
- + The phandle resolution mechanism
- + Overlay overlap removal checks
- + Device tree variants (or quirks).

Overlays Describe Hardware

- + Hardware may not be static; not known at boot time.
- + Capes, Hats, Expansion boards
- + FPGAs
- + Weird topology/device requirements
- + Or hardware is static, but using overlays is easier to manage. I 0s of board variants, would require a different DTB for each. Hard to do in the bootloader. Easier just to use an overlay.
- + Useful even on busses that can be probed. I2C devices on a PCI/ USB host bus device.

CONFIG_OF_DYNAMIC

- + Allows modification of the Live Device Tree at runtime.
- → Not very widely used until now only on Power.
- + Destructive editing of the live tree
 - + Non atomic
 - + Changes cannot be reverted
- + No connection to the bus driver model; changes to the live tree do not get reflected.
- + Part of the puzzle, but not enough as it was.

Part I: Reworking OF_DYNAMIC

- + /proc → /sys (gcl)
- + struct device_node now a kobj (gcl)
- + drivers/of/dynamic.c
- + Semantics of the of_reconfig notifiers have changed.
- + Major new user is dt selftests. Test case data dynamically inserted.
- + Already accepted in mainline (3.17)

Part 2: Dynamic Resolution (foo.dts)

Dynamic Resolution (qux.dts)

Resolving phandles

- + Phandles are pointers to other parts in the tree. For example pinmuxing, interrupt-parent etc.
- + Phandles are internally represented by a single 32 scalar value and are assigned by the DTC compiler when compiling
- + Extension to the DTC compiler required, patchset already in v2, minor rework is required.
- + "dtc: Dynamic symbols & fixup support (v2)"

Changes made to the DT Compiler

+ ABSOLUTELY NO CHANGES TO THE DTB FORMAT.

- + -@ command line option global enable.
- + Generates extra nodes in the root (__symbols__, __fixups__, __local_fixups__) containing resolution data.
- + /plugin/ marks a device tree fragment/object (controls generation of __fixups__ and __local_fixups__ nodes).
- + To perform resolution the base tree needs to be compiled using the -@ option and causes generation of __symbols__ node only.

Compiling foo.dts (base tree)

```
$ dtc -0 dtb -o foo.dtb -b 0 -@ foo.dts && fdtdump foo.dtb
/ {
    bar = <0 \times 000000001>;
    foo {
         linux, phandle = <0x00000001>;
         phandle = <0 \times 000000001>;
    };
    __symbols__ {
         F00 = "/foo";
    };
};
```

Compiling qux.dts (object)

```
$ dtc -0 dtb -o qux.dtbo -b 0 -@ qux.dts && fdtdump qux.dtbo
/ {
    qux = <0x00000001>;
    quux = <0xdeadbeef>;
    baz {
        linux, phandle = <0x000000001>;
        phandle = <0 \times 000000001>;
    };
    __symbols__ { BAZ = "/baz"; };
    __fixups__ { F00 = "/:quux:0"; };
    __local_fixups__ { fixup = "/:qux:0"; };
};
```

How the resolver works

- + Get the max device tree phandle value from the live tree + 1.
- + Adjust all the local phandles of the tree to resolve by that amount.
- + Using the __local__fixups__ node information adjust all local references by the same amount.
- + For each property in the ___fixups__ node locate the node it references in the live tree. This is the label used to tag the node.
- + Retrieve the phandle of the target of the fixup.
- + For each fixup in the property locate the node:property:offset location and replace it with the phandle value.

Part 3: Changesets/Transactions

- + A Device Tree changeset is a method which allows us to apply a set of changes to the live tree.
- + Either the full set of changes apply or none at all.
- + Only after a changeset is applied notifiers are fired; that way the receivers only see coherent live tree states.
- + A changeset can be reverted at any time.
- + Part of mainline as of 3.17.

Changesets in kernel API

- + Issue of_changeset_init() to prepare the changeset.
- + Perform your changes using of_changeset_
 {attach_node|detach_node|add_property|
 remove_property|update_property}()
- + Lock the tree by taking the of_mutex;
- + Apply the changeset using of_changeset_apply();
- + Unlock the tree by releasing of_mutex.
- + To revert everything of_changeset_revert();

Changesets helpers

- + Using changesets manually is a chore.
- + "of: changesets: Introduce changeset helper methods"
- + Dynamically allocates memory; to wit instead of using the raw API,

```
struct property *prop;
prop = kzalloc(sizeof(*prop)), GFP_KERNEL);
prop->name = kstrdup("compatible");
prop->value = kstrdup("foo,bar");
prop->length = strlen(prop->value) + 1;
of_changeset_add_property(ocs, np, prop);
```

+ While using the helper API
of_changeset_add_property_string(ocs, np, "compatible", "foo,bar");

Device Tree Overlay format

```
/plugin/;
/ {
    /* set of per-platform overlay manager properties */
    fragment@0 {
        target = <&target-label>; /* or target-path */
        __overlay__ {
            /* contents of the overlay */
        };
    };
    fragment@1 {
        /* second overlay fragment... */
    };
};
```

Device Tree Overlay in kernel API

- + Get your device tree overlay blob in memory using a call to request_firmware() call, or linking with the blob is fine.
- + Use of_fdt_unflatten_tree() to convert to live tree format.
- + Call of_resolve_phandles() to perform resolution.
- + Call of_overlay_create() to create & apply the overlay.
- + Call of_overlay_destroy() to remove and destroy the overlay. Note that removing overlapping overlays must be removed in reverse sequence.

New functionality in the pipeline

- + The target is a fixed point in the base device tree. Problematic if you have plan to connect the same hardware device to different slots.
- + Indirect targets solve this by having a re-direction method.
- + Posted a patch but Guenter's posted a better one reworked:)

Overlays, some times a good idea.

- + Overlays are powerful. Sometimes too powerful.
- + Good uses:
 - + Pluggable expansion boards with an identifying method.
 - + Hardware hackers testing designs
 - + FPGAs
 - + Anything that is a result of an action that changes the hardware topology (i.e. DRM monitor connections)

Overlays sometimes a bad idea.

- + Static changes to a board revision can be expressed via an Overlay, but it's late in the boot sequence. Early stuff (like regulators and clocks) the changes cannot affect those. Better to use a quirk (or variant)
- + Generating device tree nodes and properties automatically. I.e. PCI/USB device node generation (either firmware assisted or not). Changesets is the way to go.
- + General rule: if the resulting change in the kernel tree requires smarts, it's best to create everything via changesets.

Overlays and tools for sanity.

- + Device Tree overlays represent a big change for the device tree in the kernel. Where as of old the device tree was something static; now it's something that can change at runtime.
- + We could use some new tools to help us when creating them (compile time) and some kernel tooling to help when applying them (run time).

Compile time overlay tooling

- + Right now the changes to DTC are minimal.
- + Overlay is compiled without a reference to the base DTS.
- + Need an option to compile against a base DTS to validate that the overlay will load.
- + For testing purposes a method to generate at compile time the DTS resulting from an application of an overlay.
- + New APIs are even more demanding for example portable connector based overlays will need property matching.
- + DT diff? Generate an overlay to patch DTBs.

Compile time overlay tooling

- + Device Tree overlays represent a big change for the device tree in the kernel. Where as of old the device tree was something static; now it's something that can change at runtime.
- + We could use some new tools to help us when creating them (compile time) and some kernel tooling to help when applying them (run time).
- + Frank's NOTE:
 - + Overlays tools needed: generating, test, validation
- + From Rob's email comments:
 - + How to test an overlay applies?
 - + Generating a dtb from dts + overlay dts.
 - + Generating an overlay from a diff of old and new dts (overlay as a way to update old dtbs)

Runtime time overlay tooling

- + Not just an overlay problem. There is no acceptable type information for properties.
- + That means that one could modify the kernel live tree with properties that make no sense.
- + How to carry type information (and perform checks).
- + of_reconfig notifiers could be used, but doing it manually is madness.
- + Need to store the type information in the DT itself.

Device Tree probe order and parallel device probing - Pantelis

- + Making the phandle resolver to work means that phandles and the location where they are references are tracked.
- + Makes it possible to track dependencies of one subtree to another.
 - + Device references a DMA channel? That device is dependent on the DMA controller driver.
 - + We can create a schedule of device probes.
 - + Trivially we can create a parallel schedule of device probes.

Why probe order is a problem?

- + Not all drivers handle correctly EPROBE_DEFER.
- + Excessive defers slow down kernel boot.
- + People pepper the kernel with subsys_init() calls to force ordering.
- + Device tree dependency tracking not the first time attempted.
- + Deferred probe patches are floating around.

Driver core changes request?

- + The order of probe calls is not the order of calling device_create(). It is actually much later when the driver is matched to a device.
- + Making all this work for device tree is OK, but we need to handle other methods (yay for x86).
- + Device core should track dependencies and probe order, backend should fill it in.

Thank you for listening

Devicetree Overlay use at Juniper Networks

Guenter Roeck groeck@juniper.net

System Overview

- PTX5000 Packet Transport Router
 - Routing Engine
 - Routing protocols, administrative tasks
 - Interfaces to other cards in the system
 - 8 x FPC (Flexible PIC Concentrator)
 - 2 x PIC per FPC
 - Control Board
 - 9 x SIB (Switch Interface Board) per CB
 - All cards identified using I2C EEPROMs
 - Card connectors use multiple interface types
 - I2C, GPIO, PCIe, SERDES, ...
 - Various CPU types
 - P2020, P5020, P5040, x86

Devicetree overlay use

- All OIR capable cards managed with devicetree overlays
 - RE
 - FPCs, Fan tray, power supply, ...
 - FPC
 - PICs
 - Control Board
 - SIBs
- Each card represented as 'connector' node in devicetree data

'connector' nodes

```
pic0 {
       compatible = "jnx,pic-connector", "simple-bus";
       slot = <0>:
       auto-enable:
       ovname = "jnx pic0", "jnx pic0 pwr";
       presence-detect-gpios = <&gpio20 148 0x1>; /* active low */
       attention-button-gpios = <&gpio20 150 0x1>; /* active low */
       power-enable-gpios = <&gpio20 154 0x0>; /* active high */
       power-status-gpios = <&gpio20 151 0x0>; /* active high */
       reset-gpios = <&gpio20 153 0x1>;
                                               /* active low */
       power-enable-timeout = <2000>;
                                               /* in ms */
       attention-button-holdtime = <3000>;
                                               /* in ms */
                                            /* in ms */
       activation-timeout = <5000>:
       debounce-interval = <1>;
       led-green = <&pic0 green>;
       led-red = <&pic0 red>;
       i2c-bus {
         #address-cells = <1>:
         #size-cells = <0>;
         i2c-parent = <&pic0i2c>;
         eeprom@54 {
            compatible = "atmel,24c02":
            req = <0x54>:
            ideeprom;
     };
};
```

Connector driver

Functionality

- Manages card insertion and removal
- Responsible for loading and removing devicetree overlays
- State machine with 10 states and 12 events

Status

- Reliably loads and removes overlays
- Some limitations and concerns

Limitations

- Power management
 - After enabling power, chips may be immediately visible on bus
 - PCIe: hotplug driver attempts to load driver before overlay is loaded
 - Kind of solved by using layered overlays
 - First overlay inserted after card identified, prior to enabling power
 - Second overlay inserted after power enabled and stable

Limitations

- Indirect target support
 - Currently requires information within overlay for each slot
 - Problematic if card is re-used in a different chassis
 - Limited scalability
 - Proposal: Simplify API by providing reference(s) from calling code
 - of_overlay_indirect() gets reference(s) instead of slot number as parameter

Limitations

- No DT / DT Overlay support on x86
 - Mandatory for us
 - Other solutions either not feasible or not scalable
 - ACPI
 - Not supported on all architectures
 - No overlays
 - Platform data is clumsy
 - Requires new driver / code for each new card
 - Card management from user space does not work
 - Yes, we tried ...
 - Implemented and working with small patch set on top of upstream kernel