Linux Matchmaking

Helping devices and drivers find each other

FOSDEM 2024 - Kernel Devroom

Ahmad Fatoum – a.fatoum@pengutronix.de
About Me

Ahmad Fatoum
Pengutronix e.K.
a3f
a.fatoum@pengutronix.de

- Kernel and Bootloader Porting
- Driver and Graphics Development
- System Integration
- Embedded Linux Consulting
Tale of a kernel update

- `git rebase --onto (or `umpf build`)
- `make oldconfig`
- `make`
- Deploy the updated kernel artifacts
Tale of a kernel update

- `git rebase --onto (or umpf build)`
- `make oldconfig`
- `make`
- `Deploy the updated kernel artifacts`
- `...`
- `System hangs at boot`
  - If you have a known good → `git bisect`
  - If you don't → need to debug
Example Breakage

- After updating and running `make olddefconfig` (or `menuconfig`), system no longer boots
  - Culprit commit: c20e8c5b1203 ("mfd: rk808: Split into core and i2c")

```c
-config MFD_RK808
+config MFD_RK8XX
+     bool
+     select MFD_CORE
+config MFD_RK8XX_I2C
     tri-state "Rockchip RK805/RK808/RK809/RK817/RK818 Power Management Chip"
+     select MFD_RK8XX
```

- RK8xx is the system's power management IC
  - Not much the kernel drivers are willing to do without it being available...
Resolving missing dependencies

- Resolution is trivial once you find the issue, but how do we get there without comparing to a known good and no errors?
  - Need to retrace driver attempts at initializing the devices
Device/Driver Model

- struct **bus_type**

- struct **device_driver**

- struct **device**
Device/Driver Model

- struct bus_type
  - int (*match)(struct device *, struct device_driver *)

- struct device_driver
  - int (*probe)(struct device *)
  - int (*remove)(struct device *);

- struct device
Device/Driver Model: Matching

```c
struct pci_dev {
    unsigned short vendor;
    unsigned short device;
    struct device dev;
    struct resource resource[DEVICE_COUNT_RESOURCE];
};

struct pci_driver {
    const struct pci_device_id *id_table;
    int (*probe)(struct pci_dev *dev, const struct pci_device_id *id);
    void (*remove)(struct pci_dev *dev);
    int (*sriov_configure)(struct pci_dev *dev, int num_vfs);
    struct device_driver driver;
};

struct pci_device_id {
    __u32 vendor, device;          /* Vendor and device ID or PCI_ANY_ID*/
    kernel_ulong_t driver_data;    /* Data private to the driver */
};
```
static int platform_match(struct device *dev, struct device_driver *drv)
{
    struct platform_device *pdev = to_platform_device(dev);
    struct platform_driver *pdrv = to_platform_driver(drv);

    /* When driver_override is set, only bind to the matching driver */
    if (pdev->driver_override)
        return !strcmp(pdev->driver_override, drv->name);

    /* Attempt an OF style match first */
    if (of_driver_match_device(dev, drv))
        return 1;

    /* Then try ACPI style match */
    if (acpi_driver_match_device(dev, drv))
        return 1;

    /* Then try to match against the id table */
    if (pdrv->id_table)
        return platform_match_id(pdrv->id_table, pdev) != NULL;

    /* fall-back to driver name match */
    return (strcmp(pdev->name, drv->name) == 0);
}

{/soc} {
    compatible = "simple-bus";

    timer@fffec600 {
        compatible = "arm,cortex-a9-twd-timer";
        reg = <0xfffec600 0x100>;
        interrupts = <1 13 0xf01>;
        clocks = <&mpu_periph_clk>;
    }
};
Once a match is found, a device will be *probed*:
The driver checks device existence, requests resources and registers with relevant kernel API (e.g. `register_netdev`)

- Driver is happy
  → return 0 and device is bound

- Driver doesn't recognize the device
  → return -ENODEV or -ENXIO to silently skip match

- Driver encounters an error and needs to cleanup
  → return any other (negative) error code
Device Dependencies

- Devices often need resources exported by other devices:
  - Generic
    - pinctrl ("init", "default" states)
    - DMA configuration
  - Bus-specific. For platform devices:
    - Clock assignment: Initial assignment of parents and rates
    - Single Power domain power-up
  - Device-specific
    - Clocks, Multiple Powerdomains, GPIOs, Resets, PHYs, ... and a whole lot more
Device Dependencies

- What if device exporting resource is unavailable?
  - **Avoidance:**
    - initcalls
    - Makefile ordering
# Many drivers will want to use DMA so this has to be made available # really early.
obj-$(CONFIG_DMADEVICES) += dma/

# SOC specific infrastructure drivers.
obj-y += soc/
obj-$(CONFIG_PM_GENERIC_DOMAINS) += pmdomain/

# regulators early, since some subsystems rely on them to initialize
obj-$(CONFIG_REGULATOR) += regulator/
Reordering Driver Registration

```c
#define core_initcall(fn) __define_initcall(fn, 1)
#define core_initcall_sync(fn) __define_initcall(fn, 1s)
#define postcore_initcall(fn) __define_initcall(fn, 2)
#define postcore_initcall_sync(fn) __define_initcall(fn, 2s)
#define arch_initcall(fn) __define_initcall(fn, 3)
#define arch_initcall_sync(fn) __define_initcall(fn, 3s)
#define subsys_initcall(fn) __define_initcall(fn, 4)
#define subsys_initcall_sync(fn) __define_initcall(fn, 4s)
#define fs_initcall(fn) __define_initcall(fn, 5)
#define fs_initcall_sync(fn) __define_initcall(fn, 5s)
#define rootfs_initcall(fn) __define_initcall(fn, rootfs)
#define device_initcall(fn) __define_initcall(fn, 6)
#define device_initcall_sync(fn) __define_initcall(fn, 6s)
#define late_initcall(fn) __define_initcall(fn, 7)
#define late_initcall_sync(fn) __define_initcall(fn, 7s)
```

# Many drivers will want to use DMA so this has to be made available
# really early.
obj-$(CONFIG_DMADEVICES) += dma/

# SOC specific infrastructure drivers.
obj-y += soc/
obj-$(CONFIG_PM_GENERIC_DOMAINS) += pmdomain/

# regulators early, since some subsystems rely on them to initialize
obj-$(CONFIG_REGULATOR) += regulator/

mfg: power-domain@MT8183_POWER_DOMAIN_MFG {
  domain-supply = <&mt6358_vgpu_reg>;
  #power-domain-cells = <1>;
};
Device Dependencies

- What if device exporting resource is unavailable?
  - Avoidance:
    - initcalls
    - Makefile ordering
  - Detection
    - `-EPROBE_DEFER`
Probe Deferral

- Driver signals missing dependency by returning -EPROBE_DEFER (== -517)
- Device probe will be cleaned up as if probe had failed
- Probe is retried at a later time after other devices have been probed

```c
icc_path = devm_of_icc_get(&pdev->dev, "usb1");
if (IS_ERR(icc_path)) {
    if (PTR_ERR(icc_path) != -EPROBE_DEFER)
        dev_err(dev, "couldn't get interconnect: %pe\n", icc_path);
    return PTR_ERR(icc_path);
}
```
**Probe Deferral**

- Probe deferrals are listed in `sysfs`

```bash
root@DistroKit:~ cat /sys/kernel/debug/devices_deferred
32f10100.usb
32f10108.usb
381f0040.usb-phy
382f0040.usb-phy
32ec0000.blk-ctrl
38330000.blk-ctrl
32f10000.blk-ctrl
```

- ... and also printed after `deferred_probe_timeout` (10 if `CONFIG_MODULES=y`)

```bash
[ 16.611797] platform 32f10100.usb: deferred probe pending
[ 16.621405] platform 32f10108.usb: deferred probe pending
[ 16.631022] platform 381f0040.usb-phy: deferred probe pending
[ 16.640967] platform 382f0040.usb-phy: deferred probe pending
[ 16.650899] platform 32ec0000.blk-ctrl: deferred probe pending
[ 16.660483] platform 38330000.blk-ctrl: deferred probe pending
[ 16.670102] platform 32f10000.blk-ctrl: deferred probe pending
```

- If only there was a reason for why the probe was deferred...
```c
if (IS_ERR(icc_path)) {
    if (PTR_ERR(icc_path) != -EPROBE_DEFER)
        dev_err(dev, "couldn't get interconnect: %pe\n", icc_path);
    return PTR_ERR(icc_path);
}

if (IS_ERR(icc_path))
    return dev_err_probe(dev, PTR_ERR(icc_path), "failed to get interconnect\n");
```

Benefits:
- compact
- Includes error code
- Stores probe error reason message into `struct device`
### Probe Deferral

- SysFS also lists probe deferral reasons if provided

```bash
root@DistroKit:~ cat /sys/kernel/debug/devices_deferred
32f10100.usb   platform: supplier 32f10000.blk-ctrl not ready
32f10108.usb   platform: supplier 32f10000.blk-ctrl not ready
381f0040.usb-phy platform: supplier 32f10000.blk-ctrl not ready
382f0040.usb-phy platform: supplier 32f10000.blk-ctrl not ready
32ec0000.blk-ctrl  imx8m-blk-ctrl: failed to get noc entries
38330000.blk-ctrl  imx8m-blk-ctrl: failed to get noc entries
32f10000.blk-ctrl  imx8mp-blk-ctrl: failed to get noc entries
```

... and since v6.8-rc1, on probe deferall timeout too:

```bash
```
Tracing

- earlycon
- ignore_loglevel initcall_debug dyndbg="file dd.c +p"
- clk_ignore_unused pd_ignore_unused

- ftrace=function_graph ftrace_graph_max_depth=3
  ftrace_graph_filter=probe_func

Then in userspace: cat /sys/kernel/tracing/trace

→ Wishlist: Dump ftrace buffer during on probe_func return without patching the kernel.
   Maybe already possible with bootconfig? Ongoing discussion
Device Dependencies

- What if device exporting resource is unavailable?
  - Avoidance:
    - initcalls
    - Makefile ordering
    - Firmware device links (fw_devlink)
  - Detection
    - -EPROBE_DEFER
fw_devlink

- Parse firmware-provided description for dependencies
  - currently supports 25 device tree bindings
  - Follows device properties and manually created links
- Reflect these dependencies in software
- Walk the dependency graph to minimize probe deferral
- Probe deferral can stil happen
  - Waiting for module to be inserted
  - Cyclic dependencies
- Device links inspectable in /sys/class/devlink
Image Resources

- Kernel Recipes Mascot by Emma Tizzoni: https://www.flickr.com/photos/hupstream/53273034553/in/album-72177720312084838
- [PCI-BUS]: https://docs.oracle.com/cd/E19455-01/805-7378/6j6un038j/index.html
Summary

- There's a lot of ways to match devices to drivers
- There's a lot of resources that devices need for proper operations
- There's a lot of ways device driver probes may never succeed
  → When you find a new one, stick in a `dev_err_probe()` to make the world a tiny bit better

Questions?