Intel(R) Management Engine (ME) Client bus API

Rationale

The MEI character device is useful for dedicated applications to send and receive data to the many FW appliance found in Intel's ME from the user space. However, for some of the ME functionalities it makes sense to leverage existing software stack and expose them through existing kernel subsystems.

In order to plug seamlessly into the kernel device driver model we add kernel virtual bus abstraction on top of the MEI driver. This allows implementing Linux kernel drivers for the various MEI features as a stand alone entities found in their respective subsystem. Existing device drivers can even potentially be re-used by adding an MEI CL bus layer to the existing code.

MEI CL bus API

A driver implementation for an MEI Client is very similar to any other existing bus based device drivers. The driver registers itself as an MEI CL bus driver through the `struct mei_cl_driver` structure defined in :file:`include/linux/mei_cl_bus.c`

System Message: ERROR/3 (`D:\onboarding-resources\sample-onboarding-resources\linux-master\Documentation\driver-api\mei\[linux-master][Documentation][driver-api][mei]mei-client-bus.rst`, line 26); *backlink*

Unknown interpreted text role "file".

```
struct mei_cl_driver {
        struct device_driver driver;
        const char *name;

        const struct mei_cl_device_id *id_table;

        int (*probe)(struct mei_cl_device *dev, const struct mei_cl_id *id);
        int (*remove)(struct mei_cl_device *dev);
};
```

The mei_cl_device_id structure defined in :file:`include/linux/mod_devicetable.h` allows a driver to bind itself against a device name.

System Message: ERROR/3 (`D:\onboarding-resources\sample-onboarding-resources\linux-master\Documentation\driver-api\mei\[linux-master][Documentation][driver-api][mei]mei-client-bus.rst`, line 44); *backlink*

Unknown interpreted text role "file".

```
struct mei_cl_device_id {
        char name[MEI_CL_NAME_SIZE];
        uuid_le uuid;
        __u8    version;
        kernel_ulong_t driver_info;
};
```

To actually register a driver on the ME Client bus one must call the :c:func:`mei_cl_add_driver` API. This is typically called at module initialization time.

System Message: ERROR/3 (`D:\onboarding-resources\sample-onboarding-resources\linux-master\Documentation\driver-api\mei\[linux-master][Documentation][driver-api][mei]mei-client-bus.rst`, line 56); *backlink*

Unknown interpreted text role "c:func".

Once the driver is registered and bound to the device, a driver will typically try to do some I/O on this bus and this should be done through the :c:func:`mei_cl_send` and :c:func:`mei_cl_recv` functions. More detailed information is in :ref:`api` section.

System Message: ERROR/3 (`D:\onboarding-resources\sample-onboarding-resources\linux-master\Documentation\driver-api\mei\[linux-master][Documentation][driver-api][mei]mei-client-bus.rst`, line 59); *backlink*

Unknown interpreted text role "c:func".

System Message: ERROR/3 (`D:\onboarding-resources\sample-onboarding-resources\linux-master\Documentation\driver-api\mei\[linux-master][Documentation][driver-api][mei]mei-client-bus.rst`, line 59); *backlink*

Unknown interpreted text role "c:func".

System Message: ERROR/3 (`D:\onboarding-resources\sample-onboarding-resources\linux-master\Documentation\driver-api\mei\[linux-master][Documentation][driver-api][mei]mei-client-bus.rst`, line 59); *backlink*

Unknown interpreted text role "ref".

In order for a driver to be notified about pending traffic or event, the driver should register a callback via :c:func:`mei_cl_devev_register_rx_cb` and :c:func:`mei_cldev_register_notify_cb` function respectively.

System Message: ERROR/3 (`D:\onboarding-resources\sample-onboarding-resources\linux-master\Documentation\driver-api\mei\[linux-master][Documentation][driver-api][mei]mei-client-bus.rst`, line 63); *backlink*

Unknown interpreted text role "c:func".

System Message: ERROR/3 (`D:\onboarding-resources\sample-onboarding-resources\linux-master\Documentation\driver-api\mei\[linux-master][Documentation][driver-api][mei]mei-client-bus.rst`, line 63); *backlink*

Unknown interpreted text role "c:func".

API:

System Message: ERROR/3 (`D:\onboarding-resources\sample-onboarding-resources\linux-master\Documentation\driver-api\mei\[linux-master][Documentation][driver-api][mei]mei-client-bus.rst`, line 71)

Unknown directive type "kernel-doc".

```
.. kernel-doc:: drivers/misc/mei/bus.c
    :export: drivers/misc/mei/bus.c
```

Example

As a theoretical example let's pretend the ME comes with a "contact" NFC IP. The driver init and exit routines for this device would look like:

```
#define CONTACT_DRIVER_NAME "contact"

static struct mei_cl_device_id contact_mei_cl_tbl[] = {
        { CONTACT_DRIVER_NAME, },

        /* required last entry */
        { }
};
MODULE_DEVICE_TABLE(mei_cl, contact_mei_cl_tbl);

static struct mei_cl_driver contact_driver = {
        .id_table = contact_mei_tbl,
        .name = CONTACT_DRIVER_NAME,

        .probe = contact_probe,
        .remove = contact_remove,
};

static int contact_init(void)
{
        int r;

        r = mei_cl_driver_register(&contact_driver);
        if (r) {
                pr_err(CONTACT_DRIVER_NAME ": driver registration failed\n");
                return r;
        }

        return 0;
}

static void __exit contact_exit(void)
{
        mei_cl_driver_unregister(&contact_driver);
}

module_init(contact_init);
module_exit(contact_exit);
```

And the driver's simplified probe routine would look like that:

```
int contact_probe(struct mei_cl_device *dev, struct mei_cl_device_id *id)
{
        [...]
        mei_cldev_enable(dev);

        mei_cldev_register_rx_cb(dev, contact_rx_cb);

        return 0;
}
```

In the probe routine the driver first enable the MEI device and then registers an rx handler which is as close as it can get to registering a threaded IRQ handler. The handler implementation will typically call :c:func:`mei_cldev_recv` and then process received data.

System Message: ERROR/3 (`D:\onboarding-resources\sample-onboarding-resources\linux-master\Documentation\driver-api\mei\[linux-master][Documentation][driver-api][mei]mei-client-bus.rst`, line 137); *backlink*

Unknown interpreted text role "c:func".

```
#define MAX_PAYLOAD 128
#define HDR_SIZE 4
static void conntact_rx_cb(struct mei_cl_device *cldev)
{
        struct contact *c = mei_cldev_get_drvdata(cldev);
        unsigned char payload[MAX_PAYLOAD];
        ssize_t payload_sz;

        payload_sz = mei_cldev_recv(cldev, payload,  MAX_PAYLOAD)
        if (reply_size < HDR_SIZE) {
                return;
        }

        c->process_rx(payload);

}
```

MEI Client Bus Drivers

System Message: ERROR/3 (`D:\onboarding-resources\sample-onboarding-resources\linux-master\Documentation\driver-api\mei\[linux-master][Documentation][driver-api][mei]mei-client-bus.rst`, line 164)

Unknown directive type "toctree".

```
.. toctree::
   :maxdepth: 2

   hdcp
   nfc
```