Device Interfaces

Introduction

Device interfaces are the logical interfaces of device classes that correlate directly to userspace interfaces, like device nodes.

Each device class may have multiple interfaces through which you can access the same device. An input device may support the mouse interface, the 'evdev' interface, and the touchscreen interface. A SCSI disk would support the disk interface, the SCSI generic interface, and possibly a raw device interface.

Device interfaces are registered with the class they belong to. As devices are added to the class, they are added to each interface registered with the class. The interface is responsible for determining whether the device supports the interface or not.

Programming Interface

```
struct device_interface {
        char                    * name;
        rwlock_t                lock;
        u32                     devnum;
        struct device_class     * devclass;

        struct list_head        node;
        struct driver_dir_entry dir;

        int (*add_device)(struct device *);
        int (*add_device)(struct intf_data *);
};

int interface_register(struct device_interface *);
void interface_unregister(struct device_interface *);
```

An interface must specify the device class it belongs to. It is added to that class's list of interfaces on registration.

Interfaces can be added to a device class at any time. Whenever it is added, each device in the class is passed to the interface's add_device callback. When an interface is removed, each device is removed from the interface.

Devices

Once a device is added to a device class, it is added to each interface that is registered with the device class. The class is expected to place a class-specific data structure in

struct device::class_data. The interface can use that (along with
other fields of struct device) to determine whether or not the driver
and/or device support that particular interface.

Data
~~~~

```
struct intf_data {
        struct list_head        node;
        struct device_interface * intf;
        struct device           * dev;
        u32                     intf_num;
};

int interface_add_data(struct interface_data *);
```

The interface is responsible for allocating and initializing a struct
intf_data and calling interface_add_data() to add it to the device's list
of interfaces it belongs to. This list will be iterated over when the device
is removed from the class (instead of all possible interfaces for a class).
This structure should probably be embedded in whatever per-device data
structure the interface is allocating anyway.

Devices are enumerated within the interface. This happens in
interface_add_data()
and the enumerated value is stored in the struct intf_data for that device.

sysfs
~~~~~

Each interface is given a directory in the directory of the device
class it belongs to:

Interfaces get a directory in the class's directory as well:

```
   class/
   `-- input
       |-- devices
       |-- drivers
       |-- mouse
       `-- evdev
```

When a device is added to the interface, a symlink is created that points
to the device's directory in the physical hierarchy:

```
   class/
   `-- input
       |-- devices
       |   `-- 1 -> ../../../root/pci0/00:1f.0/usb_bus/00:1f.2-1:0/
       |-- drivers
       |   `-- usb:usb_mouse -> ../../../bus/drivers/usb_mouse/
       |-- mouse
       |   `-- 1 -> ../../../root/pci0/00:1f.0/usb_bus/00:1f.2-1:0/
       `-- evdev
           `-- 1 -> ../../../root/pci0/00:1f.0/usb_bus/00:1f.2-1:0/
```

Future Plans

A device interface is correlated directly with a userspace interface for a device, specifically a device node. For instance, a SCSI disk exposes at least two interfaces to userspace: the standard SCSI disk interface and the SCSI generic interface. It might also export a raw device interface.

Many interfaces have a major number associated with them and each device gets a minor number. Or, multiple interfaces might share one major number, and each will receive a range of minor numbers (like in the case of input devices).

These major and minor numbers could be stored in the interface structure. Major and minor allocations could happen when the interface is registered with the class, or via a helper function.