rfkill - RF kill switch support

1. Introduction

The rfkill subsystem provides a generic interface to disabling any radio transmitter in the system. When a transmitter is blocked, it shall not radiate any power.

The subsystem also provides the ability to react on button presses and disable all transmitters of a certain type (or all). This is intended for situations where transmitters need to be turned off, for example on aircraft.

The rfkill subsystem has a concept of "hard" and "soft" block, which differ little in their meaning (block == transmitters off) but rather in whether they can be changed or not:
 - hard block: read-only radio block that cannot be overriden by software
 - soft block: writable radio block (need not be readable) that is set by the system software.

2. Implementation details

The rfkill subsystem is composed of three main components:
 * the rfkill core,
 * the deprecated rfkill-input module (an input layer handler, being replaced by userspace policy code) and
 * the rfkill drivers.

The rfkill core provides API for kernel drivers to register their radio transmitter with the kernel, methods for turning it on and off and, letting the system know about hardware-disabled states that may be implemented on the device.

The rfkill core code also notifies userspace of state changes, and provides ways for userspace to query the current states. See the "Userspace support" section below.

When the device is hard-blocked (either by a call to rfkill_set_hw_state() or from query_hw_block) set_block() will be invoked for additional software block, but drivers can ignore the method call since they can use the return value of the function rfkill_set_hw_state() to sync the software state instead of keeping track of calls to set_block(). In fact, drivers should use the return value of rfkill_set_hw_state() unless the hardware actually keeps track of soft and hard block separately.

3. Kernel API

Drivers for radio transmitters normally implement an rfkill driver.

Platform drivers might implement input devices if the rfkill button is just that, a button. If that button influences the hardware then you need to implement an rfkill driver instead. This also applies if the platform provides a way to turn on/off the transmitter(s).

For some platforms, it is possible that the hardware state changes during suspend/hibernation, in which case it will be necessary to update the rfkill core with the current state is at resume time.

To create an rfkill driver, driver's Kconfig needs to have

```
depends on RFKILL || !RFKILL
```

to ensure the driver cannot be built-in when rfkill is modular. The !RFKILL case allows the driver to be built when rfkill is not configured, which which case all rfkill API can still be used but will be provided by static inlines which compile to almost nothing.

Calling rfkill_set_hw_state() when a state change happens is required from rfkill drivers that control devices that can be hard-blocked unless they also assign the poll_hw_block() callback (then the rfkill core will poll the device). Don't do this unless you cannot get the event in any other way.

5. Userspace support

The recommended userspace interface to use is /dev/rfkill, which is a misc character device that allows userspace to obtain and set the state of rfkill devices and sets of devices. It also notifies userspace about device addition and removal. The API is a simple read/write API that is defined in linux/rfkill.h, with one ioctl that allows turning off the deprecated input handler in the kernel for the transition period.

Except for the one ioctl, communication with the kernel is done via read() and write() of instances of 'struct rfkill_event'. In this structure, the soft and hard block are properly separated (unlike sysfs, see below) and userspace is able to get a consistent snapshot of all rfkill devices in the system. Also, it is possible to switch all rfkill drivers (or all drivers of a specified type) into a state which also updates the default state for hotplugged devices.

After an application opens /dev/rfkill, it can read the current state of all devices. Changes can be either obtained by either polling the descriptor for hotplug or state change events or by listening for uevents emitted by the rfkill core framework.

Additionally, each rfkill device is registered in sysfs and emits uevents.

rfkill devices issue uevents (with an action of "change"), with the following environment variables set:

RFKILL_NAME

RFKILL_STATE
RFKILL_TYPE

The contents of these variables corresponds to the "name", "state" and "type" sysfs files explained above.

For further details consult Documentation/ABI/stable/dev-rfkill and Documentation/ABI/stable/sysfs-class-rfkill.