HIDRAW - Raw Access to USB and Bluetooth Human Interface Devices

The hidraw driver provides a raw interface to USB and Bluetooth Human Interface Devices (HIDs). It differs from hiddev in that reports sent and received are not parsed by the HID parser, but are sent to and received from the device unmodified.

Hidraw should be used if the userspace application knows exactly how to communicate with the hardware device, and is able to construct the HID reports manually. This is often the case when making userspace drivers for custom HID devices.

Hidraw is also useful for communicating with non-conformant HID devices which send and receive data in a way that is inconsistent with their report descriptors. Because hiddev parses reports which are sent and received through it, checking them against the device's report descriptor, such communication with these non-conformant devices is impossible using hiddev. Hidraw is the only alternative, short of writing a custom kernel driver, for these non-conformant devices.

A benefit of hidraw is that its use by userspace applications is independent of the underlying hardware type. Currently, hidraw is implemented for USB and Bluetooth. In the future, as new hardware bus types are developed which use the HID specification, hidraw will be expanded to add support for these new bus types.

Hidraw uses a dynamic major number, meaning that udev should be relied on to create hidraw device nodes. Udev will typically create the device nodes directly under /dev (eg: /dev/hidraw0). As this location is distribution- and udev rule-dependent, applications should use libudev to locate hidraw devices attached to the system. There is a tutorial on libudev with a working example at:

```
http://www.signal11.us/oss/udev/
https://web.archive.org/web/2019*/www.signal11.us
```

The HIDRAW API

read()

read() will read a queued report received from the HID device. On USB devices, the reports read using read() are the reports sent from the device on the INTERRUPT IN endpoint. By default, read() will block until there is a report available to be read. read() can be made non-blocking, by passing the O_NONBLOCK flag to open(), or by setting the O_NONBLOCK flag using fcntl().

On a device which uses numbered reports, the first byte of the returned data will be the report number; the report data follows, beginning in the second byte. For devices which do not use numbered reports, the report data will begin at the first byte.

write()

The write() function will write a report to the device. For USB devices, if the device has an INTERRUPT OUT endpoint, the report will be sent on that endpoint. If it does not, the report will be sent over the control endpoint, using a SET_REPORT transfer.

The first byte of the buffer passed to write() should be set to the report number. If the device does not use numbered reports, the first byte should be set to 0. The report data itself should begin at the second byte.

ioctl()

Hidraw supports the following ioctls:

HIDIOCGRDESCSIZE:
    Get Report Descriptor Size

This ioctl will get the size of the device's report descriptor.

HIDIOCGRDESC:
    Get Report Descriptor

This ioctl returns the device's report descriptor using a hidraw_report_descriptor struct. Make sure to set the size field of the hidraw_report_descriptor struct to the size returned from HIDIOCGRDESCSIZE.

HIDIOCGRAWINFO:
    Get Raw Info

This ioctl will return a hidraw_devinfo struct containing the bus type, the vendor ID (VID), and product ID (PID) of the device. The bus type can be one of:

```
- BUS_USB
- BUS_HIL
- BUS_BLUETOOTH
- BUS_VIRTUAL
```

which are defined in uapi/linux/input.h.

HIDIOCGRAWNAME(len):
Get Raw Name

This ioctl returns a string containing the vendor and product strings of the device. The returned string is Unicode, UTF-8 encoded.

HIDIOCGRAWPHYS(len):
Get Physical Address

This ioctl returns a string representing the physical address of the device. For USB devices, the string contains the physical path to the device (the USB controller, hubs, ports, etc). For Bluetooth devices, the string contains the hardware (MAC) address of the device.

HIDIOCSFEATURE(len):
Send a Feature Report

This ioctl will send a feature report to the device. Per the HID specification, feature reports are always sent using the control endpoint. Set the first byte of the supplied buffer to the report number. For devices which do not use numbered reports, set the first byte to 0. The report data begins in the second byte. Make sure to set len accordingly, to one more than the length of the report (to account for the report number).

HIDIOCGFEATURE(len):
Get a Feature Report

This ioctl will request a feature report from the device using the control endpoint. The first byte of the supplied buffer should be set to the report number of the requested report. For devices which do not use numbered reports, set the first byte to 0. The returned report buffer will contain the report number in the first byte, followed by the report data read from the device. For devices which do not use numbered reports, the report data will begin at the first byte of the returned buffer.

HIDIOCSINPUT(len):
Send an Input Report

This ioctl will send an input report to the device, using the control endpoint. In most cases, setting an input HID report on a device is meaningless and has no effect, but some devices may choose to use this to set or reset an initial state of a report. The format of the buffer issued with this report is identical to that of HIDIOCSFEATURE.

HIDIOCGINPUT(len):
Get an Input Report

This ioctl will request an input report from the device using the control endpoint. This is slower on most devices where a dedicated In endpoint exists for regular input reports, but allows the host to request the value of a specific report number. Typically, this is used to request the initial states of an input report of a device, before an application listens for normal reports via the regular device read() interface. The format of the buffer issued with this report is identical to that of HIDIOCGFEATURE.

HIDIOCSOUTPUT(len):
Send an Output Report

This ioctl will send an output report to the device, using the control endpoint. This is slower on most devices where a dedicated Out endpoint exists for regular output reports, but is added for completeness. Typically, this is used to set the initial states of an output report of a device, before an application sends updates via the regular device write() interface. The format of the buffer issued with this report is identical to that of HIDIOCSFEATURE.

HIDIOCGOUTPUT(len):
Get an Output Report

This ioctl will request an output report from the device using the control endpoint. Typically, this is used to retrive the initial state of an output report of a device, before an application updates it as necessary either via a HIDIOCSOUTPUT request, or the regular device write() interface. The format of the buffer issued with this report is identical to that of HIDIOCGFEATURE.

Example

In samples/, find hid-example.c, which shows examples of read(), write(), and all the ioctls for hidraw. The code may be used by anyone for any purpose, and can serve as a starting point for developing applications using hidraw.

Document by:

Alan Ott <alan@signal11.us>, Signal 11 Software