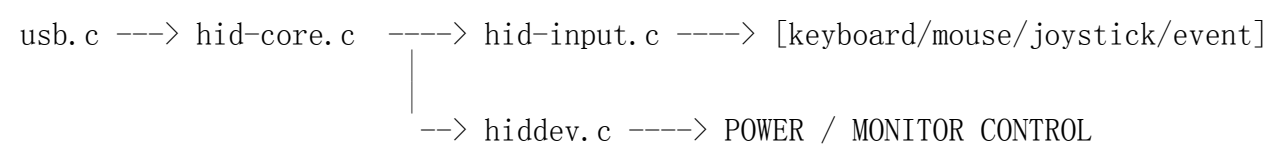
Care and feeding of your Human Interface Devices

INTRODUCTION

In addition to the normal input type HID devices, USB also uses the human interface device protocols for things that are not really human interfaces, but have similar sorts of communication needs. The two big examples for this are power devices (especially uninterruptable power supplies) and monitor control on higher end monitors.

To support these disparate requirements, the Linux USB system provides HID events to two separate interfaces:
* the input subsystem, which converts HID events into normal input device interfaces (such as keyboard, mouse and joystick) and a normalised event interface - see Documentation/input/input.txt
* the hiddev interface, which provides fairly raw HID events

The data flow for a HID event produced by a device is something like the following :



In addition, other subsystems (apart from USB) can potentially feed events into the input subsystem, but these have no effect on the hid device interface.

USING THE HID DEVICE INTERFACE

The hiddev interface is a char interface using the normal USB major, with the minor numbers starting at 96 and finishing at 111. Therefore, you need the following commands:

```
mknod /dev/usb/hiddev0 c 180 96
mknod /dev/usb/hiddev1 c 180 97
mknod /dev/usb/hiddev2 c 180 98
mknod /dev/usb/hiddev3 c 180 99
mknod /dev/usb/hiddev4 c 180 100
mknod /dev/usb/hiddev5 c 180 101
mknod /dev/usb/hiddev6 c 180 102
mknod /dev/usb/hiddev7 c 180 103
mknod /dev/usb/hiddev8 c 180 104
mknod /dev/usb/hiddev9 c 180 105
mknod /dev/usb/hiddev10 c 180 106
mknod /dev/usb/hiddev11 c 180 107
mknod /dev/usb/hiddev12 c 180 108
mknod /dev/usb/hiddev13 c 180 109
mknod /dev/usb/hiddev14 c 180 110
mknod /dev/usb/hiddev15 c 180 111
```

So you point your hiddev compliant user-space program at the correct interface for your device, and it all just works.

Assuming that you have a hiddev compliant user-space program, of course. If you need to write one, read on.

THE HIDDEV API
This description should be read in conjunction with the HID specification, freely available from http://www.usb.org, and conveniently linked of http://www.linux-usb.org.

The hiddev API uses a read() interface, and a set of ioctl() calls.

HID devices exchange data with the host computer using data bundles called "reports". Each report is divided into "fields", each of which can have one or more "usages". In the hid-core, each one of these usages has a single signed 32 bit value.

read():
This is the event interface. When the HID device's state changes, it performs an interrupt transfer containing a report which contains the changed value. The hid-core.c module parses the report, and returns to hiddev.c the individual usages that have changed within the report. In its basic mode, the hiddev will make these individual usage changes available to the reader using a struct hiddev_event:

```
struct hiddev_event {
    unsigned hid;
    signed int value;
};
```

containing the HID usage identifier for the status that changed, and the value that it was changed to. Note that the structure is defined within <linux/hiddev.h>, along with some other useful #defines and structures. The HID usage identifier is a composite of the HID usage page shifted to the 16 high order bits ORed with the usage code. The behavior of the read() function can be modified using the HIDIOCSFLAG ioctl() described below.

ioctl():
This is the control interface. There are a number of controls:

HIDIOCGVERSION - int (read)
Gets the version code out of the hiddev driver.

HIDIOCAPPLICATION - (none)
This ioctl call returns the HID application usage associated with the hid device. The third argument to ioctl() specifies which application index to get. This is useful when the device has more than one application collection. If the index is invalid (greater or equal to the number of application collections this device has) the ioctl returns -1. You can find out beforehand how many application collections the device has from the num_applications field from the hiddev_devinfo structure.

HIDIOCGCOLLECTIONINFO - struct hiddev_collection_info (read/write)
This returns a superset of the information above, providing not only application collections, but all the collections the device has. It also returns the level the collection lives in the hierarchy.

The user passes in a hiddev_collection_info struct with the index field set to the index that should be returned. The ioctl fills in the other fields. If the index is larger than the last collection index, the ioctl returns -1 and sets errno to -EINVAL.

HIDIOCGDEVINFO - struct hiddev_devinfo (read)
Gets a hiddev_devinfo structure which describes the device.

HIDIOCGSTRING - struct hiddev_string_descriptor (read/write)
Gets a string descriptor from the device. The caller must fill in the "index" field to indicate which descriptor should be returned.

HIDIOCINITREPORT - (none)
Instructs the kernel to retrieve all input and feature report values from the device. At this point, all the usage structures will contain current values for the device, and will maintain it as the device changes. Note that the use of this ioctl is unnecessary in general, since later kernels automatically initialize the reports from the device at attach time.

HIDIOCGNAME - string (variable length)
Gets the device name

HIDIOCGREPORT - struct hiddev_report_info (write)
Instructs the kernel to get a feature or input report from the device, in order to selectively update the usage structures (in contrast to INITREPORT).

HIDIOCSREPORT - struct hiddev_report_info (write)
Instructs the kernel to send a report to the device. This report can be filled in by the user through HIDIOCSUSAGE calls (below) to fill in individual usage values in the report before sending the report in full to the device.

HIDIOCGREPORTINFO - struct hiddev_report_info (read/write)
Fills in a hiddev_report_info structure for the user. The report is looked up by type (input, output or feature) and id, so these fields must be filled in by the user. The ID can be absolute -- the actual report id as reported by the device -- or relative -- HID_REPORT_ID_FIRST for the first report, and (HID_REPORT_ID_NEXT | report_id) for the next report after report_id. Without a-priori information about report ids, the right way to use this ioctl is to use the relative IDs above to enumerate the valid IDs. The ioctl returns non-zero when there is no more next ID. The real report ID is filled into the returned hiddev_report_info structure.

HIDIOCGFIELDINFO - struct hiddev_field_info (read/write)
Returns the field information associated with a report in a hiddev_field_info structure. The user must fill in report_id and report_type in this structure, as above. The field_index should also be filled in, which should be a number from 0 and maxfield-1, as returned from a previous HIDIOCGREPORTINFO call.

HIDIOCGUCODE - struct hiddev_usage_ref (read/write)
Returns the usage_code in a hiddev_usage_ref structure, given that given its report type, report id, field index, and index within the

field have already been filled into the structure.

HIDIOCGUSAGE - struct hiddev_usage_ref (read/write)
Returns the value of a usage in a hiddev_usage_ref structure. The usage to be retrieved can be specified as above, or the user can choose to fill in the report_type field and specify the report_id as HID_REPORT_ID_UNKNOWN. In this case, the hiddev_usage_ref will be filled in with the report and field information associated with this usage if it is found.

HIDIOCSUSAGE - struct hiddev_usage_ref (write)
Sets the value of a usage in an output report. The user fills in the hiddev_usage_ref structure as above, but additionally fills in the value field.

HIDIOGCOLLECTIONINDEX - struct hiddev_usage_ref (write)
Returns the collection index associated with this usage. This indicates where in the collection hierarchy this usage sits.

HIDIOCGFLAG - int (read)
HIDIOCSFLAG - int (write)
These operations respectively inspect and replace the mode flags that influence the read() call above. The flags are as follows:

HIDDEV_FLAG_UREF - read() calls will now return struct hiddev_usage_ref instead of struct hiddev_event. This is a larger structure, but in situations where the device has more than one usage in its reports with the same usage code, this mode serves to resolve such ambiguity.

HIDDEV_FLAG_REPORT - This flag can only be used in conjunction with HIDDEV_FLAG_UREF. With this flag set, when the device sends a report, a struct hiddev_usage_ref will be returned to read() filled in with the report_type and report_id, but with field_index set to FIELD_INDEX_NONE. This serves as additional notification when the device has sent a report.