Introduction

Architecture

Input subsystem is a collection of drivers that is designed to support all input devices under Linux. Most of the drivers reside in drivers/input, although quite a few live in drivers/hid and drivers/platform.

The core of the input subsystem is the input module, which must be loaded before any other of the input modules - it serves as a way of communication between two groups of modules:

Device drivers

These modules talk to the hardware (for example via USB), and provide events (keystrokes, mouse movements) to the input module.

Event handlers

These modules get events from input core and pass them where needed via various interfaces - keystrokes to the kernel, mouse movements via a simulated PS/2 interface to GPM and X, and so on.

Simple Usage

For the most usual configuration, with one USB mouse and one USB keyboard, you'll have to load the following modules (or have them built in to the kernel):

```
input
mousedev
usbcore
uhci_hcd or ohci_hcd or ehci_hcd
usbhid
hid_generic
```

After this, the USB keyboard will work straight away, and the USB mouse will be available as a character device on major 13, minor 63:

```
crw-r--r--   1 root     root      13,  63 Mar 28 22:45 mice
```

This device is usually created automatically by the system. The commands to create it by hand are:

```
cd /dev
mkdir input
mknod input/mice c 13 63
```

After that you have to point GPM (the textmode mouse cut&paste tool) and XFree to this device to use it - GPM should be called like:

```
gpm -t ps2 -m /dev/input/mice
```

And in X:

```
Section "Pointer"
    Protocol    "ImPS/2"
    Device      "/dev/input/mice"
    ZAxisMapping 4 5
EndSection
```

When you do all of the above, you can use your USB mouse and keyboard.

Detailed Description

Event handlers

Event handlers distribute the events from the devices to userspace and in-kernel consumers, as needed.

evdev

`evdev` is the generic input event interface. It passes the events generated in the kernel straight to the program, with timestamps. The event codes are the same on all architectures and are hardware independent.

This is the preferred interface for userspace to consume user input, and all clients are encouraged to use it.

See :ref:`event-interface` for notes on API.

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\linux-master\Documentation\input\(linux-master)(Documentation)(input)input.rst, line 95); *backlink*

Unknown interpreted text role "ref".

The devices are in /dev/input:

```
crw-r--r--   1 root     root      13,  64 Apr  1 10:49 event0
crw-r--r--   1 root     root      13,  65 Apr  1 10:50 event1
crw-r--r--   1 root     root      13,  66 Apr  1 10:50 event2
crw-r--r--   1 root     root      13,  67 Apr  1 10:50 event3
...
```

There are two ranges of minors: 64 through 95 is the static legacy range. If there are more than 32 input devices in a system, additional evdev nodes are created with minors starting with 256.

keyboard

`keyboard` is in-kernel input handler and is a part of VT code. It consumes keyboard keystrokes and handles user input for VT consoles.

mousedev

`mousedev` is a hack to make legacy programs that use mouse input work. It takes events from either mice or digitizers/tablets and makes a PS/2-style (a la /dev/psaux) mouse device available to the userland.

Mousedev devices in /dev/input (as shown above) are:

```
crw-r--r--   1 root     root      13,  32 Mar 28 22:45 mouse0
crw-r--r--   1 root     root      13,  33 Mar 29 00:41 mouse1
crw-r--r--   1 root     root      13,  34 Mar 29 00:41 mouse2
crw-r--r--   1 root     root      13,  35 Apr  1 10:50 mouse3
...
...
crw-r--r--   1 root     root      13,  62 Apr  1 10:50 mouse30
crw-r--r--   1 root     root      13,  63 Apr  1 10:50 mice
```

Each `mouse` device is assigned to a single mouse or digitizer, except the last one - `mice`. This single character device is shared by all mice and digitizers, and even if none are connected, the device is present. This is useful for hotplugging USB mice, so that older programs that do not handle hotplug can open the device even when no mice are present.

CONFIG_INPUT_MOUSEDEV_SCREEN_[XY] in the kernel configuration are the size of your screen (in pixels) in XFree86. This is needed if you want to use your digitizer in X, because its movement is sent to X via a virtual PS/2 mouse and thus needs to be scaled accordingly. These values won't be used if you use a mouse only.

Mousedev will generate either PS/2, ImPS/2 (Microsoft IntelliMouse) or ExplorerPS/2 (IntelliMouse Explorer) protocols, depending on what the program reading the data wishes. You can set GPM and X to any of these. You'll need ImPS/2 if you want to make use of a wheel on a USB mouse and ExplorerPS/2 if you want to use extra (up to 5) buttons.

joydev

`joydev` implements v0.x and v1.x Linux joystick API. See :ref:`joystick-api` for details.

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\linux-master\Documentation\input\(linux-master)(Documentation)(input)input.rst, line 156); *backlink*

Unknown interpreted text role "ref".

As soon as any joystick is connected, it can be accessed in /dev/input on:

```
crw-r--r--   1 root     root      13,   0 Apr  1 10:50 js0
crw-r--r--   1 root     root      13,   1 Apr  1 10:50 js1
crw-r--r--   1 root     root      13,   2 Apr  1 10:50 js2
crw-r--r--   1 root     root      13,   3 Apr  1 10:50 js3
...
```

And so on up to js31 in legacy range, and additional nodes with minors above 256 if there are more joystick devices.

Device drivers

Device drivers are the modules that generate events.

hid-generic

`hid-generic` is one of the largest and most complex driver of the whole suite. It handles all HID devices, and because there is a very wide variety of them, and because the USB HID specification isn't simple, it needs to be this big.

Currently, it handles USB mice, joysticks, gamepads, steering wheels, keyboards, trackballs and digitizers.

However, USB uses HID also for monitor controls, speaker controls, UPSs, LCDs and many other purposes.

The monitor and speaker controls should be easy to add to the hid/input interface, but for the UPSs and LCDs it doesn't make much sense. For this, the hiddev interface was designed. See Documentation/hid/hiddev.rst for more information about it.

The usage of the usbhid module is very simple, it takes no parameters, detects everything automatically and when a HID device is inserted, it detects it appropriately.

However, because the devices vary wildly, you might happen to have a device that doesn't work well. In that case #define DEBUG at the beginning of hid-core.c and send me the syslog traces.

usbmouse

For embedded systems, for mice with broken HID descriptors and just any other use when the big usbhid wouldn't be a good choice, there is the usbmouse driver. It handles USB mice only. It uses a simpler HIDBP protocol. This also means the mice must support this simpler protocol. Not all do. If you don't have any strong reason to use this module, use usbhid instead.

usbkbd

Much like usbmouse, this module talks to keyboards with a simplified HIDBP protocol. It's smaller, but doesn't support any extra special keys. Use usbhid instead if there isn't any special reason to use this.

psmouse

This is driver for all flavors of pointing devices using PS/2 protocol, including Synaptics and ALPS touchpads, Intellimouse Explorer devices, Logitech PS/2 mice and so on.

atkbd

This is driver for PS/2 (AT) keyboards.

iforce

A driver for I-Force joysticks and wheels, both over USB and RS232. It includes Force Feedback support now, even though Immersion Corp. considers the protocol a trade secret and won't disclose a word about it.

Verifying if it works

Typing a couple keys on the keyboard should be enough to check that a keyboard works and is correctly connected to the kernel keyboard driver.

Doing a `cat /dev/input/mouse0` (c, 13, 32) will verify that a mouse is also emulated; characters should appear if you move it.

You can test the joystick emulation with the `jstest` utility, available in the joystick package (see :ref:`joystick-doc`).

> **System Message: ERROR/3 (`D:\onboarding-resources\sample-onboarding-resources\linux-master\Documentation\input\(linux-master)(Documentation)(input)input.rst`, line 249); *backlink***
>
> Unknown interpreted text role "ref".

You can test the event devices with the `evtest` utility.

Event interface

You can use blocking and nonblocking reads, and also select() on the /dev/input/eventX devices, and you'll always get a whole number of input events on a read. Their layout is:

```
struct input_event {
        struct timeval time;
        unsigned short type;
        unsigned short code;
        unsigned int value;
};
```

`time` is the timestamp, it returns the time at which the event happened. Type is for example EV_REL for relative movement, EV_KEY for a keypress or release. More types are defined in include/uapi/linux/input-event-codes.h.

`code` is event code, for example REL_X or KEY_BACKSPACE, again a complete list is in include/uapi/linux/input-event-codes.h.

`value` is the value the event carries. Either a relative change for EV_REL, absolute new value for EV_ABS (joysticks ...), or 0 for EV_KEY for release, 1 for keypress and 2 for autorepeat.

See :ref:`input-event-codes` for more information about various even codes.

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\linux-master\Documentation\input\(linux-master)(Documentation)(input)input.rst, line 281); *backlink*

Unknown interpreted text role "ref".