

0. Disclaimer

This program is free software; you can redistribute it and/or modify it under the terms of the GNU General Public License as published by the Free Software Foundation; either version 2 of the License, or (at your option) any later version.

This program is distributed in the hope that it will be useful, but WITHOUT ANY WARRANTY; without even the implied warranty of MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the GNU General Public License for more details.

You should have received a copy of the GNU General Public License along with this program; if not, write to the Free Software Foundation, Inc., 59 Temple Place, Suite 330, Boston, MA 02111-1307 USA

Should you need to contact me, the author, you can do so either by e-mail - mail your message to <vojtech@ucw.cz>, or by paper mail: Vojtech Pavlik, Simunkova 1594, Prague 8, 182 00 Czech Republic

For your convenience, the GNU General Public License version 2 is included in the package: See the file COPYING.

1. Introduction

This is a collection of drivers that is designed to support all input devices under Linux. While it is currently used only on for USB input devices, future use (say 2.5/2.6) is expected to expand to replace most of the existing input system, which is why it lives in drivers/input/ instead of drivers/usb/.

The centre of the input drivers 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:

1.1 Device drivers

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

1.2 Event handlers

These modules get events from input 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.

2. 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.txt

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

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 has to be created.
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.

3. Detailed Description

3.1 Device drivers

Device drivers are the modules that generate events. The events are however not useful without being handled, so you also will need to use some of the modules from section 3.2.

3.1.1 usbhid

usbhid is 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

input.txt

interface, but for the UPSs and LCDs it doesn't make much sense. For this, the hiddev interface was designed. See Documentation/usb/hiddev.txt 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.

3.1.2 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.

3.1.3 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.

3.1.4 wacom

This is a driver for Wacom Graphire and Intuos tablets. Not for Wacom PenPartner, that one is handled by the HID driver. Although the Intuos and Graphire tablets claim that they are HID tablets as well, they are not and thus need this specific driver.

3.1.5 iforce

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

3.2 Event handlers

Event handlers distribute the events from the devices to userland and kernel, as needed.

3.2.1 keybdev

keybdev is currently a rather ugly hack that translates the input events into architecture-specific keyboard raw mode (Xlated AT Set2 on x86), and passes them into the handle_scancode function of the keyboard.c module. This works well enough on all architectures that keybdev can generate rawmode on, other architectures can be added to it.

The right way would be to pass the events to keyboard.c directly,

input.txt

best if keyboard.c would itself be an event handler. This is done in the input patch, available on the webpage mentioned below.

3.2.2 mousedev

mousedev is also a hack to make 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. Ideally, the programs could use a more reasonable interface, for example evdev

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 programs 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.

3.2.3 joydev

Joydev implements v0.x and v1.x Linux joystick api, much like drivers/char/joystick/joystick.c used to in earlier versions. See joystick-api.txt in the Documentation subdirectory for details. 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.

input.txt

3.2.4 evdev

evdev is the generic input event interface. It passes the events generated in the kernel straight to the program, with timestamps. The API is still evolving, but should be useable now. It's described in section 5.

This should be the way for GPM and X to get keyboard and mouse events. It allows for multihead in X without any specific multihead kernel support. The event codes are the same on all architectures and are hardware independent.

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
...
```

And so on up to event31.

4. Verifying if it works

Typing a couple keys on the keyboard should be enough to check that a USB 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 Documentation/input/joystick.txt).

You can test the event devices with the 'evtest' utility available in the LinuxConsole project CVS archive (see the URL below).

5. Event interface

Should you want to add event device support into any application (X, gpm, svgalib ...) I <vojtech@ucw.cz> will be happy to provide you any help I can. Here goes a description of the current state of things, which is going to be extended, but not changed incompatibly as time goes:

You can use blocking and nonblocking reads, 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.

input.txt

Type is for example EV_REL for relative moment, EV_KEY for a keypress or release. More types are defined in include/linux/input.h.

'code' is event code, for example REL_X or KEY_BACKSPACE, again a complete list is in include/linux/input.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.