# HID设备驱动

HID设备表示人机接口（交互）设备（human interface device），USB\_HID设备表示通过USB总线连接的HID设备。HID设备驱动位于/drivers/hid/目录下，USB\_HID设备驱动位于/drivers/hid/usbhid/目录下。

### 9.7.1驱动框架

### 9.7.2设备与驱动

HID设备驱动模型中定义并注册了总线bus\_type实例**hid\_bus\_type**，用于HID设备和驱动。HID总线是一个虚拟的总线，用于管理HID类设备和驱动，真实的设备可能挂载在USB总线、I2C总线等。

HID设备由hid\_device结构体表示，HID设备驱动由hid\_driver结构体表示，都定义在/include/linux/hid.h头文件。

#### 1数据结构

##### ■hid\_device

hid\_device结构体定义如下（/include/linux/hid.h）：

struct hid\_device {

\_\_u8 \***dev\_rdesc**; /\*报告描述字\*/

unsigned dev\_rsize; /\*报告描述字大小\*/

\_\_u8 \*rdesc;

unsigned rsize;

struct hid\_collection \***collection**; /\* List of HID collections \*/

unsigned collection\_size; /\* Number of allocated hid\_collections \*/

unsigned maxcollection; /\* Number of parsed collections \*/

unsigned maxapplication; /\* Number of applications \*/

\_\_u16 bus; /\* BUS ID \*/

\_\_u16 group; /\* Report group \*/

\_\_u32 vendor; /\* Vendor ID \*/

\_\_u32 product; /\* Product ID \*/

\_\_u32 version; /\* HID version \*/

enum hid\_type **type**; /\*设备类型(mouse, kbd, ...) \*/

unsigned country; /\* HID country \*/

struct hid\_report\_enum report\_enum[HID\_REPORT\_TYPES];

struct work\_struct led\_work; /\* delayed LED worker \*/

struct semaphore driver\_lock; /\* protects the current driver, except during input \*/

struct semaphore driver\_input\_lock; /\* protects the current driver \*/

struct device **dev**; /\*通用device结构体成员\*/

struct hid\_driver \***driver**; /\*HID驱动，见下文\*/

struct hid\_ll\_driver \***ll\_driver**; /\*底层HID驱动回调函数，见下文\*/

#ifdef CONFIG\_HID\_BATTERY\_STRENGTH

struct power\_supply \*battery;

\_\_s32 battery\_min;

\_\_s32 battery\_max;

\_\_s32 battery\_report\_type;

\_\_s32 battery\_report\_id;

#endif

unsigned int status; /\* see STAT flags above \*/

unsigned claimed; /\* Claimed by hidinput, hiddev? \*/

unsigned **quirks**; /\* Various quirks the device can pull on us \*/

bool io\_started; /\* Protected by driver\_lock. If IO has started \*/

struct list\_head **inputs;** /\*双链表头，链接hid\_input结构体实例\*/

void \*hiddev; /\* The hiddev structure \*/

void \*hidraw;

int minor; /\*从设备号\*/

int open; /\* is the device open by anyone? \*/

char name[128]; /\*名称\*/

char phys[64]; /\* Device physical location \*/

char uniq[64]; /\* Device unique identifier (serial #) \*/

void \***driver\_data**; /\*驱动所需的数据\*/

/\* temporary hid\_ff handling (until moved to the drivers) \*/

int (\*ff\_init)(struct hid\_device \*);

/\* hiddev event handler 事件处理器\*/

int (\***hiddev\_connect**)(struct hid\_device \*, unsigned int);

void (\*hiddev\_disconnect)(struct hid\_device \*);

void (\*hiddev\_hid\_event) (struct hid\_device \*, struct hid\_field \*field,struct hid\_usage \*, \_\_s32);

void (\*hiddev\_report\_event) (struct hid\_device \*, struct hid\_report \*);

...

};

hid\_device结构体中主要成员简介如下：

**●collection：**指向hid\_collection结构体，定义如下：

struct hid\_collection {

unsigned type;

unsigned usage;

unsigned level;

};

**●type：**设备类型，由hid\_type枚举类型表示，定义如下：

enum hid\_type {

HID\_TYPE\_OTHER = 0,

HID\_TYPE\_USBMOUSE,

HID\_TYPE\_USBNONE

};

**●quirks：**标识设备比较特殊的特性，标记值定义如下（/include/linux/hid.h）：

#define MAX\_USBHID\_BOOT\_QUIRKS 4

#define HID\_QUIRK\_INVERT 0x00000001

#define HID\_QUIRK\_NOTOUCH 0x00000002

#define HID\_QUIRK\_IGNORE 0x00000004

#define HID\_QUIRK\_NOGET 0x00000008

#define HID\_QUIRK\_HIDDEV\_FORCE 0x00000010

#define HID\_QUIRK\_BADPAD 0x00000020

#define HID\_QUIRK\_MULTI\_INPUT 0x00000040

#define HID\_QUIRK\_HIDINPUT\_FORCE 0x00000080

#define HID\_QUIRK\_NO\_EMPTY\_INPUT 0x00000100

#define HID\_QUIRK\_NO\_INIT\_INPUT\_REPORTS 0x00000200

#define HID\_QUIRK\_ALWAYS\_POLL 0x00000400

#define HID\_QUIRK\_SKIP\_OUTPUT\_REPORTS 0x00010000

#define HID\_QUIRK\_SKIP\_OUTPUT\_REPORT\_ID 0x00020000

#define HID\_QUIRK\_NO\_OUTPUT\_REPORTS\_ON\_INTR\_EP 0x00040000

#define HID\_QUIRK\_FULLSPEED\_INTERVAL 0x10000000

#define HID\_QUIRK\_NO\_INIT\_REPORTS 0x20000000

#define HID\_QUIRK\_NO\_IGNORE 0x40000000

**●driver：**指向HID设备驱动hid\_driver结构体，定义见下文。

**●ll\_driver：**指向hid\_ll\_driver结构体，定义见下文。

**●inputs：**双链表成员，链接hid\_input结构体实例，定义如下：

struct hid\_input {

struct list\_head list;

struct hid\_report \***report**; /\*指向hid\_report结构体，定义见下文\*/

struct input\_dev \***input**; /\*指向input\_dev结构体\*/

};

hid\_report结构体定义如下：

struct hid\_report {

struct list\_head list;

unsigned id; /\*ID\*/

unsigned type; /\*报告类型\*/

struct hid\_field \*field[HID\_MAX\_FIELDS]; /\* fields of the report \*/

unsigned maxfield; /\* maximum valid field index \*/

unsigned size; /\* size of the report (bits) \*/

struct hid\_device \*device; 　　/\*关联的HID设备\*/

};



##### ■hid\_driver

HID设备驱动hid\_driver结构体定义如下（/include/linux/hid.h）：

struct hid\_driver {

char \*name; /\*名称\*/

const struct hid\_device\_id \***id\_table**; /\*匹配的设备ID列表\*/

struct list\_head **dyn\_list**; /\*链接由写new\_id属性添加的hid\_dynid实例\*/

spinlock\_t dyn\_lock;

int (\*probe)(struct hid\_device \*dev, const struct hid\_device\_id \*id); /\*探测函数\*/

void (\*remove)(struct hid\_device \*dev); /\*移除设备函数（热插拔）\*/

const struct hid\_report\_id \***report\_table**;

int (\*raw\_event)(struct hid\_device \*hdev, struct hid\_report \*report,u8 \*data, int size);

const struct hid\_usage\_id \***usage\_table**;

int (\*event)(struct hid\_device \*hdev, struct hid\_field \*field,struct hid\_usage \*usage, \_\_s32 value);

void (\*report)(struct hid\_device \*hdev, struct hid\_report \*report);

\_\_u8 \*(\*report\_fixup)(struct hid\_device \*hdev, \_\_u8 \*buf,unsigned int \*size);

int (\*input\_mapping)(struct hid\_device \*hdev,struct hid\_input \*hidinput, struct hid\_field \*field,

struct hid\_usage \*usage, unsigned long \*\*bit, int \*max);

int (\*input\_mapped)(struct hid\_device \*hdev,struct hid\_input \*hidinput, struct hid\_field \*field,

struct hid\_usage \*usage, unsigned long \*\*bit, int \*max);

void (\*input\_configured)(struct hid\_device \*hdev,struct hid\_input \*hidinput);

void (\*feature\_mapping)(struct hid\_device \*hdev,struct hid\_field \*field,struct hid\_usage \*usage);

#ifdef CONFIG\_PM

int (\*suspend)(struct hid\_device \*hdev, pm\_message\_t message);

int (\*resume)(struct hid\_device \*hdev);

int (\*reset\_resume)(struct hid\_device \*hdev);

#endif

struct device\_driver **driver**; /\*通用驱动device\_driver结构体成员\*/

};

hid\_driver结构体主要成员简介如下：

●**id\_table：**指向hid\_device\_id结构体数组，定义如下（/include/linux/mod\_devicetable.h），表示匹配设备的ID：

struct hid\_device\_id {

\_\_u16 bus; /\*总线编号，如BUS\_USB，/include/uapi/linux/input.h\*/

\_\_u16 group;

\_\_u32 vendor;

\_\_u32 product;

kernel\_ulong\_t driver\_data;

};

●**dyn\_list**：链接由写new\_id属性添加的hid\_dynid实例（含hid\_device\_id结构体，/drivers/hid/hid-core.c），表示驱动匹配设备的ID。

##### ■hid\_ll\_driver

hid\_ll\_driver是底层驱动回调函数，结构体定义如下（/include/linux/hid.h）：

struct hid\_ll\_driver {

int (\*start)(struct hid\_device \*hdev); /\*由驱动hid\_driver中probe()函数调用\*/

void (\*stop)(struct hid\_device \*hdev); /\*remove()函数调用\*/

int (\*open)(struct hid\_device \*hdev); /\*输入层打开操作调用\*/

void (\*close)(struct hid\_device \*hdev); /\*输入层关闭操作调用\*/

int (\*power)(struct hid\_device \*hdev, int level);

int (\*parse)(struct hid\_device \*hdev);

void (\*request)(struct hid\_device \*hdev,struct hid\_report \*report, int reqtype);

int (\*wait)(struct hid\_device \*hdev);

int (\***raw\_request**) (struct hid\_device \*hdev, unsigned char reportnum,\_\_u8 \*buf, size\_t len,

unsigned char rtype,int reqtype); /\*必须定义的函数 \*/

int (\*output\_report) (struct hid\_device \*hdev, \_\_u8 \*buf, size\_t len);

int (\*idle)(struct hid\_device \*hdev, int report, int idle, int reqtype);

};

#### 2注册驱动

注册/注销hid\_driver实例的函数声明如下（/include/linux/hid.h）：

#define hid\_register\_driver(driver) \

**\_\_hid\_register\_driver**(driver, THIS\_MODULE, KBUILD\_MODNAME)

void hid\_unregister\_driver(struct hid\_driver \*)：注销hid\_driver实例函数。

实质的注册hid\_driver实例函数\_\_hid\_register\_driver()定义如下（/drivers/hid/hid-core.c）：

int \_\_hid\_register\_driver(struct hid\_driver \*hdrv, struct module \*owner,const char \*mod\_name)

{

int ret;

hdrv->driver.name = hdrv->name;

hdrv->driver.bus = &**hid\_bus\_type**; /\*总线\*/

hdrv->driver.owner = owner;

hdrv->driver.mod\_name = mod\_name;

INIT\_LIST\_HEAD(&hdrv->dyn\_list);

spin\_lock\_init(&hdrv->dyn\_lock);

ret = **driver\_register(&hdrv->driver)**; /\*注册驱动\*/

if (ret)

return ret;

ret = driver\_create\_file(&hdrv->driver, &driver\_attr\_**new\_id**); /\*添加属性文件\*/

if (ret)

driver\_unregister(&hdrv->driver);

return ret;

}

注册hid\_driver实例的函数比较简单，主要是将驱动关联到hid\_bus\_type总线，注册内嵌device\_driver结构体成员，为驱动添加/创建new\_id属性文件。

new\_id属性是一个只写属性，用于添加驱动区配设备的ID，并探测设备。写属性函数将创建hid\_dynid结构体实例，添加到 hid\_driver实例**dyn\_list**双链表。

#### 3创建/添加设备

真实的HID设备可能挂载在USB总线、I2C总线等，HID总线是一个虚拟的总线，用于管理系统所有的HID类设备和驱动。在USB总线、I2C总线等设备匹配驱动的probe()函数中，需要为设备分配并添加表示HID设备的 hid\_device实例。

##### ■分配hid\_device

**hid\_allocate\_device()**函数是分配hid\_device实例的接口函数，定义如下（/drivers/hid/hid-core.c）：

struct hid\_device \*hid\_allocate\_device(void)

{

struct hid\_device \*hdev;

int ret = -ENOMEM;

hdev = kzalloc(sizeof(\*hdev), GFP\_KERNEL); /\*分配hid\_device实例\*/

... /\*错误处理\*/

device\_initialize(&hdev->dev);

hdev->dev.release = hid\_device\_release;

hdev->dev.bus = **&hid\_bus\_type;**  /\*HID总线\*/

hid\_close\_report(hdev);

init\_waitqueue\_head(&hdev->debug\_wait);

INIT\_LIST\_HEAD(&hdev->debug\_list);

spin\_lock\_init(&hdev->debug\_list\_lock);

sema\_init(&hdev->driver\_lock, 1);

sema\_init(&hdev->driver\_input\_lock, 1);

return hdev;

}

##### ■添加hid\_device实例

**hid\_add\_device()**函数是添加hid\_device实例的接口函数，定义如下（/drivers/hid/hid-core.c）：

int hid\_add\_device(struct hid\_device \*hdev)

{

static atomic\_t id = ATOMIC\_INIT(0);

int ret;

if (WARN\_ON(hdev->status & HID\_STAT\_ADDED))

return -EBUSY;

if (hid\_ignore(hdev)) /\*检查是否忽然这个设备，/drivers/hid/hid-core.c\*/

return -ENODEV;

if (!hdev->ll\_driver->**raw\_request**) { /\*必须定义raw\_request()函数\*/

hid\_err(hdev, "transport driver missing .raw\_request()\n");

return -EINVAL;

}

ret = hdev->ll\_driver->**parse(hdev)**; /\*分析设备报告描述字，成功返回0\*/

...

if (!**hdev->dev\_rdesc**) /\*hdev->dev\_rdesc不能为空，报告描述字\*/

return -ENODEV;

/\*扫描组信息，hid\_ignore\_special\_drivers默认值为0\*/

if (hid\_ignore\_special\_drivers ||(!hdev->group &&!hid\_match\_id(hdev, hid\_have\_special\_driver))) {

**ret = hid\_scan\_report(hdev)**; /\*正确返回0\*/

...

}

dev\_set\_name(&hdev->dev, "%04X:%04X:%04X.%04X", hdev->bus,

hdev->vendor, hdev->product, atomic\_inc\_return(&id)); /\*设置设备名称\*/

hid\_debug\_register(hdev, dev\_name(&hdev->dev));

ret = **device\_add**(&hdev->dev); /\*添加设备\*/

if (!ret)

hdev->status |= HID\_STAT\_ADDED;

else

hid\_debug\_unregister(hdev);

return ret; /\*成功返回0\*/

}

#### 4 HID总线

HID总线实例定义如下（/drivers/hid/hid-core.c）：

static struct bus\_type **hid\_bus\_type** = {

.name = "hid",

.dev\_groups = hid\_dev\_groups,

.match = **hid\_bus\_match**, /\*匹配函数\*/

.probe = **hid\_device\_probe**, /\*探测函数\*/

.remove = hid\_device\_remove,

.uevent = hid\_uevent,

};

初始化函数hid\_init()主要工作就是注册hid\_bus\_type实例，源代码请读者自行阅读。

##### ■匹配函数

HID总线设备与驱动匹配函数hid\_bus\_match()定义如下（/drivers/hid/hid-core.c）：

static int hid\_bus\_match(struct device \*dev, struct device\_driver \*drv)

{

struct hid\_driver \*hdrv = container\_of(drv, struct hid\_driver, driver);

struct hid\_device \*hdev = container\_of(dev, struct hid\_device, dev);

return **hid\_match\_device**(hdev, hdrv) != NULL; /\*匹配函数\*/

}

hid\_match\_device()函数定义如下：

static const struct hid\_device\_id \*hid\_match\_device(struct hid\_device \*hdev,struct hid\_driver \*hdrv)

{

struct hid\_dynid \*dynid;

spin\_lock(&hdrv->dyn\_lock);

list\_for\_each\_entry(dynid, &hdrv->**dyn\_list**, list) { /\*遍历hid\_dynid实例\*/

if (**hid\_match\_one\_id**(hdev, &dynid->id)) {

spin\_unlock(&hdrv->dyn\_lock);

return &dynid->id;

}

}

spin\_unlock(&hdrv->dyn\_lock);

return **hid\_match\_id(hdev, hdrv->id\_table)**; /\*返回匹配hid\_device\_id实例\*/

}

hid\_match\_device()函数首先遍历hdrv->dyn\_list双链表中hid\_dynid实例，调用hid\_match\_one\_id()函数检查是否与其hid\_device\_id成员匹配。

hid\_match\_one\_id()函数用于检查hid\_device实例是否与hid\_device\_id列表项匹配，函数定义如下：

static bool hid\_match\_one\_id(struct hid\_device \*hdev,const struct hid\_device\_id \*id)

{

return (id->bus == HID\_BUS\_ANY || id->bus == hdev->bus) &&

(id->group == HID\_GROUP\_ANY || id->group == hdev->group) &&

(id->vendor == HID\_ANY\_ID || id->vendor == hdev->vendor) &&

(id->product == HID\_ANY\_ID || id->product == hdev->product); /\*4项都要检查\*/

}

若hid\_device实例与hdrv->dyn\_list双链表中实例没有匹配项，则再调用hid\_match\_id()函数检查HID设备与hdrv->id\_table列表是否有匹配的项。

hid\_match\_id(hdev, hdrv->id\_table)函数遍历hdrv->id\_table列表中hid\_device\_id实例检查其与HID设备是否匹配，检查函数也是hid\_match\_one\_id()函数，见上文。

##### ■探测函数

添加设备或驱动时，将在挂载的总线上查找匹配的驱动或设备（调用总线匹配函数），若设备与驱动匹配将调用总线（或驱动）的探测函数。

HID总线探测函数hid\_device\_probe()定义如下（/drivers/hid/hid-core.c）：

static int hid\_device\_probe(struct device \*dev)

{

struct hid\_driver \*hdrv = container\_of(dev->driver,struct hid\_driver, driver);

struct hid\_device \*hdev = container\_of(dev, struct hid\_device, dev);

const struct hid\_device\_id \*id;

int ret = 0;

if (down\_interruptible(&hdev->driver\_lock))

return -EINTR;

if (down\_interruptible(&hdev->driver\_input\_lock)) {

ret = -EINTR;

goto unlock\_driver\_lock;

}

hdev->io\_started = false;

if (!**hdev->driver**) {

id = **hid\_match\_device**(hdev, hdrv);

... /\*错误处理\*/

**hdev->driver = hdrv**;

if (hdrv->probe) {

ret = hdrv->probe(hdev, id);

} else { /\* default probe \*/

ret = **hid\_open\_report(hdev)**;

if (!ret)

ret = **hid\_hw\_start**(hdev, HID\_CONNECT\_DEFAULT);

}

if (ret) {

hid\_close\_report(hdev);

hdev->driver = NULL;

}

}

unlock:

if (!hdev->io\_started)

up(&hdev->driver\_input\_lock);

unlock\_driver\_lock:

up(&hdev->driver\_lock);

return ret;

}

### 9.7.3 USB\_HID驱动