20.3.4 USB骨架程序(1)

20.3.4 USB骨架程序(1)

Linux内核源代码中的driver/usb/usb-skeleton.c文件为我们提供了一个最基础的USB驱动程序，即USB骨架程序，可被看做一个最简单的USB设备驱动实例。尽管具体USB设备驱动千差万别，但其骨架则万变不离其宗。

首先看看USB骨架程序的usb_driver结构体定义，如代码清单20.16所示。

代码清单20.16 USB骨架程序的usb_driver结构体

```
1 static struct usb_driver skel_driver =
2 {
3.name =  'skeleton',
4.probe = skel_probe,
5.disconnect = skel_disconnect,
6.id_table = skel_table,
7 };
```

从上述代码第6行可以看出，它所支持的USB设备的列表数组为skel_table[]，其定义如代码清单20.17所示。

代码清单20.17 USB骨架程序的id_table

```
1  /* 本驱动支持的USB设备列表 */
2  static struct usb_device_id skel_table [] =
{
3      { USB_DEVICE(USB_SKEL_VENDOR_ID,
4        USB_SKEL_PRODUCT_ID) },
5      { }
6  };
7  MODULE_DEVICE_TABLE (usb, skel_table);
```

对上述usb_driver的注册和注销发生在USB骨架程序的模块加载与卸载函数内，如代码清单20.18所示，其分别调用了usb_register()和usb_deregister()。

代码清单20.18 USB骨架程序的模块加载与卸载函数

```
1  static int __init usb_skel_init(void)
2  {
3 int result;
4
5 /* 注册USB驱动 */
6 result = usb_register(&skel_driver);
7 if (result)
8  err('usb_register failed. Error number %d',
result);
9
10return result;
11 }
12 static void __exit usb_skel_exit(void)
13 {
14    /* 注销USB驱动 */
15    usb_deregister(&skel_driver);
```

```
16 }
```

usb_driver的probe()成员函数中，会根据usb_interface的成员寻找第一个批量输入和输出端点，将端点地址、缓冲区等信息存入为USB骨架程序定义的usb_skel结构体，并将usb_skel实例的指针传入usb_set_intfdata()作为USB接口的私有数据，最后，它会注册USB设备，如代码清单20.19所示。

代码清单20.19 USB骨架程序的探测函数

```
1  static int skel_probe(struct usb_interface
*interface, const struct usb_device_id *id)
2  {
3 struct usb_skel *dev = NULL;
4 struct usb_host_interface *iface_desc;
5 struct usb_endpoint_descriptor *endpoint;
6 size_t buffer_size;
7 int i;
8 int retval = -ENOMEM;
9
10/* 分配设备状态的内存并初始化 */
11dev = kzalloc(sizeof(*dev), GFP_KERNEL);
12if (dev == NULL) {
13 err('Out of memory');
14 goto error;
15}
16kref_init(&dev->kref);
17sema_init(&dev->limit_sem,
WRITES_IN_FLIGHT);
18
19dev->udev =
usb_get_dev(interface_to_usbdev(interface));
20dev->interface = interface;
21
22/* 设置端点信息 */
23/* 仅使用第一个bulk-in和bulk-out */
24iface_desc = interface->cur_altsetting;
25for (i = 0; i < iface_desc-
>desc.bNumEndpoints; ++i) {
26 endpoint = &iface_desc->endpoint[i].desc;
27
28 if (!dev->bulk_in_endpointAddr &&
29     ((endpoint->bEndpointAddress &
USB_ENDPOINT_DIR_MASK)
30    == USB_DIR_IN) &&
31     ((endpoint->bmAttributes &
USB_ENDPOINT_XFERTYPE_MASK)
32    == USB_ENDPOINT_XFER_BULK)) {
33  /* 找到了一个批量IN端点 */
34  buffer_size = le16_to_cpu(endpoint-
>wMaxPacketSize);
35  dev->bulk_in_size = buffer_size;
36  dev->bulk_in_endpointAddr = endpoint-
>bEndpointAddress;
37  dev->bulk_in_buffer = kmalloc(buffer_size,
GFP_KERNEL);
38  if (!dev->bulk_in_buffer) {
39   err('Could not allocate bulk_in_buffer');
40   goto error;
41  }
42 }
43
44 if (!dev->bulk_out_endpointAddr &&
45     ((endpoint->bEndpointAddress &
USB_ENDPOINT_DIR_MASK)
46    == USB_DIR_OUT) &&
47     ((endpoint->bmAttributes &
```

```
USB_ENDPOINT_XFERTYPE_MASK)
48    == USB_ENDPOINT_XFER_BULK)) {
49  /* 找到了一个批量OUT端点 */
50  dev->bulk_out_endpointAddr = endpoint-
>bEndpointAddress;
51 }
52}
53if (!(dev->bulk_in_endpointAddr && dev-
>bulk_out_endpointAddr)) {
54 err('Could not find both bulk-in and bulk-
out endpoints');
55 goto error;
56}
57
58/* 在设备结构中保存数据指针 */
59usb_set_intfdata(interface, dev);
60
61/* 注册USB设备 */
62retval = usb_register_dev(interface,
&skel_class);
63if (retval) {
64 /* something prevented us from registering
this driver */
65 err('Not able to get a minor for this
device.');
66 usb_set_intfdata(interface, NULL);
67 goto error;
68}
69
70/* 告知用户设备依附于什么node */
71info('USB Skeleton device now attached to
USBSkel-%d', interface->minor);
72return 0;
73
74 error:
75if (dev)
76 kref_put(&dev->kref, skel_delete);
77return retval;
78 }
```

usb_skel结构体可以被看作一个私有数据结构体，其定义如代码清单20.20所示，应该根据具体的设备量身定制。

代码清单20.20 USB骨架程序的自定义数据结构usb_skel

```
1  struct usb_skel
2  {
3 struct usb_device * udev;    /* 该设备
的usb_device指针 */
4 struct usb_interface * interface;  /* 该设备
的usb_interface指针 */
5 struct semaphore limit_sem;    /* 限制进程写的
数量 */
6 unsigned char *  bulk_in_buffer;  /* 接收数据
的缓冲区 */
7 size_t   bulk_in_size;    /* 接收缓冲区大小 */
8 _ _u8  bulk_in_endpointAddr;   /*批量IN端点的地
址 */
9 _ _u8  bulk_out_endpointAddr;  /* 批量OUT端点
的地址 */
10struct kref  kref;
11 };
```

USB骨架程序的断开函数会完成探测函数相反的工作，即设置接口数据为NULL，注销USB设备，如代码清单20.21所示。

代码清单20.21 USB骨架程序的断开函数

```
1  static void skel_disconnect(struct
usb_interface *interface)
2  {
3 struct usb_skel *dev;
4 int minor = interface->minor;
5
6 /* 阻止skel_open()与skel_disconnect()的竞争 */
7 lock_kernel();
8
9 dev = usb_get_intfdata(interface);
10usb_set_intfdata(interface, NULL);
11
12/* 注销usb设备，释放次设备号 */
13usb_deregister_dev(interface, &skel_class);
14
15unlock_kernel();
16
17/* 减少引用计数 */
18kref_put(&dev->kref, skel_delete);
19
20info('USB Skeleton #%d now disconnected',
minor);
21 }
```