浅析 libusb 控制接口和 mountd 守护进程处理 uevent 切换 usb 设备的实现

浅析 libusb 控制接口和 mountd 守护进程处理 uevent 切换 usb 设备的实现

1. UMS mode (USB Mass-Storage mode) [ums]

2. 从 usb_gadget_register_driver 的实现来看, insmod xxx . ko, 然后重新插拔一下 usb cable , 那么 pc 再次枚举到的设备就是 insmod xxx . ko 对应的设备了 [luther . gliethttp]

3. init 进程没有对 change event 事件进行处理, mountd 守护进程的 detect_thread 线程会等待该 uevent 事件到来, 然后卸载前一个 ko, 加载欲成为设备的相应 ko 驱动 [luther . gliethttp].

4. 其实 uevent 已经在 udc_uevent 中将 ret 数值打入了 uevent strings 中了 MODE=;NET_ENUM=;TESTMODE=;直接解析就可以了, 所以就不需要 read_usb_switch 了 [luther . gliethttp].

5. libusb 库链接程序时, 可以使用 -static -lusb 选项, 将 libusb 静态编译到程序中, 这样其他 pc 就不用单独安装 libusb 了 [luther . gliethttp].

==
=

usb_init =>我的是 strncpy (usb_path , , sizeof (usb_path) - 1); 或者

usb_find_busses =>然后将 usb 目录下的所有文件目录路径名 bus->dirname 添加到 struct usb_bus *usb_busses = NULL;

链表上, 比如: /dev/bus/usb 下的 001 和 002 目录等.

usb_find_devices =>根据文件目录路径名遍历 usb_busses 文件夹下的所有文件对应的 char 节点, 如果合法将 dev 添加 bus->devices 设备链表上.

同时将 char 节点的文件路径名作为访问文件节点的路径名存储起来.

//比如打开 host 控制器的 2 号 hub 下插入的第 1 个设备

```
    struct usb_device *right_dev;
  right_dev = NULL;

    for (bus = usb_get_busses(); bus; bus = bus->next) {
     struct usb_device *dev;

     for (dev = bus->devices; dev; dev = dev->next) {
     if (dev->descriptor.idVendor == vendor &&
dev->descriptor.idProduct == product) {
     right_dev = dev;
        DevicesN ++;
     }
        }
    }

    return right_dev;
}
```

然后调用 usb_open 打开 find_device （）返回的 usb_device 设备，

```
usb_dev_handle *usb_open(struct usb_device *dev)
{
 usb_dev_handle *udev;

 udev = malloc(sizeof(*udev));
  if (!udev)
    return NULL;

 udev->fd = -1;
 udev->device = dev;
 udev->bus = dev->bus;
 udev ->config = udev ->interface = udev ->altsetting = -1;

  if (usb_os_open (udev) < 0) {
    free (udev);
    return NULL;
  }

  return udev ;
}
int usb_os_open (usb_dev_handle *dev)
{
 dev ->fd = device_open (dev->device );

  return 0;
}
static int device_open (struct usb_device *dev)
{
  char filename [PATH_MAX+1];
  int fd ;

 snprintf (filename , sizeof (filename ) - 1, ,
  usb_path , dev ->bus->dirname , dev ->filename );
//比如打开 host 控制器的 2 号 hub 下插入的第 1 个设备



 fd = open(filename , O_RDWR);
  if (fd < 0) {
  fd = open(filename , O_RDONLY);
```

```
    if (fd < 0)
    USB_ERROR_STR(-errno ,                         ,
   filename , strerror (errno ));
   }

   return fd ;
}
==================================================================
=
```

链接程序时,可以使用 -static -lusb 选项,将 libusb 静态编译到程序中,这样其他 pc 就不用单独安装 libusb 了[luther . gliethttp].

以下代码摘自 libusb -0. 1. 12

```
#define USB_MAXDRIVERNAME 255

struct usb_getdriver {
    unsigned int interface ;
    char driver [USB_MAXDRIVERNAME+1];
};
#define IOCTL_USB_CONTROL  _IOWR('U' , 0, struct usb_ctrltransfer )
#define IOCTL_USB_BULK     _IOWR('U' , 2, struct usb_bulktransfer )
#define IOCTL_USB_RESETEP  _IOR('U' , 3, unsigned int )
#define IOCTL_USB_SETINTF  _IOR('U' , 4, struct usb_setinterface )
#define IOCTL_USB_SETCONFIG  _IOR('U' , 5, unsigned int )
#define IOCTL_USB_GETDRIVER  _IOW('U' , 8, struct usb_getdriver )
#define IOCTL_USB_SUBMITURB  _IOR('U' , 10, struct usb_urb )
#define IOCTL_USB_DISCARDURB  _IO('U' , 11)
#define IOCTL_USB_REAPURB  _IOW('U' , 12, void *)
#define IOCTL_USB_REAPURBNDELAY  _IOW('U', 13, void *)
#define IOCTL_USB_CLAIMINTF  _IOR('U' , 15, unsigned int )
```

```
#define IOCTL_USB_RELEASEINTF _IOR('U', 16, unsigned int)
#define IOCTL_USB_CONNECTINFO _IOW('U', 17, struct usb_connectinfo)
#define IOCTL_USB_IOCTL _IOWR('U', 18, struct usb_ioctl)
#define IOCTL_USB_HUB_PORTINFO _IOR('U', 19, struct
usb_hub_portinfo)
#define IOCTL_USB_RESET     _IO ('U', 20)
#define IOCTL_USB_CLEAR_HALT _IOR('U', 21, unsigned int)
#define IOCTL_USB_DISCONNECT _IO('U', 22)
#define IOCTL_USB_CONNECT _IO('U', 23)
int usb_reset (usb_dev_handle *dev)
{
  int ret;

 ret = ioctl (dev->fd, IOCTL_USB_RESET, NULL);
  if (ret)
   USB_ERROR_STR(-errno,                         , strerror (errno));

  return 0;
}

int usb_get_driver_np (usb_dev_handle *dev, int interface, char *name,
     unsigned int namelen)
{
  struct usb_getdriver getdrv;
  int ret;

 getdrv.interface = interface;
 ret = ioctl (dev->fd, IOCTL_USB_GETDRIVER, &getdrv);
  if (ret)
```

```
   USB_ERROR_STR(-errno ,                                   ,
strerror  (errno ));

   strncpy (name, getdrv .driver , namelen - 1);
  name [namelen - 1] = 0;

   return  0;
}

int usb_detach_kernel_driver_np    (usb_dev_handle  *dev, int interface  )
{
   struct  usb_ioctl command  ;
   int ret ;

  command. ifno  = interface   ;
  command. ioctl_code  = IOCTL_USB_DISCONNECT;
  command. data  = NULL;

  ret  = ioctl  (dev->fd , IOCTL_USB_IOCTL, &command);
   if  (ret )
   USB_ERROR_STR(-errno ,
                        ,
      interface         , strerror  (errno ));

   return  0;
}
===============================================================
=
devh = usb_open (dev);
```

```
ret = usb_get_driver_np （devh， 0， buf ， sizeof （buf））；
ret = usb_detach_kernel_driver_np （devh， 0）；//断开设备
ret = usb_claim_interface （devh， 0）；//改变一个 usb 设备的接口，一个接口就是一个独立的功能
ret = usb_set_altinterface （devh， 0）；//切换的实际动作并不在这里执行，而是由处理 uevent 事件的 mountd 完成[luther.gliethttp].
ret = usb_bulk_write （devh，endpoint ，message ，length ， 0）；
ret = usb_release_interface （devh， 0）；
ret = usb_close （devh）；

int usb_bulk_write （usb_dev_handle *dev，int ep，char *bytes ，int size ，
    int timeout ）
{
  /* Ensure the endpoint address is correct */
  return usb_urb_transfer （dev，ep ，USB_URB_TYPE_BULK，bytes ，size ，
    timeout ）；
}
=>usb_urb_transfer
=>ret = ioctl （dev->fd ，IOCTL_USB_SUBMITURB，&urb）；//提交写操作
```

usb_detach_kernel_driver_np

=>ioctl （dev->fd ，IOCTL_USB_IOCTL，&command）；

=>kernel 中调用 usb_driver_release_interface

=>来将 dev 和 driver 拆开，同时 device_is_registered 如果有匹配的 driver 了=>device_release_driver 释放和 dev 匹配上的 driver 彼此链表.

```
int usb_claim_interface （usb_dev_handle *dev，int interface ）
{
  int ret ；
```

```
 ret   = ioctl  (dev->fd , IOCTL_USB_CLAIMINTF, &interface  );
  if  (ret  < 0)  {
    if  (errno  == EBUSY&& usb_debug  > 0)
      fprintf  (stderr  ,        that  you have permissions  to  write  to  %s/%s
and, if you don't, that you set up hotplug

                                                                ,
dev->bus->dirname , dev ->device ->filename );

  USB_ERROR_STR(-errno ,         not  claim  interface   %d:    , interface   ,
    strerror   (errno ));
  }

 dev  ->interface    = interface    ;

  return   0;
}
```

usb_claim_interface

=>ioctl (dev->fd , IOCTL_USB_CLAIMINTF, &interface);

=>case USBDEVFS_CLAIMINTERFACE

=>kernel 中调用 proc_claiminterface 来重新设定 usb 设备的接口,接口信息有 usb 设备描述符和接口描述符中指定

```
int  usb_set_altinterface     (usb_dev_handle  *dev,  int  alternate   )
{
  int  ret  ;
  struct   usb_setinterface setintf      ;

  if  (dev->interface    <  0)
```

```
                (-EINVAL);

 setintf   . interface    = dev ->interface   ;
 setintf   . altsetting    = alternate   ;

 ret   = ioctl  (dev->fd , IOCTL_USB_SETINTF, &setintf  ) ;
   if  (ret  <  0)
   USB_ERROR_STR(-errno ,                                    ,
   dev   ->interface  , alternate   ,  strerror   (errno )) ;

 dev  ->altsetting    = alternate   ;

   return   0;
}
```

usb_set_altinterface

=>ioctl (dev->fd , IOCTL_USB_SETINTF, &setintf) ;

=>case USBDEVFS_SETINTERFACE:

=>kernel proc_setintf 改变 kernel 中对该 usb 设备的接口序号和描述符,同时 u

```
 sb_control_msg   (dev, usb_sndctrlpipe    (dev,  0) ,
         USB_REQ_SET_INTERFACE      , USB_RECIP_INTERFACE,
         alternate           , interface   ,  NULL,  0,  5000) ;
```

下发数据到 usb 设备,让设备改变相应接口对应的驱动程序,这样设备当断开 usb 总线或者 hub 发送 reset 复位总线时,usb

设备就会发送指定接口对应的接口描述符下的 endpoint 端点信息供 kernel 使用.

===
=

include /linux /usbdevice_fs . h

```
 define   USBDEVFS_MAXDRIVERNAME 255

struct   usbdevfs_getdriver      {
      unsigned   int  interface    ;
      char  driver   [USBDEVFS_MAXDRIVERNAME + 1];
};
#define   USBDEVFS_CONTROL _IOWR('U', 0, struct   usbdevfs_ctrltransfer       )
#define   USBDEVFS_BULK _IOWR('U', 2, struct   usbdevfs_bulktransfer      )
#define   USBDEVFS_RESETEP _IOR('U' , 3, unsigned   int )
#define   USBDEVFS_SETINTERFACE _IOR('U', 4, struct
usbdevfs_setinterface      )
#define   USBDEVFS_SETCONFIGURATION _IOR('U', 5, unsigned   int )
#define   USBDEVFS_GETDRIVER _IOW('U', 8, struct   usbdevfs_getdriver     )
#define   USBDEVFS_SUBMITURB _IOR('U', 10, struct   usbdevfs_urb   )
#define   USBDEVFS_SUBMITURB32 _IOR('U', 10, struct   usbdevfs_urb32   )
#define   USBDEVFS_DISCARDURB _IO('U', 11)
#define   USBDEVFS_REAPURB _IOW('U', 12, void   *)
#define   USBDEVFS_REAPURB32 _IOW('U', 12, __u32 )
#define   USBDEVFS_REAPURBNDELAY _IOW('U', 13, void   *)
#define   USBDEVFS_REAPURBNDELAY32 _IOW('U', 13, __u32 )
#define   USBDEVFS_DISCSIGNAL _IOR('U' , 14, struct
usbdevfs_disconnectsignal      )
#define   USBDEVFS_CLAIMINTERFACE _IOR('U', 15, unsigned   int )
#define   USBDEVFS_RELEASEINTERFACE _IOR('U', 16, unsigned   int )
#define   USBDEVFS_CONNECTINFO _IOW('U' , 17, struct   usbdevfs_connectinfo    )
#define   USBDEVFS_IOCTL _IOWR('U' , 18, struct   usbdevfs_ioctl      )
#define   USBDEVFS_IOCTL32 _IOWR('U' , 18, struct   usbdevfs_ioctl32     )
#define   USBDEVFS_HUB_PORTINFO _IOR('U', 19, struct
usbdevfs_hub_portinfo     )
```

```
 define  USBDEVFS_RESET _IO('U' , 20)
#define  USBDEVFS_CLEAR_HALT _IOR('U', 21, unsigned  int )
#define  USBDEVFS_DISCONNECT _IO('U', 22)
#define  USBDEVFS_CONNECT _IO('U', 23)
static   int  proc_getdriver  (struct  dev_state  *ps , void __user  *arg )
{
     struct  usbdevfs_getdriver gd   ;
     struct  usb_interface   *intf  ;
     int  ret  ;

     if  (copy_from_user (&gd, arg , sizeof  (gd)))
          return  -EFAULT;
  intf    = usb_ifnum_to_if  (ps->dev, gd . interface  );
     if  (!intf   || !intf  ->dev. driver  )
     ret      = -ENODATA;
     else  {
          strncpy  (gd. driver  , intf  ->dev. driver  ->name,
                    sizeof  (gd. driver  ));
     ret      = (copy_to_user  (arg , &gd, sizeof  (gd)) ? -EFAULT: 0);
     }
     return  ret  ;
}

static   int  usbdev_ioctl  (struct  inode  *inode , struct   file   *file  ,
               unsigned  int  cmd, unsigned  long  arg )
{
     struct  dev_state   *ps  = file  ->private_data   ;
     struct  usb_device   *dev  = ps ->dev;
     void __user   *p  = (void __user   *) arg ;
```

```
    int ret = -ENOTTY;

    if (!(file->f_mode & FMODE_WRITE))
        return -EPERM;
  usb_lock_device (dev);
    if (!connected (ps)) {
    usb_unlock_device (dev);
        return -ENODEV;
    }

    switch (cmd) {
    case USBDEVFS_CONTROL:
    . . .
}
const struct file_operations usbdev_file_operations = {
    . owner = THIS_MODULE,
    . llseek = usbdev_lseek ,
    . read = usbdev_read ,
    . poll = usbdev_poll ,
    . ioctl = usbdev_ioctl ,
    . open = usbdev_open ,
    . release = usbdev_release ,
};
int __init usb_devio_init (void )
{
    . . .
#define USB_DEVICE_DEV MKDEV(USB_DEVICE_MAJOR, 0)
#define USB_MAJOR 180
#define USB_DEVICE_MAJOR 189
```

```
 define  USB_MAXBUS         64
#define  USB_DEVICE_MAX    USB_MAXBUS*128
   retval      = register_chrdev_region      (USB_DEVICE_DEV, USB_DEVICE_MAX,
                                             );
   cdev_init      (&usb_device_cdev  ,  &usbdev_file_operations      );
   retval      = cdev_add  (&usb_device_cdev  , USB_DEVICE_DEV,
USB_DEVICE_MAX);//                     cdev_map
      . . .
}
drivers  /usb/core /usb. c
static    int  __init usb_init       (void )
{
      . . .
   retval      = usb_devio_init     () ;
      . . .
}
subsys_initcall     (usb_init  ) ;
=================================================================
=
drivers  /usb/gadget /file_storage    . c
static    int  __init fsg_init       (void )
{
      int        rc       ;
      struct    fsg_dev         *fsg ;

      if   ((rc  = fsg_alloc    ())  != 0)
            return   rc  ;
   fsg      = the_fsg   ;
      if   ((rc  = usb_gadget_register_driver        (&fsg_driver   ))  != 0)
```

```
                    (&fsg->ref , fsg_release );
    return rc ;
}
module_init (fsg_init );
static struct usb_gadget_driver fsg_driver = {
    ...
    . bind          = fsg_bind ,
    ...
};
fsg_bind
=>fsg->thread_task = kthread_create (fsg_main_thread , fsg ,
                          );
=>fsg_main_thread
=>do_scsi_command

static int do_scsi_command (struct fsg_dev *fsg )
{
    ...
#if defined (CONFIG_USB_MODE_SWITCH)
    case SC_USB_MODESWITCH:

        if (fsg->cmnd[11]==0x35) //switch to usbnet
        {
      printk          (                                          ,
fsg->cmnd[11]);
      udc_sysfs_data          . mode=USB_SWITCH_CMD;
      udc_kobject_uevent          (&udc_sysfs_data , KOBJ_CHANGE);
//init 进程没有对 change event 事件进行处理,mountd 守护进程的
detect_thread 线程会等待该 uevent 事件到来,然后
```