FreeBSD 设备驱动动程序开发讨论

马勇
mayong@ercist.iscas.ac.cn

基础软件中心

November 21, 2005

Outline

Outline

FreeBSD驱动程序的组成

- 必要的数据结构(假设设备名为mypci,下同)
  - 包括一些设备的描述以及与操作系统的接口，主要有：mypci_softc, mypci_cdevsw（字符设备), mypci_methods[], mypci_driver.
- 通用总线子系统
  - 对内
  - 主要功能是对底层的设备进行操作，包括探测识别，初始化，驱动加载，卸载，关闭，挂起恢复等。
- 设备访问入口
  - 对外
  - 主要是对一些应用程序对设备操作进行支持，包括设备的打开、关闭、读写以及IO控制。是驱动程序对外的接口。

一个设备驱动程序的框架

```
#include ...
struct mypci_softc {...};
static devclass_t mypci_devclass;
static struct cdevsw mypci_cdevsw = {
.d_open = mypci_open,...};
static int mypci_open(...){...};
...close(),read(),write(),ioctl()...
-----------------------------------------------------------
static int mypci_probe(device_t dev){...};
...attach(),detach(),shutdown()...
static device_method_t mypci_methods[]={
DEVMETHOD(device_probe, mypci_probe),...};
static driver_t mypci_driver = {
"mypci", mypci_methods, sizeof(struct mypci_softc),};
DRIVER_MODULE(mypci, pci, mypci_driver, mypci_devclass, 0, 0);
```

驱动程序中各个函数的执行顺序

- probe—>attach
- —>open—>(read/write/ioctl)—>close
- —>(suspend/resume)—>(detach/shutdown)

Outline

softc

- mypci_softc:
  - 包含私有的驱动程序数据和驱动程序资源的描述符。总线代码会自动按需要为每个设备分配一个softc描述符。驱动程序用此结构来存储一些设备在系统中的信息，如：内存资源，终端，I/O端口资源，以及各种地址和标记。在驱动程序加载过程中为其赋值。
  - struct mypci_softc {
    bus_space_tag_t bst;
    bus_space_handle_t bsh;
    dev_t dev0;
    dev_t dev1;
    u_int32_t open_mask;
    u_int32_t read_mask;
    struct resource *res;
    int rid; };

cdevsw

- mypci_cdevsw:
  - 指定字符设备提供的接口例程，每个设备提供5 个例程: open、close、read、write、和特殊功能(用来实现ioctl 系统调用)。可以缺少其中任何一个。如果应该忽略在这个例程上的一个调用，(比如，在不需要设置的一个非独占设备上的open) cdevsw 条目可以给出为nulldev; 如果它应该被考虑为一个错误，(比如在一个只读设备上写)则使用nodev.
  - static struct cdevsw mypci_cdevsw = {
    .d_version = D_VERSION,
    .d_name = "mypci",
    .d_flags = D_NEEDGIANT,
    .d_open = mypci_open,
    .d_close = mypci_close,
    .d_read = mypci_read,
    .d_write = mypci_write,
    .d_ioctl = mypci_ioctl};

methods

- mypci_methods[]:
  - 定义驱动程序加载卸载过程的各个函数：设备的识别和探测，驱动加载、卸载，设备的挂起恢复，以及设备的关闭。其中设备的识别探测和设备的加载必须包含，其他的按照需要设定。
  - static device_method_t mypci_methods[] = {
    DEVMETHOD(device_identify, mypci_identify),
    DEVMETHOD(device_probe, mypci_probe),
    DEVMETHOD(device_attach, mypci_attach),
    DEVMETHOD(device_detach, mypci_detach),
    0, 0 };

driver

- mypci_driver:
  - 即设备的驱动程序。关联驱动程序名和methods[]，定义驱动程序的大小（一般为softc结构的大小）。
  - static driver_t skeleton_driver = {
    "skeleton",
    skeleton_methods,
    sizeof(struct skeleton_softc), };

Outline

探测与识别：probe

函数probe在已知（或pnp）位置探测设备。对于已经部分配置的设备，这个例程也能够提供设备特定的对某些参数的自动侦测。设备描述符结构mypci_softc由系统在调用探测例程之前分配。如果探测例程返回错误，描述符会被系统自动取消分配。因此如果出现探测错误，驱动程序必须保证取消分配探测期间它使用的所有资源，且确保没有什么能够阻止描述符被安全地取消分配。如果探测成功完成，描述符将由系统保存并在以后传递给例程mypci_attach()。

Sjy22b加密卡的探测函数

- static int sjy22b_probe(device_t dev){
  pci_get_vendor(dev), pci_get_device(dev));
  if (pci_get_vendor(dev) == 0x10b5) {
  printf("We've got the Cryptgraphic Card!");
  return (0);
  }
  return (ENXIO);
  }

注：如果设备不存在则必须返回值“ENXIO”。其他错误值可能表示其他条件。零或负值(只在有多个驱动都适合时出现)意味着成功。大多数驱动程序返回零表示成功。

连接与初始化：attach

如果探测例程返回成功并且系统选择连接那个驱动程序，则连接例程负责将驱动程序实际连接到系统，并进行设备的初始化工作。如果探测例程返回0 ，则连接例程期望接收完整的设备结构softc，此结构由探测例程设置。同时，如果探测例程返回0，它可能期望这个设备的连接例程应当在将来的某点被调用。在sjy22b的驱动中其主要功能有：I/O、内存资源的申请和激活（由于没有实用中断方式所以未申请中断资源），PCI寄存器及基地址寄存器(BAR)的初始化，softc结构的赋值。DMA buffer的分配以及在/dev中建立新的设备。这个函数是驱动程序中的核心部分，是设备能否正常工作和实现功能的关键。

一个例子: sjy22b的attach（简化版）

- static int sjy22b_attach(device_t dev){ ...
  res = bus_alloc_resource_any(dev, SYS_RES_IOPORT, &
  rid, RF_ACTIVE);
  card_base[0]= rman_get_bushandle(res);
  sc->res0 = res;
  ...
  outl(card_base[0] + 0xf0,0xfffe0000);
  outl(card_base[0]+0x04, 0x20000000);
  ...
  sc->dev = make_dev(& sjy22b_cdevsw, 0, UID_ROOT,
  GID_WHEEL, 0644, "sjy22b0");}

不必需的：detach, shutdown,suspend and resume

这些功能不是设备驱动程序必须的，属于可选项。其中detach函数用于可以动态加载卸载型的驱动程序，其功能是在模块卸载前分离设备，如果硬件支持热插拔，这是一个很重要的特性。；shutdown用于在系统关闭前关闭设备，对于大多数ISA和PCI设备而言不需要特殊动作，因此这个函数并非真正必需，因为不管怎样重启动时设备会被重新初始化。但有些设备必须按特定步骤关闭，以确保在软重启后能被正确地检测到（对于很多使用私有识别协议的设备特别有用）。很多情况下，需要在设备寄存器中禁用DMA和中断，并停止将要进行的传输；suspend用于在系统进入节能状态前挂起设备；resume用于从节能状态返回后恢复设备的活动状态。

- A pity: We didn't do a proper detach:(

资源类型

总线资源与每个设备相关联。通过类型和类型中的数字标识它们。对于一般总线总线，定义了下面的类型：

- SYS_RES_IRQ - 中断号
- SYS_RES_MEMORY - 映射到系统内存空间的设备内存的范围
- SYS_RES_IOPORT - 设备I/O寄存器的范围
- SYS_RES_DRQ - ISA DMA通道号

对资源的操作

对资源能够执行三种类型的动作：

- set/get
- allocate/release
- activate/deactivate

Set设置资源使用的范围。Allocation保留出请求的范围，使得其它设备不能再占用（并检查此范围没有被其它设备占用）。Activation执行必要的动作使得驱动程序可以访问资源（例如，对于内存，它将被映射到内核的虚拟地址空间）。ours:

- res = bus_alloc_resource_any(dev, SYS_RES_IOPORT, &rid, RF_ACTIVE);

操作资源的主要函数有：bus_set_resource（）设置资源范围；bus_get_resource（）获取资源范围；bus_delete_resource（）删除资源；bus_alloc_resource（）或bus_alloc_resource_any（）分配资源。

基地址寄存器

为了对PCI设备做些有用的事情，需要从PCI配置空间获取Base Address Registers (BARs)。获取BAR时的PCI特定的细节被抽象在函数bus_alloc_resource()中。

- res = bus_alloc_resource_any(dev, SYS_RES_MEMORY, &rid, RF_ACTIVE);
  sc->bar1_bt = rman_get_bustag(res);
  sc->bar1_bh = rman_get_bushandle(res);

每个基址寄存器的句柄被保存在softc 结构中，以便以后可以使用它们向设备写入。然后就能使用这些句柄与bus_space_*函数一起读写设备寄存器。

- bus_space_read_2(sc->bar1_bt, sc->bar1_bh, address);
- bus_space_write_2(sc->bar1_bt, sc->bar1_bh, address, value);

这些函数以8位，16位和32位的版本存在，应当相应地使用bus_space_read|write_1|2|4。

中断资源

中断按照和分配内存资源相似的方式从面向对象的总线代码分配。首先，必须从父总线分配IRQ资源，然后必须设置中断处理函数来处理这个IRQ。

- sc->irqres = bus_alloc_resource(dev, SYS_RES_IRQ, &(sc->irqid),0, 0, 1, RF_SHAREABLE RF_ACTIVE);

  ...

  error = bus_setup_intr(dev, sc->irqres, INTR_TYPE_MISC,my_handler, sc, &(sc->handler));

在设备的分离例程detach中必须注意一些问题。分离时必须停顿设备的中断流，并移除中断处理函数。一旦bus_teardown_intr()返回，必须保证中断处理函数不会再被调用，并且所有可能已经执行了这个中断处理函数的线程都已经返回。

Outline

一般操作：Open,close,read and write of device nodes

对于设备文件的一般操作的函数主要
是：open，close，read和write。
进程通过系统调用read()/write()对文件进行io处理，它由文件描述符指定对哪里进行i/o，文件描述符是0以上的整数，它在各个进程的struct proc的成员struct filedesc *p_fd(struct filedesc)(sys/filedesc.h))保留的struct file(sys/file.h)进行选择添加。对struct file，它含有从文件的头的输入输出的byte位置，输入操作，输出操作，输入输出控制，输入输出的准备状态的检查，执行close的routine，以及描述io处理对象的信息。

Open the device node

系统调用open()(kern/vfs_syscalls.c)就是把包含路径信息的v-node找寻出来，为了对它进行io处理，先要对struct file进行初始化，然后返回文件描述符。从路径名查找v-nodehe和io准备操作由vn_open()(kern/vfs_vnops.c)承担。vn_open()通过namei()(kern/vfs_lookup.c)查找路径对应的v-node名，由VOP_OPEN()调用不同的v-node定义的准备过程routine。对于字符设备文件则主要调用驱动程序中的open（）函数。设备文件打开以后对它的读写操作就是执行设备驱动中定义的mypci_read,mypci_write函数进行处理了。close于此类似。

特殊功能接口：Ioctl

对于设备的特殊功能，主要通过ioctl来实现。ioctl是应用程序对设备进行一些高级操作的接口，其函数结构为：

- static int sjy22b_ioctl(struct cdev *dev, u_long cmd, caddr_t addr, int flag, struct thread *td){

  ...
  switch(cmd){
  case MACRO1: ...
  case MACRO2: ...
  ...
  default: ...}
  ...}

An example (application part)

```
#include ...
#define RSA_GENKEYPAIR _IOWR('s',6,struct PARAM_INFO)
int fd,i,error;
struct PARAM_INFO param;
...
 fd=open("/dev/sjy22b0",O_RDWR);
 if(fd==-1)
      err(1,"Open");
 if(ioctl(fd, RSA_GENKEYPAIR, \&param)==-1)
      err(1,"RSA_GENKEYPAIR");
...
```

驱动中与之对应的操作(ioctl)

```
static int sjy22b\_ioctl(struct cdev *dev, u\_long cmd,
caddr\_t addr, int flag, struct thread *td){
    struct PARAM_INFO *kparam;
    struct PARAM_INFO paramtmp;
    case RSA_GENKEYPAIR:
            kparam=(struct PARAM_INFO *)addr;
            paramtmp.inlen=kparam->inlen;
            paramtmp.outlen=kparam->outlen;
            copyin(kparam->inp,parin,
            sizeof(char)*kparam->inlen);
            paramtmp.inp=parin;
            copyin(kparam->outp,parout,kparam->outlen);
            paramtmp.outp=parout;
            F206Function(\&paramtmp,0);
            copyout(paramtmp.outp,kparam->outp,
            sizeof(char)*kparam->outlen);
            break; ...}
```

有关ioctl的参数

- ioctl(fd, RSA_GENKEYPAIR, &param);
- #define RSA_GENKEYPAIR _IOWR('s',6,struct PARAM_INFO)

其中_IOWR定义的宏在sys/ioccom.h中,_IOW()从用户空间向内核空间写数据，_IOR()用户空间读内核空间数据，_IOWR()双向，_IO()不写也不读。

- _IOW():在内核中重新分配一块与传入数据结构一样大小的空间，并将传递的参数值赋给新的结构。地址与所传递的地址不同。
- _IOR():在内核中重新分配空间，但是新申请的空间都是空的。即不从用户空间读取任何东西。

注：当要从内核中拷贝数据到一个内存地址时要用_IOWR();

Outline

使驱动程序在内核配置文件中可以选择配置

- 在/use/src/sys/dev 中建立sjy22b文件夹并将.c 和.h拷入。
- 修改/usr/src/sys/conf/files.i386加入“dev/sjyssb/sjy22b.c optional cryptcard”
- 修改内核配置文件GENERIC，加入“device cryptcard”，重新编译安装

注意：

- 把sjy22b.c中的“#include xxx.h”改为“#include dev/sjy22b/xxx.h”。
- GENERIC中device后面设备名中间不能含有数字：sjy22b-wrong,cryptcard-right.