Драйвер устройства, имеющий дело с PCI

Matej Košík
14.02.2008

Аннотация

Мы описываем простой драйвер устройства для операционной системы MINIX. Программный код перемежается с описаниями. Исключительной целью своей мы ставили проложить путь последующим, более практичным драйверам.

1 Эскиз

Во всех отношениях, такой `faun` драйвер может быть рассматриваем как нормальный MINIX драйвер. Это процесс пользовательского пространства, его поведение подобно демону, таким образом что в нормальном состоянии он блокирован до тех пор, пока не произойдёт нечто, требующее обслуживания, см. гл. 3 в [1].

Рисунок 1 показывает соответствие процессов, вовлечённых во взаимодействия с нашим `faun` драйвером.

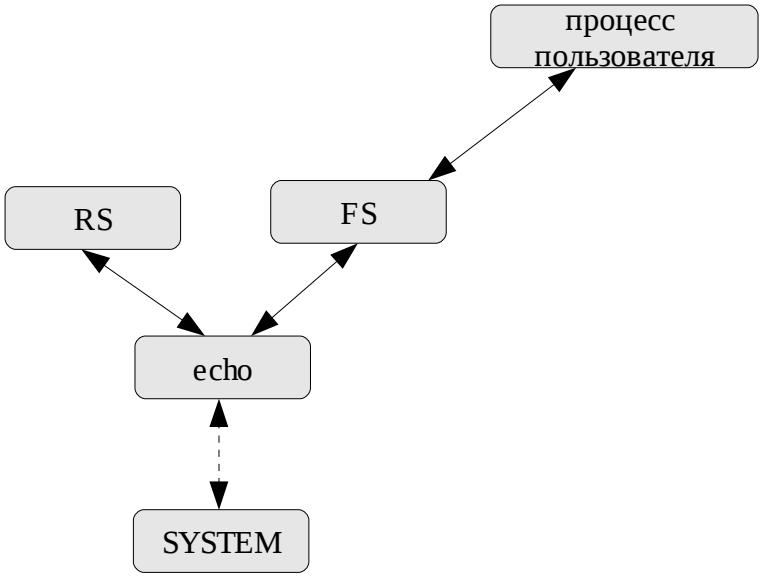

Рисунок 1. Взаимодействующие процессы

Когда пользовательские процессы пытаются выполнить операции с файлами[1] их запросы транслируются в сообщения, посылаемые файловой системе (FS). Когда процесс файловой системы видит, что пользовательские процессы пытаются выполнять операции над специальным файлом, то, в соответствии со старшим номером, он перераспределяет эти запросы к процессу соответствующего драйвера устройства. Пунктирная стрелка между `faun` драйвером и процессом `SYSTEM` должна напоминать нам, что даже если драйвер `faun` не нуждается в общении с `SYSTEM` задачей, нормальные драйверы устройств обычно это делают.

1 такие как `open()`, `close()`, `read()`, `write()` и другие.

m_type	IO_ENDPT	COUNT	ADDRESS
DEV_READ	endpt	count	address
DEV_WRITE	endpt	count	address
DEV_OPEN	endpt		
DEV_CLOSE	endpt		

Таблица 1. Поля во входящих сообщениях, которые мы используем.

Сообщения, принимаемые драйвером от процесса файловой системы, всегда представлены в формате m2, как описано в главных файлах определений MINIX. Таблица 1 представляет допустимые сообщения (DEV_READ, DEV_WRITE, DEV_OPEN, DEV_CLOSE), которые наш драйвер понимает, и какие поля приходящих сообщений он использует[2].

```
<faun.c> ≡
   <header files>
   <global variables>
   <function prototypes>
   <main function>
   <auxiliary reply function>
   <read handler>
   <status handler>
```

2 Реализация

Мы включаем эти заголовочные файлы везде, где потребуется.

```
<header files> ≡
#include "../drivers.h"
#include "../libpci/pci.h"

#include <stdio.h>
#include <stdlib.h>
#include <string.h>
```

Все глобальные переменные, которые нам понадобятся. Содержимое buffer определяет строку, которая может быть получена чтением устройства /dev/faun.

```
<global variables> ≡
#define BUFFER_SIZE 4096
PRIVATE char buffer[BUFFER_SIZE];
PRIVATE int current_buffer_size;
```

Прототипы функций, которые мы будем определять позже,

```
<function prototypes> ≡
FORWARD _PROTOTYPE( void reply, (int replyee, int proc, int s));
FORWARD _PROTOTYPE( void do_read, (message* m_ptr));
FORWARD _PROTOTYPE( void do_status, (message* m_ptr));
```

Как вы можете видеть, поведение драйвера подобно демону. Он блокирован в цикле до тех пор, пока некоторое сообщение (с некоторым запросом обслуживания) не придёт. Когда

2 Полный вид представления сообщений, их различных форматов, извлечение их полей в языке C выглядит неуклюже. Если только вы не намереваетесь писать часть операционной системы на некотором языке высокого уровня, который поддерживает описание передачу сообщений и обработку сообщений на уровне языка.

поступит сообщение, управление диспетчируется соответствующей функции, в зависимости от типа принятого сообщения. Ветвь по умолчанию также жизненно необходима. Она не только гарантирует поглощение запросов к этому драйверу, которые он не умеет обрабатывать, но также отвечает за запросы к серверу реинкарнаций (RS). Если драйвер не отвечает RS, он будет безоговорочно остановлен и перезапущен.

<main *function*> ≡

```
PUBLIC int main(void)
{
   message mess;
   int result;
   while (TRUE) {
      if (receive(ANY, &mess) != OK) continue;
      switch(mess.m_type) {
      case DEV_READ:
         do_read(&mess);
         break;
      case DEV_WRITE:
      case DEV_OPEN:
      case DEV_CLOSE:
         reply(mess.m_source, mess.IO_ENDPT, OK);
         break;
      case DEV_STATUS:
         do_status(&mess);
         break;
      default:
         reply(mess.m_source, mess.IO_ENDPT, EINVAL);
         break;
      }
   }
}
```

Вспомогательные функции для конструирования и отсылки ответа.

`Replyee` процесс (FS или RS), который посылает нам сообщение и мы отвечаем на него;
`process` процесс пользовательского пространства, который запрашивает FS осуществления I/O операции;
`status` результат I/O операции;

***<auxiliary* reply *function>* ≡**

```
PRIVATE void reply(int replyee, int process, int status)
{
   message m;
   int s;
   m.m_type = TASK_REPLY; /* ritual dance */
   m.REP_STATUS = status;
   m.REP_ENDPT = process;
   if (OK != (s=send(replyee, &m)))
      panic("faun", "sending reply failed", s);
}
```

Функция обслуживающая запросы чтения к устройству `/dev/faun` device. The `sys_vircopy()` позволяет нам копировать последовательность байт между различными адресными пространствами. В этом случае мы копируем байты из нашего адресного пространства в адресное пространство клиента (который пытается читать из `/dev/faun`).

***<read handler>* ≡**

```
PRIVATE void do_read(message* m_ptr)
{
   int result;
   int source = m_ptr->m_source;
```

```
   int end_point = m_ptr->IO_ENDPT;
   int count = m_ptr->COUNT;
   int vid, did, subvid, subdid;
   vir_bytes address = (vir_bytes) m_ptr->ADDRESS;
   int bus_number, device_number, function_number;
   buffer[0] = 0;
   bus_number = 0;
   device_number = 0;
   function_number = 0;
   sprintf(buffer, "buffer size, bus, device, function,
           vid, did, subvid, subdid\n\n");
   for (bus_number = 0;
        bus_number < PCIIC_MAX_BUSNUM;
        bus_number++) {
      for (device_number = 0;
           device_number < PCIIC_MAX_DEVNUM;
           device_number++) {
         for (function_number = 0;
              function_number < PCIIC_MAX_FUNCNUM;
              function_number++) {
            vid = get_vid(bus_number, device_number, function_number);
            did = get_did(bus_number, device_number, function_number);
            subvid = get_subvid(bus_number, device_number, function_number);
            subdid = get_did(bus_number, device_number, function_number);
            if (vid == NO_VID)
               break;
            sprintf(buffer + strlen(buffer), "%d, %d, %d, %d, 0x%X, "
                    "0x%X, 0x%X, 0x%X\n",
                    strlen(buffer), bus_number, device_number, function_number,
                    vid, did, subvid, subdid);
            sprintf(buffer + strlen(buffer), "%s\n", pci_vid_name(vid));
            sprintf(buffer + strlen(buffer), "%s\n\n", pci_dev_name(vid, did));
         }
      }
   }
   current_buffer_size = strlen(buffer);
   count = current_buffer_size < count ? current_buffer_size : count;
   result = sys_vircopy(SELF, D, (vir_bytes) buffer,
                        end_point, D, address, count);
   if(result != OK)
      panic("faun", "sys_vircopy failed", result);
   /* Tell the client how many bytes were actually emitted so
    * that it can tell it to the client which requested this I/O
    * operation.
    */
   reply(source, end_point, count);
}
```

Ритуальные танцы (необходимо файловой системе).

<status handler> ≡

```
PRIVATE void do_status(message* m_ptr)
{
   m_ptr->m_type = DEV_NO_STATUS;
   send(m_ptr->m_source, m_ptr);
}
```

Этот запутанный программный код не подлежит компиляции в Linux. Он должен быть загружен в Minix и компилирован там. В порядке упрощения перемещений файлов мы имеем:

- добавлена `push` цель в GNU `Makefile` используемый в Linux;
- `Makefile` под MINIX автоматически загружает (`pull`) новые версии всех файлов

перед компиляцией;

Конкретные цели имеют следующие предназначения:

`pull` свежий программный код загружается из Linux;

`$(DRIVER)` создать бинарный драйвер устройства;

`clean` удалить все временные (обновляемые) файлы;

`install` установить драйвер устройства в надлежащее место;

`up` поднять сервис обслуживания драйвера устройства

`refresh` остановить драйвер устройства, обновить программный код, перекомпилировать и стартовать драйвер;

`down` остановить сервис обслуживания драйвера устройства. Скриптовая команда: `ps ax | grep $(DRIVER) | sort -n | head -1 | awk '{print $$1}'` используется для определения PID уже выполняющегося экземпляра сервиса драйвера устройства.

<Makefile> ≡

```
DRIVER=faun
MAJOR=30
URL=http://10.0.0.11/~kosik/$(DRIVER)
#URL=http://147.175.98.219/~kosik/$(DRIVER)

all: up
$(DRIVER): pull
	gcc -o $(DRIVER) $(DRIVER).c ../libpci/pci.c -lsys -lsysutil
pull:
	urlget $(URL)/$(DRIVER).c > $(DRIVER).c && \
	urlget $(URL)/Makefile > Makefile
clean:
	rm -f $(DRIVER) $(DRIVER).c *~
install: /sbin/$(DRIVER)
/sbin/$(DRIVER): $(DRIVER)
	install -o root -cs $? $@
up: install
	if [ ! -c /dev/$(DRIVER) ]; then \
		mknod /dev/$(DRIVER) c $(MAJOR) 0; \
	fi && \
	service up /sbin/$(DRIVER) -dev /dev/$(DRIVER)
down:
	service down \
	  ps ax | grep $(DRIVER) | sort -n | head -1 | awk   {print $$1}
refresh: install
	if [ ! -c /dev/$(DRIVER) ]; then \
		mknod /dev/$(DRIVER) c $(MAJOR) 0; \
	fi && \
	service refresh \
	  ps ax | grep $(DRIVER) | sort -n | head -1 | awk   {print $$1}
```

A Требования

В процессе построения faun драйвера, вы должны инсталлировать следующие пакеты:

- `glib`
- `binutils`
- `gcc-3.4`
- `gnu-coreutils`

B Пример выполнения

Если вы выполняете этот драйвер на MINIX под QEMU (версия 0.8.2), и вы прочитаете их

`faun` драйвера следующей командой:

```
dd if=/dev/faun count=1 bs=4096 2> /dev/null
```

- то вы получите следующий ввывод:

```
buffer size, bus, device, function, vid, did, subvid, subdid
62, 0, 0, 0, 0x8086, 0x1237, 0x0, 0x1237
Intel Corporation
440FX - 82441FX PMC [Natoma]
151, 0, 1, 0, 0x8086, 0x7000, 0x0, 0x7000
Intel Corporation
82371SB PIIX3 ISA [Natoma/Triton II]
249, 0, 1, 1, 0x8086, 0x7010, 0x0, 0x7010
Intel Corporation
82371SB PIIX3 IDE [Natoma/Triton II]
347, 0, 2, 0, 0x1013, 0xB8, 0x0, 0xB8
Cirrus Logic
GD 5446
407, 0, 3, 0, 0x10EC, 0x8029, 0x0, 0x8029
Realtek Semiconductor Co., Ltd.
RTL-8029(AS)
```

C Example output

Это приложение содержит информацию, полученную этим драйвером на ноутбуке HP Pavilion dv4000:

```
buffer size, bus, device, function, vid, did, subvid, subdid
62, 0, 0, 0, 0x8086, 0x2590, 0x103C, 0x2590
Intel Corporation
Mobile 915GM/PM/GMS/910GML Express Processor to DRAM Controller
189, 0, 2, 0, 0x8086, 0x2592, 0x103C, 0x2592
Intel Corporation
Mobile 915GM/GMS/910GML Express Graphics Controller
305, 0, 2, 1, 0x8086, 0x2792, 0x103C, 0x2792
Intel Corporation
Mobile 915GM/GMS/910GML Express Graphics Controller
421, 0, 28, 0, 0x8086, 0x2660, 0x0, 0x2660
Intel Corporation
82801FB/FBM/FR/FW/FRW (ICH6 Family) PCI Express Port 1
538, 0, 29, 0, 0x8086, 0x2658, 0x103C, 0x2658
Intel Corporation
82801FB/FBM/FR/FW/FRW (ICH6 Family) USB UHCI #1
651, 0, 29, 1, 0x8086, 0x2659, 0x103C, 0x2659
Intel Corporation
82801FB/FBM/FR/FW/FRW (ICH6 Family) USB UHCI #2
764, 0, 29, 2, 0x8086, 0x265A, 0x103C, 0x265A
Intel Corporation
82801FB/FBM/FR/FW/FRW (ICH6 Family) USB UHCI #3
877, 0, 29, 3, 0x8086, 0x265B, 0x103C, 0x265B
Intel Corporation
82801FB/FBM/FR/FW/FRW (ICH6 Family) USB UHCI #4
990, 0, 30, 0, 0x8086, 0x2448, 0x0, 0x2448
Intel Corporation
82801 Mobile PCI Bridge
1076, 0, 31, 0, 0x8086, 0x2641, 0x103C, 0x2641
Intel Corporation
82801FBM (ICH6M) LPC Interface Bridge
1180, 0, 31, 1, 0x8086, 0x266F, 0x103C, 0x266F
Intel Corporation
82801FB/FBM/FR/FW/FRW (ICH6 Family) IDE Controller
1297, 6, 5, 0, 0x8086, 0x4220, 0x103C, 0x4220
Intel Corporation
PRO/Wireless 2200BG Network Connection
February 17, 2008 faun.nw 11
```

```
1401, 6, 6, 0, 0x104C, 0x8031, 0xF000, 0x8031
Texas Instruments
PCIxx21/x515 Cardbus Controller
1498, 6, 7, 0, 0x10EC, 0x8139, 0x103C, 0x8139
Realtek Semiconductor Co., Ltd.
RTL-8139/8139C/8139C+
1599, 6, 7, 1, 0x10EC, 0x8139, 0x103C, 0x8139
Realtek Semiconductor Co., Ltd.
RTL-8139/8139C/8139C+
1700, 6, 7, 2, 0x10EC, 0x8139, 0x103C, 0x8139
Realtek Semiconductor Co., Ltd.
RTL-8139/8139C/8139C+
1801, 6, 7, 3, 0x10EC, 0x8139, 0x103C, 0x8139
Realtek Semiconductor Co., Ltd.
RTL-8139/8139C/8139C+
1902, 6, 7, 4, 0x10EC, 0x8139, 0x103C, 0x8139
Realtek Semiconductor Co., Ltd.
RTL-8139/8139C/8139C+
2003, 6, 7, 5, 0x10EC, 0x8139, 0x103C, 0x8139
Realtek Semiconductor Co., Ltd.
RTL-8139/8139C/8139C+
2104, 6, 7, 6, 0x10EC, 0x8139, 0x103C, 0x8139
Realtek Semiconductor Co., Ltd.
RTL-8139/8139C/8139C+
2205, 6, 7, 7, 0x10EC, 0x8139, 0x103C, 0x8139
Realtek Semiconductor Co., Ltd.
RTL-8139/8139C/8139C+
```

Источники информации:

[1] Andrew S. Tanenbaum and Albert S. Woodhull. Operating Systems: Design and Implementation. Pearson Prentice Hall, 2006.

Перевод: О.Цилюрик , 06.12.2009

Оригинал находится по адресу:

http://www.altair.sk/mediawiki/upload/d/d1/Faun.pdf