by Dr. B. Thangaraju
<balasubramanian.thangaraju(at)wipro.com>

*About the author:*
Dr. B. Thangaraju erhielt seinen Doktor in Physik an der Universität von Bharathidasan, Tamil Nadu und arbeitete fünf Jahre lang als Assistent am indischen Institut der Wissenschaften. Er forschte über transparente, leitende Dünnschicht−Oxide (Transparent Conducting Oxides: TCO), Sprühpyrolyse, photoakustische Methoden und p−n Übergänge in Chalkogenid−Gläsern. Er hat zehn Arbeiten in anerkannten wissenschaftlichen Zeitschriften veröffentlicht und seine Forschungsergebnisse auf mehr als sieben internationalen und nationalen Konferenzen präsentiert.

Zur Zeit arbeitet er als Manager bei Wipro Technologies in Indien. Seine augenblicklichen Studien− und Forschungsgebiete sind der Linux Kernel, Gerätetreiber und Echtzeit−Linux.

# Sichere Port−Zuweisung für Linuxgerätetreiber

*Abstract*:

Einen Gerätetreiber zu schreiben ist eine Herausforderung und manchmal eine abenteuerliche Aufgabe. Sobald das Gerät in der init_module Routine des Treibers registriert ist, sollten die Ressourcen für das Gerät zugeteilt werden. Eine der Hauptressourcen für das Gerät ist der Ein/Ausgabe Port. Der Programmierer muß sehr darauf bedacht sein, einen ungenutzten Bereich der Port−Adressen zu vergeben. Zuerst soll der Treiber prüfen, ob der Bereich schon genutzt wird oder noch frei ist und danach die Ports für das Gerät anfordern. Wenn das Modul wieder aus dem Speicher entfernt wird, müssen die Ports wieder freigegeben werden. Dieser Artikel diskutiert die Schwierigkeiten einer zuverlässigen Zuweisung von Ein−/Ausgabeports für Linux Gerätetreiber.

## Einleitung

Die Hauptsorge eines Gerätetreiberentwicklers ist die Zuteilung der Ressourcen. Die Ressourcen sind I/O Ports, Speicher und IRQs. Dieser Arikel wird versuchen, die Grundlagen von I/O Subsystemen zu erläutern und die Wichtigkeit der Ressourceneinteilung, insbesondere der I/O Ports. Er erklärt auch, wie man nach freien Port−Adressen für Geräte sucht, sie anfordert und wieder freigibt.

Die Grundelemente der Hardware wie Ports, Busse und Controller versorgen eine breite Auswahl an Ein/Ausgabe−Geräten. Die Gerätetreiber bieten eine einheitliche Schnittstelle zum I/O Subsystem, ähnlich wie die Systemaufrufe eine Schnittstelle zwischen den Applikationen und dem Betriebssystem bereitstellen. Es gibt viele Arten von Geräten, die an den Computer angeschlossen sind, zum Beispiel Massenspeicher wie Festplatten, Magnetbänder, CD−ROMs und Floppies, Benutzerschnittstellen wie Tastatur, Maus und Bildschirm, Datenübertragungsgeräte wie Netzwerkkarten und Modems. Trotz dieser sehr unterschiedlichen Gerätetypen muß man nur verstehen, wie die Grundkonzepte aussehen, nach denen Geräte angeschlossen werden und mit deren Hilfe Software die Hardware kontrolliert.

## Grundlegendes Konzept

Ein Gerät besteht aus zwei Teilen, einerseits der elektronischen Komponente, dem Steuerbaustein oder auch Controller genannt und andererseits dem mechanischen Teil. Der Steuerbaustein ist mit dem System über den Systembus verbunden. Normalerweise ist jedem Controller ein Satz (konfliktfreier) Portadressen zugewiesen. I/O Ports umfassen vier Registersätze, nämlich Status, Control, Data−In und Data−Out. Im Status−Register stehen Bits, die der Host lesen kann und die anzeigen, ob das aktuelle Kommando erfolgreich beendet wurde, ob ein weiteres Byte zum Lesen oder Schreiben bereit ist oder ob irgendwelche Fehler auftraten. In das Control−Register schreibt der Host, um ein Kommando zu starten oder den Zustand eines Gerätes zu ändern. In das Data−In Register gelangen Eingaben, über Data−Out werden Ausgaben an das System gesendet.

Die grundlegende Schnittstelle zwischen dem Prozessor und einem Gerät ist also ein Satz Kontroll− und Statusregister. Wenn der Prozessor ein Programm ausführt und auf eine das Gerät betreffende Anweisung stößt, dann führt er diese aus, indem er einen Befehl an das passende Gerät sendet. Die Steuereinheit führt die angeforderte Aktion aus, setzt die entsprechenden Bits im Status−Register und wartet. Der Prozessor ist dafür zuständig, den Status des Gerätes regelmäßig zu überprüfen, bis er sieht, daß die Operation abgeschlossen wurde. Der Parallel−Port−Treiber (Druckeranschluß) zum Beispiel fragt die Bereitschaft des Druckers ab, Ausgabedaten zu empfangen. Wenn der Drucker nicht bereit ist, wird der Treiber eine Zeit lang schlafen (der Prozessor kann währenddessen sinnvolle Arbeit verrichten) und es immer wieder versuchen, bis der Drucker bereit ist. Dieser sogenannte 'Polling'−Mechnismus verbessert die Systemleistung, andernfalls würde das System unnötig auf das Gerät warten, ohne eine sinnvolle Aufgabe zu erfüllen.

Die Register haben eine wohldefinierte Adresse im I/O−Bereich. Im allgemeinen werden diese Adressen beim Booten zugeordnet. Dabei benutzen sie Parameter, die in einer Konfigurationsdatei abgelegt sind und jedem Gerät kann ein Adressbereich zugeordnet werden, wenn das Gerät dauerhaft angeschlossen ist. Das bedeutet, wenn der Kernel die Gerätetreiber für bestehende Geräte enthält, dann können die zugeteilten I/O Port Adressbereiche im **proc**−Verzeichnis gespeichert werden. Man kann die Adressbereiche, die das System gerade benutzt durch Eingabe von **$cat /proc/interrupts** ansehen. Die erste Spalte der Ausgabe zeigt den Adressbereich und die zweite Spalte das Gerät, dem diese Ports gehören. Einige Betriebssysteme haben die Möglichkeit, Gerätetreiber als Modul ins laufende System zu laden. Dadurch kann ein beliebiges Gerät an das laufende System angeschlossen und durch ein dynamisch nachgeladenes Treibermodul kontrolliert und darauf zugegriffen werden.

Das Konzept der Gerätetreiber ist sehr abstrakt und stellt den untersten Softwarelevel dar, der auf dem Computer läuft, da es direkt mit den Hardware−Features des Gerätes verknüpft ist. Jeder Gerätetreiber verwaltet einen speziellen Gerätetyp. Es gibt zeichen−, block− oder netzwerkorientierte Typen. Wenn eine

Anwendung Zugriff auf das Gerät anfordert, stellt der Kernel den Kontakt zum passenden Treiber her. Der Treiber gibt dann das Kommando an das spezielle Gerät aus. Der Treiber ist eine Ansammlung von Funktionen: er hat Einsprungpunkte wie open, close, read, write, ioctl, llseek etc. Beim Laden des Moduls wird die init_module ( ) Funktion aufgerufen, beim Entfernen die cleanup_module ( ) Funktion. Das Gerät ist im Treiber in der init_module ( ) Routine registriert.

Wenn ein Gerät über init_module ( ) registriert wird, dann werden die für die Arbeit notwendigen Ressourcen für das Gerät wie I/O Ports, Speicher und IRQ−Lines in der Funktion selber zugewiesen. Wenn man dem Gerät eine falsche Speicheradresse zuteilt, gibt der Kernel die Fehlermeldung **segmentation fault** aus. Im Falle von I/O Ports aber wird der Kernel keine Meldung wie **wrong I/O port** ausgeben, sondern bereits vergebene Ports zuweisen, was zu einem Systemcrash führt. Beim Entfernen des Moduls sollte das Gerät wieder abgemeldet werden, das heißt die Major Number wird freigegeben, ebenso die Ressourcen durch die cleanup_module ( ) Funktion.

Die Hauptaufgabe des Gerätetreibers ist Lesen und Schreiben auf den I/O Ports. Daher sollte der Treiber sicher sein, daß die Port Adressen von dem Gerät exklusiv genutzt werden. Es darf kein anderes Gerät geben, welches diesen Adressbereich verwendet. Um das sicherzustellen, muß der Treiber zuerst überprüfen, ob die Port Adresse bereits in Gebrauch ist oder nicht: sobald feststeht, daß die Adressen frei sind, kann er den Kernel auffordern, den Adressbereich seinem Gerät zuzuweisen.

## Zuverlässige Port−Zuweisung

Jetzt werden wir sehen, wie man mit Kernel−Funktionen die Ressourcenzuteilung und −freigabe durchführt. Der praktische Ansatz ist am Linux 2.4 Kernel ausprobiert worden. Daher ist die komplette Durchführung nur auf Linux Betriebssysteme anwendbar und nur begrenzt für andere UNIX Varianten.

Zunächst wird ein Portbereich auf seine Verfügbarkeit getestet durch

int check_region (unsigned long start, unsigned long len);

die Funktion gibt Null zurück, wenn der Adressbereich verfügbar ist, beziehungsweise weniger als Null oder einen negativen Fehlercode ( −EBUSY oder −EINVAL), wenn er schon genutzt wird. Die Funktion erwartet zwei Argumente: **start** gibt den Beginn eines zusammenhängenden Bereichs an, und **len** die Anzahl an Ports dieses Bereiches.

Sobald ein Port verfügbar ist, sollte er für das Gerät reserviert werden. Das geschieht durch den Aufruf der request_region Funktion:

struct resource *request_region (unsigned long start, unsigned long len, char *name);

Die ersten beiden Argumente sind die gleichen wie vorher, die Zeigervariable **name** ist der Name des Gerätes, dem die Adresse zugewiesen wird. Die Funktion gibt einen Zeiger auf "struct resource" zurück. Resource structure wird verwendet, um die Resourcenbereiche zu beschreiben, die in <linux/ioport.h> deklariert sind. Die strukturierte Variable hat das folgende Format:

```
struct resource {
        const char *name;
        unsigned long start, end;
        unsigned long flags;
        struct resource *parent, *sibiling, *child;
```

```
};
```

Wenn das Modul wieder aus dem Kernel entfernt wird, sollte der Port wieder für die Nutzung durch andere Geräte freigegeben werden. Dafür müssen wir die release_region ( ) Funktion in cleanup_module ( ) verwenden. Die Syntax der Funktion lautet:

void release_region ( unsigned long start, unsigned long len);

Die zwei Argumente bedeuten wieder das gleiche wie zuvor. Die drei genannten Funktionen sind eigentlich Makros, die in <linux/ioport.h> deklariert sind.

# Beispielcode für Geräteanschluss–Zuweisung

Das folgende Progamm zeigt die Zuordnung und Wieder–Freigabe von Ports für einen dynamisch geladenen Gerätetreiber.

```
#include <linux/fs.h.>
#include <linux/ioport.h.>

struct file_operations fops;
unsigned long start, len;

int init_module (void)
{
 int status;
 start = 0xff90;
 len   = 0x90;

 register_chrdev(254,"your_device",&fops);

 status =  check_region (start, len);
 if (status == 0) {
     printk ("The ports are availave in that range\n");
     request_region(start,len,"your_device");
 } else {
     printk ("The ports are already in use. Try other range.\n");
     return (status);
 }
 return 0;
}

void cleanup_module (void)
{
 release_region(start, len);
 printk ("ports are freed successfully\n");
 unregister_chrdev(254,"your_device");}
 printk (" your device is unregistered\n");
}
```

Um Verwirrungen zu vermeiden, wurden in diesem Beispielcode Fehlerprüfung und 'major number' Vergabe vermieden. Sobald der Port erfolgreich zugeordnet wurde, können wir dies im /proc Verzeichnis überprüfen:
**$cat /proc/ioports**

# Kernel I/O Port–Funktion–Optionen für Treiber

Linux bietet eine Auswahl an Funktionen, um auf I/O Ports lesend oder schreibend zuzugreifen, die vom Format der Ports abhängen. Ports können 8, 16 oder 32 Bit breit sein. Die Linux kernel headers <asm/io.h> definieren die integrierten Funktionen, um auf I/O Ports zuzugreifen. Um 8 Bit, 16 Bit oder 32 Bit Ports zu lesen (inx) und zu schreiben (outx), verwendet man die folgenden Funktionen:

```
__u8 inb (unsigned int port);
void outb (__u8 data, unsigned int port);

__u16 inw (unsigned int port);
void outw(__u16 data, unsigned int port);

__u32 inl (unsigned int prot);
void outl (__u32 data, unsigned int port);
```

Um Zeichenketten mit mehr als einem Datenwort effizient zu übertragen, gibt es folgende Funktionen:

```
void insb(unsigned int port, void *addr, unsigned long count);
void outsb(unsigned int port, void *addr, unsigned long count);
```

addr ist die Speicherstelle, an die geschrieben oder von der gelesen wird, und count stellt die Anzahl der Einheiten dar, die übertragen werden sollen. Daten werden gelesen von oder geschrieben auf den Port "port".

```
void insw(unsigned int port, void *addr, unsigned long count);
void outsw(unsigned int port, void *addr, unsigned long count);
```

Lesen oder schreiben von 16 Bit Werten an einen einzelnen 16 Bit Port.

```
void insl(unsigned int port, void *addr, unsigned long count);
void outsl(unsigned int port, void *addr, unsigned long count);
```

Lesen oder schreiben von 32 Bit Werten an einen einzelnen 32 Bit Port

## Danksagung

Der Autor dankt hiermit **Herrn Jayasurya V**, Manager Talent Transformation, Wipro Technologies, Indien, für das kritische Lesen des Manuskriptes.

## Literatur


- Linux Device Drivers (2nd Edition), vom Alessandro Rubini and Jonathan Corbet. Verfügbar bei O'Reilly:http://linux.oreilly.com
- Linux Kernel 2.4 Internals: http://tldp.org/LDP/lki/index.html

- Linux Kernel Module Programming Guide: http://tldp.org/LDP/lkmpg/mpg.html
- The Linux Kernel (older guide, 1998): http://tldp.org/LDP/tlk/tlk.html

---


| | |
|---|---|
| Webpages maintained by the LinuxFocus Editor team<br>© Dr. B. Thangaraju<br>"some rights reserved" see linuxfocus.org/license/<br>http://www.LinuxFocus.org | Translation information:<br>en −−> −− : Dr. B. Thangaraju <balasubramanian.thangaraju(at)wipro.com><br>en −−> de: Robert Gummi <smachaga(at)sp.zrz.tu−berlin.de> |


2005−01−11, generated by lfparser_pdf version 2.51