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

About the author:
Il dott. B. Thanfaraju ha conseguito un master in fisica dall'Università Bharathidasan di Tamil Nadu e ha lavorato come ricercatore per cinque anni nell'Istituto Indiano di Scienze in India. Ha fatto ricerche sugli ossidi trasparenti e conduttori (TCO), film sottili, Spray Pyrolysis, Tecniche Foto−Acustiche e p− a n− transizioni dei vetri di Chalcogenide. Ha pubblicato 10 articoli di ricerca in rinomati giornali internazionali. Ha anche presentato le sue ricerche in più di sette conferenze nazionali e internazionali.

Attualmente lavora come manager alla Talent Transformation, Wipro Technologies, India. Le sue attuali aree di ricerca, studio e diffusione di conoscenze sono Linux Kernel, Driver per periferiche e Real Time Linux.

Allocare correttamente una porta (indirizzo) per un driver di periferica

Abstract:

Scrivere un driver è un lavoro avventuroso e una sfida. Una volta che la periferica è registrata nel driver della routine del init_module, devono essere allocate le risorse per la periferica. Una della principali risorse per una periferica è la porta di I/O. I driver collegati dinamicamente, lo sviluppatore deve fare attenzione ad allocare il campo inutilizzato di indirizzi per le sue periferiche. Primo, il driver deve provare se un campo di porte sono in uso o sono libere, poi le porte devono essere richieste per la periferica. Quando il modulo è rimosso dal kernel, le porte devono essere liberate. Questo articolo discute la complessità di allocare correttamente una porta (indirizzo) per un driver Linux di periferica.

Introduzione

La maggior preoccupazione di uno sviluppatore di driver è allocare le risorse per la periferica. Le risorse sono le porte di I/O, la memoria e gli IRQ. Questo articolo tenta di spiegare i fondamenti del sottosistema di I/O e l'importanza di allocare le risorse, principalemnte le risorse che si occupano delle porte di I/O. Chiarisce

anche come provare, richiedere e liberare gli indirizzi per le periferiche.

Gli elementi hardware di base, come le porte, i bus e i controllori di periferiche, accomodano una grande varietà di periferiche di I/O. I driver presentano un'interfaccia di accesso uniforme alle periferiche per accedere al sottosistema di I/O come le chiamate di sistema forniscono un'interfaccia standard tra le applicazioni e il sistema operativo. Ci sono molti tipi di periferiche, che sono collegate al computer, per esempio, periferiche di memorizzazione come i dischi, i nastri, i CDROM e i floppy, interfaccie verso l'uomo come tastiera, mouse e video, periferiche di trasmissione come schede di rete e modem. Abbiamo bisogno solamente di capire alcuni concetti base su come le periceriche sono collegate e come il software può controllare l'hardware.

Concetti fondamentali

Una periferica consiste di due parti, una parte elettrinica, che viene chiamata controllore della periferica e un'altra è quella meccanica. Il controllore è colleagato al sistema tramite un bus. Tipicamente, un set di indirizzi (non in conflitto) sono collegati a ciascun controller. le porte di I/O comprendono quattro set di registri come lo status, controllo, dati in input e dati in output. Il registro di stato ha dei bit che possono essere letti dall'host, che indicano se il comando corrente è completato o se un byte è pronto per essere letto o scritto o per un messaggio di errore. Il registro di controlllo è scritto dall'host per lanciare un comando o per cambiare il modo di una periferica. Il registro data−in è per avere un input e data−out è per spedire un output al sistema.

L'interfaccia base tra il processore e una periferica è un set di registri di stato e di controllo. Quando il processore sta eseguendo un programma e incontra un'istruzione relativa ad una periferica, esegue l'istruzione spedendo un comando alla periferica appropriata. Il controllore esegue l'azione richiesta e poi imposta il bit appropriato nel registro di stato e aspetta. È compito del processore controllare periodicamente lo stato della periferica finché non trova non è terminata l'operazione. Per esempio, il driver della porta parallela (usato dalla stampante) normalmente interroga la stampante per vedere se è pronta per accettare dell'output, e se la stampante non è pronta, il driver aspetta un po' (il processore può fare un lavoro utile), e prova ancora e ancora finché la stampante è pronta. Il meccanismo di interrogazione aumenta le prestazioni del sistema, altrimenti il sistema aspetterebbe inutilmente la periferica senza fare alcun utile lavoro.

I registri hanno un ben definito indirizzo nello spazio di I/O. Generalmente, questi indirizzi sono assegnati all'avvio, utilizzando un set di parametri specificati in un file di configurazione usato per costruire il sistema e, un campo di indirizzi può essere allocato per ciascuna periferica, se la periferica è collegata staticamente. Questo significa che il kernel contiene i driver per le periferiche presenti, le porte di I/O allocate per le periferiche possono essere memorizzate nella directory **proc**. Potete controllare il campo di indirizzi per le periferiche che attualmente il vostro sistema sta utilizzando con **cat /proc/interrupts**. La prima colonna dell'output mostra i campi delle porte e la seconda colonna è la periferica che detiene quelle porte. Alcuni sistemi operativi hanno la possibilità di caricare i moduli driver dinamicamente quando il sistema è attivo. Così, ogni nuova periferica può essere collegata al sistema, mentre il sistema è attivo e può essere controllata / usata attraverso il modulo driver dinamico di ogni nuova periferica collegata.

Il concetto di driver di periferica è un po' astratto ed è il più basso livello di software che opera su un computer, dato che è direttamente collegato alle capacità hardware della periferica. Ogni driver controlla un solo tipo di periferica. I tipi possono essere a caratteri, blocchi o di rete. Se un'applicazione richiede la periferica, il kernel chiama il driver appropriato. Il driver quindi spedisce il comando alla specifica periferica. Il driver è una collezione di funzioni: ha molti entry point come open, close, read, write, ioctl, llseek etc. Quando voi caricate un modulo, è chiamata la funzione init_module () e quando il modulo viene rimosso, è chiamata la funzione cleanup_module (). La periferica è registrata in un driver della routine di init_module ().

Quando una periferica è registrata su init_module (), allora le risorse per la periferica come le porte di I/O, la memoria e gli IRQ sono allocate nella funzione stessa, le quali sono necessarie al driver per le corrette operazioni della periferica. Se allocate degli indirizzi di memoria sbagliati per la periferica, il kernel mostra un messaggio di errore tipo **segmentation fault**. Ma nel caso delle porte di I/O, il kernel non ritorna alcun errore del tipo **porte I/O sbagliate** ma assegna delle porte già utilizzate, che appartengono a periferiche esistenti che manderanno il vostro sistema in crash. Quando rimuovete il modulo, la periferica verrà deregistrata, così il major number verrà rilasciato e le risorse verranno liberate nella funzione cleanup_module ().

Il lavoro più frequente di un driver è leggere e scrivere le porte di I/O. Così, il vostro driver deve essere sicuro che gli indirizzi siano usati in esclusiva dalla periferica. Ogni altra periferica non deve usare questo campo di indirizzi. Per assicurarsi di questo il driver deve per prima cosa provare se gli indirizzi sono già in uso o no: Quando il driver trova che gli indirizzi non sono in uso, può chiedere al kernel di allocare il campo di indirizzi alla sua periferica.

Allocare correttamente una porta (indirizzo)

Adesso vedremo come implementare l'allocazione e la libnerazione delle risorse con le funzioni del kernel. L'approccio pratico è sperimentato su un kernel Linux 2.4. Quindi l'implementazione completa è applicabile solo a sistemi operativi Linux alcuni estesi ad altre varianti UNIX.

Primo, per provare il campo di porte disponibili o meno per una periferica si fa

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

la funzione ritorna zero se il campo di indirizzi è disponibile o meno di zero o un codice di errore (−EBUSY or −EINVAL) se è già utilizzato. La funzione accetta due parametri: **start** è l'inizio della regione (o campo di I/O) contigua e **len** è il numero di porte nella regione.

Se la porta è disponibile, può essere allocata per la periferica, con la funzione request_region.

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

I primi due parametri sono gli stessi che abbiamo visto prima, la variabile puntatore a carattere **name** è il nome della periferica, il cui indirizzo è stato allocato. La funzione ritorna il tipo di puntatore alla struttura resource. La struttura è usata per descrivere il campo delle risorse, che è dichiarato in <linux/ioport.h>. La struttura contiene questo formato:

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

Quando il modulo è rimosso dal kernel, la porta deve essere rilasciata per le altre periferiche per questo dobbiamo usare la funzione release_region ( ) nel cleanup_module ( ). La sintassi della funzione è

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

La spiegazione dei due argomenti è la stessa di prima. Le suddette tre funzioni sono attualmente delle macro che sono dichiarate in <linux/ioport.h>.

Esempio di driver per allocare una porta

Il seguente programma spiega come allocare e deallocare le porte per le vostre periferiche caricate dinamicamente.

```
#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");
}
```

Per evitare confusione, il controllo di errore e l'allocazione dinamica del major number sono evitate in questo codice di esempio. Quando la porta è allocata con successo, possiamo controllarlo nella directory proc:
cat /proc/ioports

Opzioni delle funzioni kernel per le porte I/O dei driver

Linux supporta una varietà di funzioni dipendenti dalla dimensione delle porte, per leggere e scrivere sulle porte di I/O. Le porte possono essere di dimensione 8, 16 o 32 bit. L'header del kernel Linux <asm/io.h> definisce le funzioni interne per accedere alle porte di I/O. Per leggere (inx) e scrivere (outx) porte a 8 bit, 16 bit e 32 bit, sono utilizzate le seguenti funzioni:

__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);

Per le versioni stringa che permettono di trasferire efficientemente più di un dato alla volta usando le seguenti funzioni:

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

addr è la locazione di memoria dove trasferire o ricevere e count è il numero di unità da trsferire. I dati sono letti o scritti alla singola porta port.

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

legge o scrive un valore di 16 bit su una singola porta a 16 bit.

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

legge o scrive un valore di 32 bit su una singola porta a 32 bit.

Ringraziamenti

L'autore è molto grato a **Mr. Jayasurya V**, Manager, Talent Transformation, Wipro Technologies, India, per la critica lettura di questo manoscritto.

Riferimenti

- Linux Device Drivers (2nd Edition), by Alessandro Rubini and Jonathan Corbet. Il libro è disponibile presso 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