Linux Device Drivers

**Kernel 2.6**

Dr. Wolfgang Koch
Friedrich Schiller University Jena
Department of Mathematics and
Computer Science
Jena, Germany
wolfgang.koch@uni-jena.de

1. Introduction
   - Device Driver
   - I/O Control
   - Using Ports in User Space
   - gcc Inline Assembler
   - Kernel Space
   - Literature, Web Sites

2. Kernel Modules
   - "Hello World" Module
   - Debug-Print, printk etc.
   - Loading, Unloading, insmod
   - Compiling, Kernelversion
   - Monitoring, Parallel Port
   - Writing to tty
   - Command Line Arguments
3. Char Drivers

- File Operations
- Device Files, Major & Minor Numbers
- file_operations Structure
- register_chardev, Choice of Major Number
- mknod
- register_chardev_region – the new way

3. Char Drivers – cont.

- read( ), put_user( )
- open( ), release( ), Usage Count
- file Structure, lseek( )
- write( ), get_user( )
- Race conditions, Atomic Variables
- Spinlocks, Semaphores

4. Advanced Char Drivers

- Use of Minor Numbers
- Device Control, ioctl( )
- Access Control
- Writing to /proc Files
- Memory, kmalloc()
- Time, Delay

5. Interrupts

- Blocking I/O
- poll( )
- Asynchronous Notification
- Hardware Management
- Interrupts
- The Bottom Half
Device Driver

**Driver:**
A set of functions (of software) that manipulate a hardware device
Today as a part of (better: an extension to) the operating system
A container for a collection of subroutines that the OS calls to perform various operations that relate to a hardware device.

Device Driver

A set of functions that manipulate a hardware device
– as a part of (an extension to) the operating system:
• OS needs / provides access to I/O devices
• uniform programming surface, "Virtual Device"
• necessary privileges
• concurrency, multi tasking system – applications share physical hardware

Device Driver

A set of functions that manipulate a hardware device
– as a part of (an extension to) the operating system:
a software layer that lies between the applications and the actual device – hides the details, provides a standardized surface
applications use system calls –
open(), read(), write(), ioctl(), ... close()
if we exchange a device, we simply change the driver – the applications are not affected

Kernel Modules

A set of functions that manipulate a hardware device
– as a part of (an extension to) the operating system:
in Linux: **Kernel Modules**
can be built separately from the rest of the kernel and loaded at runtime when needed
Kernel Modules

in Linux: Kernel Modules

this lectures:
- hands-on tutorial to write kernel module drivers
- (simple character drivers, without DMA, PnP, USB, ...)
- no multiprocessor (SMP) machines - synchronisation

Kernel version 2.6 (also 2.4)

Warning:
we have full privileges, can do a lot of damage
### Ports

**Address Recognition:**
- e.g. 1st serial port (COM1): 0x3F8 – 0x3FF
- memory mapped or separate I/O space (Intel)
- memory space: Load, Store, Move; I/O ports: In, Out
- base address: flash ROM (old: DIL switches)
- lower address bits: address of internal registers

### Synchronisation by Polling

```plaintext
do ; while ( (inb(SR) & 0x80) == 0 ); // busy wait
outb ( Databyte, DR ); // impl. reset of SR
```

### Synchronisation by Interrupt

- The CPU is responsible for synchronisation, addressing and byte counting
- The device must be slower than the CPU
- Polling – waste of CPU time, fast reaction
- Interrupt – takes time to accept

→ Interrupt at the beginning of a block, then polling
Handshaking

For fast devices we may need a second status line from the port to the device – Acknowledge
device and port "shake hands"
→ again with polling or interrupt

DMA – Direct Memory Access

If the CPU is responsible for addressing and byte counting:

Start:
Load R2, Count
Load AR1, StartAddr

Loop:
In R1, DR // synchr. not considered
Store (AR1), R1
Inc AR1
Dec R2
Jnz Loop // very expensive ==> DMA

DMA Controller

DMAC is 2nd Bus Master
responsible for addressing, counting, synchronisation
Cycle steal modus or Burst modus

Using Ports in User Space

#include <sys/io.h>

unsigned char inb(unsigned short int port);
void outb(unsigned char value,
unsigned short int port);

analogous: inw(), inl(), outw(), outl(), inb_p() ...

inline functions, compile: gcc -O -Wall ...
Using Ports in User Space

```c
#include <sys/io.h>
#define PORT 0x378       // parallel port

int main()
{
    outb(0x25, PORT);
}

> ./exe                   (or root: # ./exe)
 Speicherzugriffsfehler

export LANG=en_US ==> Segmentation fault
export LANG=hu_HU ==> Szegmentálási hiba
( 2.4: Szegmentálási hiba )
```

provide root privileges to an executable program:

```bash
> ll exe                        // ls -l -rwxr-xr-x    1 nwk   users    3160 ...
exe # chown root:root exe # chmod a+s exe -rwsr-sr-x    1 root  root     3160 ...

> ./exe        ==>   Result iopl: 0
s - setuid, setgid, process gets eff. user ID/ group ID of the file owner (root), not of the caller
```
gcc Inline Assembler

unsigned char inb(unsigned short int port) {
    unsigned char val;
    asm volatile("inb %l,%0": "a" (val): "d" (port));
    return val;
}

asm(instructions : outp operands : inp operands                          
   [: clobbered registers]); operands: "constraints" (C_expr)  ->  %0, %1 ... > gcc -O -S      ==>  inb %dx,%al

gcc Inline Assembler

GNU assembler syntax

mov from, to
movl %eax, %ebx     // longword (32 bit)
movw %ax,  %bx      // word (16 bit)
movb %al,  %bl      // byte ( 8 bit)

movl $0x386, %edi   // immediate operand
movb (%esi), %al    // indirect memory reference

( look at the output of  'gcc -S source.c' )

 GCC Inline Assembler

asm volatile("outb %b0,%w1":"a" (val),"Nd" (port));

no output operands

operand %0: byte, "a"  val -> %al
operand %1: word; "Nd" port -> byte constant
   or %dx

===> outb %al, $255
or  outb %al, %dx

constraints:

"m"  - output
"r"  - general register
"M"  - memory
"g"  - general register, memory or immediate
"s"  - different from all input operands
"a"  - %eax ... "d"  - %edx
"D"  - %edi, "S" - %esi
"N"  - byte constant (0 .. 255)
int xy=5;
asm("rorb %b0": "=r" (xy): "0" (xy));

printf("%02x\n",xy);

// rotate right 1 byte: 0x05 -> 0x82

output operand %0
input operand "0" -> same register as %0

--> rorb %al
(*rorb %0* --> rorb %eax --> Warning)

---

asm ("movl %2,%0; incl %2; movl %2,%1":
"=r" (m), "=r"(k) : "r" (n));

--> n=37, m=38, k=38  not correct!!
( movl %eax,%eax; incl %eax; movl %eax,%edx)

asm ("movl %2,%0; incl %2; movl %2,%1":
"=r" (m), "=r"(k) : "r" (n));

--> n=37, m=37, k=38  OK
( movl %eax,%ecx; incl %eax; movl %eax,%edx)

---

User Space – Kernel Space

kernel, modules  –  kernel space
applications  –  user space

one task of the OS: independent operation of programs and protection against unauthorized access to resources

the CPU enforces protection of system software from applications
( i386: 4 rings – supervisor mode (kernel space): ring 0,
user mode (user space): ring 3 )

---

references (Inline Assembler):
gcc documentation:
http://gcc.gnu.org/onlinedocs/gcc-3.3.1/gcc/Extended-Asm.html

Intel IA-32 Architecture Software Developer’s Manuals:
User Space – Kernel Space

modules – kernel space      highest privilege level
applications – user space      low privilege level
some operations are disallowed

both modes also have their own address space

code can switch from one level to another only through a limited
number of gates:    applications issue "system calls"
(then the kernel code executing the system call can access data in
the callers address space, interrupt handler cannot)

User Space – Kernel Space

modules – kernel space      highest privilege level
applications – user space      low privilege level

user: full C library                –       kernel: system calls only

conventional debugger hanging application – simply kill it,
we can do any damage in kernel space

user mem: swappable, won’t occupy RAM when it is not in use
kernel module: 2.4 not interruptible by the scheduler (time slices)

User Space – Kernel Space

user space          kernel space
full C library                   system calls only
#include <stdlib.h>          #include <linux/*>
conventional debuggers       debugging difficult
hanging harmless → kill                  strange system errors, reboot
swappable                           remains in memory (fast)
preemptive                           2.4 nonpreemptive
                                        must be reentrant anyhow
                                        be careful – macros
                                        very small stack

Literature

Jonathan Corbet, Alessandro Rubini, Greg Kroah-Hartman
LINUX Device Drivers, 3rd Edition
O'Reilly, 2005
ISBN 0-596-00590-3
available online as .pdf files:
<table>
<thead>
<tr>
<th>Literature</th>
</tr>
</thead>
</table>
| A. Rubini, J. Corbet  
*Linux Device Drivers, 2nd Edition*  
O'Reilly, 2001  
ISBN 0-596-00008-1  
(covers versions 2.0 – 2.4)  
available online as .pdf files:  
http://www.oreilly.com/catalog/linuxdrive2/  

<table>
<thead>
<tr>
<th>Literature</th>
</tr>
</thead>
</table>
| Peter Jay Salzman, Michael Burian, Ori Pomerantz  
*The Linux Kernel Module Programming Guide*  
80 p., 2005 (Version 2.6)  
http://www.tldp.org/LDP/lkmpg/2.6/html/  
(HTML)  
http://www.tldp.org/LDP/lkmpg/2.6/lkmpg.pdf  
other guides: http://www.tldp.org/guides.html  
(tldp: The Linux Documentation Project) | 44 |

<table>
<thead>
<tr>
<th>Literature</th>
</tr>
</thead>
</table>
| J. Quade, E.-K. Kunst  
Linux-Treiber entwickeln:  
Gerätetreiber für Kernel 2.6 systematisch eingeführt  
dpunkt.verl. Heidelberg, 2004  
ISBN 3-89864-238-0  
425 S. (Bib: INF, DE 2000, LINUX, Q1)  

<table>
<thead>
<tr>
<th>Literature</th>
</tr>
</thead>
</table>
| T. Aivazian  
*Linux Kernel 2.4 Internals*  
77 p, 2002  
http://www.tldp.org/LDP/lki/  
(HTML)  
http://www.tldp.org/LDP/lki/lki.pdf  
(Version 2.6 not available) | 45 |
Literature

W. Maurer
Linux Kernelarchitektur:
Konzepte, Strukturen und Algorithmen
von Kernel 2.6
Hanser München, 2004
ISBN  3-446-22566-8
770 S.
Infos, Downloads: www.linux-kernel.de (interesting links)

Literature

S. Gold, S. van der Meer, S. Burkett, M. Welsh
The Linux Programmers Guide
rather old (1995)
http://www.tldp.org/LDP/lpg
http://www.ssc.com/mirrors/LDP/LDP/lpg/

Web Sites

http://www.linux.org/docs
http://www.linuxdocs.org/
http://www.tldp.org/guides.html
http://www.kernel.org/ (developers)

The linux-kernel mailing list FAQ:
http://www.tux.org/lkml