Hardware-Programmierung mit Linux/NASM

Daniel Grün

12. Februar 2007

Zusammenfassung

Direkte Ansteuerung der Parallel-Schnittstelle mittels NASM unter Linux nur unter Einsatz von reinen Assembler-Routinen und Linux System Calls, aufgezeigt am Beispiel der Ansteuerung eines Druckers

1 Theorie

1.1 Zielsetzung

Es soll ein Drucker am Parallelport unter Linux mit möglichst einfachen Mitteln angesprochen werden. Insbesondere soll dabei kein vorgefertigter Treiber zum Einsatz kommen, sondern der Drucker direkt über Portzugriffe gesteuert werden. Das Programm ist in NASM geschrieben und soll möglichst ohne Befehle der C-Bibliothek auskommen. Alle nötigen Funktionen sind daher direkt in Assembler zu implementieren.

1.2 Hardwareprogrammierung mit DOS/Assembler

Unter einem Betriebssystem wie DOS ist der Zugriff auf Hardwareinstanzen z.B. mit Assembler ohne Beschränkungen möglich:

```
mov     ax,'A'          ; byte to write
mov     dx,0x378        ; destination
out     dx,al           ; write al to dx

mov     dx,0x37a        ; source port
in      al,dx           ; read to al
```

Dieser Beispielcode schreibt unter DOS ohne Weiteres ein *A* auf Register `0x378` und liest das Register `0x37a` nach `al` ein.

Insbesondere ist es auch möglich, BIOS-Interrupts direkt zu verwenden:

```
mov     ah,0
mov     al,'A'          ; character: 'A'
mov     dx,1            ; send to LPT2
int     0x17            ; BIOS LPT interrupt
```

Somit ist es hier nicht weiter schwierig, den Drucker entweder über BIOS-Befehle oder direkte Port-Ansteuerung anzusprechen.

1.3 Portprogrammierung mit Linux/NASM

Der Linux-Kernel erlaubt im Normalfall die unter DOS gegebenen Möglichkeiten des direkten Port-Zugriffs und der BIOS-Interrupts nicht. Dies ist unvermeidlich, da Linux als Mehrbenutzer- und Multitasking-System andere Sicherheitsansprüche als DOS stellt (Dateirechte, Sicherheit gegen unbefugten Zugriff bzw. Cracking).

Während der Zugriff auf alle Interrupts außer dem Kernel-Interrupt `0x80` unter Linux unmöglich bleibt, kann der Portzugriff freigeschaltet werden. Hierzu gibt es unter C zwei Möglichkeiten: die Befehle `ioperm` und `iopl`, wobei der erstere zur Freischaltung einiger Ports, der zweite zur globalen Freischaltung aller dient.

Diese Befehle beruhen auf System-Calls des Linux-Kernels, die auch von Assembler aus angewandt werden können, wie weiter unten Beschrieben ist. Einem vom Benutzer `root` gestarteten Prozess kann so der direkte Portzugriff wie unter DOS ermglicht werden.

Im Folgenden kann dann mit `in` und `out` wie unter DOS auf die freigeschalteten Ports (hier `0x378` bis `0x37a`) zugegriffen werden.

2 Realisierung

Das Programm zum Ausdrucken einer Textdatei ist folgenden Schritten realisiert:

1. Portzugriff freischalten
2. Textdatei öffnen
3. Zeichen lesen und drucken
4. Programmende

2.1 Portzugriff freischalten

Die Freischaltung der Ports für die eine parallele Schnittstelle bei `0x378` funktioniert z.B. mit `ioperm` wie folgt. Die nötigen Parameter werden dabei in die Register geladen und dann der System-Call 101 (`sys_ioperm`) ausgeführt:

```
mov     eax, 101        ; sys_ioperm
mov     ebx, 0x378      ; from port
mov     ecx, 3          ; number of subsequent ports
mov     edx, 1          ; turn permission on
int     0x80            ; call kernel
```

Im Folgenden kann dann mit `in` und `out` wie unter DOS auf die freigeschalteten Ports (hier `0x378` bis `0x37a`) zugegriffen werden.

2.2 Textdatei öffnen

Dem Programm wird als einziger Parameter (bei Programmstart der Reihenfolge nach im Stack gespeichert) der Name der Textdatei übergegeben, die gedruckt werden soll. Die folgende Routine mit System Call `sys_open` und Fehlerüberprüfung liest den Dateinamen und öffnet die Datei im Read-Only-Modus.

```
mov     eax,5   ; sys_open
pop     ebx     ; argc
pop     ebx     ; argv[0]: program name
pop     ebx     ; should be the argument: filename
mov     ecx,0   ; mode = O_RDONLY -- we hope it exists
int     0x80    ; call kernel

test    eax,eax ; check returned file descriptor
jns     cont2   ; continue if positive (not signed)

call    error_exit     ; this jumps to an error exit

cont2:
push    eax     ; save fd
```

Im Stack befindet sich nun der *file descriptor*, ein Integerwert, über den mittels `read` aus der Datei gelesen werden kann.

2.3 Zeichen lesen und drucken

Jedes Zeichen wird nun einzeln gelesen und gedruckt, bis die Datei vollständig verarbeitet ist.

2.3.1 Zeichen lesen

Zum Lesen eines Zeichens wird der System Call `sys_read` eingesetzt:

```
mov     eax,3   ; sys_read
pop     ebx     ; file descriptor from stack
mov     ecx,buffer ; memory address to read to
mov     edx,1   ; number of bytes to read
int     0x80    ; kernel call
```

2.3.2 Zeichen drucken

Um ein Zeichen an den Drucker zu senden, muss eine Reihe von Befehlen ausgeführt werden. Bei einfachen Nadeldruckern ist das folgende Protokoll erfolgreich:

- zu druckendes Byte an Datenport senden (per `out`-Befehl, siehe Listing)
- Statusbyte lesen (per `in`-Befehl)
- Strobe-Bit des Statusbytes auf 'high' setzen und an Status-Port zurückschreiben
- Strobe-Bit des Statusbytes wieder auf 'low' setzen und an Status-Port zurückschreiben
- Acknowledge-Bit des Statusbytes abwarten

Die Schreib- und Lesevorgänge funktionieren wie weiter oben beschrieben (siehe auch Listing). Zum Setzen und Überprüfen des Strobe- und Acknowledge-Bits des Statusbytes werden binäre Operatoren eingesetzt. So setzt ein `or` mit `0x01` das erste Bit (Strobe) 'high', ein `and` mit `0xfe` das selbe Bit 'low'. Ein `and` mit `0x40` liefert setzt genau dann das 'zero'-Flag, wenn das sechste Bit (Acknowledge) 'low' ist, der Drucker den Empfang des Zeichens also bestätigt hat.

2.3.3 Wartezeiten

Auch wenn man das Acknowledge-Bit abwartet, geht die Ausgabe der Zeichen oft zu schnell. Der Portbaustein braucht eine gewisse Zeit zur Verarbeitung der Signale. Die nötige Verzögerung kann mittels des System Calls `sys_nanosleep` erzeugt werden, der den jeweiligen Prozess für eine gewissen Zeitdauer anhält. Als Parameter wird eine Datenstruktur verlangt, die sich wie folgt zusammensetzt:

```
struc timespec
  second:   resd 1
  nanosec:  resd 1
endstruc
```

Es wird eine konstante Instanz dieser Datenstruktur eingeführt, die die gewünschte Wartezeit enthält (hier 500000ns, siehe Listing). Man übergibt die Datenstruktur an den System Call wie folgt:

```
mov eax,162     ; sys_nanosleep
mov ebx,temp1   ; load time structure.
mov ecx,0       ; no safety net in case of signals
int 80h         ; sys_nanosleep
```

Da `nanosleep` durch Signale unterbrochen werden kann, ist es möglich, eine zweite Datenstruktur in `ecx` anzugeben, die in diesem Fall die noch nicht vergangene Wartezeit speichert. Hier soll darauf verzichtet werden.

2.3.4 Textausgabe

Um den Programmablauf besser nachvollziehen zu können, werden einige Meldungen sowie der gedruckte Text auch auf dem Bildschirm ausgegeben. Hierzu wird der System Call `sys_write` auf file descriptor 1 (entspricht `stdout`) eingesetzt:

```
mov     eax,4   ; sys_write
mov     ebx,1   ; stdout
mov     ecx,tgreeting  ; string at memory address tgreeting
mov     edx,37  ; length of the string
int     0x80    ; kernel call
```

Die selbstgeschriebene Funktion `putc` schreibt ein Zeichen, indem sie den Stack Pointer als Speicheradresse nutzt.

3 Programmlisting

Der folgende kommentierte Programmtext kann mittels NASM (hier Version 0.98) auf Linux-Systemen kompiliert werden.

```
; print.asm:
; prints a file given as command line parameter to lpt1
; using only system calls and direct port access
; (c) Daniel Gruen, 2006
; License: GNU General Public License (GPL) Version 2
; No Warranty whatsoever!

section .text
global  _start

lp1_port equ   0x378    ; my lp port, change this accordingly

struc timespec          ; model for "nanosleep" structure .
second:   resd 1
nanosec:  resd 1
endstruc

_start:
        mov     eax,4   ; sys_write
        mov     ebx,1   ; stdout
        mov     ecx,tgreeting
        mov     edx,37  ; length
        int     0x80

        ; step 0: get port permissions
        mov     eax, 101        ; sys_ioperm
        mov     ebx, lp1_port   ; from
        mov     ecx, 3          ; num
        mov     edx, 1          ; turn on
        int     0x80            ; kernel call

        inc     eax             ; is return value 0 = ok?
        jnz     cont1           ; go on unless eax was -1

        call    error_exit

cont1:
        ; read & print characters
        ; step 1: open file
        mov     eax,5   ; sys_open
        pop     ebx     ; argc
        pop     ebx     ; argv[0]
        pop     ebx     ; argv [1] should be the argument: filename
        mov     ecx,0   ; O_RDONLY -- we hope it exists
        int     0x80    ; kernel call
```

```
        test    eax,eax ; check returned value
        jns     cont2   ; go on if not signed (=if positive)

        call    error_exit

cont2:
        push    eax     ; save fd

        mov     eax,4   ; sys_write
        mov     ebx,1   ; stdout
        mov     ecx,tinit
        mov     edx,41  ; length
        int     0x80    ; kernel call

rploop:
        ; step 2: read char
        mov     eax,3   ; sys_read
        pop     ebx     ; file descriptor (fd)
        mov     ecx,buffer ; address to read to
        mov     edx,1   ; should be type size_t...
        int     0x80    ; kernel call

        push    ebx     ; save fd
        push    eax     ; save number of read bytes

        mov     ax,[buffer]
        call    putc

        ; step 3: print char

        ; step 3a: send char on data port
        mov     ax,[buffer]
        mov     dx,lp1_port
        out     dx,al

        ; step 3b: read status byte
        mov     dx,lp1_port+2
        in      al,dx

        ; step 3c: set strobe high
        or      al,0x01

        ; step 3d: write high strobe status byte
        mov     dx,lp1_port+2
        out     dx,al

        ; step 3e: read status byte
        mov     dx,lp1_port+2
        in      al,dx
```

```
        ; step 3f: set strobe low
        and     al,0xfe

        ; step 3g: write low strobe status byte
        mov     dx,lp1_port+2
        out     dx,al

        ; step 3h: wait for ack signal
        mov     ebx,100
lack:   mov     dx,lp1_port+2

        mov     dx,lp1_port+2
        in      al,dx

        and     al,0x40

        jz      pend

        mov eax,162     ; sys_nanosleep
        mov ebx,temp1   ; time strucure
        mov ecx,0       ; no safetz net in case of signals
        int 80h         ; kernel call

        push    ebx
        mov     eax,'w'
        call    putc
        pop     ebx

        dec     ebx
        jnz     lack    ; acknowledge waiting loop

        ; step 4: repeat if neccessary

pend:
        mov eax,162     ; sys_nanosleep
        mov ebx,temp1   ; time strucure
        mov ecx,0       ; no safety net in case of signals
        int 80h         ; kernel call

        pop     eax
        dec     eax
        jz      rploop

        mov     ax,10
        call    putc

        ; step 5: close file
        pop     ebx     ; fd
```

```
        mov     eax,1   ; sys_exit
        mov     ebx,0
        int     0x80

error_exit:
        mov     eax,4   ; sys_write
        mov     ebx,1   ; stdout
        mov     ecx,terror
        mov     edx,28  ; length
        int     0x80    ; kernel call

        mov     eax,1   ; outta here (sys_exit)
        mov     ebx,1   ; return code: error
        int     0x80    ; kernel_call
        ret

putc:   push    eax     ; puts char in eax
        mov     eax,4   ; sys_write
        mov     ebx,1   ; file descriptor -> stdout
        mov     ecx,esp ; stack pointer -> data address
        mov     edx,1   ; length 1
        int     0x80    ; kernel call
        pop     eax     ; restore eax
        ret

section .data
tgreeting:
        db      'This program will call your printer.', 10, 0
tinit:
        db      'Finished initialization. Now printing...', 10, 0
terror:
        db      'An error occured. Exiting...', 10, 0
timeout_count:
        db      9 ; cycles to wait till timeout
strobe_wait:
        db      200
buffer:
        times 4 db 0
temp1      istruc timespec
           at second,  dd 0
           at nanosec, dd 500000
           iend
```