Direct I/O-address access within Linux

During the programming of an embedded system it is mostly unavoidable to address some hardware components directly. In the ideal case, the drivers (driver devices) were directly located over the hardware, so that the operating system could take access to the hardware. The application itself uses only the interfaces of the operating system to access to the hardware. The hardware is completely enclosed from the application. By changing hardware components you have only to modify the involved device driver, the application remain unchanged.

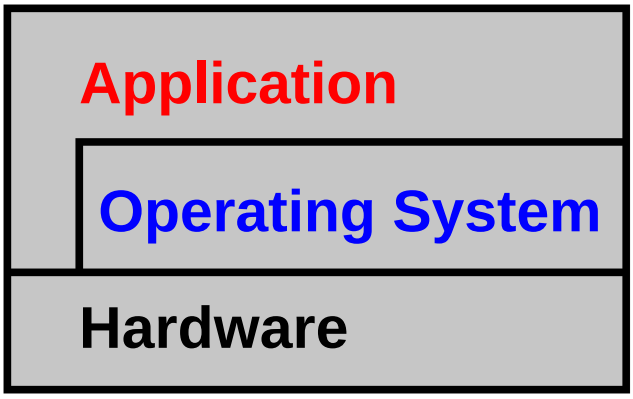

Figure 1: Layer of an embedded system

For office PCs this procedure is very reasonable. For a word-processing program it is not necessary to have direct access on a printer. For that, a printer driver is available. In the industrial environment of embedded systems you must have an own device driver for every small hardware device. That is not what we want. The reasons are various. The development of a device driver is a very extensive job that can be really good controlled only from a few programmers. Further it is very difficult to test device drivers because they work as a component of the operating system. In addition of that, it is not possible to use normal troubleshooting tools (debugger). The interface between an application program and a driver is normally defined by the operating system. That means, that in many cases exotic hardware components can not be added to the system without problems. In regard to an embedded system application it is important to know that the driver application breaks basically into two parts during the hardware access. It is very difficult to service these parts on a later point of time. Considering these advantages and disadvantages during the programming of an embedded system it is absolute acceptable to access directly to the hardware when running under Linux. There is a maximum of flexibility for the developer.

```
// 8-Bit Counter for DIL/NetPC PIO (Linux-Version).
// Written by KDW (kdw@ist1.de) - 24.Jan.2001

#ifdef HAVE_CONFIG_H
#include <config.h>
#endif

#include <stdio.h>
#include <stdlib.h>
#include <fcntl.h>
#include <unistd.h>
#include <asm/io.h>

#define  CSCIR  0x22      // Chip Setup and Control Index Register
#define  CSCDR  0x23      // Chip Setup and Control Data Register
#define  PAMR  0xa5       // PIO Port A Mode Register
#define  PADR  0xa9       // PIO Port A Data Register

void main (void) {

   int iCnt;

   printf ("  Start Binary Counter for Port A...\n");
   printf ("  Current Counter Value=   0");

   // Set SC410 for DIL/NetPC PIO Port A = Output

   ioperm (CSCIR, 2, 1);
   outb (PAMR, CSCIR);     // IndexPtr.= IndexRegister A5h
   outb (0xff, CSCDR);     // IndexRegister A5h= 0xff
   ioperm (CSCIR, 2, 0);

   // Run Counter until User Break by CTRL-C...

   for (;;) {

      // Write 8-bit Binary Counter Value to Port A...

      for (iCnt= 0; (iCnt < 256); iCnt++) {

         ioperm (CSCIR,2,1);
         outb (PADR,CSCIR);          // IndexPtr.= IndexRegister A9h
         outb (iCnt & 0xff,CSCDR);   // IndexRegister A9h= iCnt
         ioperm (CSCIR,2,0);

         printf ("\r  Current Counter Value= %3d", iCnt);
         fflush (stdout);
         usleep (100000);
      }
   }
}
```

Listing 1: Direct access on 80x86-I/O-addresses under Linux

At an embedded PC system based on 80x86-processors it is needed at first to access the I/O-addresses directly. Listing 1 shows an example of access on the parallel ports of the DIL/NetPC. For the access to the I/O-addresses there is a C-function named outb (...) used. This function writes a byte to the desired address in the I/O-address range of an 80x86-processor.

Before you can directly access to the I/O-hardware with an `outb (...)`-command you have to afford access by the function `ioperm (...)`. Linux runs on an 80 x 86-processor in a particular protected mode from different protection layers. The lowest protection layer with all rights and minimum protection is reserved through the operating system. In a higher layer with limited rights and maximum protection mechanisms runs the application. To become access to I/O-addresses from this layer and to avoid a segmentation fault, you have to unlock the desired I/O-address range before by using the function call `ioperm (...)`.

The function `ioperm (...)` is Linux specific. It allows an application program to access to the 80x86-I/O-addresses within the range 0x000 to 0x3ff. Please note: a program, which uses this function, need administrator rights (super user rights) by execution.

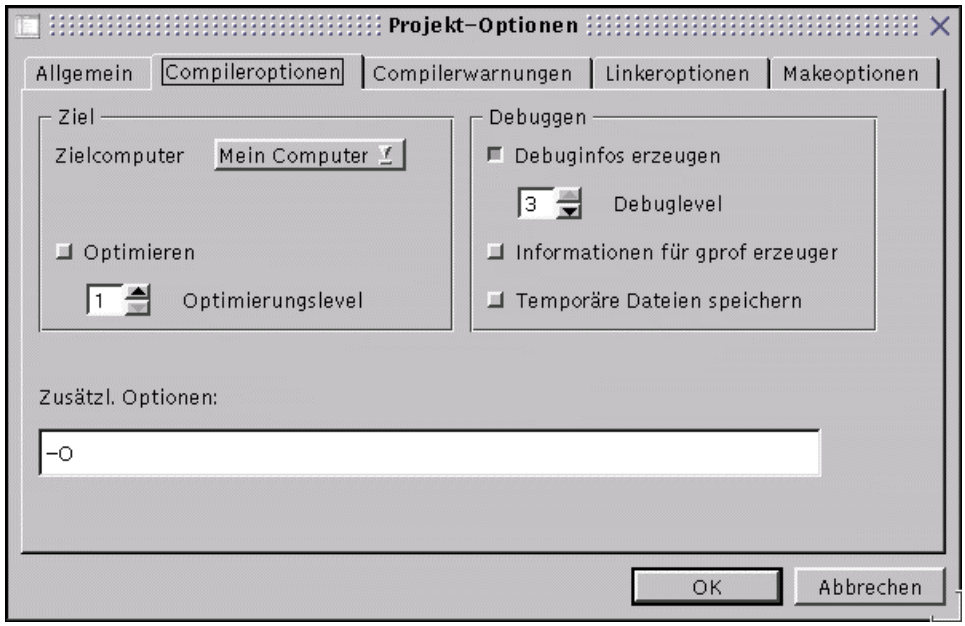

Figure 2: Project parameter for the compiling of listing 1

If this example is supposed to be translated under KDevelop you have to set a special parameter for the project. The C function `outb (...)` is an inline-macro. That is the reason that the *gnu* C compiler requires the parameter -o. As a result you should choose the menu item project before the first compiling of a project with `outb(...)`-function calls. Now select the item options and select the tab `Compileroptions` as you can see in figure 2. In the field additional options please enter the parameter `-o`.

After that, you are able to translate the example from listing 1 without any error reports of the *gnu*-C-compiler.

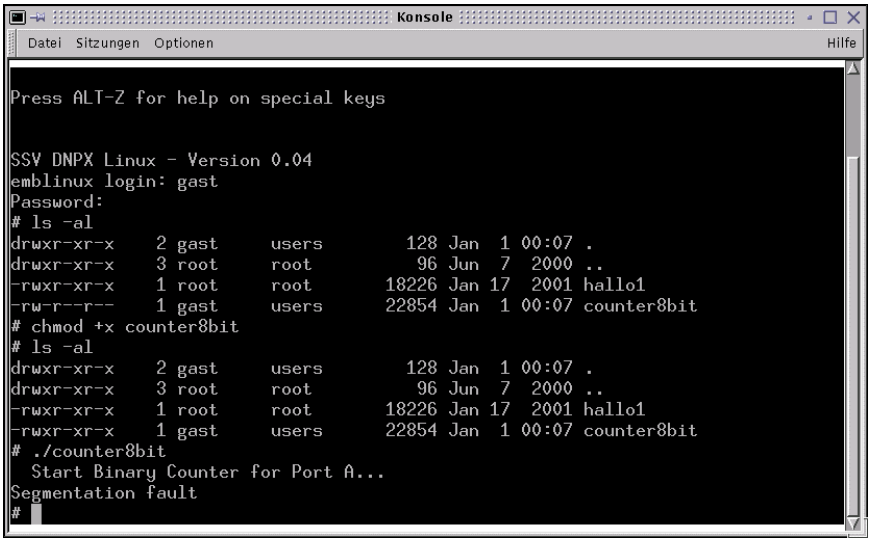

Figure 3: Start of an I/O-access program without administrator rights

Normally, a program with direct I/O-access can only be tested onto the embedded system. The development system generally decrees not about the appropriate hardware under the respective I/O-addresses. Under circumstances this can cause a fatal mistake, if the addresses on the development systems are used by other functions than on the embedded system.

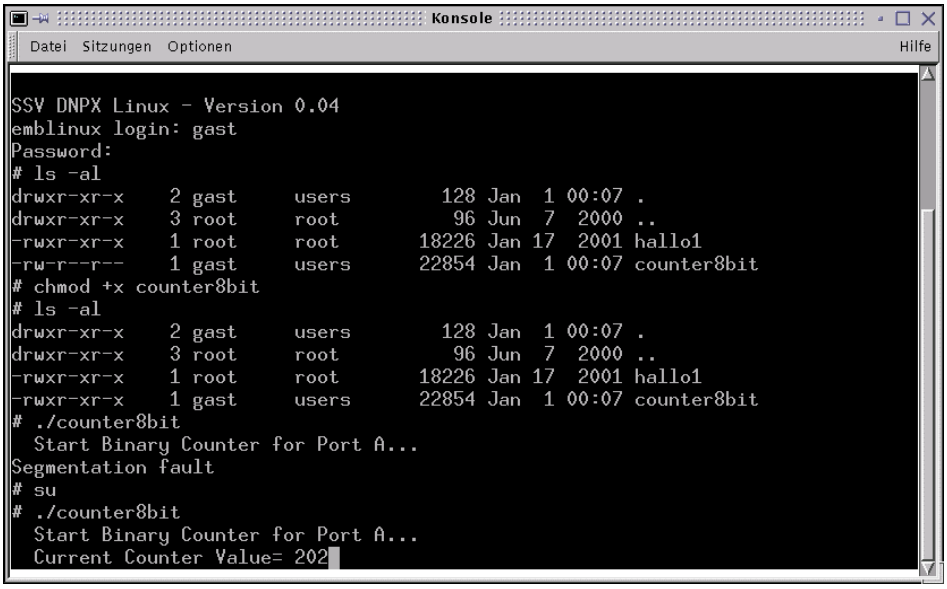

Figure 4: Start of an I/O-access program with administrator rights

That is the reason that a program like the example from listing 1 should be translated to the RAM-disk of the embedded system. This has to be done via FTP directly after the translation. Figure 3 shows the try to start such a program without administrator rights. Linux will receive the error report `“Segmentation fault”`. In the figure 4 the administrator rights were present.

The program can be started without problems. In this figure it is recognizable, that the administrator rights were taken via the su-command. Normally, for this is a password necessary. In this case the DIL/NetPC was configured that no password is required.

Under Linux for the programmer there are numerous functions available to access directly onto the I/O-ports of the 80x86. These functions can be segmented in three groups. The table 1 shows an overview.

Functions	**Group**
outb, outw, outl, outsb, outsw, outsl	I/O Port Output
inb, inw, inl, insb, insw, insl	I/O Port Input
outb_p, outw_p, outl_p, inb_p, inw_p, inl_p	Paused I/O Port Access

Table 1: Overview of the functions for the I/O-port access

With these functions there are not many wishes about the direct I/O-access remaining. Also an user who had programmed a micro controller without an operating system in C before should not have any problems on an embedded Linux system.

Figures

Tables

Contact

SSV Embedded Systems
Heisterbergallee 72
D-30453 Hannover
Tel. +49-(0)511-40000-0
Fax. +49-(0)511-40000-40
E-mail: sales@ist1.de
Internet: www.ssv-embedded.de

Document History (Emblinx9e.doc)

Revision	Date		Name
1.00	29.08.2001	First Version.	KDW