Linux Parport programming

Hardware view of the parport

The parallel port (or printer port, IEEE1284 standard) has 8 data lines, 5 status lines (/ACK, BUSY, PAPER_OUT, SELECT and ERROR) and 4 control lines (/STROBE, /INIT, /SELECT_IN and /AUTO_FEED).

Access via raw IO (not recommended)

This has many disadvantages (it's not portable, needs root privileges, ...) and is mentioned here only for educational purpose.

This will under NO CIRCUMSTANCES work with USB-to-Parallel converters.
This is ONLY for ISA-Bus based 8255-compatible devices.
Unless you are developing a device driver in kernel space, do NOT use it.

```
#include <stdio.h>
#include <sys/io.h>

#define BASEADDR 0x378

int main(int argc, char **argv)
{
        int i;

        if(ioperm(BASEADDR, 4, 1) < 0) {
                fprintf(stderr, "could not get i/o permission. are you root ?\n");
                return 5;
        }
        for(i=0; i < 256; i++) {
                outb(i, BASEADDR);
        }
        ioperm(BASEADDR, 4, 0);
        return 0;
}
```

Access via /dev/lpX

This allows writing the 8 data bits (but not reading them) according to the centronics standard, i.e. after writing of the data, a pulse on STROBE is triggered and an acknowledgement pulse on ACK is expected afterwards. Also the BUSY line might have influence on this. You can also read all other status lines.
A huge advantage is also, that this (in contrary of the raw i/o method) will also work with USB-to-Parport converters.

```
#include <stdio.h>
#include <unistd.h>
#include <fcntl.h>
#include <sys/ioctl.h>
#include <linux/lp.h>

#define DEVICE "/dev/lp0"

int main(int argc, char **argv)
{
        int fd, k;

        if((fd=open(DEVICE, O_WRONLY)) < 0) {
                fprintf(stderr, "can not open %s\n", DEVICE);
```

```
                return 10;
        }
        ioctl(fd, LPRESET);                     /* generate RESET pulse */
        if(ioctl(fd, LPGETSTATUS, &k) < 0) {
                fprintf(stderr, "getstatus ioctl failed\n");
                return 10;
        }
        k^=LP_PBUSY;                            /* invert PBUSY */
        printf("BUSY = %s\n", ((k & LP_PBUSY) == LP_PBUSY)?"HI":"LO");
        printf("POUT = %s\n", ((k & LP_POUTPA) == LP_POUTPA)?"HI":"LO");
        printf("ERR  = %s\n", ((k & LP_PERRORP) == LP_PERRORP)?"HI":"LO");
        printf("SEL  = %s\n", ((k & LP_PSELECD) == LP_PSELECD)?"HI":"LO");
        close(fd);
        return 0;
}
```

Full access via /dev/parportX

This gives you full control of the parallel port: You can read and write all data lines, set all control lines (including STROBE) and read all status lines (including ACK).
Prefer this method if you are going to read and write data to any homebrew electronics (because here, e.g. the STOBE line is NOT set automatically and the driver also does NOT wait for an ACK pulse).

```
#include <stdio.h>
#include <fcntl.h>              /* open() */
#include <sys/types.h>          /* open() */
#include <sys/stat.h>           /* open() */
#include <asm/ioctl.h>
#include <linux/parport.h>
#include <linux/ppdev.h>

#define DEVICE "/dev/parport0"

int main(int argc, char **argv)
{
        struct ppdev_frob_struct frob;
        int fd;
        int mode;

        if((fd=open(DEVICE, O_RDWR)) < 0) {
                fprintf(stderr, "can not open %s\n", DEVICE);
                return 10;
        }
        if(ioctl(fd, PPCLAIM)) {
                perror("PPCLAIM");
                close(fd);
                return 10;
        }

        /* put example code here ... */

        ioctl(fd, PPRELEASE);
        close(fd);
        return 0;
}

/* trivial example how to write data */
int write_data(int fd, unsigned char data)
{
        return(ioctl(fd, PPWDATA, &data));
}

/* example how to read 8 bit from the data lines */
int read_data(int fd)
{
        int mode, res;
        unsigned char data;

        mode = IEEE1284_MODE_ECP;
        res=ioctl(fd, PPSETMODE, &mode);        /* ready to read ? */
        mode=255;
        res=ioctl(fd, PPDATADIR, &mode);        /* switch output driver off */
        printf("ready to read data !\n");
```

```
        fflush(stdout);
        sleep(10);
        res=ioctl(fd, PPRDATA, &data);  /* now fetch the data! */
        printf("data=%02x\n", data);
        fflush(stdout);
        sleep(10);
        data=0;
        res=ioctl(fd, PPDATADIR, data);
        return 0;
}

/* example how to read the status lines. */
int status_pins(int fd)
{
        int status;

        ioctl(fd, PPRSTATUS, &status);
        status^=PARPORT_STATUS_BUSY; /* BUSY needs to get inverted */

        printf("BUSY   = %s\n",
                ((status & PARPORT_STATUS_BUSY)==PARPORT_STATUS_BUSY)?"HI":"LO");
        printf("ERROR  = %s\n",
                ((status & PARPORT_STATUS_ERROR)==PARPORT_STATUS_ERROR)?"HI":"LO");
        printf("SELECT = %s\n",
                ((status & PARPORT_STATUS_SELECT)==PARPORT_STATUS_SELECT)?"HI":"LO");
        printf("PAPEROUT = %s\n",
                ((status & PARPORT_STATUS_PAPEROUT)==PARPORT_STATUS_PAPEROUT)?"HI":"LO");
        printf("ACK = %s\n",
                ((status & PARPORT_STATUS_ACK)==PARPORT_STATUS_ACK)?"HI":"LO");
        return 0;
}

/* example how to use frob ... toggle STROBE on and off without messing
   around the other lines */
int strobe_blink(int fd)
{
        struct ppdev_frob_struct frob;

        frob.mask = PARPORT_CONTROL_STROBE;     /* change only this pin ! */
        while(1) {
                frob.val = PARPORT_CONTROL_STROBE;      /* set STROBE ... */
                ioctl(fd, PPFCONTROL, &frob);
                usleep(500);
                frob.val = 0;                           /* and clear again */
                ioctl(fd, PPFCONTROL, &frob);
                usleep(500);
        }
}
```

List of ioctl()s for /dev/parportX access

```
# Data register
ioctl(fd, PPRDATA, &r);         /* read the data register */
ioctl(fd, PPWDATA, &r);         /* write the data register */

# control register
ioctl(fd, PPRCONTROL, &r);      /* read the control register */
ioctl(fd, PPWCONTROL, &r);      /* write the control register */
ioctl(fd, PPFCONTROL, &frob);   /* "atomic" modify (read - write at once) */

# status register
ioctl(fd, PPRSTATUS, &r);       /* read the status register */
```

Download example programs

- Currently unavailable. Use cut and paste.

Liked this page? Then this might be also interesting for you:

- Linux serport programming