

```

/* dl.c -- download hex code to ezusb by Andrew Burgess aab@cichlid.com */

/* Based on usbstress and ezload by Thomas Sailer, original copyright follows:

* Copyright (C) 1999-2000
*   Thomas Sailer (sailer@ife.ee.ethz.ch)
*
* This program is free software; you can redistribute it and/or modify
* it under the terms of the GNU General Public License as published by
* the Free Software Foundation; either version 2 of the License, or
* (at your option) any later version.
*
* This program is distributed in the hope that it will be useful,
* but WITHOUT ANY WARRANTY; without even the implied warranty of
* MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
* GNU General Public License for more details.
*
* You should have received a copy of the GNU General Public License
* along with this program; if not, write to the Free Software
* Foundation, Inc., 675 Mass Ave, Cambridge, MA 02139, USA.
*/

#define _GNU_SOURCE
#include <stdarg.h>
#include <stdio.h>
#include <signal.h>
#include <unistd.h>
#include <stdlib.h>
#include <errno.h>
#include <sys/types.h>
#include <sys/stat.h>
#include <sys/ioctl.h>
#include <sys/poll.h>
#include <fcntl.h>
#include <string.h>
#include "getopt.h"

#include "/usr/src/linux/include/asm/types.h"
#include "/usr/src/linux/include/linux/usbdevice_fs.h"

#define VENDOR_REQUEST_OUT 0x40
#define VENDOR_REQUEST_IN 0xC0
#define ANCHOR_LOAD 0xA0
#define CPU_CONTROL_REGISTER 0x7F92

struct usbdevice *dev;
int verbose;
int overlap;
int timeout = 2000;

struct usb_device_descriptor {
    u_int8_t  bLength;
    u_int8_t  bDescriptorType;
    u_int8_t  bcdUSB[2];
    u_int8_t  bDeviceClass;
    u_int8_t  bDeviceSubClass;
    u_int8_t  bDeviceProtocol;
    u_int8_t  bMaxPacketSize0;
    u_int8_t  idVendor[2];
    u_int8_t  idProduct[2];
    u_int8_t  bcdDevice[2];
    u_int8_t  iManufacturer;
    u_int8_t  iProduct;
    u_int8_t  iSerialNumber;
    u_int8_t  bNumConfigurations;
};

#define USB_DT_DEVICE_SIZE sizeof(struct usb_device_descriptor)

struct usbdevice {
    int fd;
    struct usb_device_descriptor desc;
};

const char *usb_devicefs_mountpoint = "/proc/bus/usb";

/* SYS ERROR -- error with a valid errno */

void sys_error(char *fmt,...) {

```

```

char buf[1000];
va_list ap;
int errno_save = errno;

va_start(ap, fmt);
vsprintf(buf, fmt, ap);
va_end(ap);
strcat(buf, ": ");
strcat(buf, sys_errlist[errno_save]);
fprintf(stderr, buf);
fprintf(stderr, "\n");
exit(2);
}

/* ERROR */

void error(char *fmt,...) {
char buf[1000];
va_list ap;

va_start(ap, fmt);
vfprintf(stderr, fmt, ap);
fprintf(stderr, "\n");
va_end(ap);
exit(2);
}

/* PP STR */

char *pp_str(const char *str) {
static char buf[100];
char c, *p = buf;

while(c = *str++) {
switch(c) {
case 7:
*p++ = '\\';
*p++ = '7';
break;
case '\\n':
*p++ = '\\';
*p++ = 'n';
break;
case '\\r':
*p++ = '\\';
*p++ = 'r';
break;
default:
*p++ = c;
break;
}
}
*p = 0;
return buf;
}

/* USB CLOSE */

void usb_close() {
if(!dev)
return;
close(dev->fd);
free(dev);
}

/* PARSE DEV */

int parsedev(int fd, unsigned int *bus, unsigned int *dev, int vendorid, int productid) {
char buf[16384];
char *start, *end, *lineend, *cp;
int devnum = -1, busnum = -1, vendor = -1, product = -1;
int ret;

if (lseek(fd, 0, SEEK_SET) == (off_t)-1)
return -1;
ret = read(fd, buf, sizeof(buf)-1);
if (ret == -1)
return -1;
end = buf + ret;

```

```

    *end = 0;
    start = buf;
ret = 0;
    while (start < end) {
        lineend = strchr(start, '\n');
        if (!lineend)
            break;
        *lineend = 0;
        switch (start[0]) {
case 'T': /* topology line */
            if ((cp = strstr(start, "Dev#=")) {
                devnum = strtoul(cp + 5, NULL, 0);
            } else
                devnum = -1;
            if ((cp = strstr(start, "Bus=")) {
                busnum = strtoul(cp + 4, NULL, 0);
            } else
                busnum = -1;

            break;

            case 'P':
                if ((cp = strstr(start, "Vendor=")) {
                    vendor = strtoul(cp + 7, NULL, 16);
                } else
                    vendor = -1;
                if ((cp = strstr(start, "ProdID=")) {
                    product = strtoul(cp + 7, NULL, 16);
                } else
                    product = -1;
                if (vendor != -1 && product != -1 && devnum >= 1 && devnum <= 127 &&
                    busnum >= 0 && busnum <= 999 &&
                    (vendorid == vendor || vendorid == -1) &&
                    (productid == product || productid == -1)) {
                    if (bus)
                        *bus = busnum;
                    if (dev)
                        *dev = devnum;
                    ret++;
                }
                break;
        }
        start = lineend + 1;
    }
    return ret;
}

/* USB OPEN BY NUMBER */

struct usbdevice *usb_open_bynumber(unsigned int busnum, unsigned int devnum, int vendorid, int productid) {
    struct usbdevice *dev;
    struct usb_device_descriptor desc;
    unsigned int vid, pid;
    char devsfile[256];
    int ret, fd;

    snprintf(devsfile, sizeof(devsfile), "%s/%03u/%03u", usb_devicefs_mountpoint, busnum, devnum);
    if ((fd = open(devsfile, O_RDWR)) == -1)
        return NULL;
    if ((ret = read(fd, &desc, sizeof(desc))) != sizeof(desc)) {
        if (ret > 0)
            errno = EIO;
        close(fd);
        return NULL;
    }
    vid = desc.idVendor[0] | (desc.idVendor[1] << 8);
    pid = desc.idProduct[0] | (desc.idProduct[1] << 8);
    if ((vid != vendorid && vendorid == -1) ||
        (pid != productid && productid == -1)) {
        errno = -ENOENT;
        close(fd);
        return NULL;
    }
    if (!(dev = malloc(sizeof(struct usbdevice))) {
        close(fd);
        return NULL;
    }
    dev->fd = fd;
    dev->desc = desc;
    return dev;
}

```

```

}

/* USB OPEN */

struct usbdevice *usb_open(int vendorid, int productid, unsigned int timeout) {
    struct usb_device_descriptor desc;
    time_t starttime, curtime;
    unsigned int busnum, devnum;
    char devsfile[256];
    long timediff;
    int ret;
    struct pollfd pfd;

    snprintf(devsfile, sizeof(devsfile), "%s/devices", usb_devicefs_mountpoint);
    time(&starttime);
    if ((pfd.fd = open(devsfile, O_RDONLY)) == -1)
        return NULL;
    for (;;) {
        ret = parsedev(pfd.fd, &busnum, &devnum, vendorid, productid);
        if (ret < 0) {
            close(pfd.fd);
            return NULL;
        }
        if (ret > 0)
            break;
        time(&curtime);
        timediff = curtime - starttime;
        timediff = timeout - timediff;
        if (timediff <= 0) {
            close(pfd.fd);
            errno = ETIMEDOUT;
            return NULL;
        }
        if (timediff > 10)
            timediff = 10;
        pfd.events = POLLIN;
        ret = poll(&pfd, 1, timediff * 1000);
        if (ret < 0) {
            close(pfd.fd);
            return NULL;
        }
    }
    close(pfd.fd);
    return usb_open_bynumber(busnum, devnum, vendorid, productid);
}

/* USB CONTROL MSG */

int usb_control_msg(unsigned char requesttype, unsigned char request,
                    unsigned short value, unsigned short index, unsigned short length,
                    void *data, unsigned int timeout)
{
    struct usbdevfs_ctrltransfer ctrl;
    int i;

    ctrl = (struct usbdevfs_ctrltransfer){ requesttype, request, value, index, length, timeout, data };
    i = ioctl(dev->fd, USBDEVFS_CONTROL, &ctrl);
    if(i < 0)
        error("USBDEVFS_CONTROL(rqt=0x%x rq=0x%x val=%u idx=%u len=%u) error %s",
              requesttype, request, value, index, length, strerror(errno));
    return i;
}

/* WRITE CPU CS */

void writecpucs(unsigned char buf) {
    int r;

    // usb_control_msg(struct usb_device *dev, unsigned int pipe, __u8 request, __u8 requesttype, __u16 value, __u16
    index, void *data, __u16 size, int timeout);
    r = usb_control_msg(VENDOR_REQUEST_OUT, ANCHOR_LOAD, CPU_CONTROL_REGISTER, 0, 1, &buf, timeout);
    if(r != 1)
        error("writecpucs(0x%02x) failed %d", buf, r);
}

void reset_anchor(void) {
    if(verbose)
        displaycpucs();
    writecpucs(1);
}

```

```

    if(verbose)
        displaycpucs();
}

void unreset_anchor(void) {
    if(verbose)
        displaycpucs();
    writecpucs(0);
    if(verbose)
        displaycpucs();
}

/* DISPLAY MEM */

void displaymem() {
    unsigned char buf[16];
    unsigned addr, u;
    int r;

    printf("EZUSB Memory contents:\n");
    for(addr = 0; addr < 0x2000; addr += 16) {
        r = usb_control_msg(VENDOR_REQUEST_IN, ANCHOR_LOAD, addr, 0, 16, buf, timeout);
        if(r != 16)
            error("usb_control_msg(sz=%d,addr=0x%04x) returned %d", 16, addr, r);
        printf("%04x:", addr);
        for (u = 0; u < 16; u++)
            printf(" %02x", buf[u]);
        printf("\n");
    }
    r = usb_control_msg(VENDOR_REQUEST_IN, ANCHOR_LOAD, CPU_CONTROL_REGISTER, 0, 1, buf, timeout);
    if(r != 1)
        error("readcpucs returned %d\n", r);
    printf("CPUCS: %02x\n", buf[0]);
}

/* DISPLAY CPU CS */

int displaycpucs() {
    unsigned char buf;
    int r;

    r = usb_control_msg(VENDOR_REQUEST_IN, ANCHOR_LOAD, CPU_CONTROL_REGISTER, 0, 1, &buf, timeout);
    if(r != 1)
        error("readcpucs returned %d\n", r);
    printf("CPUCS: %02x\n", buf);
    return 0;
}

/* HEX DIGIT */

int hexdigit(const char *s) {
    if (*s >= '0' && *s <= '9')
        return *s - '0';
    if (*s >= 'A' && *s <= 'F')
        return *s - 'A' + 10;
    if (*s >= 'a' && *s <= 'f')
        return *s - 'a' + 10;
    error("not a hex digit '%s'", pp_str(s));
}

/* HEX BYTE */

int hexbyte(const char *s) {
    int a, b;

    a = hexdigit(s);
    b = hexdigit(s+1);
    return (a << 4) | b;
}

/* HEX WORD */

int hexword(char *s) {
    return (hexbyte(s) << 8) + hexbyte(s+2);
}

/* LOAD HEX */

void load_hex(char *filename, void (*callback)(unsigned char *buf, int address, int len)) {

```

```

    FILE *f;
    unsigned char buf[1000], *p;
    unsigned checksum;
    int h, b, end, line=0, len, address, type, i;
    unsigned char goods[1000];
    unsigned char memory[10000];

    if(!(f = fopen(filename, "r")))
        sys_error("open %s", filename);

    memset(memory, 0, sizeof(memory)); // use to check for overlapping hex records

    // :02000000215786
    // : 02 0000 00 21 57 86
    // : len addr type data checksum
    // type=0 for data, 1 for end of record

    while (fgets(buf, sizeof(buf), f)) {
        line++;
        buf[strlen(buf)-1] = 0; // eat \n
        p = buf;
        if (buf[0] != ':')
            error("invalid intel hex record '%s'", buf);
        p++; // skip ':'
        checksum = 0;
        len = hexbyte(p);
        checksum += len;
        p += 2;
        address = hexword(p);
        // uuuuuugly, maybe make checksum global?
        checksum += address & 0xFF;
        checksum += address >> 8;
        p += 4;
        type = hexbyte(p);
        checksum += type;
        if(type == 1)
            break;
        p += 2;
        for(b=0; b<len; b++,p+=2) {
            i = hexbyte(p);
            goods[b] = i;
            checksum += i;
        }
        checksum += hexbyte(p);
        if((checksum & 0xFF) != 0)
            error("%s:%d checksum error %02x", filename, line, checksum & 0xFF);

        // any overlap?
        if(address+len > sizeof(memory))
            error("memory too small %d", address+len);

        for(i=0; i<len; i++)
            if(memory[address+i]) {
                printf("%04x already written\n", address+i);
                overlap++;
            }
        // remember that we wrote here already
        for(i=0; i<len; i++)
            memory[address+i] = 1;

        (*callback)(goods, address, len);
    }
    fclose(f);
}

int byte_count;

/* CALLBACK */

void callback(unsigned char *buf, int address, int len) {
    int r;
    if(len > 64)
        error("64 max");
    r = usb_control_msg(VENDOR_REQUEST_OUT, ANCHOR_LOAD, address, 0, len, buf, timeout);
    if(verbose) {
        int i, count=0;
        printf("%04x ", address);
        for(i=0; i<len && i<16; count++,i++)
            printf(" %02x", buf[i]);
    }
}

```

```

    for(;count<=16;count++)
        printf(" ");
    for(i=0; i<len && i<16; i++)
        printf("%c", isalnum(buf[i]) ? buf[i] : '.');
    if(len >= 16)
        printf(" <more>");
    printf("\n");
}
if(r != len)
    error("usb_control_msg(address=%04x len=%04x buf=%08x) result=%d expected %d",
        address, len, buf, r, len );
byte_count += len;
}

/* ANCHOR DOWNLOAD HEX */

int anchor_download_hex(char *filename) {
    byte_count = 0;
    reset_anchor();
    load_hex(filename, callback);
    //displaymem();
    printf("downloaded %d bytes\n", byte_count);
    if(overlap)
        printf("%d overlaps (not good)\n", overlap);
    unreset_anchor();
}

/* USAGE */

void usage(void) {
    error("usage: dl [-v] -h <hexfile>");
}

/* MAIN */

int main(int argc, char *argv[]) {
    int pid = 0xBeef;
    static const struct option long_options[] = {
        { "usbdevfs", 1, 0, 'D' },
        { 0, 0, 0, 0 }
    };
    int c, err = 0, busnum = 1, devnum = -1, fdownload = 0;
    char *par = NULL, *hex_filename = NULL;
    struct sigaction sigact;
    struct usb_device_descriptor desc;

    printf("dl by Andrew Burgess, based on code by Thomas Sailer\n");
    while ((c = getopt_long(argc, argv, "vD:d:b:p:h:", long_options, NULL)) != EOF) {
        switch (c) {
            case 'v':
                verbose=1;
                break;
            case 'h':
                hex_filename = strdup(optarg);
                break;

            case 'D':
                usb_devicefs_mountpoint = optarg;
                break;

            case 'p':
                par = optarg;
                break;

            case 'd':
                devnum = strtoul(optarg, NULL, 0);
                break;

            case 'b':
                busnum = strtoul(optarg, NULL, 0);
                break;

            default:
                usage();
                break;
        }
    }
    if(!hex_filename)
        usage();
}

```

```
#define VIRGIN_PID 0x2131
{
    struct { int vendor; int product; } trys[] = {
        { 0x0547, VIRGIN_PID },
        { 0x0547, 0xBEEF },
        { 0x1234, 0x5678 },
        { 0 }
    };
    int t;
    for(t=0; trys[t].vendor != 0; t++ ) {
        printf("Try %04x:%04x\n", trys[t].vendor, trys[t].product);
        if(devnum != -1)
            dev = usb_open_bynumber(busnum, devnum, trys[t].vendor, trys[t].product);
        else
            dev = usb_open(trys[t].vendor, trys[t].product, 0);
        if(dev)
            break;
    }
}
if(!dev)
    error("No USB device found");
anchor_download_hex(hex_filename);
usb_close();
exit(0);
}
```