DNW源码解析

2013年04月23日 09:33:21 亲亲台宝 阅读数：4790 标签： usb c 更多

一、 数据传输格式

USB Tx format:

addr(4)+size(4)+data(n)+cs(2)

addr：下载到开发板RAM的目标地址， 4字节

size：传输文件的大小， 4字节

data：文件数据流， n字节

cs： checksum校验和， 2字节

二、 MenuUsbTransmit发送文件流程

1. 打开usb输出管道文件open_file(outPipe)，如果失败，报错；
2. 选择并打开所要传输的文件；
3. 获取文件大小；fileSize=GetFileSize(hFile,NULL);
4. 分配一个(fileSize+10)大小的缓冲区；4+4+2=10
5. 将文件读入缓冲区txBuf+8（即保留前8个字节）开始处；
6. 将所设置的下载地址赋给txBuf[0:3]4个字节；downloadAddress;
7. 将fileSize+10赋给txBuf[4:7]4个字节
8. 将校验和赋给缓冲区的最后2字节(txBuf+8+fileSize))=cs;
9. 启动usb传输UsbTxFile()

DNW编译记录：

DNW的源码下载之后，直接编译和运行——当然这是你迫切想去做的了。不过此时可能遇麻烦，是一系列“头文件无法识别”的错误。此时你要做的：

1.将WINDDK下的WXP目录下的头文件添加到vc下的include文件中。此时你只需要将E:\WINDDK\2600\inc\wxp中的 wxp文件，"ctrl+c"&&"ctrl+v"到C:\Program Files\Microsoft Visual Studio\VC98\Include中。当然，前提是你已经安装了WINDDK和VC到相应的目录之下。这样你的编译器便不会对这些重要的“.h”视而不见了。

2.将WINDDK目录E:\WINDDK\2600\lib\wxp\i386下的usbd.lib,setupapi.lib添加到vc下的lib文件中

3.如果碰到：fatal error LNK1112: module machine type "IA64" conflicts with target machine type "IX86"（模块计算机类型“IA64”与目标计算机类型“IX86”冲突），首先，进入VC下的LINK，如图

并且要将Project Options下的内容手动修改成图中的内容——我特别指的是machine:IX86和最后一行的\lib\i386\... 然后将E:\WINDDK\2600\lib\wxp\i386下的库文件mfc42u.lib到C:\Program Files\Microsoft Visual Studio\VC98\Lib之中。注意，不是E:\WINDDK\2600\lib\wxp\ia64目录下的那个mfc42u.lib

Linux下编译记录：

编译PC端USB驱动和写入工具

dnw_linux.tgz压缩包文件结构如下

```
dnw_linux/
dnw_linux/secbulk/
dnw_linux/secbulk/Makefile
dnw_linux/secbulk/secbulk.c
dnw_linux/dnw/
dnw_linux/dnw/dnw.c
```

其中secbulk.c是PC端USB驱动, dnw.c是写入工具

编译驱动之前先修改secbulk.c

找到#define BULKOUT_BUFFER_SIZE
修改为
```
#define BULKOUT_BUFFER_SIZE 512
```

找到
```
static struct usb_device_id secbulk_table[]= {
{ USB_DEVICE(0x04e8, 0x1234)},
{ }
};
```
修改为上面的样子

接下来编译

```
$cd secbulk
$make -C /lib/modules/`uname -r`/build M=`pwd` modules
```

加载编译好的驱动

```
$sudo insmod ./secbulk.ko
```

注意，每次下载前都需要加载驱动，或者可以设置为开机自动加载
Ubuntu中，假设驱动文件在/opt/dnw_linux/secbulk/

则修改/etc/rc.d/rc.local文件，末尾加入

```
sudo insmod /opt/dnw_linux/secbulk/secbulk.ko
```

即完成开机自动加载驱动模块

接下来编译dnw写入工具
先打开dnw.c修改
找到
```
printf("Writing data...\n");
size_t remain_size = file_stat.st_size+10;
size_t block_size = remain_size / 100;
size_t writed = 0;
```

在它前面加上2行代码，如下：

```
file_buffer [file_stat.st_size + 8] = sum & 0xff;
file_buffer [file_stat.st_size + 9] = sum >> 8;

printf("Writing data...\n");
size_t remain_size = file_stat.st_size+10;
size_t block_size = remain_size / 100;
size_t writed = 0;
```

编译dnw

```
$gcc -o dnw dnw.c
```

编译成功后生成可执行dnw

使用DNW下载

启动开发板，进入minicom，并将开发板和PC用USB电缆连接，此时用dmesg命令可以看到secbulk驱动加载：
```
[ 283.677772] usb 1-1: new full speed USB device using uhci_hcd and address 2
[ 284.084835] usb 1-1: configuration #1 chosen from 1 choice
[ 284.140430] secbulk:secbulk probing...
[ 284.140482] secbulk:bulk out endpoint found!
```
说明驱动可以使用

重起开发板，别进linux系统，按任意键进入uboot界面，
输入

```
dnw 50008000
```

当出现“USB host is connected. Waiting a download.”时，
在PC端Linux上用dnw工具写入要下载的文件，例如我要写入/tmp/zImage

```
$./dnw /tmp/zImage
```

写入完成后提示成功
```
100% 312349 bytes OK
```

至此，dnw在linux下使用一切正常

210通过DNW下载文件说明：

4. dnw0.5修改说明
4.1 winMain()->Register(HINSTANCE hInst)-> 回调函数WndProc()->设置一个定时器（用来定时探测usb是否连接）->消息检测循环
4.2 按下菜单栏中USB Port的Transmit->MenuUsbTransmit(HWND hwnd)(在此修改的代码)，以下为主要修改或添加的代码
4.2.1查看驱动知道GUID要修改为 DEFINE_GUID(GUID_CLASS_I82930_BULK, 0xa5dcbf10, 0x6530, 0x11d2, 0x90, 0x1f, 0x00, 0xc0, 0x4f,0xb9, 0x51, 0xed);
4.2.2 UsbSendAckData()函数是3.1中用来发送“ATUD”通知210板子的, 现在这个函数是放在MenuUsbTransmit中，所以只有按下菜单栏中USB Port的Transmit，才会通知210板子下载，也可放入WndProc()中设置的定时器服务函数中，这样一打开dnw便进行连接。
4.3.3 FilebufToTxbuf()函数为3.8 和3.9 协议

DNW-LINUX源码分析：

源代码地址：http://code.google.com/p/dnw-linux/

参考文章：http://www.cnblogs.com/QuLory/archive/2012/11/16/2773389.html

http://blog.csdn.net/yming0221/article/details/7211396

1.原理

DNW原理就是通过PC端软件把要烧写的镜像（uboot，kernel，fs）通过usb口写进usb设备的RAM中，然后USB设备再把RAM里的数据写到rom（nandflash，emmc等）中实现固化程序。想比较直接从SD端口直接固化程序麻烦了许多，但是对于很多没有sd卡接口的设备却是必须的.

2.使用

下载源代码，然后进入目录。输入命令sudo make install，注意这里需要root权限。

Makefile文件如下：

```
 3 driver_src = `pwd`/src/driver
  4 dnw_src = src/dnw
  5
  6 all: driver dnw
  7
  8 driver:
  9     make -C /lib/modules/`uname -r`/build M=$(driver_src) modules
 10
 11 dnw:
 12     make -C $(dnw_src)
 13
 14 install: all
 15     make -C $(dnw_src) install
 16     make -C /lib/modules/`uname -r`/build M=$(driver_src) modules_install
 17     cp dnw.rules /etc/udev/rules.d/
 18     depmod
 19
 20 clean:
 21     make -C $(dnw_src) clean
 22     make -C /lib/modules/`uname -r`/build M=$(driver_src) clean
```

make指令编译出应用和驱动，没有进行安装，所以不须要root权限

```
gexueyuan@gexueyuan:~/Downloads/dnw$ make
make -C /lib/modules/`uname -r`/build M=`pwd`/src/driver modules
make[1]: Entering directory `/usr/src/linux-headers-3.2.0-36-generic'
  CC [M]  /home/gexueyuan/Downloads/dnw/src/driver/secbulk.o
/home/gexueyuan/Downloads/dnw/src/driver/secbulk.c: In function ‘secbulk_write’:
/home/gexueyuan/Downloads/dnw/src/driver/secbulk.c:60:14: warning: comparison of
 distinct pointer types lacks a cast [enabled by default]
  Building modules, stage 2.
  MODPOST 1 modules
  CC      /home/gexueyuan/Downloads/dnw/src/driver/secbulk.mod.o
  LD [M]  /home/gexueyuan/Downloads/dnw/src/driver/secbulk.ko
make[1]: Leaving directory `/usr/src/linux-headers-3.2.0-36-generic'
make -C src/dnw
make[1]: Entering directory `/home/gexueyuan/Downloads/dnw/src/dnw'
gcc -g -o dnw dnw.c
make[1]: Leaving directory `/home/gexueyuan/Downloads/dnw/src/dnw'
gexueyuan@gexueyuan:~/Downloads/dnw$ 
```

make install则需要root权限。

在pc端使用dnw将需要下载的镜像文件写入usb设备ram

```
$sudo ./dnw [-a load_addr] /filepath/filename
```

3.源代码分析：

驱动文件secbulk.c这个文件没什么好说的，usb设备驱动的模型，填入对应代码，要注意的是

```
static struct usb_device_id secbulk_table[]= {
        { USB_DEVICE(0x5345, 0x1234) }, /* FS2410 */
        { USB_DEVICE(0x04e8, 0x1234) }, /* EZ6410 */
        { }
};
```

这里设置的pid和vid要与设备对应，否则驱动无法识别。

应用程序dnw.c

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <malloc.h>
#include <errno.h>
#include <sys/types.h>
#include <sys/stat.h>
#include <sys/time.h>
#include <unistd.h>
#include <fcntl.h>
#include <stdint.h>

const char* dev = "/dev/secbulk0";    //dnw所创建的设备文件，要对其写入
#define BLOCK_SIZE      (1*1024*1024) //设置的写入块大小1MB

struct download_buffer {
        uint32_t        load_addr;  /* load address */
        uint32_t        size; /* data size *///size=地址（4位）+大小（4位）+数据+校验（2位）=
        uint8_t         data[0];//0长度数组，指向数据
        /* uint16_t checksum; */数组后紧接着的两位是校验位
};

static int _download_buffer(struct download_buffer *buf)//从缓存写入到usb设备文件
{
        int fd_dev = open(dev, O_WRONLY);//打开设备
        if( -1 == fd_dev) {
                printf("Can not open %s: %s\n", dev, strerror(errno));
                return -1;
        }

        printf("Writing data...\n");
        size_t remain_size = buf->size;//写入文件的剩余大小
        size_t block_size = BLOCK_SIZE;//每次写入的大小
        size_t writed = 0;//已经写入的文件大小
        while(remain_size>0) {
                size_t to_write = remain_size > block_size ? block_size : remain_size;//每次写入的实际大小
                if( to_write != write(fd_dev, (unsigned char*)buf + writed, to_write)) {
                        perror("write failed");
                        close(fd_dev);
                        return -1;
                }
                remain_size -= to_write;
                writed += to_write;
                printf("\r%02zu%%\t0x%08zX bytes (%zu K)",
                        (size_t)((uint64_t)writed*100/(buf->size)),
                        writed,
                        writed/1024);//打印写入的百分比
                fflush(stdout);//将缓存写入文件
        }
        printf("\n");
        close(fd_dev);
        return 0;
}

static inline void cal_and_set_checksum(struct download_buffer *buf)
{
        uint16_t sum = 0;
        int i;

        for(i = 0; i < buf->size; i++) {
                sum += buf->data[i];
        }
        *((uint16_t*)(&((uint8_t*)buf)[buf->size - 2])) = sum;//校验码赋值给最后一个word
}

static struct download_buffer* alloc_buffer(size_t data_size)//分配空间的函数
{
        struct download_buffer  *buffer = NULL;
        size_t total_size = data_size + sizeof(struct download_buffer) + 2;buffer=文件大小+结构体前两项的大小+2

        buffer = (typeof(buffer))malloc(total_size);
        if(NULL == buffer)
                return NULL;
        buffer->size = total_size;
        return buffer;//返回指向结构体的指针
}

static void free_buffer(struct download_buffer *buf)
{
        free(buf);
}

static struct download_buffer *load_file(const char *path, unsigned long load_addr)//载入文件到缓存
{
        struct stat             file_stat;
        struct download_buffer  *buffer = NULL;
        unsigned long           total_size;
        int                     fd;

        fd = open(path, O_RDONLY);//通过路径打开文件，获得fd文件标识符
        if(-1 == fd) {
                printf("Can not open file %s: %s\n", path, strerror(errno));
                return NULL;
        }

        if( -1 == fstat(fd, &file_stat) ) {//获取文件的属性
                perror("Get file size filed!\n");
                goto error;
        }

        buffer = alloc_buffer(file_stat.st_size);//给buffer分配空间（文件占用空间+结构体空间+2位校验）
        if(NULL == buffer) {
                perror("malloc failed!\n");
                goto error;
        }
        if( file_stat.st_size !=  read(fd, buffer->data, file_stat.st_size)) {//将文件写入buffer->data
                perror("Read file failed!\n");
                goto error;
        }

        buffer->load_addr = load_addr;//填充结构体
        cal_and_set_checksum(buffer);//校验数据

        return buffer;

error:
        if(fd != -1)
                close(fd);
        if( NULL != buffer )
                free(buffer);
        return NULL;
}

static int download_file(const char *path, unsigned long load_addr)
{
        struct download_buffer *buffer;
        struct timeval __start, __end;
        long __time_val = 0;
        float speed = 0.0;

        buffer = load_file(path, load_addr);//将文件载入到buffer中
        gettimeofday(&__start,NULL);
        if (buffer != NULL) {
                if (_download_buffer(buffer) == 0) {//将缓存中的数据写入usb口
                        gettimeofday(&__end,NULL);
                        __time_val = (long)(__end.tv_usec - __start.tv_usec)/1000 + \
                                (long)(__end.tv_sec - __start.tv_sec) * 1000;
                        speed = (float)buffer->size/__time_val/(1024*1024) * 1000;
                        printf("speed: %fM/S\n",speed);
                        free_buffer(buffer);
                } else {
                        free_buffer(buffer);
                        return -1;
                }
        } else
                return -1;
}

int main(int argc, char* argv[])
{
        unsigned load_addr = 0x57e00000;
        char* path = NULL;
        int     c;

        while ((c = getopt (argc, argv, "a:h")) != EOF)
        switch (c) {
        case 'a':
                load_addr = strtol(optarg, NULL, 16);
                continue;
        case '?':
        case 'h':
        default:
usage:
                printf("Usage: dwn [-a load_addr] <filename>\n");
                printf("Default load address: 0x57e00000\n");
                return 1;
        }
        if (optind < argc)
                path = argv[optind];
        else
                goto usage;

        printf("load address: 0x%08X\n", load_addr);
        if (download_file(path, load_addr) != 0) {
                return -1;
        }

        return 0;
}
```

需要注意的几点：

1. struct download_buffer结构体含有一个零长度数组，不占用结构体空间长度，可以灵活分配空间。

2.cal_and_set_checksum检验函数通过指针偏移量写入到正确位置，位于分配空间的最后两位，数据段零长度数组后面2位。

3.load_addr参数要根据不同的设备，来设定，不指定具体地址，将会采用默认地址0x57e00000