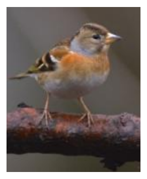


E-mail:yugui.hu@hotmail.com

版本信息	修改信息	作者
V1.0	初始化版本，android 开发常用命令以及调试方法	Hu YuGui
V1.1-20110421	基于 Glibc 的 android rootfs 制作(Busybox 需使用动态库链接，编译器使用 arm-2008q3)	Hu YuGui

Android 驱动开发手册

1.android 系统编译命令

```
(1).make -C ../littleton-kernel O=$PWD/../LITTLETON_OBJ ARCH=arm CROSS_COMPILE=arm-none-linux-gnueabi- zImage
(2).mkbootfs out/target/product/roewe_v2/root | minigzip > out/target/product/roewe_v2/ramdisk.img
(3).mkbootimg --kernel arch/arm/boot/zImage --ramdisk ../out/target/product/roewe_v2/ramdisk.img -o ./boot.img
(4).fastboot flash kernel ../out/target/product/roewe_v2/boot.img
(5).fastboot reboot
```

(6).编译驱动模块的 Makefile

```
obj-m := cis_core.o
obj-m += cis_protocol.o
obj-m += audi_uart_key.o
obj-m += cis_uart.o

KERNELDIR ?= /lib/modules/$(shell uname -r)/build
PWD := $(shell pwd)
default:
    $(MAKE) -C $(KERNELDIR) M=$(PWD) modules
clean:
    rm *.o *.mod.c *.ko
```

2.yaffs2 & SDcard 挂载

```
mount -o rw,remount -t yaffs2 /dev/block/mtdblock3 /system
mount -rw -t vfat -o remount /dev/blockmmcblk1 /sdcard          #重新挂在 SDCARD
```

3.setprop & getprop

(1).ro.sf.lcd_density=240/160   #UI 缩放比例

(2).ro.allow.mock.location=1

(3).ro.debuggable=1   #adb shell 为 root 权限

(4).persist.service.adb.enable=1

4.android busybox[can't access tty; job control turned off]

(1).Why do I keep getting "sh: can't access tty; job control turned off" errors? Why doesn't Control-C work within my shell?This isn't really a uClibc question, but I'll answer it here anyways. Job control will be turned off since your shell can not
obtain a controlling terminal. This typically happens when you run your shell on /dev/console. The kernel will not provide a controlling terminal on the /dev/console device. Your should run your shell on a normal tty such as tty1 or ttyS0 and everything will work perfectly. If you REALLY want your shell to run on /dev/console, then you can hack your kernel (if you are into that sortof thing) by changing drivers/char/tty_io.c to change the lines where it sets "noctty = 1;" to instead set it to "0". I recommend you instead run your shell on a real console...
or change /system/core/init/init.c console_name="/dev/console" to "/dev/ttyS*".

(2)rm /system/bin/ls

busybox ln -s /system/bin/busybox /system/bin/ls

5./root/.kermrc 配置

```
set line /dev/ttyUSB0
set speed 115200
set carrier-watch off
set handshake none
set flow-control none
robust
set file type bin
set file name lit
set rec pack 1000
set send pack 1000
set window 5
```

```
log session /tmp/kermit.log append
c
```

6.git 下载代码

```
git clone git://192.168.9.201/pateo/roewe_android.git/tools/repo.git tools/repo.git
./tools/repo.git/repo init -u git://192.168.9.201/pateo/roewe_android.git/platform/manifest.git -b red_coral
./tools/repo.git/repo sync
./tools/repo.git/repo start red_coral --all
```

使用 git 生成 patch

```
git format-patch commit-id
git am *.patch
```

7.InKaNet 系统激活码

RoEWe8

#*08973	3G 模块激活
#*98723	工厂模式
#*4367349863	保持 Log 到 SD 卡

8.How to open/create the android's ramdisk.img

```
(1).mv ramdisk.img ramdisk.cpio.gz
(2).gunzip ramdisk.cpio.gz
(3).mkdir ramdisk
(4).cd ramdisk
(5).cpio -i -F ../ramdisk.cpio

(6)find . |cpio -o -H newc | gzip -9 > ../ramdisk.img
```

9.shell script example

```
#!/system/bin/sh
```

```
echo "Start to burn the Linux kernel..."
echo "=================================="
#Get the size of the boot.img
KERNEL_SIZE=`./busybox ls -l |./busybox awk '{if ($9=="boot.img") print $5}'`
echo "kernel image size:$KERNEL_SIZE"

ERASE_SIZE=$((($KERNEL_SIZE/131072+1)*131072))
echo "erase size:$ERASE_SIZE"

./mtd_debug info /dev/mtd/mtd3

echo "<1>.start to erase kernel partition"
./mtd_debug erase    /dev/mtd/mtd3 0 $ERASE_SIZE

echo "<2>.start to write kernel image to flash"
./mtd_debug write    /dev/mtd/mtd3 0 $KERNEL_SIZE boot.img

echo "=================================="
echo "Finish!"
```

10.android keycode define

```
framworks/base/include/ui/keycodelabels.h
    { "SOFT_LEFT", 1 },
    { "SOFT_RIGHT", 2 },
    { "HOME", 3 },
    { "BACK", 4 },
    { "CALL", 5 },
    { "ENDCALL", 6 },
    { "STAR", 17 },
    { "POUND", 18 },
    { "DPAD_UP", 19 },
    { "DPAD_DOWN", 20 },
    { "DPAD_LEFT", 21 },
    { "DPAD_RIGHT", 22 },
    { "DPAD_CENTER", 23 },
    { "VOLUME_UP", 24 },
    { "VOLUME_DOWN", 25 },
    { "POWER", 26 },
    { "CAMERA", 27 },
    { "CLEAR", 28 },
    { "COMMA", 55 },
    { "PERIOD", 56 },
    { "ALT_LEFT", 57 },
    { "ALT_RIGHT", 58 },
    { "SHIFT_LEFT", 59 },
    { "SHIFT_RIGHT", 60 },
    { "TAB", 61 },
    { "SPACE", 62 },
    { "SYM", 63 },
    { "EXPLORER", 64 },
    { "ENVELOPE", 65 },
    { "ENTER", 66 },
    { "DEL", 67 },
    { "GRAVE", 68 },
    { "MINUS", 69 },
    { "EQUALS", 70 },
    { "LEFT_BRACKET", 71 },
    { "RIGHT_BRACKET", 72 },
    { "BACKSLASH", 73 },
    { "SEMICOLON", 74 },
    { "APOSTROPHE", 75 },
    { "SLASH", 76 },
    { "AT", 77 },
    { "NUM", 78 },
    { "HEADSETHOOK", 79 },
    { "FOCUS", 80 },
    { "PLUS", 81 },
    { "MENU", 82 },
    { "NOTIFICATION", 83 },
    { "SEARCH", 84 },
    { "MEDIA_PLAY_PAUSE", 85 },
    { "MEDIA_STOP", 86 },
    { "MEDIA_NEXT", 87 },
    { "MEDIA_PREVIOUS", 88 },
    { "MEDIA_REWIND", 89 },
    { "MEDIA_FAST_FORWARD", 90 },
    { "MUTE", 91 },
```

input keyevent 4;输入 BACK 按键

11.TSLIB 使用

五点校准：

```
int axis_table[] = {54194, 39, -1920576, -719, -36922, 33514374, 16 };    #axis_table[6] = 65536,使用移位是为 16
int sample_x, sample_y;     #定义一个坐标变量
tem_x = (axis_table[2] + axis_table[0]*sample_x + axis_table[1]*sample_y) >> axis_table[6];
tem_y = (axis_table[5] + axis_table[3]*sample_x + axis_table[4]*sample_y) >> axis_table[6];
tem_x = (tem_x > X_MAX) ? X_MAX : tem_x;
tem_x = (tem_x < X_MIN) ? X_MIN : tem_x;
tem_y = (tem_y > Y_MAX) ? Y_MAX : tem_y;
tem_y = (tem_y < Y_MIN) ? Y_MIN : tem_y;
```

```
#/bin/sh
echo "Compile the TSLIB testsuite."
arm-none-linux-gnueabi-gcc -static fbutils.c font_8x16.c font_8x8.c ts_calibrate.c -o TSLIB
echo "OK!"
```

ts_calibrate.c 修改：

```
put_cross(50, 50, 1);     #后面一个参数为查找颜色表的序号
put_cross(xres - 50, 50, 1);
put_cross(xres - 50, yres - 50, 1);
put_cross(50, yres - 50, 1);
put_cross(xres/2, yres/2, 1);
```

12.BC 的使用

```
echo "ibase=16;3FF" |bc
echo "scale=3;1/3" |bc
```

13.find ./ -name *.c |xargs grep -nr --color "read"

14.mkdosfs -F 32 /dev/block/mmcblk0p1 #格式化 MMC 卡

15.将 Inand 当作 U 盘

```
insmod g_file_storage.ko file=/dev/block/vold/179:0
```

16.How to compile the SDL.

```
#Writed by pecker.hu@gmail.com
#Date:2010/11/02 Nanjing
#!/bin/sh

SDL_DIR=$PWD/SDL-1.2.14
ZLIB_DIR=$PWD/zlib-1.2.5
LIBPNG_DIR=$PWD/libpng-1.4.4
SDL_IMAGE_DIR=$PWD/SDL_image-1.2.10
GUICHAN_DIR=$PWD/guichan-0.8.2
CROSS_COMPILE=arm-linux
INSTALL_DIR=$PWD/install

echo "==========================================="
echo "1.SDL(Simple DirectMedia Layer)"
echo "2.Zlib-1.2.5"
echo "3.libpng-1.4.4"
echo "4.SDL_image-1.2.10"
echo "5.GUICHAN librarys"
echo "6.GUICHAN example->"
echo "q.Exit Menu"
echo "==========================================="
echo "Please select menu>"

read number

case $number in
        "1")
        echo "Start to compile the SDL-1.2.14"
        cd $SDL_DIR
        make distclean
        ./configure --prefix=$INSTALL_DIR --disable-video-photon --disable-video-cocoa --disable-video-directfb --enable-video-fbcon
        --disable-video-ps2gs --disable-video-ps3 --disable-video-svga --disable-video-vgl --disable-video-wscons --disable-video-xbios
        --disable-video-gem  --disable-video-dummy  --disable-video-opengl  --disable-video-x11  --disable-dga  --disable-input-tslib
        --disable-audio --disable-cdrom --disable-joystick --disable-loadso --disable-sdl-dlopen --host=$CROSS_COMPILE
        make all
        make install
        ;;

        "2")
        echo "Start to compile the zlib"
        export CC=$CROSS_COMPILE-gcc
        export AR=$CROSS_COMPILE-ar
        export RANLIB=$CROSS_COMPILE-ranlib
        cd $ZLIB_DIR
        make distclean
        ./configure --prefix=$INSTALL_DIR
        make all
        make install
        ;;
```

```
"3")
echo "Start to compile the libpng"
cd $LIBPNG_DIR
make distclean
./configure --prefix=$INSTALL_DIR --enable-static --host=$CROSS_COMPILE CFLAGS="-I$INSTALL_DIR/include
-L$INSTALL_DIR/lib -lz"
make all
make install
;;

"4")
echo "Start to compile the SDL_image"
cd $SDL_IMAGE_DIR
make distclean
./configure --prefix=$INSTALL_DIR --enable-static --host=$CROSS_COMPILE --disable-sdltest --enable-bmp --disable-jpg
--disable-lbm --disable-pcx --enable-png --disable-tga --disable-tif --disable-xcf --disable-xpm --disable-xv
SDL_CFLAGS="-I$INSTALL_DIR/include/SDL" SDL_LIBS="-L$INSTALL_DIR/lib -lSDL" CFLAGS="-I$INSTALL_DIR/include
-L$INSTALL_DIR/lib -lpng -lz"
make all
make install
;;

"5")
echo "Start to compile the GUICHAN librarys"
cd $GUICHAN_DIR
make distclean
./configure --prefix=$INSTALL_DIR --host=$CROSS_COMPILE --enable-force-sdl --enable-force-sdlimage
CXXFLAGS="-I$INSTALL_DIR/include -I$INSTALL_DIR/include/SDL -L$INSTALL_DIR/lib -lSDL_image -lSDL -lpng -lz -lpthread"
make all
make install
;;

"6")
cd $GUICHAN_DIR/examples
```

```
        echo "=========================================="
        echo "1.sdlhelloworld"
        echo "2.sdlwidgets"
        echo "q.exit menu"
        echo "=========================================="
        echo "Please select menu>"
        read testcasenum

        case $testcasenum in
            "1")
            echo "Compile GUICHAN example:sdlhelloworld."
            $CROSS_COMPILE-g++  -static  sdlhelloworld.cpp  -o  sdlhelloworld  -I$INSTALL_DIR/include  -I$INSTALL_DIR/include/SDL
            -L$INSTALL_DIR/lib -lguichan_sdl -lguichan -lSDL_image -lSDL -lpng -lz -lpthread
            $CROSS_COMPILE-strip sdlhelloworld
            ;;

            "2")
            echo "Compile GUICHAN example:sdlwidgets."
            $CROSS_COMPILE-g++   -static   sdlwidgets.cpp   -o   sdlwidgets   -I$INSTALL_DIR/include   -I$INSTALL_DIR/include/SDL
            -L$INSTALL_DIR/lib -lguichan_sdl -lguichan -lSDL_image -lSDL -lpng -lz -lpthread
            $CROSS_COMPILE-strip sdlwidgets
            ;;
        esac
        ;;

        "q")
        echo "exit menu!!!"
        exit 0
        ;;
esac
exit 0
```

When use "-static" flag to compile the image,we should include the static librarys with order

17.Android PM

(1).启动命令行参数：no_console_suspend

(2).cat /sys/power/wake_lock

(3).I2c 电源需 CPU 睡眠后方可断开

18.Android miniRootfs 制作

1). Use the ARM EABI to compile the kernel（AEABI [=y]）

2).使用 arm-none-linux-gnueabi-gcc 静态编译 busybox(make defconfig/make menuconfig/make all/make install)

3).创建以下文件夹

```
cd _install
mkdir dev etc proc sys tmp usr
mkdir etc/init.d
ln -s bin/busybox init
```

4).在 dev 目录创建必要的设备节点

```
cp /dev/console dev/
#不创建 console 这个节点，系统停住，最后输出以下 log
#[        2.084228] Freeing init memory: 140K
#[        2.088012] Warning: unable to open an initial console.
cp /dev/null dev/
cp /dev/tty2 dev
cp /dev/tty3 dev
cp /dev/tty4 dev
```

5).脚本代码(/etc/init.d/rcS & /init.rc)

```
#/init.rc
#!/bin/sh
export PATH=/bin:/sbin
echo "~~~~~~~~$PWD/init.rc~~~~~~~~"
mount -t proc none /proc
mount -t sysfs none /sys
mdev -s #实现 udev 功能
mount -t yaffs2 /dev/mtdblock4 /system
mount -t yaffs2 /dev/mtdblock6 /data

#/etc/init.d/rcS
```

#!/bin/sh

export PATH=/bin:/sbin

echo "~~~~$PWD/rcS~~~~"

echo "Rootfs Author:Hu Yugui"

echo "E-mail:yugui.hu@hotmail.com"

/init.rc

6).打包 ramdisk.img

find . |cpio -o -H newc | gzip -9 > ../ramdisk.img

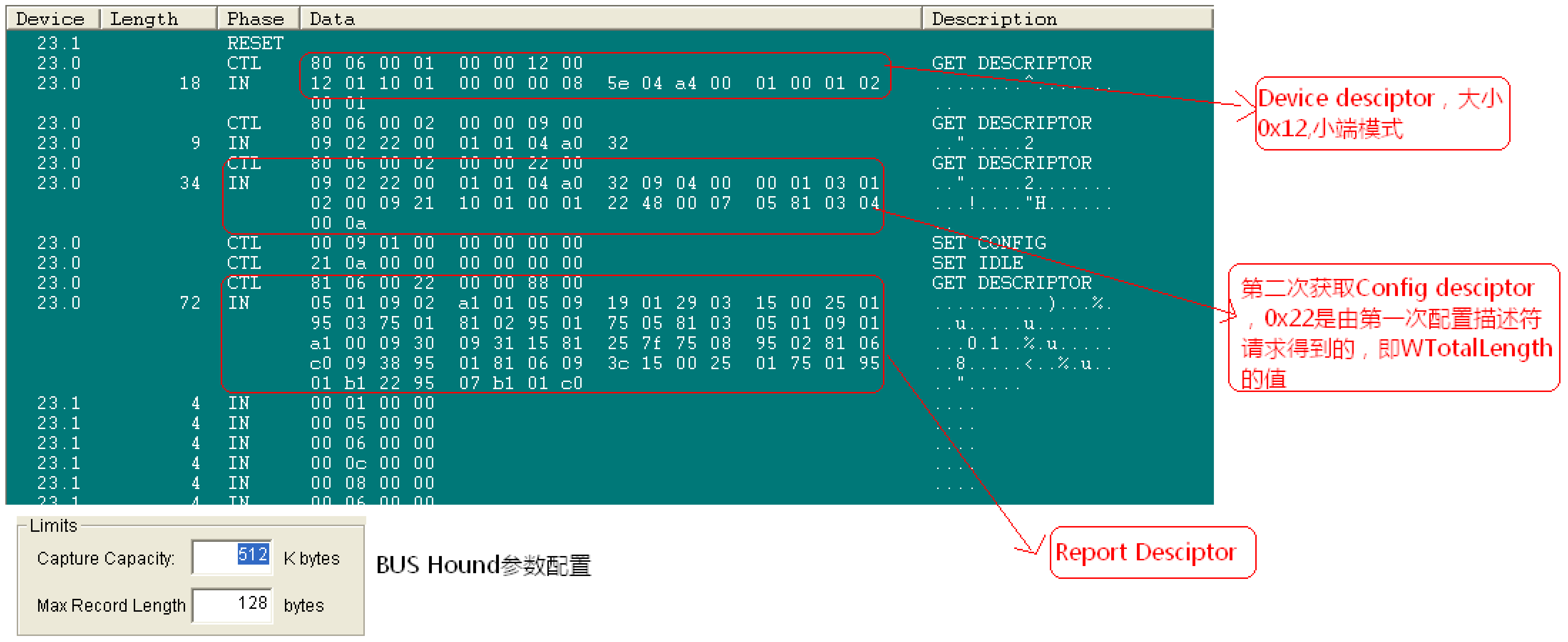
mkbootimg --kernel /huyugui/roewe_redcoral/out/target/product/roewe_v2/obj/KERNEL_OBJ/arch/arm/boot/zImage --ramdisk ramdisk.img -o miniroot.img

7).rootfs 使用标准 glibc 库(armv5:arm-2008q3/arm-none-linux-gnueabi/libc/lib/*)

/etc/profile 中需要导出环境变量：export LD_LIBRARY_PATH=/lib:/usr/lib

Busybox 不要使用静态编译，否者动态库不可用

19.USB HID Spec 分析



```
Device  Length  Phase  Data                                                     Description
  23.1          RESET
  23.0          CTL    80 06 00 01  00 00 12 00                                 GET DESCRIPTOR
  23.0      18  IN     12 01 10 01  00 00 00 08  5e 04 a4 00  01 00 01 02       ........^.......
                       00 01                                                    ..
  23.0          CTL    80 06 00 02  00 00 09 00                                 GET DESCRIPTOR
  23.0       9  IN     09 02 22 00  01 01 04 a0  32                             .."....2
  23.0          CTL    80 06 00 02  00 00 22 00                                 GET DESCRIPTOR
  23.0      34  IN     09 02 22 00  01 01 04 a0  32 09 04 00  00 01 03 01       .."....2........
                       02 00 09 21  10 01 00 01  22 48 00 07  05 81 03 04       ...!...."H......
                       00 0a                                                    ..
  23.0          CTL    00 09 01 00  00 00 00 00                                 SET CONFIG
  23.0          CTL    21 0a 00 00  00 00 00 00                                 SET IDLE
  23.0          CTL    81 06 00 22  00 00 88 00                                 GET DESCRIPTOR
  23.0      72  IN     05 01 09 02  a1 01 05 09  19 01 29 03  15 00 25 01       ..........)...%.
                       95 03 75 01  81 02 95 01  75 05 81 03  05 01 09 01       ..u.....u.......
                       a1 00 09 30  09 31 15 81  25 7f 75 08  95 02 81 06       ...0.1..%.u.....
                       c0 09 38 95  01 81 06 09  3c 15 00 25  01 75 01 95       ..8.....<..%.u..
                       01 b1 22 95  07 b1 01 c0                                 .."....
  23.1       4  IN     00 01 00 00                                              ....
  23.1       4  IN     00 05 00 00                                              ....
  23.1       4  IN     00 06 00 00                                              ....
  23.1       4  IN     00 0c 00 00                                              ....
  23.1       4  IN     00 08 00 00                                              ....
  23.1       4  IN     00 06 00 00
```

Limits
Capture Capacity: 512 K bytes
Max Record Length 128 bytes

Device desciptor，大小0x12,小端模式
第二次获取Config desciptor，0x22是由第一次配置描述符请求得到的，即WTotalLength的值
Report Desciptor

BUS Hound参数配置

注:通过 Config Desciptor 的请求可以得到 Config Desciptor,Interface Desciptor & EndPointer Desciptor

20.z-modem 工具 lrzsz 发送和接收

1）修改该 Makefile 文件（android 下静态编译）：

```
1 # Makefile for Unix/Xenix rz and sz programs
2 # Some targets may not be up to date
3 CC=arm-none-linux-gnueabi-gcc
```

4 OFLAG= -O –static

2）make posix

Export RZSZLINE=/dev/modem

rz 用于接收文件，sz 用于发送文件

21.mtd-utils 工具集使用

下载 mtd-utils-1.2.0-HYG.tgz

arm-none-linux-gnueabi-gcc -static nandwrite.c -o nandwrite -I$PWD/include #nand 烧写工具

arm-none-linux-gnueabi-gcc -static mtd_debug.c -o mtd_debug -I$PWD/include #nand erase 工具

(1)./mtd_debug erase /dev/mtd/mtd4 0 分区大小

(2)./nandwrite –a –o /dev/mtd/mtd4 system.img #yaffs2 image

(3) ./nandwrite /dev/mtd/mtd4 boot.img #kernel image #***烧写之前需要 erase***

[mtd-utils bugfix] Many people encountered this problerm, creating a image by mkyaffs2image, then write it into a nand falsh with nandwrite, then mounted failed.
This is because mkyaffs2image didn't know the oob layout of a NAND flash,
so it put the yaffs2 tags at the offset 0 of oob area, nandwrite didn't
put it at right position when writing oobdata.

22./dev/loop0 设备使用

可以将文件挂载成块设备，并格式化成相应的文件系统（ext3，FAT，etc）

# dd if=/dev/zero of=FS_on_file bs=1k count=10000

# losetup /dev/loop0 FS_on_file 将文件装载到回环设备上

# mkfs -t ext3 /dev/loop0 格式化文件

# mkdir FS_on_file0

# mount /dev/loop0 ./FS_on_file0/ 挂载

# umount /dev/loop0

# losetup -d /dev/loop0

23.poll 函数使用（APP &kernel）

static unsigned int apm_poll(struct file *fp, poll_table * wait)

```
{
    struct apm_user *as = fp->private_data;

    poll_wait(fp, &apm_waitqueue, wait); //该函数将执行 poll 函数的进程加入到等待队列头
    return queue_empty(&as->queue) ? 0 : POLLIN | POLLRDNORM;
}
```

##根据返回的结果决定是否阻碍应用程序##

ret=poll((struct pollfd *)&event,1,5000); //监测 event，一个对象，等待 5000 毫秒后超时,-1 为无限等待

例子：

```
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <sys/ioctl.h>
#include <sys/types.h>
#include <sys/stat.h>
#include <fcntl.h>     /*文件控制*/
#include <sys/select.h>
#include <sys/time.h>   /*时间方面的函数*/
#include <errno.h>   /*有关错误方面的宏*/
#include<sys/poll.h>   //poll()
#include<fcntl.h>
#include<string.h>    //memset()

int main(void)
{
        int fd,key_value,ret;
        struct pollfd event;  //创建一个 struct pollfd 结构体变量，存放文件描述符、要等待发生的事件
        fd=open("/dev/key",O_RDWR);
        if(fd<0){
                perror("open /dev/key error!\n");
                exit(1);
        }
        printf("open /dev/key sucessfully!\n");
        while(1){  //poll 结束后 struct pollfd 结构体变量的内容被全部清零，需要再次设置
                memset(&event,0,sizeof(event)); //memst 函数对对象的内容设置为同一值
                event.fd=fd;  //存放打开的文件描述符
                event.events=POLLIN;      //存放要等待发生的事件
                ret=poll((struct pollfd *)&event,1,5000); //监测 event，一个对象，等待 5000 毫秒后超时,-1 为无限等待

                //判断 poll 的返回值，负数是出错，0 是设定的时间超时，整数表示等待的时间发生
                if(ret<0){
                        printf("poll error!\n");
                        exit(1);
                }
                if(ret==0){
                        printf("Time out!\n");
                        continue;
                }
                if(event.revents&POLLERR){   //revents 是由内核记录的实际发生的事件，events 是进程等待的事件
                        printf("Device error!\n");
                        exit(1);
                }
                if(event.revents&POLLIN){
                        read(fd,&key_value,sizeof(key_value));
                        printf("Key value is '%d'\n",key_value);
                }
        }
        close(fd);
        return 0;
}
```

Linux Kernel Debug

1.Kernel 反汇编

arm-none-linux-gnueabi-objdump -S   vmlinux > kernel.asm   #带源码的反汇编

arm-none-linux-gnueabi-gcc –g hello.c –o hello   #带调试信息

Panic 分析：

有自己编译的 vmlinux：   使用 gdb

EIP is at list_del+0xa/0x61

这告诉我们，list_del 函数有 0x61 这么大，而 Oops 发生在 0xa 处。 那么我们先看一下 list_del 从哪里开始：

```
# grep list_del /boot/System.map-2.6.24-rc3-module
c10e5234 T plist_del
c10e53cc T list_del
c120feb6 T klist_del
c12d6d34 r __ksymtab_list_del
c12dadfc r __ksymtab_klist_del
c12e1abd r __kstrtab_list_del
c12e9d03 r __kstrtab_klist_del
```

于是我们知道，发生 Oops 时的 EIP 值是：

c10e53cc + 0xa   == c10e53d6

然后用 gdb 查看：

```
# gdb /home/arc/build/linux-2.6/vmlinux
(gdb) b *0xc10e53d6
Breakpoint 1 at 0xc10e53d6: file
/usr/src/linux-2.6.24-rc3/lib/list_debug.c, line 64.
```

gdb 中还可以这样：

```
# gdb Sources/linux-2.6.24/vmlinux
(gdb) l *do_fork+0x1f
0xc102b7ac is in do_fork (kernel/fork.c:1385).
1380
1381        static int fork_traceflag(unsigned clone_flags)
1382        {
1383                if (clone_flags & CLONE_UNTRACED)
1384                        return 0;
1385                else if (clone_flags & CLONE_VFORK) {
```

```
1386                                if (current->ptrace & PT_TRACE_VFORK)
1387                                        return PTRACE_EVENT_VFORK;
1388                        } else if ((clone_flags & CSIGNAL) != SIGCHLD) {
1389                                if (current->ptrace & PT_TRACE_CLONE)
(gdb)
```

2.Kernel Debug 输出（CONFIG_DEBUG_LL）

这里是 arch/arm/boot/compressed/head.S 的解压过程，调用了 decompress_kernel()(同目录下的 misc.c)->include/asm-arm/arch-xxx/uncompress.h 的 putc()实现。这是在 Bootloader 中初始化的，用的是物理地址，因为此时内核还没有起来。而 printascii 则是调用了汇编。printascii()位于 arch/arm/kernel/debug.S，他需要调用虚拟地址，此虚拟地址通过 machine_start 提供，而相关的宏在 include/asm/arch-xxx/debug-macro.S 实现。
debug.s 里面需要判断一下当前是否打开了 mmu，然后指定 uart 的基址。在解压阶段的 head.s，mmu 是 1:1 映射，目的是加快速度。到了内核的 head.s，就是真正的 mmu 了，此时就是虚拟地址了。

```
.macro addruart,rx
mrc p15, 0, \rx, c1, c0
tst   \rx, #1          @ 判断 MMU 是否被使能
moveq \rx, #0x40000000    @ 使用 physical 地址
movne \rx, #io_p2v(0x40000000)  @ 使用 virtual 地址
orr   \rx, \rx, #0x00700000
.endm
```

3.修改 Linux 的启动地址

(1).Command line:

```
-CONFIG_CMDLINE="console=ttyS2,115200 mem=126M@0xa0000000 mem=128M@0xc0000000 comm_v75 uart_dma android lpj=3129344"
+CONFIG_CMDLINE="console=ttyS2,115200 mem=128M@0xc0000000 comm_v75 uart_dma android lpj=3129344"
```

(2). 移除 ARCH_DISCONTIGMEM_ENABLE 支持

(3).启动参数地址修改

```
.phys_io            = 0x40000000,
-          .boot_params       = 0xa0000100,
+          .boot_params       = 0xc0000100,
           .io_pg_offst       = (io_p2v(0x40000000) >> 18) & 0xfffc,
           .map_io            = pxa_map_io,
```

(4).arch/arm/mach-xxx/include/mach/memory.h 中物理地址偏移量的修改

```
-#define PHYS_OFFSET      UL(0xa0000000)
```

```
+#define PHYS_OFFSET        UL(0xc0000000)
```

(5) arch/arm/mach-xxx/Makefile.boot 中 zImage 解压地址配置

```
-    zreladdr-y      := 0xa0008000
+    zreladdr-y      := 0xc0008000
```

(6)<Android Dir>/system/core/mkbootimg.c 中地址修改

(7)Bootloader 中启动 Linux kernel 的配置

```
#define KERNEL_RAM_BASE      (0xc0800000)
static void (*ramKernel)(int zero, int arch, u32 params) =
    (void (*)(int, int, u32)) KERNEL_RAM_BASE;
ramKernel(0, ARCH_NUMBER,    0xc0000100);        #第二个参数为 Machine ID，第三个参数为启动参数地址
```