智能家居

1）知识的回顾 总结
2）嵌入式软件的开发流程

0、项目背景
BAT 360 联合传统的家电厂商 研发智能家居系统

无线通信过程使用的硬件：
zigbee
wifi
蓝牙
NB-IOT
LORA
智能家居的难点

CMMI: 软件开发成熟度 CMMI-3
CMMI-5
敏捷开发

1、需求分析
1）LED 灯的控制
2）视频监控功能
3）音频功能
mp3 音乐
4）监控是否有人闯入（红外）
拿按键模拟 实现报警
5）g-sensor 实现计步器
6）在线升级

产出物： 需求分析文档
2、概要设计
硬件选型：
CPU
无线通信
要不要有操作系统
用哪种操作系统

设计软件模块 ， 以及模块之间的通信方式
3、详细设计
每个模块中有哪些函数
函数的参数、返回值 、出错处理
函数的执行流程图 （viso）

4、编码

5、整合测试

6、项目的发布

资料：ehome.txt ,项目笔记
ehome_v1.tar.gz, 项目第一个版本
ehome_v2.tar.gz, 项目第二个版本
...
ehome_v7.tar.gz, 项目最终代码

1、开发环境的搭建

cd /home/tarena
mkdir ehome
cd ehome/
kernel: 编译通过的内核源码
rootfs：板子挂载的根文件系统

安装交叉编译工具

烧写到开发板 uImage 并运行

cd ehome
cp kernel.tar.bz2 ./
tar xf kernel.tar.bz2
rm kernel.tar.bz2
cd kernel
cp arch/arm/configs/x6818_defconfig .config
内核打补丁：（LCD 花屏）
cp /mnt/hgfs/esd2002/ehome/env/kernel.patch ./
patch -p1 <kernel.patch
make uImage
cp arch/arm/boot/uImage /tftpboot
//让开发板使用新内核
setenv bootcmd ping 192.168.1.8\;ping 192.168.1.8\;tftp 48000000 uImage \;bootm 48000000

nfs 方式挂载根文件系统
sudo vi /etc/exports
/home/tarena/ehome/rootfs *(rw,sync,no_root_squash)
sudo /etc/init.d/nfs-kernel-server restart

//设置开发板

setenv bootargs root=/dev/nfs nfsroot=192.168.1.8:/home/tarena/ehome/rootfs ip=192.168.1.6:192.168.1.8:192.168.1.1:255.255.255.0 console=ttySAC0,115200 maxcpus=1 lcd=wy070ml tp=gslx680-linux cam=OV5645

saveenv

2、QT 程序的移植

注意：尽量使用相同版本的 QT 开发环境

2.1PC 机上的 QT 开发环境

cd /home/tarena/workdir/qt/qt_dev

sudo ./qt-opensource-linux-x64-android-5.4.1.run

安装路径 /opt/Qt5.4.1 ： 注意将/opt 目录的访问权限改成 777

sudo chmod /opt -R 777

sudo vi /home/tarena/.bashrc

export PATH=$PATH:/opt/arm-cortex_a9-eabi-4.7-eglibc-2.18/bin/:/opt/Qt5.4.1/5.4/gcc_64/bin:/opt/Qt5.4.1/Tools/QtCreator/bin

source /home/tarena/.bashrc// 让.bashrc 重新启动

cd /home/tarena/ehome

使用 qtcreator 创建一个 test 工程

在其中增加了“确定”按钮, 退出集成开发环境

qmake

make

./test

readelf -d test//查看 test 程序依赖那些库文件

libQt5Widgets.so.5

libQt5Core.so.5

2.2 编译 ARM 版本的 QT 库

http://doc.qt.io/qt-t/embedded-linux.html

1）拿到 QT 源码 官方网站 //建议直接使用达内虚拟机提供的 QT 源码

2）通过交叉编译工具编译 ARM 版本的 QT 库

cd ehome/

cp /home/tarena/workdir/qt/qt_src/qtbase-opensource-src-5.4.1.tar.xz ./

tar xf qtbase-opensource-src-5.4.1.tar.xz

cd qtbase-opensource-src-5.4.1

vi mkspecs/linux-arm-gnueabi-g++/qmake.conf

```
14 QMAKE_CC                = arm-cortex_a9-linux-gnueabi-gcc
15 QMAKE_CXX               = arm-cortex_a9-linux-gnueabi-g++
16 QMAKE_LINK              = arm-cortex_a9-linux-gnueabi-g++
17 QMAKE_LINK_SHLIB        = arm-cortex_a9-linux-gnueabi-g++
18
19 # modifications to linux.conf
20 QMAKE_AR                = arm-cortex_a9-linux-gnueabi-ar cqs
21 QMAKE_OBJCOPY           = arm-cortex_a9-linux-gnueabi-objcopy
22 QMAKE_NM                = arm-cortex_a9-linux-gnueabi-nm -P
23 QMAKE_STRIP             = arm-cortex_a9-linux-gnueabi-strip
```

./configure --help 查看 QT 源码编译前的配置选项

./configure -prefix /home/tarena/ehome/qtlib -release -opensource -qt-libpng -qt-libjpeg -plugin-sql-sqlite -widgets -qt-sql-sqlite -make libs -no-cups -no-nis -no-iconv -no-dbus -no-openssl -no-iconv -no-accessibility -no-sse2 -silent -xplatform linux-arm-gnueabi-g++ -nomake tools -nomake examples -nomake tests -qt-freetype -no-glib -strip -linuxfb -plugindir /home/tarena/ehome/qtlib/plugins

执行./configure 会根据配置生成合适的 Makefile

-prefix: 编译后的安装路径

make install

-plugindir: 插件库的安装路径

-xplatform linux-arm-gnueabi-g++

//ls mkspecs/linux-arm-gnueabi-g++

make -j4 //开 4 个线程同时编译

make install

2.3 将 QT 程序移植到开发板运行

2.3.1 编译 ARM 版本的可执行程序

0) cd /home/tarena/ehome/

cd test //QT test 程序所在目录

1) /home/tarena/ehome/qtlib/bin/qmake //生成合适的 Makefile

2) make clean

3) make

2.3.2 将可执行程序及库文件部署到开发板

注意：建议接下来的所有内容都拷贝 rootfs/home

1)mkdir /home/tarena/ehome/rootfs/home/bin -p

2)cp test /home/tarena/ehome/rootfs/home/bin/

3)mkdir /home/tarena/ehome/rootfs/home/qt

4)cp /home/tarena/ehome/qtlib/lib/ /home/tarena/ehome/rootfs/home/qt/ -a

5)mkdir /home/tarena/ehome/rootfs/home/lib

6)cp /opt/arm-cortex_a9-eabi-4.7-eglibc-2.18/arm-cortex_a9-linux-gnueabi/sysroot/lib/*.so*

/home/tarena/ehome/rootfs/home/lib/ -a

7)cp /opt/arm-cortex_a9-eabi-4.7-eglibc-2.18/arm-cortex_a9-linux-gnueabi/sysroot/usr/lib/*.so* /home/tarena/ehome/rootfs/home/lib/ -a

2.3.3 部署 QT 程序运行时使用的插件

1）cp /home/tarena/ehome/qtlib/plugins/ rootfs/home/qt/ -a

2.3.4 关于 QT 的配置文件

1）mkdir rootfs/home/etc -p

2）vi rootfs/etc/profile

```
export PATH=$PATH:/home/bin
export LD_LIBRARY_PATH=/home/qt/lib/:/home/lib/:$LD_LIBRARY_PATH
#插件库路径
export QT_QPA_PLATFORM_PLUGIN_PATH=/home/qt/plugins/
#指定 LCD 设备  经常修改的就是分辨率
export QT_QPA_PLATFORM=linuxfb:fb=/dev/fb0:size=1024x600:tty=/dev/fb0
#指定字体库的存放位置
export QT_QPA_FONTDIR=/home/qt/lib/fonts
#指定的触摸屏对应的设备文件
#确定触摸屏设备文件的方式
#hexdump /dev/input/eventX
#用手敲击屏幕看是否有数据输出
export QT_QPA_GENERIC_PLUGINS=evdevkeyboard,evdevmouse,evdevtouch:/dev/input/event1
#如果接了 USB 键盘  使用判断触摸屏设备文件的方式修改 event7
#export QWS_USB_KEYBOARD=/dev/input/event7
```

3） 使配置生效

source /home/etc/profile

2.3.5 运行程序

/home/bin/test

关于开发板 GUI 界面程序中中文显示乱码的问题：

ehome_env.rar/font.zip

1）rm /home/tarena/ehome/rootfs/home/qt/lib/fonts/* -rf

2)cp /mnt/hgfs/ehome/env/font/DroidSansFallback.ttf/home/tarena/ehome/rootfs /home/qt/lib/fonts/ //部署字体库

3）重新在开发板上执行/home/bin/test

注意：前面移植 上给 uboot 打补丁

图片才能正常显示的开发板

需要给 linux 内核打补丁

否则 QT 界面程序板子触摸位置不准确

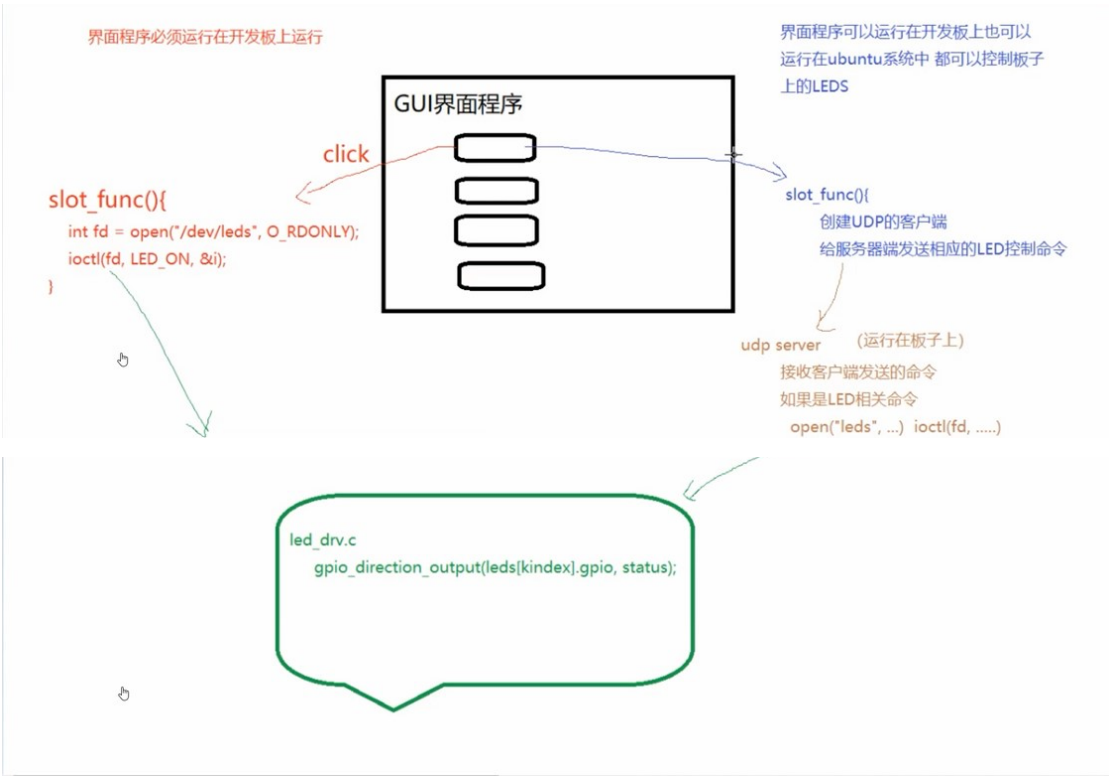
cd /home/tarena/ehome/kernel
cp .....kernel.patch ./
patch -p1 <kernel.patch

make uImage
//让板子使用新内核
cp arch/arm/boot/uImage /tftpboot/
uboot 使用 tftp 方式加载启动内核

练习： 将 QT 课程中的小游戏移植到开发板运行
问题：移植 qt 程序到开发板运行需要哪几步?

1）编译 ARM 版本的 QT 库
2）编写 QT 程序
3）使用 qtlib/bin/qmake 生成 ARM 版本的 Makfile
4) make 编译 QT 程序
5）将 test ARM 版本的 QT 库 插件库部署到开发板
6）配置 QT 所需要的环境变量
7）在开发板上执行 QT 程序

3、LED 的控制



3.1 LED 驱动程序
实质就是一个 linux 字符设备驱动

cd ehome

mkdir drivers
cd drivers
mkdir leds
cd leds
vi led_drv.c
vi Makefile
make
mkdir ../../rootfs/home/drivers
cp leds_drv.ko ../../rootfs/home/drivers/
vi test.c

编译 test.c，测试驱动程序是否好用

编写 LED 驱动 并测试是否可用： 参考 ehome_v1.tar.bz2

3.2 编写 LED 的控制应用程序

有两种方式

3.2.1 简单的方式

当点击亮灯按钮时，完成该信号的槽函数
在槽函数中
open("/dev/leds", ...)
ioctl(fd, CMD_LED_ON, &i);
修改图片

再次点击时灭灯
open("/dev/leds", ...)
ioctl(fd, CMD_LED_OFF, &i);
修改图片

GUI 界面程序一定部署到开发板才能有效果

3.2.2 复杂方式

希望 GUI 界面程序不管是运行在 PC
或者运行在开发板，都能控制开发板的 LED 状态

client (GUI 开发板/PC) Server (UDP 运行在开发板)
点击按钮发送命令 LED_ON 接收命令，根据命令 open 设备 ioctl 亮灯

1）界面客户端程序
cd /home/tarena/ehome
mkdir gui_client
cd gui_client

qtcreator

建立 client 工程 完成界面编程

先将 mainwindow 窗口的大小调整为 1024*600

添加按钮

槽函数中给 UDP 服务器发送命令

2）服务器程序

```
cd ehome
mkdir server
cd server
vi server.c    //接收命令 分发命令
    //udp server

    recvfrom(cmd)

    if(cmd == LED_ON)
        open("/dev/leds",...)
        ioctl(fd, LED_ON, 0)
vi leds.c:  根据命令不同调用操作灯的函数
vi leds_hw.c:
                open
                ioctl

arm-cor.....gcc leds.c leds_hw.c server.c -o server
cp server rootfs/home/bin/
启动 server (开发板)
启动 client    点击按钮实验是否亮灯
```

前三项在板子上运行

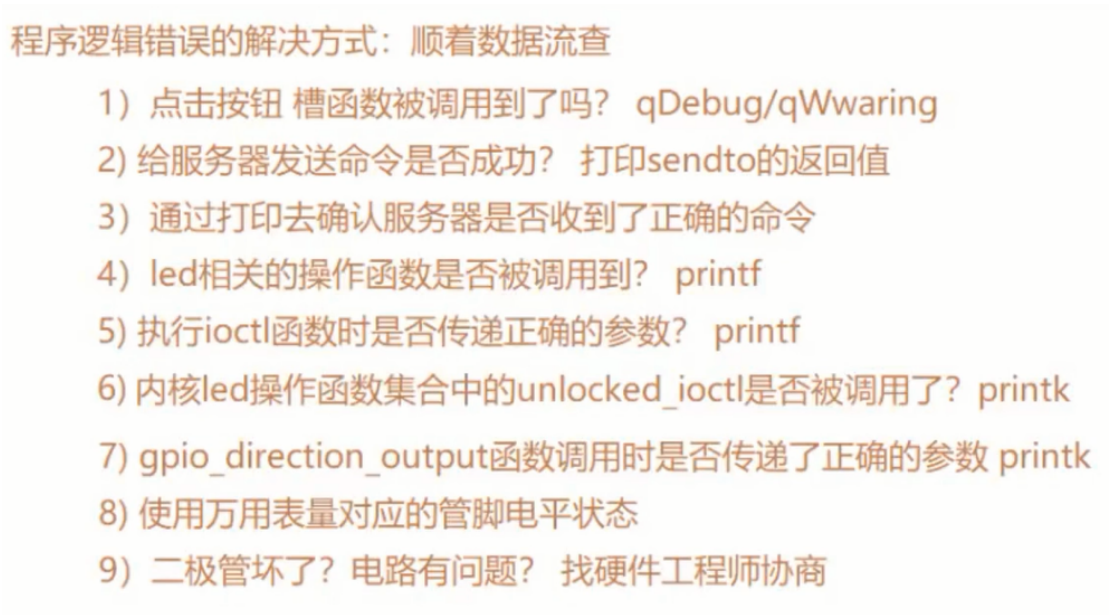
insmod /home/drivers/leds_drv.ko

/home/bin/server &

可以运行在开发板上页可以运行在 PC

/home/bin/client

注意编程过程中的调试技巧




```
#define DEBUG
#ifdef DEBUG
    /*##表示如果可变参数被忽略或为空，将使预处理器（ preprocessor ）去除掉它前面的那个逗号。*/
    #define pr_debug(fmt, ...) printf(fmt, ##__VA_ARGS__)
#else
    #define pr_debug(fmt, ...)
#endif
```

BUG 的调试，顺着数据流分析该调用的函数是否调用到

gui 界面程序点击按钮 发送命令函数是否被调用
    qWarning
命令被正常接收?
    在 server 中打印收到的命令
该命令被正常处理了吗?
    leds_operations
        leds
            ioctl
                ----
                led_ioctl

ehome_v1.tar.gz

面试过程中的重点：
1）笔试题
    考察 C 编程能力
    链表操作，其中最基本的就是链表的增删改查

2）项目

项目中实现了什么功能

问题：该功能是如何实现的?

按照混杂设备的编程框架实现了 LED 驱动程序

使用 QT 编程实现控制界面

该控制界面可以在 PC 机也可以移植到开发板运行

使用 socket 编程实现界面客户端和板子上运行的服务器端通信

界面端给服务器发命令

服务器端根据收到的命令控制 LED

3）经常被问到的问题

进程和线程的区别

进程间通信的方式，各有什么特点

网络七层模型，各层中的协议举例

TCP/UDP 的区别

UART 通信协议

I2C 通信协议

对中断的理解

系统调用的理解

对 input 子系统的理解

对 platform 的理解

给你一个芯片怎么把驱动搞定

4、视频服务器

4.1 摄像头的驱动

uvc 子系统： usb video class

内核中自带了满足 uvc 格式的摄像头驱动

如果你手中的摄像头满足 uvc 规范，该摄像头就是免驱

只需要对内核进行配置，将 uvc 模块对应的代码

编译到 uImage

如何判断摄像头满足 uvc 格式规范?

lsusb

再将摄像头插入开发板

lsusb

Bus 001 Device 003: ID 046d:0825

网络搜索 uvc 官网 有一个的页面：

列出了 uvc 框架支持的 usb 摄像头 ID

配置内核，将 uvc 子系统编译进内核

```
Device Drivers    --->
    <*> Multimedia support    --->
        [*]     Video capture adapters    --->
            [*]     V4L USB devices    --->
```

<*> USB Video Class (UVC)

make uImage

让开发板加载包含 uvc 模块的新的内核

ls /dev/video*

再次插入摄像头

ls /dev/video*

发现多了一个 video9 ，就是插入摄像头的设备文件

4.2 应用程序

4.2.1 操作摄像头，抓取图像数据

v4l2: video for linux ver2

它属于摄像头软件的中间层

向下统一摄像头驱动的格式

向上为应用软件访问控制摄像头提供统一的接口

简化应用层软件控制摄像头的编程工作

v4l2 用户态编程怎么玩?

v4l2 提供的有小程序：v4l_demo.zip/capture.c

open 设备

ioctl 设置工作参数

ioctl(fd, VIDIOC_STREAMON, &type);//开始摄像头开始工作

// 获取图像数据

ioctl(fd, VIDIOC_DQBUF, &buf);

ioctl(fd, VIDIOC_QBUF, &buf);

mjpeg-streamer 包含了按照 v4l2 框架去操作摄像头的代码

而且其中也包含了按照 http 协议向客户端发送图像数据的代码

重点：移植部署运行 mjpeg-streamer

4.2.2mjpeg-streamer 的移植:mjpg-streamer.tar.bz2

cd ehome

mkdir video

cd video

cp /mnt/hgfs/ehome/env/mjpg-streamer.tar.bz2 ./

tar xf mjpg-streamer.tar.bz2

cd mjpg-streamer/

vi README

重要的信息包括以下内容

make clean all

./mjpg-streamer ....

start.sh

vi Makefile

CC = gcc

find ./ -name "Makefile" -exec sed -i "s/CC = gcc/CC = arm-cortex_a9-linux-gnueabi-gcc/g" {} \;

sed: 文件文件

awk: 行处理

结合正则表达式 功能非常强大

make

4.2.3 部署到开发板

cp mjpg_streamer ../../rootfs/home/bin/

cp *.so ../../rootfs/home/lib/ -a

cp www/ ../../rootfs/home/ -r

4.2.4 运行

/home/bin/mjpg_streamer --help

/home/bin/mjpg_streamer -i "input_uvc.so --help"

/home/bin/mjpg_streamer -i "/home/lib/input_uvc.so -d /dev/video9 -y -r 320x240 -f 30" -o "/home/lib/output_http.so -w /home/www"

-i: 指定输入插件

-d: 指定访问的摄像头设备文件

-y: 采集图像的格式为 YUYV

-r: 采集图像的大小

-f: 帧频率

-o: 指定输出插件

-w: 网页资源文件所在目录

打开浏览器，输入“http://192.168.1.6:8080/”

如果手里没有摄像头，可以使用如下方案

/home/bin/mjpg_streamer -i "/home/lib/input_testpicture.so -r 320x240 -d 500" -o "/home/lib/output_http.so -w /home/www"

input_testpicture.so 不是采集摄像头数据

自身有两张图片

将这两张图片交替的发送给客户端

打开浏览器，输入“http://192.168.1.6:8080/”

B/S 架构：browser / server

C/S 架构：client / server

4.3 mjpg-streamer 源码分析

4.3.1 mjpg_streamer.c 高内聚 低耦合

ctags -R *

```
int main(int argc, char *argv[])
{
    /*共享库的运行阶段加载有两种方式
```

```
        gcc xxx -o a.out -lpthread
        ./a.out  操作系统加载共享库

        程序中自主主动加载共享库（插件库）
    */
                                        "input_uvc.so"
    global.in[i].handle = dlopen(global.in[i].plugin, RTLD_LAZY)
                /*找到 input_uvc.so 中的 input_init 函数对应代码在内存中的位
置*/
    global.in[i].init = dlsym(global.in[i].handle, "input_init");
    global.in[i].run = dlsym(global.in[i].handle, "input_run");
    /*执行 input_uvc.so 中的 input_init 函数*/
    global.in[i].init(&global.in[i].param, i)

                                        /*"output_http.so"*/
    global.out[i].handle = dlopen(global.out[i].plugin, RTLD_LAZY);
    global.out[i].init = dlsym(global.out[i].handle, "output_init");
    global.out[i].run = dlsym(global.out[i].handle, "output_run");
    global.out[i].init(&global.out[i].param, i)

    global.in[i].run(i)
    global.out[i].run(global.out[i].param.id);

    pause();
    return 0;

}
```

4.3.2 输入插件 plugins/input_uvc/

按照 v4l2 编程步骤去操作 uvc 格式的摄像头

官方例程：capture.c

Video for linux 2 example (v4l2 demo) - MetalSeed - 博客频道 - CSDN.NET.png

vi plugins/input_uvc/input_uvc.c

```
    /*打开摄像头 设置工作参数*/
    int input_init(input_parameter *param, int id)
    {
        init_videoIn(cams[id].videoIn, dev, width, he        ight, fps, format, 1,
cams[id].pglobal, id)
        {
            init_v4l2(vd)
            {
                /*打开"/dev/video9"设备文件*/
                vd->fd = OPEN_VIDEO(vd->videodevice, O_RDWR)
```

```
                    /*查询当前硬件的工作能力*/
                    xioctl(vd->fd, VIDIOC_QUERYCAP, &vd->cap)
                    /*图像格式设置*/
                    ret = xioctl(vd->fd, VIDIOC_S_FMT, &vd->fmt);
                    ...
                }
            }
        }
        int input_run(int id)
        {
                pthread_create(&(cams[id].threadID), NULL, cam_thread,
&(cams[id]));
        }
        void *cam_thread(void *arg)
        {
            while(!pglobal->stop)
            {
                uvcGrab(pcontext->videoIn)
                {
                    video_enable(vd)
                    {
                        /*VIDIOC_STREAMON：让摄像头开始工作*/
                        ret = xioctl(vd->fd, VIDIOC_STREAMON, &type);
                    }
                    /*获取一帧图像*/
                    xioctl(vd->fd, VIDIOC_DQBUF, &vd->buf)

                }
                /* copy JPG picture to global buffer */
                  memcpy_picture(pglobal->in[pcontext->id].buf,
pcontext->videoIn->tmpbuffer, pcontext->videoIn->buf.bytesused);
            }
        }
```

BUG 修改：input_uvc/v4l2uvc.c

```
428     do{
429     memset(&vd->buf, 0, sizeof(struct v4l2_buffer));
430     vd->buf.type = V4L2_BUF_TYPE_VIDEO_CAPTURE;
431     vd->buf.memory = V4L2_MEMORY_MMAP;
432
433     ret = xioctl(vd->fd, VIDIOC_DQBUF, &vd->buf);
434     if(ret < 0) {
435           perror("Unable to dequeue buffer");
436         // goto err;
```

```
437         }
438         }while(ret < 0);
```

4.3.3 输出插件 plugins/output_http/

将图像数据封装成 http 数据包

通过 TCP 方式下客户端发送

vi plugins/output_http/output_http.c

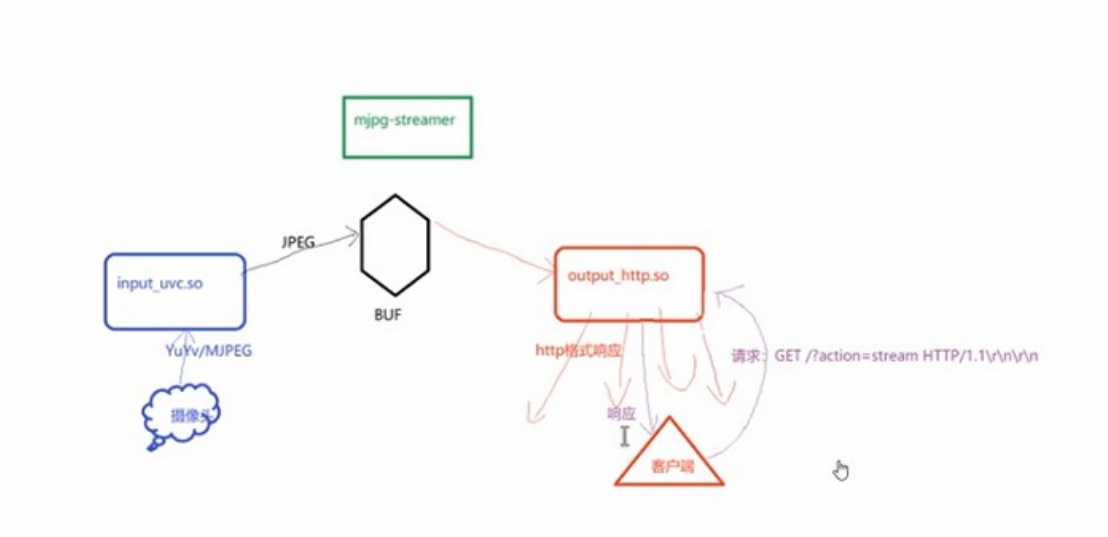
```
int output_init(output_parameter *param, int id)
{
    port = htons(8080);
    ...
}
int output_run(int id)
{
    pthread_create(&(servers[id].threadID), NULL, server_thread,
&(servers[id]));
}
  void *server_thread(void *arg)
  {
      socket(aip2->ai_family, aip2->ai_socktype, 0)
      bind(pcontext->sd[i], aip2->ai_addr, aip2->ai_addrlen)
      listen(pcontext->sd[i], 10)
      while(!pglobal->stop)
      {
          accept(pcontext->sd[i], (struct sockaddr *)&client_addr, &addr_len)
          pthread_create(&client, NULL, &client_thread, pcfd)
      }

  }
  void *client_thread(void *arg)
  {
    _readline(lcfd.fd, &iobuf, buffer, sizeof(buffer) - 1, 5)
    {
        _read(fd, iobuf, &c, 1, timeout)
        {
            read(fd, &iobuf->buffer, IO_BUFFER)
        }
    }

    else if(strstr(buffer, "GET /?action=stream") != NULL)
    {
        req.type = A_STREAM;//确定客户端请求类型
    }
```

```
                switch(req.type) {
                        case A_STREAM:
                                send_stream(lcfd.fd, input_number)
                                {
                                        while(!pglobal->stop)
                                        {
                                            /**/
                                            write(http 头信息)
                                            /*将 pglobal->buf 数据拷贝到  frame 缓冲区*/
                                              memcpy(frame,  pglobal->in[input_number].buf,
frame_size);
                                              write(fd, frame, frame_size)
                                              write(http 尾信息)
                                        }
                                }
                                break;
                }
```

5、视频客户端



5.1 HTTP 协议:http.zip

超文本传输协议

建立 TCP 传输的基础上

客户端要给服务器端发送请求 request

服务器端根据客户端的请求回送响应 response

5.2 对 mjpg-streamer 的分析得到以下内容

如果客户端发送的请求中包含"GET /?action=stream"

服务器就会将视频数据封装 HTTP 格式数据帧

不断发送给客户端

按照 http 协议 request 的数据格式

就是给服务器发送"GET /?action=stream HTTP/1.1\r\n\r\n"

5.3 编写一个 tcp 客户端程序

code_for_mjpgstreamer.rar

1）保证客户端和服务器能够联通

serverip : 开发板 IP

端口号： 8080

给服务器发送请求 "GET /?action=stream HTTP/1.1\r\n\r\n"

读取一次服务器返回的数据，并将数据打印

2）在 1）的基础上，不断地读取数据

把读到的数据保存/tmp/test.jpg 文件

保存 1M 数据后程序退出

3）能不能把 http 头信息过滤掉 把 http 尾信息过滤掉

只将图像信息内容保存到/tmp/test.jpg 文件去

hexdump -C /tmp/test.jpeg |less

服务器返回的图像是 jpeg 格式的

jpeg 图像帧在存储时是有固定的格式

JPEG 是一种有损的图像压缩算法

JPEG 文件由两部分组成： 标记码 压缩数据

FF D8 .... FF D9

参考 code_for_mjpgstreamer.zip/03 代码

4) 编写 GUI 客户端显示图像数据

定义一个新的类 camer,父类为 Qobject

camer：该类负责和 HTTP server 进行通信

a)连接服务器

connectToHost

b)给服务器发送请求"GET /?action=stream HTTP/1.1\r\n\r\n"

requestImage

c)接收 server 返回的数据

并从中过滤出"ff d8 .... ff d9"

```
void CamClient::readImage()
{
    接收数据
```

过滤出图像
发送信号 newImageReady(image)给 mainwindow
}
void MainWindow::showNewImage(QImage img)
{
/*显示图片*/
ui->imageLabel->setPixmap(QPixmap::fromImage(img));
}
通信过程中使用了 QTcpSocket
使用 QBtyeArray 缓存图像数据
显示图像使用了 QImage

注意 1：SIGNAL(readyRead())

当收到 readyRead()信号时，去调用对应的槽函数接收数据，过滤数据

camer::camer(QObject *parent) :

QObject(parent)
{
connect(&tcpSocket,SIGNAL(readyRead()),this,SLOT(readImage()));
}

注意 2： 自定义的信号
CamClient 当过滤到一个完整的图像帧发出的 newImageready
MainWindow 负责处理该信号

视频数据的压缩 H264
参考代码： 04 视频客户端
ehome_v2.tar.gz : LED + 视频客户端

问题：视频监控功能是如何实现的?
采用了免驱摄像头，只有配置内核将 UVC 子系统编译到内核中

应用程序：
视频服务器，移植的 mjpg-streamer
实现了采集图像数据 (v4l2)
按照 http 协议传输图像数据

视频客户端, QT
QTcpSocket
QImage

QByteArray
信号和槽机制
自行过滤图像数据(ffd8 ... ffd9)
将图片贴到 qLabel

6、红外报警功能

常见的人体红外感应模块 HC-SR501 ： HC-SR501.zip
用按键来模拟红外
当按键按下，有人闯入，需要报警
报警：
beep 响 (选择)
发一个短信 (gsm)

6.1 驱动程序

按键驱动

cd /home/tarena/ehome/drivers
mkdir buttons
cd buttons
vi btn_dev.c
vi btn_drv.c
vi Makefile
make
vi test.c
arm-cor...-gcc test.c -o test

cp *.ko ../../rootfs/home/drivers
cp test ../../rootfs

在开发板上安装驱动模块
insmod /home/drivers/btn_dev.ko
insmod /home/drivers/btn_drv.ko
./test 或者通过 hexdump /dev/input/eventX
按下按键观察是否正常

或者实现红外驱动

触发方式：

1）不可重复触发： 人进入感应范围 输出高电平， 延时(8~200s)之后输出低电平

2）可重复触发：人进入感应范围内 输出高电平，人如果一直在其感应范围内活动

一直输出的是高电平
直到人离开 延时(8~200s)之后输出低电平

cd /home/tarena/ehome/drivers

mkdir hc-sr501
cd hc-sr501

蜂鸣器驱动

建议按照混杂设备的架构完成

cd /home/tarena/ehome/drivers

mkdir beep
cd beep
vi beep_drv.c
vi Makefile
make
vi test.c
cp *.ko ../../rootfs/home/drivers
cp test ../../rootfs

在开发板上安装驱动模块

insmod /home/drivers/beep_drv.ko
./test

6.2 应用程序

1)vi server.c

开启新线程　读取是否有人闯入（按键按下）

如果有人闯入

开启报警功能

pthread_create(&threadID, NULL, hcsr_thread, NULL);

2) vi hcsr.c

```
void *hcsr_thread(void *arg)
{
    int btn_fd = open("/dev/input/event5",O_RDONLY);

    beep_operations(BEEP_ON);
    ...
}
```

3) vi beep.c

```
void beep_operations(int cmd){
    beep(1);
}
```

4) vi beep_hw.c

```
void beep(int on_off)
{
```

```
        int fd = open("/dev/mybeep", O_RDONLY);
        ioctl(fd, DRV_BEEP_ON, NULL);
    }
```

5) make

6) cp server ../rootfs/home/bin

7) 在开发板上执行

insmod home/drivers/beep_drv.ko
insmod home/drivers/leds_drv.ko
insmod home/drivers/btn_dev.ko
insmod home/drivers/btn_drv.ko
启动 udp server 接收客户端发送的命令
/home/bin/server &
实验时要注意的问题：

1）内核中原有的按键驱动裁剪

2）咱们安装的按键设备驱动对应的设备文件 要确定一下

```
cat /proc/bus/input/devices
vi hcsr.c
    open("/dev/input/eventX",O_RDONLY|O_NONBLOCK);
```

ehome_v3.tar.bz2

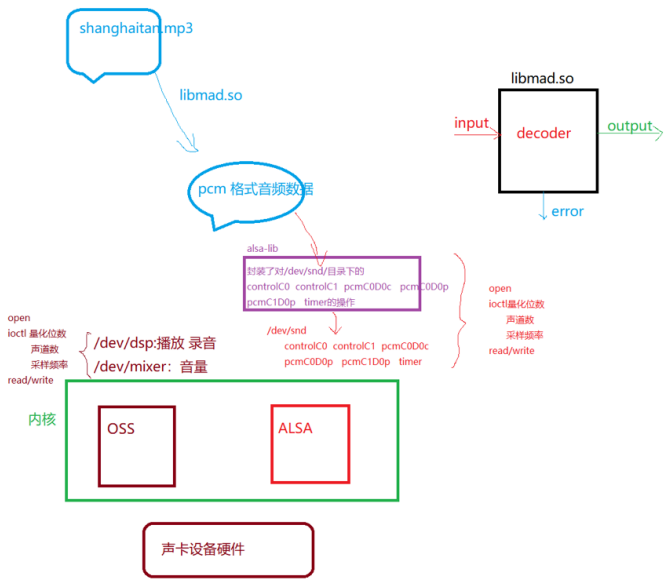
7、MP3 功能的实现 ： mp3.rar

完成 mp3 文件的解码工作，

10 分钟：MP3_format_parse.doc

将解码之后的数据写入声卡设备

20 分钟：Linux_sound_guide.doc



mp3/ALSA Audio API 使用指南(中英版) - 幻雪神界 - 博客频道 - CSDN.NET.html

7.1 基本概念

PCM: 脉冲编码调制

声音是模拟量

计算机能处理的是数字量，涉及模拟量和数字量的相互转换

录音时是模拟量转数字量

播放时是数字量转模拟量

采样频率：

每秒钟抽取声波幅度样本的次数

当采样频率应该在 40kHz 左右，保证声音不失真

量化位数：

每次采样，模拟音频信号的振幅用多少个 bit 来表示

一般使用 16bit

声道数：

有单声道和双声道之分，双声道又称为立体声

MP3：

音频数据的压缩算法

有损压缩（还原后是有失真的）

压缩比可以接近 10:1-12:1

使压

缩后的文件在回放时能够达到比较接近原音源的声音效果

7.2 MP3 文件的解码工作

MP3_format_parse.doc

其存储格式：

TAGV2

{

。。。

}

FRAME

{

帧头

{

。。。。

}

数据实体

}

FRAME

...

TAGV1

{

。。。

}

mp3 文件的解码工作，可以使用开源的库完成

mp3.rar/libmad-0.15.1b.tar.gz

libmad 库的使用步骤：

1）cd ehome/

2）mkdir mp3

cd mp3

3)cp /mnt/hgfs/ehome/env/mp3/libmad-0.15.1b.tar.gz ./

4)tar xf libmad-0.15.1b.tar.gz

5) ./configure --help

./configure CC=arm-cortex_a9-linux-gnueabi-gcc --host=arm-linux --prefix=/home/tarena/ehome/mp3/install

--host: 运行目标平台

--prefix: 安装路径 make install

用来生成合适的 Makefile

6) make && make install

7) minimad.c

libmad API 使用的演示 demo

从标准输入获取要解码的数据

进行解码

解码后的数据打印到标准输出设备

arm-cortex_a9-linux-gnueabi-gcc minimad.c -o player -L ../install/lib/ -lmad

如果出现“../install/lib//libmad.so：对‘III_imdct_l’未定义的引用”

解决方式：

make clean

make

make install

arm-cortex_a9-linux-gnueabi-gcc minimad.c -o player -L ../install/lib/ -lmad

计划将解码之后的数据输出到声卡设备

7.3 声卡的驱动程序

linux 下的声卡驱动分两种：

OSS: Open Sound System

最早出现的

非开源程序

/dev/dsp：播放录音

/dev/mixer：音量

open("/dev/dsp",...)

ioctl()设置采样频率 量化位数 声道数

write(解码后的数据)

ALSA: 开源

make menuconfig

Device Drivers --->

<*> Sound card support --->

<*> Advanced Linux Sound Architecture --->

ls /dev/snd/

controlC0

controlC1

pcmC0D0c

pcmC0D0p

pcmC1D0p

timer

7.4 alsa 库的使用

如果是 alsa 架构，提供了一套用户空间访问以上设备文件的库函数 alsa-lib-1.1.1.tar.bz2

alsa 库封装了对/dev/snd 下设备文件的操作

编译步骤：

1） cd mp3/

2） cp /mnt/hgfs/ehome/env/mp3/alsa-lib-1.1.1.tar.bz2 ./

tar xf alsa-lib-1.1.1.tar.bz2

cd alsa-lib-1.1.1/

3) ./configure CC=arm-cortex_a9-linux-gnueabi-gcc --host=arm-linux --prefix=/home/tarena/ehome/mp3/install --with-configdir=/opt/alsa --with-plugindir=/opt/alsa_lib

4) make && make install

以上库函数的使用可以参考

alsa 使用官网：http://www.alsa-project.org/alsa-doc/alsa-lib/index.html

http://www.alsa-project.org/main/index.php/Download

或者参考：

ALSA Audio API 使用指南(中英版) - 幻雪神界 - 博客频道 - CSDN.NET.html

以上两个库 libmad alsa 库的使用实例 mp3_player.c

mp3_player.c 和 minimad.c 相比修改以下内容

./player xxx.mp3

1)原来从标准输入读取解码数据

现在从 xxx.mp3 文件中读取解码数据

2）设置声卡的工作属性 44.1KHz 16bit 双声道

```
set_pcm() //参考文档中“一个最简单的回放程序”
{
/*打开声卡设备*/
rc=snd_pcm_open(&handle, "default", SND_PCM_STREAM_PLAYBACK, 0);
//硬件设置为交错模式
snd_pcm_hw_params_set_access(handle, params, SND_PCM_ACCESS_RW_INTERLEAVED);
//设置 16 位采样精度
rc=snd_pcm_hw_params_set_format(handle, params, SND_PCM _FORMAT_S16_LE);
//设置声道,1 表示单声>道，2 表示立体声
rc=snd_pcm_hw_params_set_channels(handle, params, channels);
//设置频率
snd_pcm_hw_params_set_rate_near(handle, params, &rate, &dir);
}
```

3）修改了 output

将解码的音频数据输出到声卡设备

注意：2） 3） 参考 ALSA Audio API 使用指南(中英版) - 幻雪神界 - 博客频道 - CSDN.NET.html

```
/*libmad 解码的音频帧数据写入声卡设备*/
snd_pcm_writei(handle, OutputPtr, n);
```

结合 mp3_player.c ALSA Audio API 使用指南(中英版) 分析 mp3_player.c 的修改

编译 部署 运行 player 程序

1）arm-cortex_a9-linux-gnueabi-gcc mp3_player.c -lmad -lasound -L install/lib/ -I install/include/ -o player

出现了以下错误：

libmad.so : III_....

解决方法：

```
cd libmad-0.15.1b
make clean
make
make install
cd ..
arm-cortex_a9-linux-gnueabi-gcc mp3_player.c -lmad -lasound -L install/lib/ -I install/include/ -o player
```

2）cp player ../rootfs/home/bin/

3）cp install/lib/* ../rootfs/home/lib/ -a

4）配置文件和插件库文件

注意，配置时指定了路径

--with-configdir=/opt/alsa --with-plugindir=/opt/alsa_lib

部署到开发板上时要保持一致

cp /opt/alsa/ ../rootfs/opt/ -a

cp /opt/alsa_lib/ ../rootfs/opt/ -a

5) mkdir ../rootfs/home/songs

cp /mnt/hgfs/ehome/env/mp3/*.mp3 ../rootfs/home/songs/

6) 运行

. /home/etc/profile

/home/bin/player /home/songs/I\ Will\ Always\ Love\ You Always\ Love\ You.mp3 1>/dev/null 2>&1

练习：

1）看懂 mp3_player.c

2）编译 部署到开发板运行

3）/home/bin/player /home/songs/xxx.mp3

练习：//将/home/songs 目录下的所有.mp3 文件播放一遍

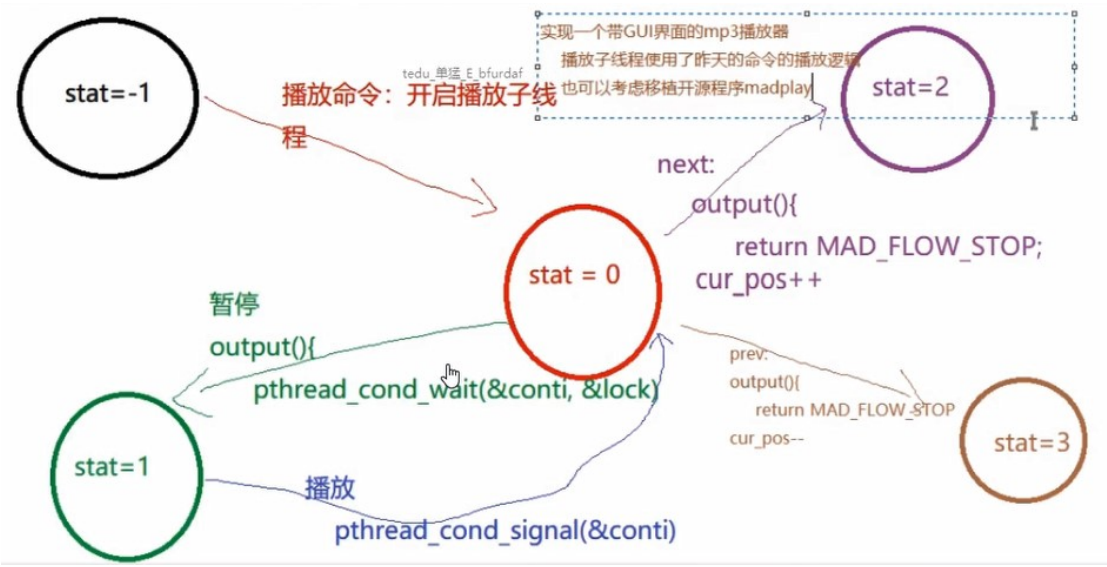
/home/bin/player /home/songs

参考 ehome_v4.tar.bz2/server/ myplayer.c

问题：mp3 播放器是如何实现的?

参考 ehome.txt 和 mp3.png 图

7.5 加一个简单的 GUI 界面



涉及到的知识点：线程的管理

播放：

1）开启新线程开始播放

或者

2) 唤醒被暂停的播放线程

pthread_cond_signal(&conti);

暂停：stat=1

修改了播放线程中的 output

//alsa 声卡的暂停,将缓冲区中提提交声卡播放数据全部提交给声卡

snd_pcm_drop(handle);

//让播放线程进入睡眠状态， 暂停了播放

pthread_cond_wait(&conti, &lock);

当再次收到播放命令 会执行到“ pthread_cond_signal(&conti); ”

//设置声卡处于就绪状态

snd_pcm_prepare(handle);

上一首

MUSIC_NEXT music_stat =2

播放线程

music_play()

{

while(遍历歌曲)

{

decode()

{

out_put

{

return MAD_FLOW_STOP//导致 decode()执行完毕

}

}

munmap

改变 cur_pos //标识要播放的目录中的第几首歌

}

}

下一首

MUSIC_PREV music_stat =3

播放线程

music_play()

{

while(遍历歌曲)

{

decode()

```
{
out_put
{
return MAD_FLOW_STOP//导致 decode()执行完毕
}
}
munmap
cur_pos = cur_pos?(cur_pos-1):0; //标识要播放的目录中的第几首歌

}
}
```

步骤：

```
. /home/etc/profile
/home/bin/start.sh    &
insmod /home/drivers/beep_drv.ko
insmod /home/drivers/btn_drv.ko
insmod /home/drivers/btn_dev.ko
insmod /home/drivers/leds_drv.ko
/home/bin/server &
```

参考代码：ehome_v4.tar.bz2

8、计步器的功能实现 （重点是 g-sensor 驱动程序的实现，重中之重 I2C 通信协议）

如何算一步：

只要 xyz 任意一轴和上个周期比有了一个比较大的变化

8.1 g-sensor 的驱动程序

1)i2c_client

配置内核将内核中原有的 g-sensor 驱动程序裁剪掉

```
cd kernel/
make menuconfig
    Device Drivers    --->
      <*> Hardware Monitoring support    --->
        < >     Freescale MMA865X 3-Axis Accelerometer
make uImage
```

在内核源码中添加自己的 i2c_client

```
vi arch/arm/plat-s5p6818/x6818/device.c
        566 static struct i2c_board_info __initdata mma865x_i2c_bdi = {
        567             .type     = "mma8653",
        568             .addr     = 0x1D//(0x3a),
        569 };
```

1480 i2c_register_board_info(2, &mma865x_i2c_bdi, 1);

make uImage

cp arch/arm/boot/uImage /tftpboot

让开发板使用新内核

2)i2c_driver

cd /home/tarena/ehome

mkdir drivers/mma8653

cd drivers/mma8653

vi mma8653_drv.c

vi Makefile

make

cp mma8653_drv.ko ../../rootfs/home/drivers

vi test.c

arm-cortex....-gcc test.c -o test

cp test ../../rootfs/

在板子上安装驱动模块

insmod /home/drivers/mma8653_drv.ko

检查对应的设备文件是否出现

ls /dev/mma865x

./test

改变板子的存放姿态，观察加速度值

面试时 被问题加速度传感器驱动程序是如何实现的？ 参考 driver/day11/bj.txt

8.2 应用程序 C/S 架构

c/s: client/server : qq

b/s browser/server : 网页版游戏

client: qt gui

周期性（博尔特）的给服务器发送命令

让服务器不断的读取 g-sensor 的数据，和上次读取到的数据作比较

客户端如果收到 1，步数+1 ，并显示

可以使用 Qtimer

server:

收到相应的命令，读取 g-sensor 的数据

上次读到的结果进行比较

如果变化比较大（人为定义），

给客户端返回 1，否则返回 0

数据的比较可以放在服务器上完成
其实也可以放在客户端完成

步骤：

```
source /home/etc/profile
insmod   /home/drivers/beep_drv.ko
insmod   /home/drivers/btn_dev.ko
insmod   /home/drivers/btn_drv.ko
insmod   /home/drivers/leds_drv.ko
insmod   /home/drivers/mma8653_drv.ko
/home/bin/start.sh &
/home/bin/server &
```

参考代码：ehome_v5.tar.bz2

9、项目的发布

9.1 配置脚本，实现自启动

/etc/init.d/rcS: 开机自启动的程序可以放入该脚本
/etc/profile: 全局对所有用户有效的环境变量

开机自启动：

vi rootfs/etc/init.d/rcS

最后加入

```
exec /home/etc/rcS
```

vi rootfs/home/etc/rcS

```
#配置开发板 IP
ifconfig eth0 192.168.1.6
ifconfig lo up
#加载内核模块
find /home/drivers/ -name *ko -exec insmod {} \;


source /home/etc/profile

#启动视频服务器
/home/bin/start.sh &

#启动 UDP 服务器
/home/bin/server &
#启动 GUI 界面程序
/home/bin/client &
```

chmod +x rootfs/home/etc/rcS

环境变量的设置

vi rootfs/etc/profile

. /home/etc/profile

重新启动开发板 看是可以自启动

自启动后功能是否有效，如果有效开始后续步骤

如果无效 调试 BUG 调试通过后开始后续步骤

9.2 分区的规划 烧写镜像到 EMMC

前提条件，该分区能够存储下要存储文件

0--------1M----------17M----------273M----------->剩余

扇区号 0x800 0x8800 0x88800

uboot kernel rootfs userdata

进入 uboot 界面 对开发板 emmc 按照上述规划重新分区

fdisk 2 3 0x100000:0x1000000 0x1100000:0x10000000 0x11100000:0

9.2.1 制作 uImage 并完成烧写

tftp 48000000 uImage

mmc write 48000000 0x800 0x3000

setenv bootcmd mmc read 48000000 800 3000 \;bootm 48000000

saveenv

9.2.2 制作 ext4 类型个文件系统 并烧写

#创建 256M 大小的 rootfs.ext4 文件

dd if=/dev/zero of=rootfs.ext4 bs=1k count=262144

du -h rootfs.ext4

sudo mkfs.ext4 rootfs.ext4

sudo mkdir /mnt/ext4

sudo mount -t ext4 -o loop rootfs.ext4 /mnt/ext4

sudo cp rootfs/* /mnt/ext4/ -a

sudo umount /mnt/ext4

cp rootfs.ext4 /tftpboot/

tftp 0x48000000 rootfs.ext4

mmc write 48000000 0x8800 0x80000

9.2.3 启动参数的修改

setenv bootargs root=/dev/mmcblk0p2 rootfstype=ext4 init=/linuxrc console=ttySAC0,115200 lcd=wy070ml tp=gslx680-linux maxcpus=1

saveenv

系统启动后：

ls /dev/mmcblk0*

cat /proc/partitions

mkfs.vfat /dev/mmcblk0p3

后续可以将以下命令放到 rcS 文件中，实现开机自动挂载

mount /dev/mmcblk0p3 /mnt/

建议：在实现系统升级功能前 还是改成通过 nfs 方式挂载根文件系统

10、系统功能升级

通过网络进行升级

通过 U 盘来实现升级功能

10.1 U 盘的手工挂载

ls /dev/sd*

插入 U 盘

ls /dev/sd*

mount /dev/sda1 /mnt

umount /mnt

10.2 u 盘自动挂载

usb.rar

/dev/sda1 设备文件自动创建

是由于热插拔事件产生，导致 mdev 程序被执行

由 mdev 来去创建的设备文件

其实可以通过设置让 mdev 在去创建/dev/sda1 设备文件

的同时，可以完成 u 盘的自动挂载

问题 1：如何配置让 mdev 既可以自动创建设备文件/dev/sda1

又可以创建后自动挂载 u 盘?

mdev.conf

问题 2： 如何修改 mdev.conf ,语法格式?

vi busybox-1.23.2/docs/mdev.txt

sd[a-z][0-9] 0:0 666 @/home/usb/usb_insert.sh /dev/$MDEV

sd[a-z][0-9], 设备文件的规则，满足该规则的

sda1 sdb2

sda 不满足

0:0 , uid:gid

666, 权限

@, 创建设备文件

/home/usb/usb_insert.sh /dev/$MDEV,

创建 sda1/sdb2...设备文件时执行/home/usb/usb_insert.sh 脚

本,

并且传递参数/dev/sda1(sdb2)

sd[a-z] 0:0 666 $/home/usb/usb_remove.sh

$, 销毁设备文件之前

实验步骤：usb.rar

1）在 rootfs/etc/mdev.conf

cp ... mdev.conf rootfs/etc/

2) mkdir rootfs/home/usb -p

cp 。。。usb_insert.sh rootfs/home/usb/

cp 。。。usb_remove.sh rootfs/home/usb/

3）创建挂载点

mkdir rootfs/mnt/usb

10.3 手工操作将 u 盘中 uImage 更新到

dd if=/mnt/usb/uImage of=/dev/mmcblk0p1

输入文件 if 指定

输出到哪去 of

10.4 当点击按钮时自动更新

system("dd if=/mnt/usb/uImage of=/dev/mmcblk0p1");

GUI: 给服务器发送命令 m

server: 收到命令 m

执行 system("dd if=/mnt/usb/uImage of=/dev/mmcblk0p1");

下午完成系统升级功能 参考：ehome_v7.tar.gz

11、ds18b20 驱动

温度传感器 ds18b20.zip

DS18B20.pdf

DS18B20 中文资料.pdf

练习：阅读 ds18b20 的芯片数据手册

思考问题，如何从该传感器中读出温度值?

1）功能

用来测量温度， 输出的是数字量的温度值

(该芯片能够完成模拟量的温度值到数字量的温度值的转换)

2）接线方式

a)独立供电方式

ds18b20 板子

GND GND
VDD 5V
DQ GPIO 管脚

b)寄生电源

ds18b20 板子
GND GND
VDD GND
DQ GPIO 管脚

ds18b20 内部有电容

当连接 DQ 的 GPIO 管脚上为高电平时 ds18b20 内部的电容充电

当连接 DQ 的 GPIO 管脚为为低电平时 ds18b20 内部的电容放电以维持 ds18b20 的能耗

3）该芯片属于一线式设备

和 CPU 通信时只使用一根线 DQ

如何操作 DQ 才能获取到温度值？具体研究 ds18b20 数据手册

4）阅读 ds18b20 datasheet

温度值用 9~12bit 的数字量表示温度值

每个 ds18b20 内部有一个唯一的 64bit 的序列号

如何与 ds18b20 进行数据交互

a)初始化

cpu 发复位信号

ds18b20 紧跟着回 presence pulse

b)rom command

cpu 发 MATCH ROM 64 序列号 （点名）

cpu 发 SKIP ROM //当总线上只有一个 ds18b20 时 默认就选中

c)function command

cpu 发 CONVERT T //告诉选中的 ds18b20 开始干活 （完成温度采集转换）

cpu 发 READ SCRATCHPAD //告诉选中的 ds18b20 输出 SCRATCHPAD 中的值
//后续 cpu 端就可以接收数据

具体初始化的时序 ： 参考 datasheet P15

复位脉冲：

空闲时 DQ 默认为高电平

CPU 端将 DQ 拉低 最少持续 480us

存在脉冲：

复位脉冲之后

ds18b20 将 DQ 拉低 持续 60~240us

收发数据的数据时序: 参考 datasheet P16

总结：如何从 ds18b20 中读取出温度值?

cpu 发送 reset 脉冲

接收 ds18b20 返回的 presence 脉冲
cpu 发送 skip rom (0xcc)
cpu 发送 CONVERT T (0x44) //让 ds18b20 开始工作

等待指定的时间 //最长是 750ms

cpu 发送 reset 脉冲
接收 ds18b20 返回的 presence 脉冲
cpu 发送 skip rom
cpu 发送 READ ROM (0x33)
cpu 接收 ds18b20 回传的数据 byte0 byte1 byte2 ... byte8

5) 驱动编程

```
cd /home/tarena/ehome/drivers
mkdir ds18b20
cd ds18b20
vi ds18b20_drv.c
vi Makefile
make
vi test.c
arm-cortex....-gcc test.c -o test

cp *.ko ../../rootfs/home/drivers
cp test ../../rootfs
```

在板子上执行

```
insmod /home/drivers/ds18b20_drv.ko
./test
```

观察温度值

个人简历:

总体原则:

1) 自身优势点放到的明显位置
2) 简历中要从事的编程方向要明确

搞驱动开发: 熟悉的 IIC GPIO UART SPI CAN ....
使用万用表 电烙铁 示波器 。。。
input 子系统 platform 总线

搞应用软件开发:
会的算法
多线程编程
进程间通信
。。。

基本格式

基本信息

名校 相关专业 英语 4 级 党员

个人技能

精通 C 编程

熟悉 C++ ARM 汇编 QT

常见算法：

深入理解进程切换任务调度机制，熟悉进程间通信方式

常见接口：IIC UART SPI

深入理解驱动编程框架 input platform ...

使用万用表 电烙铁 示波器 。。。

使用 protel 画电路原理图

项目经验：体现实现功能时使用了哪些技术

硬件平台： s5p6818 (cortex-a53)

软件平台: linux3.4

项目实现的功能：

主要负责实现的功能：

配置内核支持 uvc 子系统 移植 mjpg-streamer 搭建视频服务器 使用 QT 实现视频数据显示

编程 mma8653 芯片（I2C 接口）驱动，用户态实现计步功能

工作经历

尽量是和编程 和开发相关的工作经历

自我评价

善于专研

良好的沟通能力 团队合作

...