

# 2017 年《嵌入式操作系统》期末 project 设计报告

## 第一章 项目概述

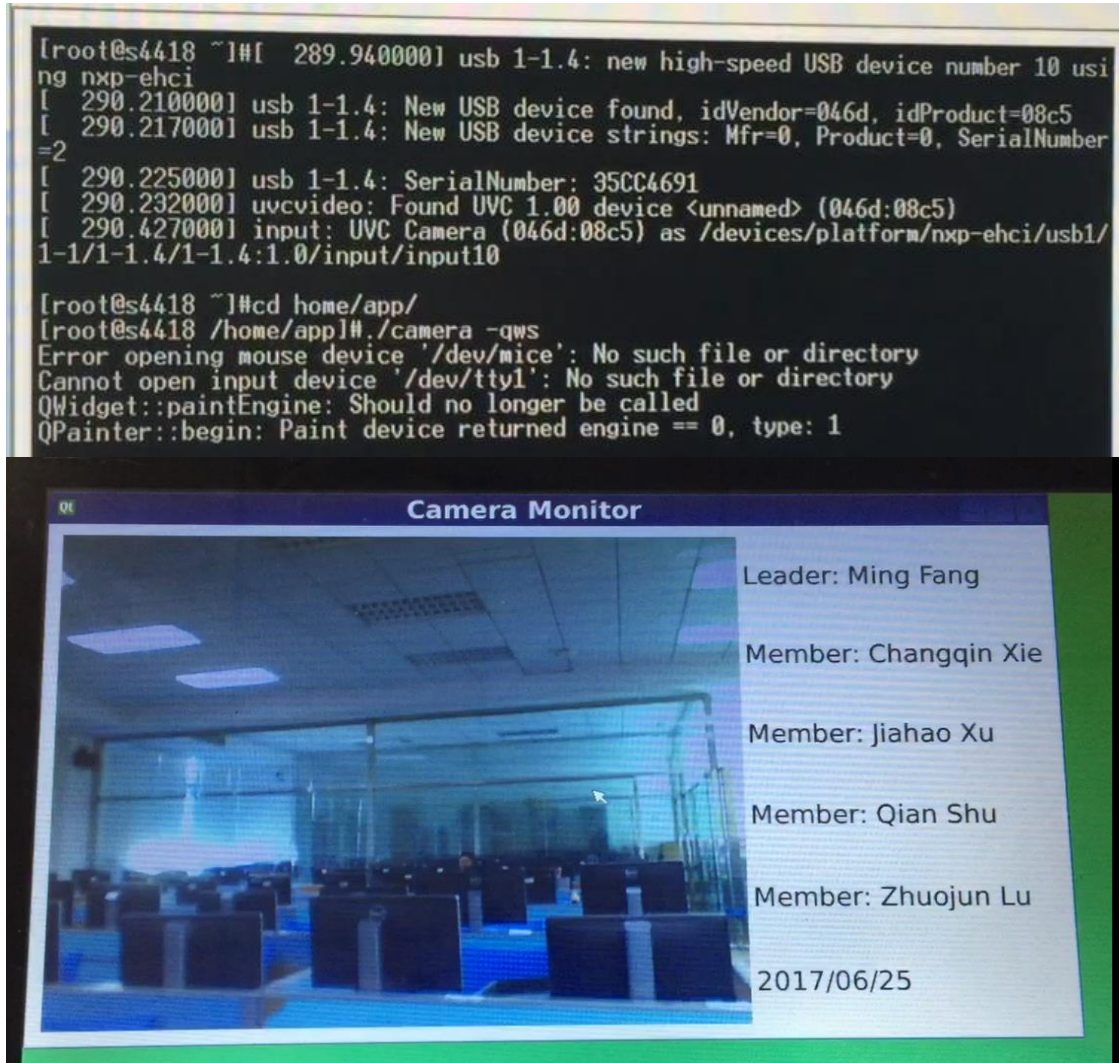
- 1. 目标：定制和移植 s4418 平台上的最小 Linux 内核与文件系统。在最小系统的基础上，加入 USB 摄像头，用 QT 编写界面，实现一个利用 USB 摄像头进行实时监控的简单系统。根据需求，内核需要有基本的功能与 USB 驱动，文件系统也要加上 QT 环境。
- 2. 已完成的功能：内核和文件系统可以成功编译制作，安装到 s4418 平台上后可以正常启动；USB 摄像头可以被正常驱动、识别；使用了 QT 设计 GUI，摄像头拍摄到的画面可以成功显示在板子的屏幕上。
- 3. 测试的效果：插入 USB 摄像头后，/dev 目录下会出现 video9 设备，UVC 驱动识别到摄像头插入，在终端上显示：uvcvideo: Found UVC 1.00 device (046d:08c5)。通过手动移动摄像头，屏幕上显示的画面也在移动，画面有一定延时，，满足监控需求。

## 第二章 项目人员组成及分工

人员	分工	占比
方铭	添加 UVC 驱动至内核，文件系统移植 QT，调试摄像头	30%
卢卓君	设计报告	18%
舒倩	最小内核与文件系统的裁剪与编译	20%
谢昌秦	调试摄像头，QT 图形化界面的代码编写与调试	20%
徐佳豪	录制视频	12%

项目地址：<https://github.com/SimonFang1/Realtime-Monitor/>

### 第三章 项目效果



根据图片，可以看出，摄像头拍摄到的图像都显示在了屏幕上；移动的过程中，图像也在跟着移动，实时监控功能已经实现了。为了让屏幕布局更好看，在显示图像的边框右边，加上了一些 QT 编写的界面，虽然内容比较简单，但也算是完成了图形化界面的设计。对于最小内核的文件系统的裁剪，无法形象的体现出来，只有通过编译后的文件大小来说明。开发板能正常烧写和运行，说明没有裁剪掉实时监控系统所需要的内核模块和系统程序。

### 第四章 项目开发过程

#### 一、最小内核的裁剪和编译

原始内核大小：

```
Image Name: Linux-3.4.39
Created: Thu Jul 6 00:42:33 2017
Image Type: ARM Linux Kernel Image (uncompressed)
Data Size: 4752248 Bytes = 4640.87 kB = 4.53 MB
Load Address: 40008000
Entry Point: 40008000
Image arch/arm/boot/uImage is ready
```

内核配置结果:

```
Image Name: Linux-3.4.39
Created: Sat Jul 8 00:35:08 2017
Image Type: ARM Linux Kernel Image (uncompressed)
Data Size: 2753424 Bytes = 2688.89 kB = 2.63 MB
Load Address: 40008000
Entry Point: 40008000
Image arch/arm/boot/uImage is ready
```

使用 `make menuconfig` 进入图形化界面

1、General setup 代码成熟度选项，它又有子项:

1.1、prompt for development and/or incomplete code/drivers

该选项是对那些还在测试阶段的代码，驱动模块等的支持。一般应该选这个选项，除非你只是想使用 Linux 中已经完全稳定的东西。但这样有时对系统性能影响挺大。

暂时选

1.2、Cross-compiler tool prefix

交叉编译工具前缀，例如：Cross-compiler tool prefix 值为: (arm-Linux-)

默认不选

1.3、Local version - append to kernel release 内核显示的版本信息，填入 64 字符以内的字符串，你在这里填上的字符口串可以用 `uname -a` 命令看到。

默认不选

1.4、Automatically append version information to the version string 自动在版本字符串后面添加版本信息,编译时需要有 perl 以及 Git 仓库支持

不选

1.5、Kernel compression mode (Gzip) ---> 有四个选项，这个选项是说内核镜像要用的压缩模式，回车一下，可以看到 `gzip,bzip2,lzma,lzo`,一般可以按默认的 `gzip`,如果要用 `bzip2,lzma,lzo` 要先装上支持

默认不选

### 1.6、Support for paging of anonymous memory (swap)

使用交换分区或交换文件来做为虚拟内存，一定要选上。

默认不选

### 1.7、System V IPC

表示系统的进程间通信 Inter Process Communication，它用于处理器在程序之间同步和交换信息，如果不选这项，很多程序运行不起来。

必选

### 1.8、POSIX Message Queues

POSIX 标准的消息队列，它同样是一种 IPC。

默认不选

### 1.9、BSD Process Accounting

用户进程访问内核时将进程信息写入文件中。通常主要包括进程的创建时间/创建者/内存占用等信息。建议最好选上。

**BSD Process Accounting version 3 file format** 使用新的第三版文件格式,可以包含每个进程的 PID 和其父进程的 PID,但是不兼容老版本的文件格式。

默认不选

### 1.10 默认不选

**1.11、Export task/process statistics through netlink (EXPERIMENTAL)** 通过 netlink 接口向用户空间导出任务/进程的统计信息,与 BSD Process Accounting 的不同之处在于这些统计信息在整个任务/进程生存期都是可用的

**Enable per-task delay accounting (EXPERIMENTAL)**

在统计信息中包含进程等候系统资源(cpu,IO 同步,内存交换等)所花费的时间

**Enable extended accounting over taskstats (EXPERIMENTAL)** 在统计信息中包含扩展进程所花费的时间

默认不选

**1.12、Auditing support** 审计支持，用于和内核的某些子模块同时工作，例如 Security Enhanced linux。只有选择此项及它的子项，才能调用有关审计的系统调用。

默认选

**1.13、Enable system-call auditing support** 支持对系统调用的审计

默认选

#### +1.14 Make audit loginuid immutable

默认不选

#### 1.14、IRQ subsystem --->

中断子系统 Support sparse irq numbering

<=== 支持稀有的中断编号，关闭

默认不选

#### 1.15、RCU Subsystem --->

非对称读写锁系统 是一种高性能的 kernel 锁机制，适用于读多写少环境 RCU Implementation (Tree-based hierarchical RCU) ---> RCU 实现机制 Tree(X) Tree-based hierarchical RCU 基本数按等级划分 Enable tracing for RCU 激活跟踪 (32) Tree-based hierarchical RCU fanout value 基本数按等级划分分列值 Disable tree-based hierarchical RCU auto-balancing

默认不选

1.16、<> Kernel .config support 这个选项允许.config 文件（即编译 LINUX 时的配置文件）保存在内核当中

不选

#### 1.17、(17) Kernel log buffer size (16 => 64KB, 17 => 128KB)

#### 1.18、[ ] Control Group support --->

cgroups 支持， 文档资料， cgroups 主要作用是给进程分组， 并可以动态调控进程组的 CPU 占用率。比如 A 进程分到 apple 组， 给予 20%CPU 占用率， E 进程分 easy 组， 给予 50%CPU 占用率， 最高 100%。我目前没有此类应用场景， 用到时会选择将其编译进去。 CPU bandwidth provisioning for FAIR\_GROUP\_SCHED 此选项允许用户定义的 CPU 带宽速率（限制）在公平的组调度运行的任务。组没有限制设置被认为是无约束和运行没有限制。

Group scheduling for SCHED\_RR/FIFO 此功能可以让您显式地分配真实的 CPU 带宽任务组。

默认选

#### +1.18 Checkpoint/restore support

默认不选

#### 1.19、-\*- Namespaces support --->

命名空间支持， 允许服务器为不同的用户信息提供不同的用户名空间服务 [\*] UTS namespace

通用终端系统的命名空间。它允许容器， 比如 Vservers 利用 UTS 命名空间来为不

同的服务器提供不同的 UTS。如果不清楚，选 N。 [\*] IPC namespace  
IPC 命名空间，不确定可以不选 [\*] User namespace (EXPERIMENTAL) User 命名空间，不确定可以不选 [\*] PID Namespaces  
PID 命名空间，不确定可以不选 [\*] Network namespace

默认不选

1.20、Automatic process group scheduling 自动进程组调度

默认不选

1.21、[ ] enable deprecated sysfs features to support old userspace tools

默认不选不选

1.22、-\*- Kernel->user space relay support (formerly relayfs)

在某些文件系统上( 比如 debugfs ) 提供从内核空间向用户空间传递大量数据的接口，我目前没有此类应用场景

默认不选

1.23、[\*] Initial RAM filesystem and RAM disk (initramfs/initrd) support

用于在真正内核装载前，做一些操作（俗称两阶段启动），比如加载 module，mount 一些非 root 分区，提供灾难恢复 shell 环境等，资料，我是期望直接从 kernel image 直接启动，所以没选它

不选

1.25、Optimize for size 这个选项将在 GCC 命令后用“-Os”代替“-O2”参数，这样可以得到更小的内核。没必要选。选上了有时会产生错误的二进制代码。

默认不选

1.26、Enable full-sized data structures for core: 在内核中使用全尺寸的数据结构. 禁用它将使得某些内核的数据结构减小以节约内存,但是将会降低性能。

默认选

1.27、Enable futex support: 快速用户空间互斥体可以使线程串行化以避免竞态条件,也提高了响应速度.禁用它将导致内核不能正确的运行基于 glibc 的程序。

默认选

1.28、Enable eventpoll support: 支持事件轮循的系统调用。

默认选

+1.28 Enable signalfd() system call

+1.28 Enable timerfd() system call

+1.28 Enable eventfd() system call

以上系统调用都用不到，不选

**1.29、Use full shmem filesystem:** 除非你在很少的内存且不使用交换内存时，才不要选择这项。后面的这四项都是在编译时内存中的对齐方式，0 表示编译器的默认方式。使用内存对齐能提高程序的运行速度，但是会增加程序对内存的使用量。内核也是一组程序呀。

**Enable VM event counters for /proc/vmstat:** 允许在/proc/vmstat 中包含虚拟内存事件计数器。 [\*] **Disable heap randomization** 禁用随机 heap (heap 堆是一个应用层的概念，即堆对 CPU 是不可见的，它的实现方式有多种，可以由 OS 实现，也可以由运行库实现,如果你愿意，你也可以在一个栈中来实现一个堆)

默认选

+1.29 Enable AIO support 默认不选

+1.29 Embedded system 不选

+1.29 Kernel Performance Events And Counters 不选

+1.29 Enable VM event counters for /proc/vmstat

不选

+1.29 Enable SLUB debugging support 不选

+1.29 Display heap randomization

**1.30、Choose SLAB allocator (SLAB) --->** 选择内存分配管理器（强烈推荐使用 SLUB）

这个选项可以让内核的基本选项和设置无效或者扭曲。这是用于特定环境中的，它允许“非标准”内核。你要是选它，你一定要明白自己在干什么。这是为了编译某些特殊用途的内核使用的，例如引导盘系统。配置标准的内核特性(为小型系统)

**Enable 16-bit UID system calls:** 允许对 UID 系统调用进行过时的 16-bit 包装。

**Sysctl syscall support** 几乎使用不到这一选项，不选它可以轻微使内核变小

**Include all symbols in kallsyms:** 在 kallsyms 中包含内核知道的所有符号,内核将会增大

300K。 **Enable support for printk:** 允许内核向终端打印字符信息,在需要诊断内核

为什么不能运行时选择。 **BUG() support:** 显示故障和失败条件(BUG 和 WARN),禁用

它将可能导致隐含的错误被忽略。 **Enable ELF core dumps:** 内存转储支持,可以帮助调试 ELF 格式的程序。 -->

**1.32、[ ] Profiling support** 不选剖面支持，用一个工具来扫描和提供计算机的剖面图。支持系统评测（对于大多数用户来说并不是必须的）

不选

1.34、Kprobes 调试内核除非开发人员，否则不选

不选

1.35、Optimize trace point call sites

不选

1.36、GCOV-based kernel profiling

☐ Enable gcov-based kernel profiling 不选

2、Enable loadable module support

整个不选

2.1 Forced module loading

允许强制加载模块

2.2 Module unloading

允许卸载已经加载的模块

2.3 Forced module unloading 允许强制卸载正在使用中的模块(比较危险)这个选项允许你强行卸除模块，即使内核认为这不安全。内核将会立即移除模块，而不管是否有人在使用它（用 `rmmod -f` 命令）。这主要是针对开发者和冲动的用户提供的功能。如果不清楚，选 N。

2.4 Module versioning support 有时候，你需要编译模块。选这项会添加一些版本信息，来给编译的模块提供独立的特性，以使不同的内核在使用同一模块时区别于它原有的模块。这有时可能会有点用。如果不清楚，选 N。允许使用其他内核版本的模块(可能会出问题)

2.5 Source checksum for all modules 为所有的模块校验源码,如果你不是自己编写内核模块就不需要它这个功能是为了防止你在编译模块时不小心更改了内核模块的源代码但忘记更改版本号而造成版本冲突。如果不清楚，选 N。

3、Enable the block layer

块设备支持,使用硬盘/USB/SCSI 设备者必选这选项使得块设备可以从内核移除。如果不选，那么 `blockdev` 文件将不可用，一些文件系统比如 `ext3` 将不可用。这个选项会禁止 SCSI 字符设备和 USB 储存设备，如果它们使用不同的块设备。选 Y，除非你知道你不需要挂载硬盘和其他类似的设备。不过此项无可选项

3.1 Support for large (2TB+) block devices and files 仅在使用大于 2TB 的块设备时需要

不选



### 3.2 Block layer SG support v4 通用 scsi 块设备第 4 版支持 选上

### 3.3 Block layer data integrity support

块设备数据完整性支持

不选

### 3.4 Block layer bio throttling support

可用于限制设备的 IO 速度

不选

**3.5 Partition Types Advanced partition selection** 如果你想要在 linux 上使用一个在其他的介质上运行着操作系统的硬盘时，选择 Y，如果你不确定时可以选 N

不选

**3.6 IO Schedulers** IO 调度器 I / O 是输入输出带宽控制，主要针对硬盘，是核心的必须的东西。这里提供了三个 IO 调度器。

#### Deadline I/O scheduler

使用轮询的调度器,简洁小巧,提供了最小的读取延迟和尚佳的吞吐量,特别适合于读取较多的环境(比如数据库)Deadline I / O 调度器简单而又紧密，在性能上和抢先式调度器不相上下，在一些数据调入时工作得更好。至于在单进程 I / O 磁盘调度上，它的工作方式几乎和抢先式调度器相同，因此也是一个好的选择。

**CFQ I/O scheduler** 使用 QoS 策略为所有任务分配等量的带宽,避免进程被饿死并实现了较低的延迟,可以认为是上述两种调度器的折中.适用于有大量进程的多用户系统 CFQ 调度器尝试为所有进程提供相同的带宽。它将提供平等的工作环境，对于桌面系统很合适。

**Default I/O scheduler (CFQ)** 默认 IO 调度器我这样理解上面三个 IO 调度器：抢先式是传统的，它的原理是一有响应，就优先考虑调度。如果你的硬盘此时在运行一项工作，它也会暂停下来先响应用户。期限式则是：所有的工作都有最终期限，在这之前必须完成。当用户有响应时，它会根据自己的工作能否完成，来决定是否响应用户。CFQ 则是平均分配资源，不管你的响应多急，也不管它的工作量是多少，它都是平均分配，一视同仁的。 (\*) Deadline

( ) CFQ

( ) No-op

## 4、System Type

未动

## 5、FIQ Mode Serial Debugger

不选

## 6、Bus Support PCCard Support

不选

## 7、kernel Features

未动

## 8、Boot Options

不选

## 9、CPU Power Management

CPU Frequency scaling

CPU idle PM support

都不选

## 10、Floating point emulation 选一个 VFP-format floating point maths

## 11、userpace

## 12、power management options 12.1 Suspend to ram and standly

待机 选择

## 12.2 Run-time PM core functionality

不选

## 12.3 power management debug support

不选

## 12.4 Advanced power management emulation

选择

## 12.5 Log time spend in suspend

不选

## 13、networking System

本次项目没有用到网络系统，所以基本上都不选

## 13.1、Networking options

☐ Packet socket

☐ Unix domain sockets

[ ] PF\_KEY sockets

[ ] TCP/IP networking

< > DECnet Support

< > ANSI/IEEE 802.2 LLC type 2 Support

< > The IPX protocol

< > Appletalk protocol support

< > CCITT X.25 Packet Layer

(EXPERIMENTAL)

< > LAPB Data Link Driver (EXPERIMENTAL)

< > Acorn Econet/AUN protocols (EXPERIMENTAL)

< > WAN router

< > Phonet protocols family

< > IEEE Std 802.15.4 Low-Rate Wireless Personal Area Networks support  
(EXPERI|

[ ] QoS and/or fair queueing --->

通过 IPRoute 切换网络设备上的 QoS 策略，我不打算使用 IP 路由

[ ] Data Center Bridging support

-\*- DNS Resolver support

< > B.A.T.M.A.N. Advanced Meshing Protocol

Network testing --->

13.2、[ ] Amateur Radio support --->

13.3、< > CAN bus subsystem support --->

13.4、< > IrDA (infrared) subsystem support --->

13.5、< > Bluetooth subsystem support --->

13.6、< > RxRPC session sockets

13.7、-- Wireless --->

13.8、< > WiMAX Wireless Broadband support --->

13.9、< > RF switch subsystem support --->

13.10、< > Plan 9 Resource Sharing Support (9P2000) (Experimental) --->

13.11、< > CAIF support --->

13.12、< > Ceph core library (EXPERIMENTAL)

内核编译命令：

```
$ cd /usr/local/src/s4418/s5p4418-kitkat/kernel
$ export PATH=../uboot/tools:$PATH
$ cp arch/arm/configs/x4418_defconfig .config
$ make menuconfig
$ make ARCH=arm CROSS_COMPILE= .. /prebuilts/gcc/linux-x86/arm/arm-eabi-4.7/bin/arm-eabi- uImage
```

## 二、最小文件系统制作

```
...
|-- bin
|   |-- busybox
|   '--- sh -> busybox
|-- dev
|   '--- console
|-- etc
|   '--- init.d
|   '--- rcS
'-- lib
    |-- ld-2.3.2.so
    |-- ld-linux.so.2 -> ld-2.3.2.so
    |-- libc-2.3.2.so
    '--- libc.so.6 -> libc-2.3.2.so
...
```

这个最小根文件系统例子只有 5 个目录，共 8 个文件。这个极小的根文件系统可以实现系统引导，并通过串行控制台为用户提供一个功能完整的命令行提示，用户可以使用 **busybox** 中启用的任意命令。

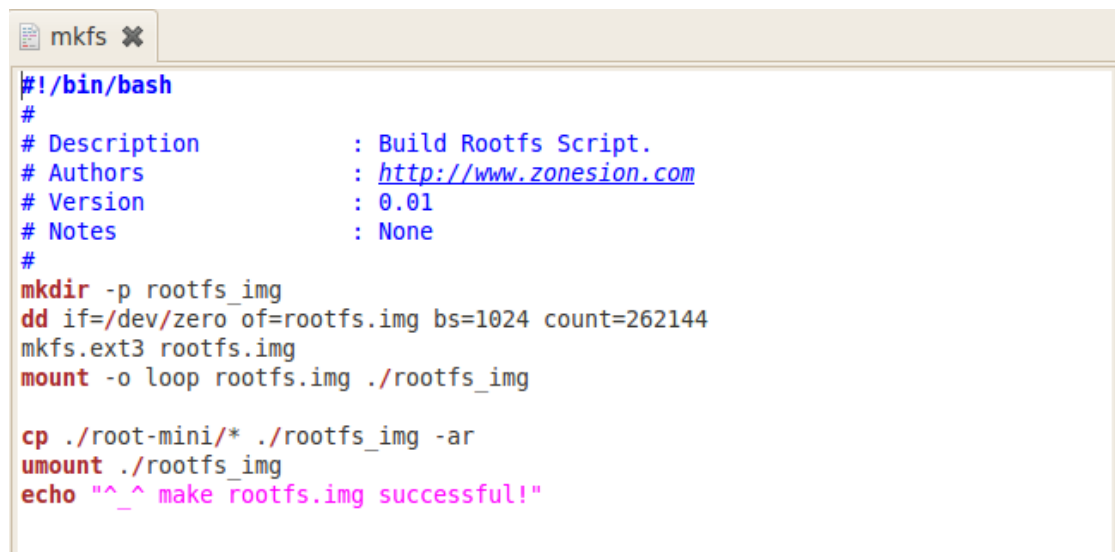
从/bin 目录开始看，在其下面已经有了可执行的 **busybox** 文件和指向 **busybox** 的软链接（soft link）**sh**，你一会儿就会明白这样做的必要性。**/dev** 目录下的文件是打开一个控制台进行输入/输出所需要的设备节点（**device node**）。**/etc/init.d** 目录下的 **rcS** 文件是由 **busybox** 启动时处理的默认的初始化脚本文件，尽管该文件并不是必需的。使用 **rcS** 文件之后就不会出现 **busybox** 发出的警告信息，该警告信息只会在 **rcS** 文件缺失的时候才出现。

上述根文件系统必需的最后一个目录项及两个文件是两个库：GLIBC（libc-2.3.2.so）库和 Linux 动态加载器（ld-2.3.2.so）。GLIBC 包括 C 标准库函数（如 printf()）以及绝大多数应用程序所依赖的其他大量库函数；Linux 动态加载器用于将二进制可执行文件加载到内存中，并且执行应用程序引用所需共享库函数的链接工作。这里包括的两个软链接是指向 ld-2.3.2.so 的 ld-linux.so.2 和指向 libc-2.3.2.so 的 libc.so.6，这些链接使这些共享库不受版本影响并且具有向后兼容的特性，在所有 Linux 系统下都能看到这类链接。

本次实验并不需要用到 busybox 的所有命令行功能，所以可以根据需求进一步裁剪。

busybox 中，Busybox Settings，Init Utilities，Shells，Coreutils 不可裁剪，其余的都可以删掉；同时也可以对 Coreutils 执行进一步的裁剪。本次实验理论上用不到任何与命令行有关的操作，只需要执行一个编译 c++ 代码后生成的可执行文件就可以了。所以将 Coreutils 中所有命令设置为 n 也可以满足要求。但是为了调试方便，保留了 cd 和 ls 的命令。

最后，再把内核和交叉编译好的 QT 移植进这个文件系统。



```
#!/bin/bash
#
# Description      : Build Rootfs Script.
# Authors         : http://www.zonesion.com
# Version        : 0.01
# Notes          : None
#
mkdir -p rootfs_img
dd if=/dev/zero of=rootfs.img bs=1024 count=262144
mkfs.ext3 rootfs.img
mount -o loop rootfs.img ./rootfs_img

cp ./root-mini/* ./rootfs_img -ar
umount ./rootfs_img
echo "^_^ make rootfs.img successful!"
```

*mkfs*

文档和光盘提供的 mkfs 脚本生成的镜像大小为 256MB，共有 262144 个扇区，每个扇区大小为 1024 字节。



*actual size*

挂载镜像后，可以看到，其实际大小为 10.6MB，因此可以把它大幅缩减为 16MB。保持扇区大小不变，修改 mkfs 脚本，令 count=16384，即虚拟镜像文件的存储介质的扇区数为 16384 个。

三、usb 摄像头驱动 在裁剪好的 linux 内核基础上，检查并确保以下模块已经加入。

```
Device Drivers --->
...<*> Multimedia support --->
.....<*> Video For Linux
.....[*] Enable Video For Linux API 1 (DEPRECATED)
.....[*] Video capture adapters --->
.....[*] V4L USB devices --->
.....<*> USB Video Class (UVC)
.....[*] UVC input events device support
.....[*] GSPCA based webcams --->
```

由于本系统的核心功能是实时监控，UVC 驱动是开发板每次启动所必须的，因此把上述模块全部加入内核，开机自动载入内核，而不是以模块方式手动加载。

#### 四、QT 界面的编写

本次实验中的 GUI 还是相对简单的，使用 Qt 的 GUI 框架即可很快完成。

Qt 为我们做好了内存管理，因此我们可以自由使用 new 关键字来创建 UI 而不需要考虑指针的销毁。本次实验，我们的 UI 布局比较简单：窗口左边是摄像头的实时画面，右边是我们的小组成员列表。

实际开发的时候，我们需要一个 `QWidget` 对象，这个对象会成为程序的主窗体，也就是整个 UI 视图树的根。然后，我们创建一个横向布局 `QHBoxLayout`，在里面依次添加一个 `QLabel` 和一个 `QVBoxLayout`，前者用于渲染摄像头画面，后者则是小组成员列表文本的容器。同样地，我们依次实例化 `QLabel` 对象，设定显示文本，然后添加到右边的 `QVBoxLayout` 中。至此，我们便完成了 Qt 的界面编写了。

值得注意的是，在 `QHBoxLayout` 中添加子控件和子布局使用的是不同的方法，前者使用 `addWidget`，而后者要使用 `addLayoutItem`。

## 第五章 项目总结

### 遇到的问题

1. 开发文档的摄像头例程无法运行，且文档中表述的二进制程序的文件名和 `Makefile` 生成的文件名不一致。开发文档使用的摄像头是模拟摄像头，本实验使用的摄像头是数字摄像头，学习此例程无助于本项目的推进。
2. 裁剪内核时，非常容易导致编译错误。内核模块之间存在一定的依赖关系，使用 `make menuconfig` 菜单添加或删除模块不能够自动添加或解除这种依赖。

解决方案：参考网络上其他人裁剪的内核的过程和原因，对照着进行裁剪。很多模块看起来是没有用的，但是由于它作为基础模块，很多模块都依赖于它，要删除必须把所有的依赖项也删除，因此在裁剪的时候一定要冷静耐心，不可贪心。对裁剪了容易导致编译失败地模块，原则上不宜裁剪掉。

### 未解决的问题

1. 使用 `tftp` 程序向开发板传送文件时，偶尔会出现数据传输到一半就卡住了，过一会儿连接被重置，提示传送失败。拔掉网线重连也无法解决这个问题。只能不停地重启实验平台，多次尝试，才能成功传送文件。
2. 编译内核时，偶尔会提示 `"mkimage" not found, U-Boot images will not be built` 的错误，`uboot/tools` 目录下已存在 `mkimage` 程序，且编译内核前执行了 `"export PATH=../uboot/tools:$PATH"`，但是系统的环境变量还是没有更新，找不到那个文件，重启系统后再次编译后就成功了。
3. 原内核不支持鼠标和键盘驱动，导致 QT 程序无法调用 USB 鼠标和键盘，系统缺少交互功能。
4. 未实现 `socket` 编程，实现远程监控功能。
5. 实验使用的罗技摄像头，在 UVC 的官网上有 `warning`，提示 UVC 虽然支持这种摄像头，但是还是会有一些意想不到的错误。系统运行一段时间后，容易发生段错误，导致 QT 程序异常退出。

### 完善项目建议

1. 系统没有交互功能，若要完善，在解决 USB 键鼠驱动问题后，可以考虑在屏幕上增加交互控件，增加播放和暂停功能。
2. 系统不支持远程监控，若要完善，可增加对远程监控的支持。
3. 可以考虑在屏幕上增加交互控件，实现监控录像回放，保存的功能。