bootloader 的启动过程和代码详见：https://github.com/luowei1995/my-boot.git

开发板上电后，首先运行start.S

1.关闭看门狗

2.设置FCLK、HCLK、PCLK比例

3.设置CPU工作模式

4.设置系统时钟为200MHz

5.使能高速缓冲

6.初始化SDRAM

7.从nandflash中复制出bootloader代码（源地址：0目的地址：_start(0x33f8 0000)数据长度：__bss_start减_start）

8.进入主函数

再运行boot.c

1.初始化uart0

2.从nandflash中复制出内核代码（源地址：0x60000目的地址：0x3000 8000数据长度：0x50 0000）

3.设置参数存入params结构体

4.最后是引导内核，利用函数指针跳到内核的第一个函数地址0x3000 8000

内核启动过程

1.架构/开发板相关的引导过程，采用汇编语言编写：

arch/arm/kernel/head.s：

```
        .section ".text.head", "ax"
ENTRY(stext)
        setmode PSR_F_BIT | PSR_I_BIT | SVC_MODE, r9 @ ensure svc mode
                                                @ and irqs disabled
        mrc     p15, 0, r9, c0, c0              @ get processor id
        bl      __lookup_processor_type         @ r5=procinfo r9=cpuid //确认内核是否支持此处理器架构
        movs    r10, r5                         @ invalid processor (r5=0)?
        beq     __error_p                       @ yes, error 'p'
        bl      __lookup_machine_type           @ r5=machinfo
        movs    r8, r5                          @ invalid machine (r5=0)?//确认内核是否支持该单板
        beq     __error_a                       @ yes, error 'a'
        bl      __vet_atags             //检查内核参数是否合法
        bl      __create_page_tables//建立页表

        /*
         * The following calls CPU specific code in a position independent
         * manner.  See arch/arm/mm/proc-*.S for details.  r10 = base of
         * xxx_proc_info structure selected by __lookup_machine_type
         * above.  On return, the CPU will be ready for the MMU to be
         * turned on, and r0 will hold the CPU control register value.
         */
        ldr     r13, __switch_data              @ address to jump to after//跳转到head-common文件
                                                @ mmu has been enabled
        adr     lr, BSYM(__enable_mmu)          @ 使能MMU
 ARM(   add     pc, r10, #PROCINFO_INITFUNC     ) /*禁止I/D cacahe*/
 THUMB( add     r12, r10, #PROCINFO_INITFUNC    )
 THUMB( mov     pc, r12                         )
ENDPROC(stext)
```

arch/arm/kernel/head-common.s：

```
__mmap_switched:
        adr     r3, __switch_data + 4

        ldmia   r3!, {r4, r5, r6, r7}
        cmp     r4, r5                          @ Copy data segment if needed复制数据段
1:      cmpne   r5, r6
        ldrne   fp, [r4], #4
        strne   fp, [r5], #4
        bne     1b

        mov     fp, #0                          @ Clear BSS (and zero fp)清除bss段
1:      cmp     r6, r7
        strcc   fp, [r6],#4
        bcc     1b

 ARM(   ldmia   r3, {r4, r5, r6, r7, sp})
 THUMB( ldmia   r3, {r4, r5, r6, r7}    )
 THUMB( ldr     sp, [r3, #16]           )
        str     r9, [r4]                        @ Save processor ID保存CPU ID
        str     r1, [r5]                        @ Save machine type保存机器类型,就是1999
        str     r2, [r6]                        @ Save atags pointer保存栈指针,就是内核参数起始地址
        bic     r4, r0, #CR_A                   @ Clear 'A' bit
        stmia   r7, {r0, r4}                    @ Save control register values
        b       start_kernel                    @跳转到第一个c语言函数start_kernel()（在init/main.c中）
ENDPROC(__mmap_switched)
```

以上内核启动的第一个阶段包括了确定内核架构、确定机器类型、建立页表、禁止 cache、使能 mmu、复制数据段、清除 boss 段、设置栈指针、保存 CPU ID、保存机器类型 ID、调用 start_kernel()等过程。

2.通用启动过程，采用 C 语言编写：

init/main.c/start_kernel()：

```
//start_kernel是所有Linux平台的通用初始化入口函数
asmlinkage void __init start_kernel(void)
{
        char * command_line;
        extern struct kernel_param __start___param[], __stop___param[];
        //获取多核处理器的ID
        smp_setup_processor_id();

        /*
         * Need to run as early as possible, to initialize the
         * lockdep hash:
         */
        //初始化哈希表(对于ARM为空)
        lockdep_init();
        //在调试的时候用
        debug_objects_early_init();

        /*
         * Set up the the initial canary ASAP:
         */
        //保护堆栈（对于ARM为空）
        boot_init_stack_canary();
        //控制器组的早期初始化
        cgroup_init_early();
        //关闭中断
        local_irq_disable();
        early_boot_irqs_off();
```

……

```
        cpuset_init();
        taskstats_init_early();//初始化任务状态相关的缓存、队列和信号量
        delayacct_init();//初始化每个任务延时计数。当一个任务等CPU运行，或者等IO同步时，都需要计算等待时间

        check_bugs();//检查CPU配置、FPU等是否非法使用不具备的功能
        //高级配置及电源接口
        acpi_early_init(); /* before LAPIC and SMP init */
        sfi_init_late();//一个轻量级的方法用于平台固件通过固定的内存页表传递信息给操作系统

        //初始化内核跟踪模块，ftrace的作用是帮助开发人员了解Linux 内核的运行时行为
        ftrace_init();

        /* Do the rest non-__init'ed, we're now alive */
        //创建内核线程init，并运行
        rest_init();
}
```

以上内核启动的第二个阶段包含了大量操作系统相关的初始化工作、设置与体系结构相关的环境、初始化控制台打印、最后启动了 init 进程。

文件系统启动过程

默认的内核命令是"noinitrd root=/dev/mtdblock3 init=/linuxrc console=ttySAC0"，当文件系统被挂载后，运行的第一个程序就是/linuxrc，/linuxrc 是一个链接，指向→/bin/busybox，因此运行的第一个程序就是 busybox（可执行程序），查看 busybox 的源码 busybox-1.13.3/init/init.c：

首先宏定义了#define INIT_SCRIPT　/etc/init.d/rcS

```
#define INITTAB      "/etc/inittab"     /* inittab file location */
#ifndef INIT_SCRIPT
#define INIT_SCRIPT  "/etc/init.d/rcS"  /* Default sysinit script. */
#endif
```

在此文件中的 parse_inittab()函数里，先打开/etc/inittab 文件，但是实际的文件目录里没有这个文件，所以打开为 NULL 进入下面的 if 语句：

```
static void parse_inittab(void)
{
#if ENABLE_FEATURE_USE_INITTAB
        char *token[4];
        parser_t *parser = config_open2("/etc/inittab", fopen_for_read);

        if (parser == NULL)
#endif
        {
                /* No inittab file -- set up some default behavior */
                /* Reboot on Ctrl-Alt-Del */
                new_init_action(CTRLALTDEL, "reboot", "");
                /* Umount all filesystems on halt/reboot */
                new_init_action(SHUTDOWN, "umount -a -r", "");
                /* Swapoff on halt/reboot */
                if (ENABLE_SWAPONOFF)
                        new_init_action(SHUTDOWN, "swapoff -a", "");
                /* Prepare to restart init when a QUIT is received */
                new_init_action(RESTART, "init", "");
                /* Askfirst shell on tty1-4 */
                new_init_action(ASKFIRST, bb_default_login_shell, "");
//TODO: VC_1 instead of ""? "" is console -> ctty problems -> angry users
                new_init_action(ASKFIRST, bb_default_login_shell, VC_2);
                new_init_action(ASKFIRST, bb_default_login_shell, VC_3);
                new_init_action(ASKFIRST, bb_default_login_shell, VC_4);
                /* sysinit */
                new_init_action(SYSINIT, INIT_SCRIPT, "");
                return;
```

最后一句执行 INIT_SCRIPT，也就是/etc/init.d/rcS，因此第二个被执行的程序就是/etc/init.d/rcS 脚本。打开 rcS 脚本文件（友善之臂提供的 rootfs_qtopia_qt4/etc/init.d/rcS）：

```
#! /bin/sh

PATH=/sbin:/bin:/usr/sbin:/usr/bin:/usr/local/bin:
runlevel=S
prevlevel=N
umask 022
export PATH runlevel prevlevel

#
#       Trap CTRL-C &c only in this shell so we can
#
trap ":" INT QUIT TSTP
/bin/hostname FriendlyARM

/bin/mount -n -t proc none /proc
/bin/mount -n -t sysfs none /sys
/bin/mount -n -t usbfs none /proc/bus/usb
/bin/mount -t ramfs none /dev

echo /sbin/mdev > /proc/sys/kernel/hotplug
/sbin/mdev -s
/bin/hotplug
# mounting file system specified in /etc/fstab
```

1.设置机器名字为 FriendlyARM
2.挂载虚拟文件系统/proc 和/sys，并且在/dev 下挂载一个 ramfs，相当于在/dev 目录覆盖上一块 SDRAM
/sys 和 挂载了 ramfs 的/dev 是正确创建设备节点的关键
3.通过 mdev -s 在/dev 目录下建立必要的设备节点文件
4.设置内核的 hotplug handler 为 mdev, 即当设备热插拔时,由 mdev 接收来自内核的消息并作出相应的回应, 比如挂载 U 盘

```
# mounting file system specified in /etc/fstab
mkdir -p /dev/pts
mkdir -p /dev/shm
/bin/mount -n -t devpts none /dev/pts -o mode=0622
/bin/mount -n -t tmpfs tmpfs /dev/shm
/bin/mount -n -t ramfs none /tmp
/bin/mount -n -t ramfs none /var
mkdir -p /var/empty
mkdir -p /var/log
mkdir -p /var/lock
mkdir -p /var/run
mkdir -p /var/tmp

/sbin/hwclock -s

syslogd
/etc/rc.d/init.d/netd start
echo "                          " > /dev/tty1
echo "Starting networking..." > /dev/tty1
sleep 1
/bin/chmod 0600 /usr/local/etc/ssh_*_key
/usr/local/sbin/sshd &
echo "                          " > /dev/tty1
echo "Starting ssh daemon..." > /dev/tty1
sleep 1
/etc/rc.d/init.d/httpd start
echo "                          " > /dev/tty1
echo "Starting web server..." > /dev/tty1
sleep 1
/etc/rc.d/init.d/leds start
echo "                          " > /dev/tty1
echo "Starting leds service..." > /dev/tty1
echo "                          "
sleep 1

/sbin/ifconfig lo 127.0.0.1
/etc/init.d/ifconfig-eth0

/bin/qtopia &
echo "                                    " > /dev/tty1
echo "Starting Qtopia, please waiting..." > /dev/tty1
```

5.就像注释中所说的,这是用来挂载其他一些常用的文件系统,并在/var 目录下(同样是ramfs,可写的)新建必要的目录。

6./sbin/hwclock -s 用来设定系统时间,从硬件 RTC 中获取

7.启动系统 log、网口、ssh、http、leds、回环地址、以太网、qtopia 图形界面等一系列服务。

自己制作的文件系统中的 rcS 脚本比较简单如下图所示:

```
#!/bin/sh
PATH=/sbin:/bin:/usr/sbin:/usr/bin
runlevel=S
prevlevel=N
umask 022
export PATH runlevel prevlevel

mount -a

echo /sbin/mdev>/proc/sys/kernel/hotplug
mdev -s

echo 'Start up successfully!'

#/bin/hostname -F /etc/sysconfig/HOSTNAME
/bin/hostname luowei
```