第10课 内核启动流程分析

1. 内核打补丁、编译、烧写、试验
2. 解压缩

tar xjf linux-2.6.22.6.tar.bz2

1. 打补丁

cd linux-2.6.22.6/

Patch –p1 < ../linux-2.6.22.6\_jz2440.patch

P1中的1代表忽略补丁内指定目录的第一个斜杠前面的内容+++是新的更改过的文件，---是原始文件

两者比较不同的地方是补丁文件

打补丁就是把补丁文件打到原始文件，对原始文件进行更改

三个＋标识增加文件

../linux-2.6.22.6\_jz2440.patch 补丁文件所处位置是linux文件夹的上一级中

1. 配置
2. 直接使用make menuconfig
3. 使用默认的配置

执行find –name “\*defconfig”

找到arch/arm/configs 找到类似文件s3c2410\_defconfig

执行make s3c2410\_defconfig

结果保存到.config文件里面

执行make menuconfig

1. 使用厂家提供的默认文件

config\_0k就是厂家提供的默认配置文件

复制为.config

cp config\_ok .config

执行make menuconfig

1. 编译make uImage

uImage=头部+vmlinux

uImage 是Uboot支持的内核，在内核文件之前加了一个头部

zImage是压缩过的uImage，内含解压方法

5>下载内核

1. 内核功能、结构、结合makefile、kconfig、进行分析

grep “CONFIG\_DM9000” -nwR

搜索包含“”内容的文件，并列出相应的行内容

-w   只显示全字符合的列。

-n在显示符合范本样式的那一列之前，标示出该列的列数编号

-R 递归搜索，即搜索子文件夹

有四类文件有CONFIG\_DM9000 宏

在头文件和C文件定义

C源码，arch/arm/plat-s3c24xx/common-smdk.c

drivers/net/Makefile:198:#obj-$(CONFIG\_DM9000) += dm9000.o

当$(CONFIG\_DM9000)配置为y时，标识编译进内核

为m时编译为可加载模块

include/config/auto.conf:142:CONFIG\_DM9000=y

配置时由.config自动生成，给makefile 使用,被顶层的makefile包含

include/linux/autoconf.h:143:#define CONFIG\_DM9000 1

显然autoconf.h是配置时由.config自动生成的，即CONFIG\_DM9000是配时自动生成，当配置为m或者y时被定义为1，C源码就可以使用。

Makefile 中体现了M和Y的区别，是编译为模块还是编译进内核

分析Makefile

inux-2.6.22.6\Documentation\kbuild \makefiles深入讲解了makefile

子目录下makefile

两个c文件的makefile编译配置，一般在子文件夹里面

编译到内核Obj-y+=a.o b.o

编译为模块

obj-m+=ab.o

ab-objs:=a.o b.o

名字ab可以任意， a.c->a.o b.c->b.o a.o b.o->ab.k

o

顶层Makefile

Make uImage uImage 在 arch\arm\makefile中定义

而执行时在顶层目录执行，所以顶层makefile一定包含了arch\arm\makefile

在495行有include $(srctree)/arch/$(ARCH)/Makefile

include/config/auto.conf 包含了自动生成的配置文件

arch\arm\makefile中有uImage: vmlinux 即uImage依赖于vmlinux

在顶层Makefiez中有all: vmlinux 如果只执行Make命令就会只生成

Vmlinux

vmlinux: $(vmlinux-lds) $(vmlinux-init) $(vmlinux-main) $(kallsyms.o) FORCE

这就是vimlinux依赖的文件

vmlinux-init := $(head-y) $(init-y)

vmlinux-main := $(core-y) $(libs-y) $(drivers-y) $(net-y)

vmlinux-all := $(vmlinux-init) $(vmlinux-main)

vmlinux-lds:=arch/$(ARCH)/kernel/vmlinux.lds

vmlinux-init := $(head-y) $(init-y)

head-y := arch/arm/kernel/head$(MMUEXT).o arch/arm/kernel/init\_task.o

init-y := init/

init-y := $(patsubst %/, %/built-in.o, $(init-y))

init 目录下的所有涉及的文件会编译成init/built\_in.o

vmlinux-main := $(core-y) $(libs-y) $(drivers-y) $(net-y)

core-y := usr/

core-y += kernel/ mm/ fs/ ipc/ security/ crypto/ block/

core-y := $(patsubst %/, %/built-in.o, $(core-y))

各子文件文件夹下的内容都会单独编译成built-in.o的文件

libs-y = lib/lib.a lib/built-in.o

drivers-y=:=drivers/built-in.o sound/built-in.o

net-y :=net/built-in.o

源文件如何编译成内核

make uImage v=1

v=1 会详细的列出编译过程

arm-linux-ld -EL -p --no-undefined -X

-o vmlinux

-T arch/arm/kernel/vmlinux.lds

arch/arm/kernel/head.o arch/arm/kernel/init\_task.o init/built-in.o --start-group usr/built-in.o arch/arm/kernel/built-in.o

arch/arm/mm/built-in.o arch/arm/common/built-in.o arch/arm/mach-s3c2410/built-in.o arch/arm/mach-s3c2400/built-in.o

arch/arm/mach-s3c2412/built-in.o arch/arm/mach-s3c2440/built-in.o arch/arm/mach-s3c2442/built-in.o

arch/arm/mach-s3c2443/built-in.o arch/arm/nwfpe/built-in.o arch/arm/plat-s3c24xx/built-in.o

kernel/built-in.o mm/built-in.o fs/built-in.o ipc/built-in.o security/built-in.o

crypto/built-in.o block/built-in.o arch/arm/lib/lib.a lib/lib.a arch/arm/lib/built-in.olib/built-in.o drivers/built-in.o sound/built-in.o net/built-in.o --end-group .tmp\_kallsyms2.o

第一个文件为 arch/arm/kernel/head.s

1. 内核启动过程分析

比较机器ID， 判断是否支持这个cpu,判断是否支持这个单板

adr r3, 3b

3b意思是向前找到标号3的位置，然后读取该内存地址的内容，存到r3中

.arch.info.init在 Include/asm-arm/mach/arch.h的宏MACHINE\_START中定义

#define MACHINE\_START(\_type,\_name) \

static const struct machine\_desc \_\_mach\_desc\_##\_type \

\_\_used \

\_\_attribute\_\_((\_\_section\_\_(".arch.info.init"))) = { \

.nr = MACH\_TYPE\_##\_type, \

.name = \_name,

#define MACHINE\_END \

};

Arch/arm/Mach/smdk2410.c

MACHINE\_START(S3C2410, "SMDK2410")

/\* Maintainer: Ben Dooks <ben@fluff.org> \*/

.phys\_io = S3C2410\_PA\_UART,

.io\_pg\_offst = (((u32)S3C24XX\_VA\_UART) >> 18) & 0xfffc,

.boot\_params = S3C2410\_SDRAM\_PA + 0x100,

.init\_irq = s3c24xx\_init\_irq,

.map\_io = smdk2410\_map\_io,

.init\_machine = smdk2410\_machine\_init,

.timer = &s3c24xx\_timer,

MACHINE\_END

代入后得到结构体，存储了uboot传输的参数

static const struct machine\_desc \_\_mach\_desc\_SMDK2410

\_\_used \

\_\_attribute\_\_((\_\_section\_\_(".arch.info.init"))) = {

.nr = MACH\_TYPE\_s3c2410,

.name = "SMDK2410",

.phys\_io = S3C2410\_PA\_UART,//IO空间的开始地址

.io\_pg\_offst = (((u32)S3C24XX\_VA\_UART) >> 18) & 0xfffc,

.boot\_params = S3C2410\_SDRAM\_PA + 0x100,

.map\_io = smdk2410\_map\_io, //实现对IO空间的映射

.init\_irq = s3c24xx\_init\_irq, //对中断进行初始化

.init\_machine = smdk2410\_init,

.timer = &s3c24xx\_timer,

};

解析uboot传入的启动参数

建立页表，

使能Mmu

arch/arm/kernel/head.s 该文件末尾包含了head-common.S"

ENTRY(stext)

执行start\_kernel

b start\_kernel head-common.S (arch\arm\kernel

asmlinkage void \_\_init start\_kernel(void) /init/main.c

start\_kernel()调用函数rest\_init();

setup\_arch()//解析uboot传入的启动参数

setup\_command\_lind //解析uboot传入的启动参数

rest\_init()

rest\_init()创建内核线程kernel\_thread(kernel\_init, NULL, CLONE\_FS | CLONE\_SIGHAND);并调用kernel\_init（）

kernel\_init（）调用prepare\_namespace()

prepare\_namespace()调用mount\_root()挂载根文件系统

kernel\_init调用init\_post()

init\_post()调用以下函数执行应用进程

run\_init\_process("/sbin/init");

run\_init\_process("/etc/init");

run\_init\_process("/bin/init");

run\_init\_process("/bin/sh");

挂接根文件系统、执行第一个应用程序

/init/do\_mounts()

prepare\_namespace（）

if (saved\_root\_name[0]) {

root\_device\_name = saved\_root\_name;

/init/do\_mounts()

static int \_\_init root\_dev\_setup(char \*line)

{

strlcpy(saved\_root\_name, line, sizeof(saved\_root\_name));

return 1;

}

//解析命令行参数，查找到含有root=的参数，将=号后面的内容存入到saved\_root\_name中

\_\_setup("root=", root\_dev\_setup);

#define \_\_setup(str, fn) \

\_\_setup\_param(str, fn, fn, 0)

#define \_\_setup\_param(str, unique\_id, fn, early) \

static char \_\_setup\_str\_##unique\_id[] \_\_initdata = str; \

static struct obs\_kernel\_param \_\_setup\_##unique\_id \

\_\_attribute\_used\_\_ \

\_\_attribute\_\_((\_\_section\_\_(".init.setup"))) \

\_\_attribute\_\_((aligned((sizeof(long))))) \

= { \_\_setup\_str\_##unique\_id, fn, early }

构建根文件系统

Uboot :启动内核

内核 ：启动应用程序

构建根文件系统

init\_post（）启动应用程序中有

sys\_open((const char \_\_user \*) "/dev/console", O\_RDWR, 0)//串口0 控制台，生成一个文件描述符

sys\_dup(0) //复制一个文件，生成一个文件描述符

sys\_dup(0) //复制一个文件，生成一个文件描述符

printf scanf stderr 对应三个文件描述符，标准输入0，标准输出1，标准错误2，都指向串口

if (execute\_command) {

run\_init\_process(execute\_command);

}

init\_setup(char \*str)

{

execute\_command = str;

}

\_\_setup("init=", init\_setup);

与bootargs=noinitrd root=/dev/mtdblock3 init=/linuxrc console=ttySAC0

run\_init\_process("/sbin/init");

run\_init\_process("/etc/init");

run\_init\_process("/bin/init");

run\_init\_process("/bin/sh");

/sbin/init到busybox的连接

Inint 进程

1 读取配置文件

2 解析配置文件

3 执行用户程序

Busybox 🡪

init\_main()

Parse\_inittab

File=fopen(INITAB,”r”)INITTAB ="/etc/inittab"打开配置文件

new\_init\_action（）

Init\_tab的格式

<id>:<runlevels>:<action>:<process>

配置文件 1、指定程序，何时执行

id=> /dev/id ,用作终端，stdin,stdout,stderr printf,scanf,err

runlevels :可以忽略

action :指示何时执行

process：应用程序或脚本

new\_init\_action(ASKFIRST, bb\_default\_login\_shell, "VC\_2");

ASKFIRST = 0x004

bb\_default\_login\_shell

bb\_default\_login\_shell[] ALIGN1 = LIBBB\_DEFAULT\_LOGIN\_SHELL;

LIBBB\_DEFAULT\_LOGIN\_SHELL "-/bin/sh"

VC\_2 "/dev/tty2"

new\_init\_action(0x004, "-/bin/sh", ""/dev/tty2"");

static void new\_init\_action(int action, const char \*command, const char \*cons)

创建一个Init\_action结构，填充

把这个结构放入Init\_action\_list链表

new\_init\_action(CTRLALTDEL, "reboot", "");

：：ctrlaltdel:reboot

new\_init\_action(SHUTDOWN, "umount -a -r", "");

::shutdown:umount –a -r

new\_init\_action(RESTART, "init", "");

::restar:init

new\_init\_action(ASKFIRST, bb\_default\_login\_shell, "");

::askfirst:-/bin/sh

new\_init\_action(ASKFIRST, bb\_default\_login\_shell, VC\_2);

tty2::askfirst:-/bin/sh

new\_init\_action(ASKFIRST, bb\_default\_login\_shell, VC\_3);

tty3::askfirst:-/bin/sh

new\_init\_action(ASKFIRST, bb\_default\_login\_shell, VC\_4);

tty4::askfirst:-/bin/sh

new\_init\_action(SYSINIT, INIT\_SCRIPT, "");

::sysinit:/etc/init.d/rcS

run\_actions(SYSINIT);

waitfor(a,0); //执行应用程序，等待它执行完毕

delete\_init\_action(a)//从

run\_actions(WAIT);

run\_actions(ONCE);

while(1)

{

run\_actions(RESPAWN);

run\_actions(ASKFIRST);

wpid=wait(NULL)//等待子进程退出

while(wpid>0) //退出后，就设置pid=0

{

a->pid=0;

}

}

action 指定程序运行的时机，取值有以下8种

action>: Valid actions include: sysinit, respawn, askfirst, wait, once,

# restart, ctrlaltdel, and shutdown.

/dev/consol /dev/null

/etc/inittab

配置文件里指定的应用程序

库

Init本身就是busybox 最小根文件系统

构建根文件系统busybox

\busybox-1.7.0\busybox-1.7.0\INSTALL中描述了如何编译busybox