嵌入式系统内核与文件系统实验

戴文谦
141180014

目录

1 实验环境介绍

1.1 硬件环境

硬件开发环境即为嵌入式实验系统开发板 x210v3。本次实验用到的硬件资源如下：

- 核心处理器：S5PV210，主频 1GHz
- 内存：512MB
- RS–232 接口 (UART2)
- RJ45 接口 (有线以太网 DM9000CEP)
- 7 英寸电容触摸屏[1]

[1]本实验仅用到其显示功能

- 功能按键 (开机/复位)
- Flash:4GB nand-Flash

1.2 软件环境

软件环境主机端主要有以下配置：

- Ubuntu 桌面环境（包含 minicom 工具）
- busybox 1.20.2 文件系统
- linux 内核 2.6.35/3.0.8 版本
- 针对 ARM 的交叉编译工具链 2011.08/2016.08 版本[1]

在嵌入式开发板 x210v3 端，厂商预装了底层的 bootloader 和 Android4.0.4 系统。

2 实验原理说明

2.1 嵌入式系统基本开发原理

嵌入式系统实验中，基本开发原理是：在 PC 上通过交叉编译工具 toolchain 生成二进制文件并通过嵌入式系统与主机的通信或外置储存载体写入其中后运行。

目标板的基本引导方式如下：上电启动 → 加载 bootloader 并进行相关硬件的初始化 → 与主机通信进行配置和传输内核、文件系统等 → bootloader 根据环境变量引导启动后续系统 → 成功加载目标板上的系统内核和文件系统，高级设备驱动 → 在目标机系统上运行嵌入式应用程序。

Minicom 是 linux 下的串口通信工具，本实验利用 minicom 实现主机与嵌入式系统系统之间的通信以及主机对嵌入式系统的控制。运用 minicom 软件，对 minicom 进行基本设置后，即可利用基于 RS–232 串口标准的的 UART 接口在主机和目标机之间进行初级引导通信。

实际上，串口通信的通信速率很低，按照 minicom[2]的波特率设置，传输一个 5MB 左右大小的内核就需要花 6 分钟以上的时间，效率极低，故在主机与目标机的后续文件传输等通讯中，改用目标机支持的有线以太网通信是必须的。本实验中，主要用到了 tftp 文件传输传输协议和 NFS 共享文件服务系统协议。

tftp 协议主要用于在目标机 bootloader 加载完成后的通信中。在 bootloader 阶段，目标机会默认采用 tftp 协议并作为客户机[3]。在此实验中，tftp 协议主要用于向开发板内存写入内核和基于 ramdisk 文件映像的文件系统。

通过主机 linux 系统架设的 NFS 服务器[4]可以方便的将主机的共享文件夹挂载在目标板上，从而在目标板上的 linux 系统上方便地使用主机上的远程文件。在目标板上的 Linux 启动后，若内核和文件系统中都支持 NFS 系统功能，且网络配置与连接正常，NFS 服务即可正常使用。

[1]包含以 GNU 工具：gcc compiler，binutil，C library，C header

[2]速率为 115200bps = 115200/1024/8KB/s = 14.0625KB/s

[3]主机的 tftp-server 已经配置好

[4]主机的 NFS 服务（portmap, NFS）都已配置好

2.2 内核系统原理

Linux 内核是后续嵌入式开发的基础，没有刷入开发板的 linux 内核则无法实现后续的在开发板上运行嵌入式软件的工作。

配置 Linux 内核所需要的源代码已经提供，本实验中有两个版本可以选择：2.6.35 版本和 3.0.8 版本，新版内核具有一些新特性、驱动、问题修复。内核源代码主要包括的内容有：

- 体系结构相关的代码[1]
- 内核函数实现代码、内核初始化代码
- 内存管理代码、文件系统代码、网络与进程通信相关代码
- 内核头文件和库文件代码
- 驱动程序代码
- 配置内核的脚本文件

只有源码而不经编译的内核是无法在机器上运行的。编译时，通过 make 命令读取 Makefile，调用交叉编译工具链，得出目标文件后打包成核心和压缩的核心映像文件。嵌入式开发板加载 bootloader 成为 tftp 协议的目标机后，即可将在主机中生成的内核映像刷写到开发板中。

为了在内核中实现功能模块的取舍和基本的配置选项、编译方式等参数，通过 make 命令编译内核前，需要通过依赖于 config.in 的配置工具[2]来产生或更改内核配置文件.config。配置工具中有多个主要可选分支项：

- Genernal setup
- System type
- Networking Options
- Device Drivers
- File systems

以上分支项将在之后的实验过程涉及时进一步说明。

在内核配置完成后，通过 make 命令，可以编译内核。在编译内核的过程中，内核顶层的 Makefile 将读取配置工具产生的.config 文件，并根据.config 文件中的设置来调用子文件夹中的 Makefile 以编译相关功能对应的代码。

[1] 本机用到 arm/s5pv210

[2] make menuconfig

2.3 文件系统原理

嵌入式开发的文件系统是在嵌入式储存设备上命名、放置、管理文件的系统。

实际上，不同于 Windows 系统[1]，Linux 下的文件系统分为 ext 系列，NFS，FFS 系列等，他们的区别在于组织文件的形式和底层的管理文件方式不同。实验中主要涉及以下几种类别的文件系统:

1. ext 文件系统：是 linux 普通磁盘上最常见的文件系统，目前主要有 ext2fs，ext3，ext4，与 windows 的典型不同在于支持快速符号链接，ext3 版本开始支持用于恢复文件的日志

2. 网络文件系统 NFS：可以通过 mount 命令将远端的文件系统挂载在本机的/mnt 文件夹下，从而方便的使用网络上其它计算机的文件

3. FLASH 文件系统：以 JFFS2、YAFFS2 文件系统为代表，是专门针对闪存设计的文件系统，特点是采用“追加式”文件系统，对 FLASH 的使用寿命损耗较小。

在实验中还有一种特殊的 Ramdisk，是将预先制作好的映像载入内存中，用内存的一部分空间来模拟一个与映像内里完全相同的硬盘分区。这类文件系统在组织文件的方式上由对映像格式化的程序来决定[2]，但由于对 ramdisk 的修改本身不会同步到映像，内存断电后又会失去数据，此类 Ramdisk 在系统关机后只能保留原有映像的内容，故也可将 Ramdisk 作为一种不同的文件系统。

Linux 的根文件系统通常包含以下几个按文件系统规范储存相应文件的文件夹：[3]

bin dev etc lib mnt proc sbin

制作根文件系统采用的是 Busybox 软件。Busybox 是一个继承了各种系统基本命令、配置、工具的软件，适用于嵌入式系统。交叉编译完成后，软件能产生一个基于 busybox 主程序组成的根文件系统，加入基本的库文件、启动文件和设备文件后即可制作成嵌入式开发板上 Linux 系统的根文件系统。

3 实验过程与内容

3.1 嵌入式系统基本开发实践

设置开发环境：编译程序用交叉编译工具链为 arm-2016.08 版。为方便调用编译工具，在终端设置以下环境变量：

```
$ export PATH=/opt/arm-2016.08/bin:$PATH
```

此后可在任意路径下直接调用编译器。编写一个简单的 C 程序 1234.c：

```
/* 1234.c */
#include <stdio.h>
```

[1]Windows 系统从改用 NT 内核开始采用 NTFS 文件系统，早期使用的是 FAT(16)、FAT32

[2]在系统中会显示为 ext/yaffs2 等

[3]以实际制作出的文件系统为例

```
int main(int argc, char* argv[])
{
    printf("Hello world!!\n");
    return 0;
}
```

在同一目录下编译程序 1234，作为后续尝试在开发板 Linux 系统上运行的程序，并将其拷贝至主机 NFS 共享目录[1]下待用：

```
$ arm-linux-gcc -o 1234 1234.c
$ cp 1234 /srv/nfs4/1234
```

为保证主机和从机能够通信且不与实验室其它电脑冲突，设定本机 IP 地址：

```
$ ifconfig eth0 192.168.208.27
```

运行 minicom，进行串口的初始化设置：输入 Ctrl+A-o，设置通信口为/dev/ttyS0，波特率为 115200bps，数据位 8 位，无奇偶校验，一个停止位。配置完成后选择保存配置并返回 minicom 主界面。minicom 配置后，将从主机引出的串口线接到开发板的 UART2 接口上，网线接到 RJ45 接口上，电源线接到供电插孔上，确认无误后才按下开发板的开机键。

上电后，主机和开发板会通过串口通信，minicom 主界面上会出现开发板的启动信息。出现“autoboot”字样时通过键盘干预停止加载后续内核工作，进入 bootloader 的交互模式。作为第一次操作，应该配置 bootloader 的环境变量，本机按如下设置：

```
x210 # setenv ipaddr 192.168.208.127
x210 # setenv serverip 192.168.208.27
x210 # setenv gatewayip 192.168.208.254
x210 # setenv netmask 255.255.0.0
x210 # setenv ramdisk root=/dev/ram rw initrd=0x40000000,8M
x210 # setenv bootargs console=ttySAC2,115200 $ramdisk
x210 # saveenv
```

输入下列命令将位于主机 tftp 文件夹[2]下已经编写好的内核和 ramdisk 文件系统[3]刷入开发板的内存中并启动该内核：

```
x210 # tftp 0x30008000 zImage2.6
x210 # tftp 0x40000000 ramdisk_img.gz
x210 # bootm 0x30008000
```

minicom 显示大段系统启动过程消息后，出现 root 用户文件系统中提供的欢迎信息并进入命令行 sh，即启动系统成功。接下来配置开发板上运行的 Linux 系统的网络功能并将主机的 NFS 共享文件夹挂载到开发板上的 Linux 系统：

[1] /srv/nfs4

[2] /srv/tftpboot/

[3] 由主机 root 用户提供

```
# ifconfig eth0 192.168.208.127
# mount 192.168.208.27:/srv/nfs4 /mnt -o nolock,proto=tcp
```

配置成功后，输入以下命令：

```
# cd /mnt
# ls
```

显示1234，说明网络配置正确且 NFS 服务正常运行。

再输入./1234，显示如下：

```
Hello world!!
```

目标板上的 Linux 系统成功运行主机交叉编译生成的 C 语言程序。

3.2 Linux 内核编译与配置

实验主机上有 Linux2.6.35 和 3.0.8 两个内核版本可选，本次试验选择 2.6.35 版本。为在主机中将源文件交叉编译为 arm 处理器的内核，需要在终端中设置环境变量：

```
$ export ARCH=arm
```

进入内核所在文件夹，首先需要载入适合 x210v3 开发板的默认配置文件[1]，再进入配置编译内核的界面：[2]

```
$ make x210ii_initrd_defconfig
$ make menuconfig
```

在图形界面中对以下几个选项进行配置：

1. Genernal setup：选择内核配置选项和编译方式等
   (a) Cross-compiler tool prefix：指定交叉编译工具的编译器。此处填入/opt/arm-2011.08/bin/arm-linux-[3]
   (b) Initial RAM filesystem and RAM disk (initramfs/initrd) support：指定内核是否支持使用 ramdisk，内核实验中必须依赖 ramdisk，故此处将此项选上。
   (c) Initramfs source file(s)：本实验中 ramdisk 会单独刷入，故应将此项置空。[4]
   (d) Optimize for size：生成体积较小的内核，嵌入式开发中一般选上此项减小内核体积。

[1] 手工配置相当繁琐

[2] 对于 3.0.8 版本为 make x210_initrd_defconfig

[3] 内核与普通编译的程序不同，只能使用 2011.08 版本的编译器

[4] 原来有填入 ramdisk 映像

(e) Configure standard kernel features (for small systems)：为嵌入式系统设计的选项，可以微调内核的一些功能。嵌入式开发中一般将此项选上，选中后才会出现选择性支持压缩 ramdisk 镜像的选项。

(f) Support initial ramdisks compressed using gzip：是否支持 gzip 压缩后的 ramdisk 映像，本实验选上此项。

2. Enable loadable module support：允许不常用的功能以模块化形式放在内核外，需要用到时再加载。嵌入式实验中一般选择此项，可以减少内存占用和内核大小。

3. System Type：设置处理器架构等相关选项，载入相应的默认配置文件后一般不需要手工调整，本实验不涉及该选项。

4. Networking support：网络配置选项

(a) Networking options：基础网络功能配置，不含 wifi，红外等功能的调节选项。

i. TCP/IP networking：使内核支持 TCP/IP 网络协议[1]。不选此项网络功能会受到极大限制，为了配置网络实现 NFS 功能，此处选上此项。

ii. IP: kernel level autoconfiguration：此功能用于在内核启动阶段就对 IP 地址自动配置，采用 Ramdisk 内存映像时，可以在系统启动后手工配置，暂不选上此项。

5. Device Drivers：驱动程序设置

(a) Generic Driver Options：驱动程序全局设置

i. Maintain a devtmpfs filesystem to mount at /dev：此项用于产生/dev 文件夹下的设备文件。选择此项，内核将在加载阶段就会自动产生设备文件并保留。[2]一般开发中都选上此项。

ii. Automount devtmpfs at /dev, after the kernel mounted the rootfs：此项用于在内核挂载根文件系统后将自动将产生的设备文件挂载到/dev 文件夹中，依赖于上一项的选择。一般在选上上一项时也会选上此项。

(b) Network device support：网络设备驱动支持

i. Ethernet：以太网驱动支持，实际上开发板向外的网路即为局域以太网，此处为了配置网络验证 NFS 功能选上此项。

6. File systems：与文件系统相关的设置，将在文件系统实验中详细说明。[3]

(a) Second extended fs support：Ext2 文件格式支持，由于后续刷入的文件系统映像是 Ext2 格式，此处需要选上。

(b) Network File Systems：网络文件系统相关设置

[1]在因特网和局域网上广泛使用，另一常用协议为 bootloader 使用的 UDP

[2]若文件系统下的/dev 包含了设备工作所需的设备文件则不需选择此项, 但选上此项依然可以加速启动

[3]见3.3

i. NFS client support：支持挂载 NFS 文件系统，为验证 NFS 功能此处选上此项。

退出配置界面后，提示 configuration written to .config，保存设置成功。调用实验主机的八线程 CPU[1]来加快编译内核：

```
$ make -j8
```

提示编译成功后，将编译成功的内核 zImage 找出并复制到 tftp 文件夹下，再进入 minicom 刷入编译后的内核和 ramdisk，可以成功启动，证明内核制作成功。[2]

3.3 创建嵌入式文件系统

根文件系统的主要组成来源于交叉编译 busybox，设置环境变量[3]后，通过 make menuconfig 命令进入 busybox 的编译设置

- Busybox Settings
  1. Build Options
     (a) Build BusyBox as a static binary(no shared libs)：对 busybox 程序执行静态编译。[4]希望系统能执行交叉编译的文件故不选此项选择加入相应的动态链接库。[5]
     (b) Cross Compiler prefix：交叉编译器设置。由于先前设置了环境变量，可直接填入编译器名称：arm-linux-
  2. Installation Options ("make install" behavior)
     – (./_install) BusyBox installation prefix：指定 BusyBox 输出文件系统所在的文件夹，这里取默认值。

配置完成后，输入以下命令，将在`./_install` 文件夹生成文件系统的雏形：

```
$ make
$ make install
```

若在内核中没有配置产生设备文件[6]，则还需要在 dev 目录下建立必要的设备：

```
$ mknod console c 5 1
$ mknod null c 1 3
$ mknod zero c 1 5
```

[1]Core i7 桌面 CPU 为 4 核心，利用超线程技术虚拟出 8 线程
[2]详细操作见3.1
[3]详细操作见3.2
[4]若只需要 busybox 提供的基本操作功能，可选择此项而不加入动态链接库
[5]相关操作见3.3
[6]见3.2 5(a)ii

进入内核编译选项界面[1]，对以下几项进行设置：

1. IP: kernel level autoconfiguration[2]，若需要配置 NFS 作为根文件系统则需要选上此项否则不会出现相关选项。

2. File systems：

   (a) Second extended fs support[3]

   (b) Ext3 journalling file system support：Ext3 文件系统支持

   (c) The Extended 4 (ext4) filesystem：Ext4 文件系统支持[4]

   (d) Miscellaneous filesystems：其他文件系统格式[5]

      i. YAFFS2 file system support：YAFFS2 文件系统支持，这种文件系统适用于现在常见的 NAND flash。如要访问嵌入式系统自带的 flash 选择此项。

      ii. Journalling Flash File System v2 (JFFS2) support：JFFS2 文件系统支持，一般用于 NOR flash，现在已经比较少见。

   (e) Root file system on NFS[6] ：选择此项则支持 NFS 作为根文件系统启动。

根据需要制作的文件系统选择相应的选项后编译内核。
接下来要补全根文件系统使其能够正常运行：

- 建立系统初始化必须的文件和启动脚本：[7]

```
$ vim etc/inittab
# /etc/inittab
::sysinit:/etc/init.d/rcS
::askfirst:-/bin/sh
::once:/usr/sbin/telnetd -l /bin/login
::ctrlaltdel:/sbin/reboot
::shutdown:/bin/umount -a -r
$ vim etc/rc
#!/bin/sh
hostname x210
mount -t proc proc /proc
/bin/cat /etc/motd
chmod +x etc/rc
$ vim etc/motd
Welcome to ARM-LINUX SYSTEM!
```

[1]详细操作见3.2
[2]详细位置见3.2 4(a)ii
[3]介绍见3.2 6a
[4]见2.3 1
[5]嵌入式系统一般存储载体为 flash 上，故主要关注 flash 相关文件格式 2.3 3中有介绍
[6]实际上是位于3.2 6(b)i之下的选项
[7]省略部分 vim 内操作提示

```
$ cd etc
$ mkdir init.d
$ cd init.d
$ ln -s ../rc rcS
```

- 建立 proc 空目录
- 建立运行程序所需动态链接库的 lib 目录，将交叉编译器下的几个库文件 ld-2.13.so，libc-2.13.so，libm-2.13.so 复制进去并建立符号链接：

```
$ ln -s ld-2.13.so ld-linux.so.3
$ ln -s libc-2.13.so libc.so.6
$ ln -s libm-2.13.so libm.so.6
$ ln -s ld-2.13.so ld-linux-armhf.so.3
```

对于单独刷入目标机内存的文件系统，需要制作文件系统的映像文件，以制作 Ext2 文件系统映像为例：[1]

```
$ dd if=/dev/zero of=ramdisk_img bs=1k count=8192
$ mke2fs ramdisk_img
$ mount ramdisk_img
$ cp -r /_install/* /mnt/ramdisk
$ umount /mnt/ramdisk
$ gzip ramdisk_img
```

将内核和 ramdisk 刷入开发板即可工作。

将 NFS 作为根文件系统，则实际上的根文件系统位于 Ubuntu 主机上的 NFS 文件夹中，故需要在 minicom 中更改 bootloader 对内核的传递参数才能正常启动：

```
$ cp -r /_install/* /srv/nfs4/nfslinux
$ minicom
x210 # setenv rootfs root=/dev/nfs rw nfsroot=192.168.208.27:/srv/nfs4/
nfslinux
x210 # setenv nfsaddrs nfsaddrs=192.168.208.127:192.168.208.27:192.168.
208.254:255.255.0.0
x210 # setenv bootargs console=ttySAC2,115200 $rootfs $nfsaddrs
x210 # saveenv
```

只需要刷入内核，启动后开发板会自动载入位于主机上的根文件系统。

还可以选择制作包含于内核中的 ramdisk，理论上只需要将先前制作的文件系统打包即可，但实际操作中包含于内核的 ramdisk 根目录下没有 *init* 无法启动，故应进行以下操作：

[1] mke2fs 为使用 ext2 文件系统格式化，改用 mkfs.ext3 等命令则可使用 ext3 文件系统与其他文件系统格式化

```
$ cd _install
$ ln -s /sbin/init init
$ find ./ -print |cpio -H newc -o |gzip -9 > ~/ramdisk.cpio.gz
```

再进入内核配置界面中，将”Initramfs source file(s)” 一项设置为`~/ramdisk.cpio.gz`[1]，重新编译即可得到包含文件系统的内核，单独刷入该内核即可正常运行系统。

虽然本实验没有直接制作一个 YAFFS2 文件系统，但可以对已有的该种文件系统进行探究：嵌入式开发板的 flash 即采用该种文件系统。无论先前采用哪种根文件系统配置方式，只要内核中配置了对 YAFFS2 文件系统的支持[2]并开启了自动产生设备文件选项[3]，便可以采取以下操作：

```
# mount /dev/mtdblock2
# mount
# chroot /mnt/mtdblock2
# ls /
```

终端上的显示说明 mtdblock2 设备实际上是一个可读写的，格式为 yaffs2 的磁盘。同时观察根目录下文件结构可知该磁盘内实际上存放的就是厂商预装的安卓系统。实际上，这种改变根目录的方法在嵌入式系统的应用上相当常见，可以通过精简的 ramdisk 文件系统启动 Linux 后，再将格式为 yaffs2 的 flash 作为后续运行系统的大型根文件系统。这样做，可以依靠 flash 的大容量，容纳后续程序的开发，载入内核模块化后的多项可选功能。

4 实验讨论与小结

4.1 部分实验问题讨论

* *研究 NFS 作为根文件系统的启动过程。*

实际上这是探究 bootloader 引导启动内核和根文件系统的问题，下面就尝试对比分析 bootloader 环境变量对启动过程的影响：

1. 开发板自带安卓系统的启动

   如按以下方式设置，不跳过 bootloader 的 autoboot 阶段可以自动启动安卓系统：

   ```
   x210 # setenv bootargs console=ttySAC2,115200 root=/dev/mtdblock2 rw
   rootfstype=yaffs2 init=/linuxrc
   ```

2. ramdisk 的启动方式[4]

   如按以下方式设置，直接输入 boot 命令而无需指定 bootm 地址即可启动该种文件系统下的内核：

[1] 具体位置见3.2 1c

[2] 见3.3 2(d)i

[3] 见3.2 5(a)ii

[4] 默认已做过3.1中的设置

```
x210 # setenv bootcmd bootm 0x30008000
```

3. NFS 文件系统下的启动方式[1]

   对比以上两种启动方式 bootloader 的环境变量，可知 bootloader 的 bootargs，bootcmd 参数决定了开发板的启动方式。

   NFS 文件系统对环境变量的设置等价于下列语句：

```
x210 # setenv bootargs console=ttySAC2,115200 root=/dev/nfs rw
nfsroot=192.168.208.27:/srv/nfs4/nfslinux 192.168.208.127:
192.168.208.27:192.168.208.254:255.255.0.0
```

   同时，根据设置界面的逻辑可知：还需要选择“内核启动阶段 IP 自动配置”选项才可能支持 NFS 文件系统作为根文件系统启动。

通过以上分析，对 NFS 文件系统的启动概述如下：

将 UART2 串行接口上的另一端作为主机控制端；将 nfs 设备文件作为启动后的根目录，可读可写；指定采用 nfs 文件系统；内核根据 bootargs 传入的网络参数自动配置好 IP 参数后，将网络上主机 nfs 文件夹下的指定目录作为实际上的根目录启动该文件系统。

而 bootcmd 参数主要决定了 bootloader 默认启动参数 boot 执行的命令内容，由于内核还是通过 tftp 方式传到开发板的内存中，故此项设置为 bootm 0x30008000 即可进行 NFS 文件系统的启动。

4.2 实验小结

本次实验对嵌入式系统的基本开发原理进行了了解，并为后面的实验打下了基础：实现了可用的内核、文件系统并将其刷写到 x210v3 开发板中，提供了后续的嵌入式程序运行环境。

通过本次实验实际上了解了以下内容：

通过 Ubuntu 桌面环境的几次操作，熟悉了不同于 Windows 系统的 Linux 系统，掌握了一些基本命令与操作方法。在 Linux 和 bootloader 相关基础知识储备下，学会了运用 minicom 程序分别通过串口和以太网在主机和开发板上进行通信。

利用自动化工具 Make 和内核源码，交叉编译工具链，根据所需功能编译了适用于嵌入式系统的 Linux 内核，对内核无用的功能做出了删减或模块化，掌握了编译内核的方法和大量内核配置参数、内核功能的含义。

构建了 Linux 根文件系统并以传统的 Ext 格式，nfs 网络格式进行实现，探究了 bootloader 的启动过程和相应传递参数的意义。

初步了解了将 ramdisk 作为引导文件系统启动开发板 Linux 后转入 flash 上 YAFFS2 文件系统的方法及其对后续开发工作中的意义。

[1]默认已做过3.3中的设置