文章编号: 1000-5013(2011)02-0161-04

AdHoc网络在嵌入式Linux上的实现

王磊, 谢维波

(华侨大学 计算机科学与技术学院, 福建 泉州 362021)

摘要: 分析和研究嵌入式Linux系统开发中的关键技术问题及解决方法. 在S3C2410处理器上成功移植适用于AdHoc网络的嵌入式Linux系统,并实现AdHoc网络的AODV路由协议. 网络性能的测试表明,所移植的Linux系统各部分运行稳定可靠,AdHoc网络运行正常.

关键词: ARM处理器; 嵌入式Linux; 无线自组织网; AODV协议

中图分类号: TP 393 **文献标志码:** A

嵌入式系统是以应用为中心,以计算机为基础,软硬件可裁剪,适用于系统对功能、可靠性、成本、功耗严格要求的专用计算机系统. AdHoc无线移动自组织网是一种特殊的对等式网络,使用无线通信技术,网络中的节点互相作为其邻居(在直接通信范围内的节点)的路由器,通过节点转发,实现节点之间的通信,因此又被称为多跳网络. AdHoc网络具有无中心、自组织、可快速展开、节点可移动和多跳等特点[1]. 这些特点使得它在战场、救灾等特殊场合的应用日渐受到人们的重视. 因此,在嵌入式系统上实现AdHoc网络更能发挥它的作用. 本文主要描述构建适用于AdHoc网络的嵌入式系统环境,成功移植AODV路由协议及在此基础上实现的AdHoc网络.

1 嵌入式Linux系统的移植

在移植之前首先要建立交叉开发环境,安装交叉编译工具[2]. 目标平台采用适用于PDA等手持设备和Internet设备的,韩国Samsung公司基于ARM920T内核开发的一款嵌入式处理器S3C2410,其Linux内核采用较新的Linux 2.6.24内核. Bootloader,Linux内核和文件系统共同构成了基本的、最小的嵌入式Linux系统.

1.1 配置编译引导装载程序vivi

Bootloader是在操作系统内核运行之前运行的一段小程序. 它初始化硬件设备、建立内存空间的映射图,从而建立适当的软硬件环境,为调用(运行)操作系统内核做好准备. Bootloader是基于特定硬件平台的. 采用韩国MIZI公司开放的源代码vivi,主要用于S3C2410处理器的开发板引导程序Bootloader. vivi的运行分为以下2个阶段[3]:

(1) 第1阶段的完成包含依赖于CPU体系结构的硬件初始化代码,包括禁止中断、初始化串口、复制自身到RAM等,最后跳到Bootloader的下一阶段(main);

(2) 第2阶段的完成包括8个部分,即完成打印出vivi的版本、对开发板进行初始化、内存映射初始化和内存管理单元的初始化、初始化堆栈、初始化mtd设备、初始化私有数据、初始化内置命令、启动内核或进入命令行状态.

在移植vivi的过程中,要修改开发板的内存设置. 在smdk.c文件中,数组default_mtd_partitions定义了存储器空间的划分. 图1是vivi启动后的默认分区. 通过part命令进行mtd分区,但重置后需要恢复默认. 分区的地址是引导程序、内核映像及文件系统下载到NandFlash的真正地址. 修改smdk.c

收稿日期: 2009-09-03

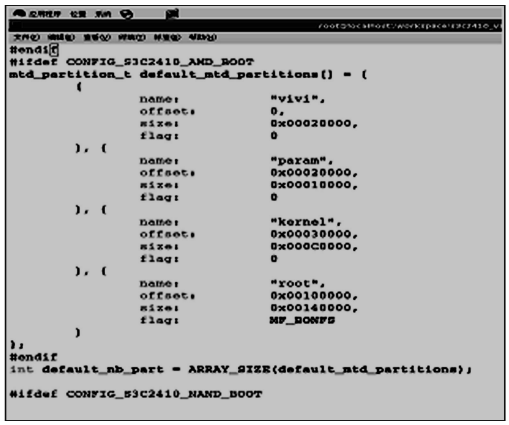
通信作者: 谢维波(1964-),男,教授,主要从事信号与信息处理、嵌入式技术的研究. E-mail:xwblxf@hqu.edu.cn.

基金项目: 福建省厦门市科技计划项目(3502Z20083047)

中内核启动参数为

char linux_cmd[] = "initrd root = /dev/mtdblock2 init = /linuxrc console = ttySAC0,115200N8 mem = 64M".

参数为 vivi 启动后的默认值,也可用 parm set 命令进行修改,是传递给内核的启动参数,其他参数可以默认. 交叉编译 vivi 成功后会生成一个 bin 文件,用 Jtag 下载到 NandFlash 中,启动目标板,并按着空格键,进入 vivi 命令行.



```
#endif
#ifdef CONFIG_S3C2410_NAND_BOOT
mtd_partition_t default_mtd_partitions[] = {
        {
                name:           "vivi",
                offset:         0,
                size:           0x00020000,
                flag:           0
        }, {
                name:           "param",
                offset:         0x00020000,
                size:           0x00010000,
                flag:           0
        }, {
                name:           "kernel",
                offset:         0x00030000,
                size:           0x000C0000,
                flag:           0
        }, {
                name:           "root",
                offset:         0x00100000,
                size:           0x00140000,
                flag:           MF_MOWFS
        }
};
#endif
int default_nb_part = ARRAY_SIZE(default_mtd_partitions);

#ifdef CONFIG_S3C2410_NAND_BOOT
```

图 1 vivi 的 Flash 分区

Fig. 1 Flash partition of vivi

1.2 配置编译 Linux 内核

Linux 2.6 内核的编译与 Linux 2.4 版本的内核的编译有些不同. Linux 2.6 内核的编译,只需要 make 命令,代替了 make dep,make zImage 和 make modules. Make menuconfig 配置内核后,只要运行 make 命令就行. 编译前要对内核代码进行一些修改,主要有如下 6 个步骤.

(1) 修改顶层的 Makefile 文件. ARCH = arm CROSS_COMPILE = /usr/local/arm/3.4.1/bin/arm-linux.

(2) 修改内核 MTD 分区. 内核 MTD 分区必须与 vivi 的分区相一致. 因为 vivi 分区中的地址是引导程序、内核映像及文件系统下载到 NandFlash 的真正地址. 内核启动时,内核并不是去读 vivi 分区中的地址,而是去读内核 MTD 分区设定的地址. 所以,如果两者不同,将导致不能正常启动内核或根文件系统. 需要注意的是,Linux 2.6.16(含)以前的内核没有分区信息,需自己添加;Linux 2.6.17(含)以后的内核文件中已含有分区信息,只需修改 arch/arm/mach-s3c2410/common-smdk.c 文件里的函数"mtd_partition smdk_default_nand_part[] = {}",使其地址与 vivi 中的分区地址一致.

(3) 禁止 Flash ECC 校验. 内核都是通过 Bootloader 写到 NandFlash 的. Bootloader 通过软件 ECC 算法产生 ECC 校验码,与内核校验 ECC 码不一样. 内核中的 ECC 码是由 S3C2410 中的 NandFlash 控制器产生的. 所以,要禁止内核 ECC 校验. 修改 s3c2410.c 代码中"chip->ecc.mode = NAND_ECC_NONE".

(4) 支持启动时挂载 devfs. 从 Linux 2.6.12 内核起,devfs 选项从内核配置中删除. 为了让内核支持 devfs 及在启动时/sbin/init 运行之前能自动挂载/dev 为 devfs 文件系统,应修改 fs/Kconfig 文件,如图 2 所示. 在文件 fs/Kconfig 中找到 menu "Pseudo filesystem"语句,然后在其后添加图 2 中所示语句.

```
config DEVFS_FS
        bool " /dev file system support(OBSOLETE)"
        defaulty
config DEVFS_MOUNT
        bool " Automatically mount at boot"
        defaulty
        depends on DEVFS_FS
```

图 2 修改 fs/Kconfig 文件

Fig. 2 Modify the fs/Kconfig file

(5) 匹配 machine ID. vivi 传递给内核的 machine ID 必须与内核代码中的 ID 一致. 系统中的 ID 为 193,其他处理器的值定义在/arch/arm/tools/mach-types 文件中.

(6) 内核配置. 在配置内核时,对需要的功能进行配置和去除不需要的模块,以此来减小内核体积. 运行 make menconfig 进行配置,在 s3c2410_defconfig 默认配置基础上进行裁减. 需要加入 MTD 分区及 NandFlash 支持、文件系统 ramdisk 支持,对于 AdHoc 网络,还需要加上无线网络、无线网卡 zd1211b 驱动、USB 驱动、模块可加载及 NetFilter 等支持.

1.3 制作根文件系统

虽然 Linux 系统的核心是内核,但文件却是用户与操作系统交互的主要工具. 根据 FHS 描述 Linux 根文件系统的目录结构,来创建嵌入式 Linux 根目录结构. 创建一个伪根目录,在其中/bin,/sbin 下存放各种可执行程序,在/etc 下存放配置文件,在/dev 目录下存放常见必要的设备节点等. 系统使用 Busybox 来创建可执行文件,若 Busybox 使用动态链接,还要在/lib 目录下包含库文件.

主要关键的细节有:负责旧版本的设备管理系统 devfs 已被废除,因此是从 Linux 2.6.18 开始的. 新版本的 udev(系统中使用的是 Busybox 提供的 mdev)是一个基于用户空间的设备管理系统,在内核

启动时并不能自动创建设备节点，需要手动在 dev/目录下创建各种节点，即设备文件.

涉及系统启动过程的设备有/dev/mtdblock * (MTD 块设备)，/dev/ttySAC * (串口)，/dev/console(控制终端)，/dev/null，只要建立以上设备就可以启动系统. 其他设备可以当系统启动后，使用“cat/proc/devices”命令查看内核中注册了哪些设备，然后一一创建相应的设备文件. 设备号可从内核源代码的 Documentation/devices. txt 中获得. 其他配置文件的编写不再赘述.

2　移植 AODV 路由协议

基于嵌入式 Linux 协议栈，在嵌入式 ARM-Linux 环境下实现 AODV 路由协议. 目前多数已经发布的 AODV 实现都利用了 Netfilter 功能框架. Netfilter 是 Linux 2. 4 内核实现报文过滤、报文处理、报文转发等的功能框架.

采用的 AODV 协议是开源的 aodv-uu-0. 9. 5，其实现方法是：AODV 协议作为用户层后台程序实现的，包括两个可装载的 Linux 内核模块(kaodv 和 ip_queue_aodv)；使用 Netfilter 来截获本地外及本地内的报文，但它是在用户层运行的. Kaodv 模块使用 Netfilter 通过返回 NF_QUEUE 来缓存用户层的所有报文，ip_queue_aodv 对用户层报文进行排队. 将所有报文的目的地址与用户层路由缓存进行匹配，在用户层缓存需要路由请求的报文，并立刻返回已存在路由的报文[4].

Aodv 路由协议移植即将 aodv-uu-0. 9. 5 移植到 ARM-Linux 中，主要有以下几个方面.

(1) 内核的配置要求. 直接编译到内核中的，也可以编译成模块 ip_queue. ko，然后挂载.

(2) 修改 Makefile. 修改内核的版本号，将 KERNEL＝＄(shell uname-r)改为 KERNEL＝2. 6. 24；修改内核的源码路径，文中的内核源码路径为/adhoc/linux-2. 6. 24，即将 KERNEL_DIR＝/lib/modules/＄(KERNEL)/build 改为 KERNEL_DIR＝/adhoc/linux-2. 6. 24.

(3) 修改参数. Linux 2. 6 的内核中的函数 ip_route_me_harder 多了 1 个参数，所以在调用的时候要增加 1 个参数. 即将 ip_route_me_harder(skb)改为 ip_route_me_harder(skb, RTN_UNSPEC).

(4) 编译后，将在当前目录下生成 aodvd 的可执行文件，在 lnx 目录中将生成 kaodv. ko 模块.

3　组建 AdHoc 无线网络及网络性能测试

使用 802. 11b/g 无线网卡 ZD1211b 作为传输设备，Linux 2. 6 内核版本加入了对该网卡的支持. 设置 essid 和 rate，运行 aodvd，将无线网卡改成 AdHoc 模式. 网络采用 1 台 PC 机作为 A 节点(192. 168. 1. 3)，两台目标平台分别作为节点 B(192. 168. 1. 5)和节点 C(192. 168. 1. 7). 其网络的拓扑结构，如图 3 所示.

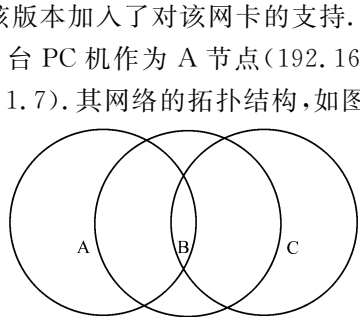


图 3　网络的拓扑结构

Fig. 3　Topology of network

3. 1　连通性测试

使用 Linux 操作系统提供的 Ping 命令来完成该项测试. Ping 命令提供了“-R”选项，可以记录下 Ping 数据包所经过的所有节点的地址. 具体有如下 3 个主要步骤.

(1) 同时打开所有节点的 AODV 功能.

(2) 检查 A 节点的路由表. 确定存在到 B 节点的路由，而没有到 C 节点的路由. B 节点的路由是由于 B 节点发送“HELLO”消息来确定的.

(3) A 节点与 C 节点通信. 通过向 C 节点发送 ping 命令来发送数据分组. 经过短时间的延迟，得到 C 节点的回复. 此时检查路由表，可以发现增加了到 C 节点的路由，A 的下一跳是 B 节点. 开始时，节点 A 没有去往节点 C 的路由，必须先发送 RREQ，启动路由发现，节点 C 收到 RREQ 后，返回 RREP(由节点 B 转发给节点 A)，由此建立路由 A-B-C.

在节点 A 执行“Ping R 192. 168. 1. 7”，结果如图 4 所示. 从图 4 可以看到，节点 A 发送给节点 C 的数据包所经过的路径是 A→B→C→B→A. 由此可知，经过 B 节点上的 AODV 模块的路由功能将 Ping 数据包转发，使得不能直接通信的节点 A 和节点 C 实现了数据传输，验证了本系统的 AODV 模块可以提供数据多跳传输的功能.

3.2 性能测试

在A节点上设置FTP服务器,通过Ping命令测试端到端的延迟和路由发现时间,通过FTP下载A节点的文件以测试吞吐量.

(1) 端到端的延迟和路由发现时间[5]. Ping包可以记录往返时间RTT(Round Trip Times),从而反映端到端的延迟.使用Ping工具来测试单链N跳Ad Hoc网络的RTT值,$N=1,2$. 测试结果表明:性能跳数1(B),2(C)端到端的延迟时间分别为1.6,3.1 ms,而其路由发现时间分别为4.8,17.5 ms.

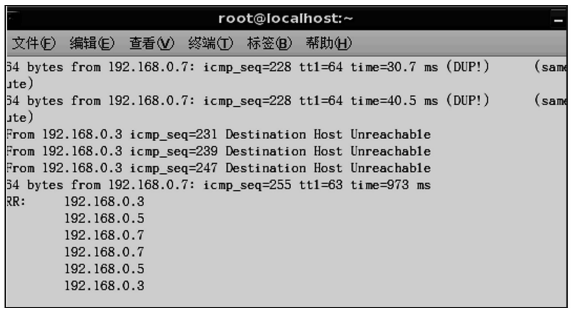


图4 连通性测试结果

Fig. 4 Results of connectivity testing

(2) 端到端的吞吐量.测试吞吐量的时候,当从节点C上通过节点B中转来下载节点A上的ftp文件时,在多个时刻进行测试,每次传输时间大概10 min左右. 性能跳数1(B),2(C)的吞吐量的多次测量结果,其平均值分别为150,56 kB·s^{-1}.

4 结论

目前,针对Ad Hoc网络路由协议的研究多是在仿真的环境中进行的. 但是,由于物理层和协议的某些不恰当、不精确的模型不能正确和真实地反映节点的移动性,因此在真实的环境中实现和评估路由协议是十分必要的. 更何况这也是Ad Hoc网络实际应用的必由之路.

在ARM处理器S3C2410上构建适用于AdHoc网络的嵌入式Linux系统,在所构建的嵌入式Linux系统上实现AdHoc网络的AODV路由协议,组建具有自主知识产权的嵌入式AdHoc网络. 同时,针对S3C2410处理器,提出嵌入式Linux系统开发中的关键技术问题及解决方法,对于AdHoc网络的实现和应用,以及基于嵌入式Linux的应用开发都具有一定的参考意义.

篇幅有限,文中仅指出系统实施过程中的关键技术细节,而省略了具体的命令步骤.下一步要做的工作是,对AODV代码进行优化及对AdHoc网络的安全性保障.

参考文献:

[1] 郑少仁,王海涛,赵志峰,等. Ad Hoc网络技术[M]. 北京:人民邮电出版社,2005.
[2] 宋凯,严丽平,甘岚,等. ARM Linux在S3C2410上的移植[J]. 计算机工程与设计,2008,29(16):4138-4140.
[3] 杨水清,张剑,施云飞,等. ARM嵌入式Linux系统开发技术详解[M]. 北京:电子工业出版社,2008.
[4] 刘焕敏. Linux平台Ad Hoc网络按需路由协议实现技术研究[D]. 长沙:国防科学技术大学,2005.
[5] 洪家军,吴金龙.利用NS-2实现Ad Hoc网络仿真平台[J]. 华侨大学学报:自然科学版,2008,29(3):375-378.

Implementation of AdHoc Network Based on Embedded Linux

WANG Lei, XIE Wei-bo

(College of Computer Science and Technology, Huaqiao University, Quanzhou 362021, China)

Abstract: The key technologies and solutions of development on the embedded Linux system are presented in this paper. An embedded Linux system applicable for the AdHoc network is transplanted on the ARM S3C2410 processor, and the AODV protocol of AdHoc network is implemented based on that. Testing on the Network performance shows that the Linux system is steady and the AdHoc network runs normally.

Keywords: ARM processor; embedded Linux; wireless ad hoc network; AODV protocol

(**责任编辑**:陈志贤 **英文审校**:吴逢铁)