嵌入式Linux系统小型化技术

介绍了Linux在嵌入式领域中的应用和宿主机、目标机开发模式，详细地给出了精简内核的实现过程。分析了glibc系统库和ELF文件格式的结构和其中的共享库裁剪技术的原理，提出并实现了一种库裁剪方案。

关键词 嵌入式；Linux；小型化

一、 概述

嵌入式Linux一般是指对标准Linux发行版本进行小型化裁剪处理之后，适合于特定嵌入式应用场合的专用 Linux操作系统。嵌入式系统通常是资源受限的系统，无论是处理器计算能力还是RAM或其他存储器容量都比较“小”。因此，如何创建一个小型化的 Linux作为操作系统开发成为首先需要考虑的问题。嵌入式Linux系统中普遍采用三层结构：核心层主要是Linux内核和模块；调用接口层是以glibc库为主的系统库；应用层是根据用户需求设计的应用程序。为了实现资源的高利用率，后两层都以ELF文件形式存在，在运行过程中对外部功能代码动 态加载。

一般来说，建立交叉平台开发环境是进行嵌入式软件开发的第一步。宿主机与目标机硬件平台的异构(处理器体系结构不 同)是采用交叉开发的根本原因。另外，由于资源有限，直接在嵌入式系统的硬件平台上开发软件不方便、甚至不可能。因此，通常采用Host／Target开 发模式，如表1。

	宿主机(Host)	目标机(Target)
硬件	PC 或者工作站，其中x86CPU占优势	嵌入式系统硬件，处理器多样化(x86，ARM，PowerPC，MIPS，68K等)
软件	Windows、Linux等桌面操作系统，丰富的集成开发环境(如WindRiver 的	软件资源有限，开发阶段通常从宿主机下载

	Tornado)	

表1 交叉平台发环境的特点

交叉平台开发环境包括交叉编译器、交叉调试器和系统仿真器，比如嵌入式Linux开发经常用的GNU工具链。开发者 需要根据目标平台来选择合适的GNU交叉编译器，然后在宿主机上面重新编译内核和其他软件，这样得到的目标代码才能拿到目标机上面运行。这个过程相当繁琐 且容易出错。宿主机和目标机一般通过以太网或者串口连接。目前，世界上出现了数以百计的嵌入式Linux开发计划和发行版本，比 如：ETLinux，LPR， μC-Linux，ThinLinux等开发源代码的项目，如表2所示。

名称	特点
ETLinux	设计用于在小型工业计算机，尤其足PC／104模块上运行
Linux Router Project	LPR 的目标是用于路由器、接入服务器、瘦服务器等网络没备和嵌入式系统，可以安装在一张软盘上。类似的项目还有Linux On A Floppy(LOAF)
μC-Linux	在没有MMU 的系统L运行的Linux。同前支持Motorola DragonBall (M68EZ328)，M68328，M68EN322， ColdFire， QUICC，ARM7TDMI，MC68EN302，Axis ETRAX，Inte]i960，PRISMA，Atari 68k等微处理器
ThinLinux	一个为嵌入式和特定应用制作的Linux发行版，运行在Intel和PC兼容硬件上

表2 几种开放源代码的嵌入式Linux发行版

另外，还有：Coventive XLinux，LineoEmbedix，LynuxWorks BlueCat，MontaVista Linux等商业公司的发行版。同时，针对实时环境，有RT-Linux、RTAI等实时扩展。近年来，越来越多的目标系统选择了性价比不断提高的x86 处理器和成熟的PC架构作为硬件平台。LinuxDevices.com网站进行的调查显示，嵌入式系统开发者在过去2年和未来2年选择x86处理器作为 目标平台的比例分别为31%

和35%，高居首位。

对于宿主机和目标机都是PC兼容平台的开发者来说，除了沿用上述模式之外，有更简单的创建小型化Linux 系统的方法：以一个常规的Linux发行版为基础，编译内核、复制所需的文件，并利用初始化RAM盘(initrd：INITial Ram Disk)机制创建根文件系统，就可以快速实现一个小型化Linux系统。

二、 小型化技术

Linux已经越来越广泛地应用于各种嵌入式设备中。但是一般的Linux发行版都非常庞大，很难用于只有有限存储 空间的嵌入式设备。所以我们必须对Linux系统进行裁剪。Linux系统大致有以下4种主要的裁剪技术，使用这些技术可以有效地减小系统的尺寸且不会影 响系统的性能。① 删除冗余文件。一般的Linux发行版中都包含很多帮助文档、辅助程序、配置文件和数据模板，在嵌入式系统中这些文件都是不必要的，完全可以删除。甚至连 配置文件中的大量注释也都可以被去掉。② 共享库裁剪。嵌入式系统的应用程序是有限的，共享库中就可能有很多永远不会被用到的冗余代码，这些代码就可以被删除。③ 采用具有同样功能的替代软件包。Linux上有许多具有相似功能的软件包，可以选择其中占存储空间较小的软件包并其移植到嵌入式设备上，用来代替原来占空 间较大那些的软件包。④ 修改源码。包括重新配置、编译软件包，去掉不需要的功能；增加软件的模块性，从而有利于提高裁剪效率；重新配置内核，去掉不需要的驱动和模块。

1、精简内核

与传统嵌入式操作系统的微内核(Micro-kernel)体系结构不同，Linux内核采用的是整体式结 构(Monolithic)，整个内核是一个单独的、非常大的程序。其优点是能够使系统的各个部分直接沟通，有效地缩短任务之间的切换时间，提高系统响应 速度。缺点也是明显的，即内核尺寸比较大，因为Linux内核不仅包括如任务调度、内存管理、中断处理等基本的操作系统功能，同时还包括文件系统、网络协 议、没备驱动程序等功能。

Linux 内核是高度模块化、可配置的，通过配置使内核具有不同的功能，从而减小内核的大小。例 如，Linux 支持的文件系统种类很多，包括 ext2、ext3、FAT、Reiserfs、JFS 等。可以根据实际情况选择所需的文件系统，比如仅仅把 ext2 文件系统编译进内核。编译内核的主要步骤如下(“#”代表命令提示符)：

```
# cd／usr／src／linux-2.4
# make menuconfig
# make dep；make clean；make bzlmage
```

编译成功的内核文件为 arch／i386／boot／bzlmage。具体方法参考内核源代码包中的 README 文件。为了进一步增加灵活性、减小内核尺寸，Linux 还提供了可加载内核模块机制，内核中的很多功能可以编译为模块，在内核运行时动态加 载，而不是直接编译进内核。然而在嵌入式 Linux 系统中更倾向于根据需要编译一个独立的内核，较少使用模块机制。这样得到的内核通常在几百 kB 甚至 1MB 左右，相对传统的嵌入式操作系统内核来说是比较大的(比如包含文件系统和网络支持的 VxWorks 内核大约 250kB)。在进行内核配置时，开发者要比较了解各功能模块之间的依赖关系，否则有可能造成编译失败。而在 VxWorks 内核的配置过程中，如果破坏了依赖关系，有比较明确的指示，从而避免这 种错误。

2、共享库裁剪

在小型化技术中，共享库裁剪容易用软件实现，做成自动裁剪工具，效果最明显。下面重点介绍共享库小型化技 术，共享库小型化的基本思想是通过提取和解析系统库内目标文件、符号的依赖关系，通过对这些依赖构造关系模型进行关系演算，根据应用程序中的符号信息，在 库目标文件一级实现系统库的小型化. 实现上分为四步：a、确定待调函数集。在 ELF 文件内部，存在一个 E1f32-Sym 数组结构的符号表，用于内部符号定义和外部符号引用，通过对这个符号表的分析可以将 ELF 应用程序中待调符号(系统函数)抽取出来，从而建立一个应用程序 -待调函数符号的多对多关系。b、确定系统库函数与目标文件的对应关系。系统库逻辑上分成：库、目标文件、符号三个层次，库和目标文件都是 ELF 格式，通 过对库的映像文件*_pic.a 和每个目

标文件中的符号表分析得到库。目标文件的定义关系、目标文件-符号定义关系和目标文件-符号调用关系。c、确定系 统库目标文件之间的相互依赖关系。通过对步骤b中相关关系的关系演算得到目标文件-目标文件的完全依赖关系。d、生成小型化系统库。通过对应用程序-待调 符号表和目标文件-目标文件依赖表的关系演算得到待调函数所依赖的目标文件集合，将它们进行重新链接即可得到最小化的库文件。

2.1、共享库裁剪技术的原理

共享库中保存着预先编译好的目标代码，一般是被应用程序反复使用的公用代码。在Linux系统中，应用程序 与库之间可以静态链接或动态链接。静态链接时，链接器从库中选取应用程序需要的代码，然后复制到生成的可执行文件中。显然，当静态库被多个程序使用时，磁 盘上、内存中都是多份冗余拷贝。动态链接时，链接器并不真的把库代码复制到可执行文件中；仅当可执行文件运行时，加载器才检查该库是否已经被其它可执行文 件加载进内存，如果内存中不存在才从磁盘上加载该库。这样多个应用程序就可以共享库中的代码的同一份拷贝，节约了存储空间。这也是嵌入式Linux系统使 用共享库的主要原因。

当使用静态链接库时，链接器会自动地只把库中被使用的模块链接到可执行文件中。但是这种方法没有用在共享库中，主要是因为在应用程序执行之前链接器并不知道应用程序最终用到了库中的哪些部分。因此要对共享库进行裁剪必须先分析动态链接的原理。

共享库和可执行文件中都有若干个符号表，其中定义了一些外部符号，分为导出(export)符号和导入 (import)符号这两种。导出符号是指在该文件中定义但可以被其它文件使用的符号，一般是可以由其它文件调用的函数；导入符号是指被该文件使用了但并 没有定义的符号，一般是被该文件调用的函数，而且导入符号一般指明了定义该符号的共享库。加载器在加载可执行文件或共享库之前会先遍历它的每个导入符号， 检查该符号的相关代码是否已在内存中，否则先查找并加载定义该符号的共享库。由于嵌入式Linux系统中的应用程序和共享库一般都是确定的，共享库中就可 能存在永远不会被别的文件调用到的导出符号，将这些符号的相应代码从共享库中删除不会影响到系

统的正常运行。
现有裁剪技术都是以上述原理实现的。下面则具体分析它的实现方法。
2.2、ELF 文件符号提取
ELF 格式是 UNIX 实验室作为应用程序二进制接口而开发和发布的，ELF 是目前广泛应用于 Linux 系统中的一种文件格式。
2.2.1、 ELF 文件进程映像加载
ELF 文件开头部分是一个 ELF Header 结构，它包含两个指针，分别指向两个数组结构：Program header table 和 Section header table，Program header table 中的数组元素对文件内部的可执行代码段进行定位；Section header table 中的数组元素保存相关重定位和动态链接信息. 装载器通过控制这两类数组实现进程映像的加载。
2.2.2、 ELF 文件的符号表和重定位过程
ELF 文件的 Section header table 中有一个类型为 SHT_DYNSYM 的 Section，该 Section 记录了创建进程映像所需要的所有符号。
a、符号值确定和符号定位，ELF 文件中字符串 ection(. shstrtab)用于保存所有字符串，ELF 头通 过 e_shstrndx 域保存节头名字字符串表(.shstrtab)的节索引。ELF 文件中符号名字域值是.shstrtab 节的一个字符索 引：Syrnbol 结构中 St_name 对应相应的字符串表一个索引，在相应的字符串表中对应其符号值，St_value 对应两类不同地址：对于文件内部 定义符号，对应该符号内容的文件内部相对地址；对于外部调用符号，对应待调符号的地址(已解析)或重定位表中的一个入口(未解析)。St_info 保存符 号的类型和相应的属性。
b、被调符号重定位。符号表中，STT-SEC-TION 对应重定位入口信息表。重定位入口以数组的形式存在于 ELF 文件中，其中的 R_offset 保存着应用于重定位行为的地址，而 R_addend 对应一个偏移用于计算要存储于重定位域中的值。R_info 中给 出受重定位影响的符号索引和重定位应用的类型。例如：当类型为 R_386_JMP_SLOT 时，符号值就对应一个.plt(过程连接表)入口的位置.
c、外部符号装载。对于外部符号代码的装载，装载器通过 lazy MODE 装载方式将外部符号代码加载到进程映像中：首次调用外部符号通过 PLT[0]中的装载代码和 PLT[1]中出栈参数将待调符号代码加载到.got 表中；以后对此待调符号的调用通过对应.got 表入口进行控制传输。
2.2.3、 ELF 文件符号提取实现

对每个参与动态链接的共享目标文件来说，其程序头表(Program header table)有一个类型为PT_DYNAMIC的入口元素。该入口所指向的段.dynamic section是一个E1f32_Dyn的结构数组.E1f32_Dyn结构中有一个属性标志d_tag和一个联合结构d_un，d_tag控制d_un中的解释。数组中下标为DT_SYMTAB的入口指向符号表。通过对符号表、重定位表、过程连接表、全局过程表的相关控制结构进行分析，完成文件定义符号 和待调符号的分离提取，算法如下：

```
symtab=.dynamic[DT_SYMTAB]->d_un. d->ptr
//根据DT_SYMTAB找到符号表的地址
for(int i=0; symtab[i]!=NULL; i++)
//对符号表中所有入口进行扫描
{swith((symtab[i] ->st_info)>>4) //根据符号类型进行操作
}…
case STB_WEAK:
case STB_GLOBOL: /*对全局性的和弱符号可用于外部文件调用*/
if(对应入口指向过程连接表入口)/*如果其指向的地址为. plt入口则在创建进程映像的时候需要重定位*/
loadrequest(symtab[i]->st_name); /*重定位并将该符号的名字在字符表找到relocate(symtab[i]->st_value)相应的值，并将其名字放人相应的关系表*/
else
loadprovide(symtab[i]->st_name); /*如果是内部定义，那么此符号名是供其它的应用程序调用*/
… }}
```

ELF文件中定义符号和待调符号由其st_value所指的目标进行区分：对应于.plt表的待调符号需要重定位； 对应于内部符号，如果是弱类型(WEAK)或全局类型(GLOBOL)则用于其它文件调用。通过对ELF格式中的待调符号的提取，建立起应用程序和符号的 依赖关系。

2.3、嵌入式系统小型化结果与分析

对各个表进行连接得到应用程序依赖的目标文件集合。将集合中的目标文件无重复记录并重新连接从而得到最小化库。前后 库中各数据对比见表3。小型化后，系统库内各组成部分显著减少，库被缩减近50%。对于日益庞大的嵌入式系统中的应用程序，根据库文件内部的依赖关系，在 其基础上对应用程序进行优化裁减，对一般性应用可

以使系统库减小40%～50%。

	小型化前	小型化后
目标文件（个）	1183	544
符号（个）	9118	4861
显性依赖（条）	3819	1887
间接依赖（条）	120273	55425
库大小	1242480byte	685032byte

表3 系统库小型化前后对照

三、 结束语

近年来嵌入式Linux技术迅速发展，各种商业和开放源码的Linux发行版为不同的硬件平台、不同的应用环境提供 了多种选择。Linux的文件系统事实上非常的庞大。构造一个嵌入式的Linux文件系统是一个很复杂的过程。如何让文件系统在保证安全的前提下精简得更紧凑、运行得更有效率，是需要深入探索的一个课题。特别是，共享库裁剪技术能将库中大部分冗余代码裁剪掉，但要求库的源码编写比较规范，不同体系结构需要 有不同的处理等。但毕竟库裁剪领域才发展不久，还不是很成熟。经过对该技术长时间的测试，相信我们能够弥补它的不足，使它能够在嵌入式Linux领域广泛 使用。通过以上方法，我们构造了一个精简的嵌入式版本的Linux文件系统，它使得内核在系统尽量精简的情况下能够运行起来. 并满足产品和系统各方面的要 求。