第 32 卷 第 9 期　　华 中 科 技 大 学 学 报(自然科学版)　　Vol. 32 No. 9
2004 年 9 月　　J. Huazhong Univ. of Sci. & Tech. (Nature Science Edition)　　Sep. 2004

嵌入式 linux 系统动态库小型化技术研究

阳富民　李文海　涂　刚

(华中科技大学 计算机科学与技术学院, 湖北 武汉 430074)

摘要: 分析了 glibc 系统库和 ELF 文件格式的结构,利用关系演算对 glibc 中的目标文件及其包含的符号进行解析,由此得到各个目标文件之间的依赖关系;通过这些依赖关系对应用程序所需要的符号进行分析,从而得到所需要的目标文件,在目标文件一级最大限度地完成对系统库的小型化.

关　键　词: 嵌入式; 小型化; 动态链接; 依赖关系

中图分类号: TP311.54　　**文献标识码**: A　　**文章编号**: 1671-4512(2004)09-0006-03

Technology of the dynamic libraries miniaturization for the embedded linux system

Yang Fumin　Li Wenhai　Tu Gang

Abstract: This paper analyzed the structure of glibc libraries and ELF file format. The relation calculus was used to parse the objective files and their symbols in the glibc library files; the dependencies in objective files were found. By analyzing the dependencies and the symbols that the applications required, the required objective files are found, the miniaturization of system library at the objective file level is achieved.

Key words: embedded; miniaturization; dynamic link; dependency relation

Yang Fumin　Prof.; College of Computer Sci. & Tech., Huazhong Univ. of Sci. & Tech., Wuhan 430074, China.

1　基于关系模型小型化流程

嵌入式 linux 系统中普遍采用三层结构[1]: 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. 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(见图 1):

a. 确定待调函数集. 在 ELF 文件内部,存在一个 Elf32_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.

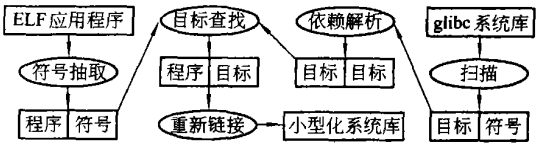

图 1　动态库小型化流程图示

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[1].

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.

收稿日期: 2003-09-04.

作者简介: 阳富民(1966-),男,教授;武汉,华中科技大学计算机科学与技术学院 (430074).

E-mail: lwhaymail@21cn.com

基金项目: 国家高技术研究发展计划资助项目 (2001AA13519002).

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,将他们进行重新链接即可得到最小化的库文件.

2　ELF 文件符号提取

ELF 格式[①]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.

2.1　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[①]和 Section header table[②], 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　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 文件中字符串表 Section (. 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[①]: Symbol 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[①];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[①]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_SECTION[①]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[②]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.

2.3　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 Elf32_Dyn 的结构数组. Elf32_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_GLOBAL:  /*对于全局性的和弱符号可以用于外部文件调用 */
      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,并将其名字放入相应的关系表 */
   else
        loadprovide(symtab[i] -> st_name);
        /*如果是内部定义,那么此符号名是供其他的应用程序调用 */
   … }}
```

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

① Toll Interface Standard(TIS) Executable and Linking Format(ELF) Specification. Version 1.2. TIS Committee, May 1995.

② Steve Chamberlain and Roland Pesch, Using ld: The GNU linker: ld version 2. Cygnus Support, January 1994.

(WEAK)或全局类型(GLOBOL)则用于其他文件调用.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.

3 嵌入式系统库关系模型构造

GNU C library 即 glibc 由一系列函数库组成,其功能函数用以执行诸如输入/输出、内存管理、串操作等基本操作,并提供对系统调用的封装.库由若干目标文件链接而成,库的完整性由目标文件的依赖完整性决定.

3.1 目标文件的两类依赖

目标文件彼此的符号调用关系使目标文件之间存在依赖,这些依赖可分为两类,一类是显性依赖,即目标文件对另一目标文件中符号发生符号调用关系而引起的依赖;另一类是指目标文件的间接依赖.目标文件 A 中的符号α在动态加载时由于对目标文件 B 中的符号β需要进行引用从而导致 A 依赖于 B,B 由于对目标文件 C 中的符号γ需要进行引用从而依赖于 C,这样 A 就对 C 产生间接传递依赖.

3.2 系统库关系模型的实现

根据系统库的三层特征,在编译成初始库后对映像文件和目标文件进行扫描形成相关层次中元素的名字表和层次间元素之间的关系表,并对这些中间表进行操作,通过关系演算将目标文件之间的两类依赖抽取出来构成目标文件依赖关系表 (见表 1):

表 1 关系结构表

表名称	属性一	属性二	说明
Symbol	symID	symName	符号名表
Object	objID	objName	目标文件名表
Library	libID	libName	库文件名表
appDep	appID	symID	应用程序和符号的联系表
Provide	symID	objID	目标文件和其提供符号的联系表
Request	objID	symID	目标文件和其需要符号的联系表
Dependency	srcID	desID	目标文件之间的全依赖表
libInclude	libID	objID	库和其包含的目标文件的联系表

a. 对每个 *.pic.a 映像文件扫描,找出库所包含的目标文件,形成库名字表 Library、目标文件名字表 Object 和库-目标文件包含关系表 LibInclude;根据前面的 ELF 符号提取算法对每个目标文件进行扫描,记录所有定义符号形成 Symbol 表,再二次扫描得到 Provide 表和 Request 表(Provide 表用于存储目标文件内部定义的符号,Request 表用于存储目标文件需要引用的符号).

b. 对 Provide 和 Request 两个表进行等值连接,从而形成初始 Dependency 表,用于存储目标文件之间的显性依赖.

c. 间接依赖由于不是目标文件的直接符号引用导致的,需要通过关系表运算进行提取:构造临时表 impDep 和变量 num. impDep 用于记录所有目标文件的间接依赖;num 为 impDep 中的记录个数.循环执行下列过程:对 Dependency,Request 和 Provide 三个表的等值连接结果输出到 impDep、如果 num 不为零则将 impDep 加入 Dependency,并且将 impDep 和 num 清空;当 num 为零时删除 impDep,则得到的 Dependency 表就是要求的全依赖关系表.

4 嵌入式系统小型化结果与分析

对 appDep,Provide 和 Dependency 三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.

本测试平台硬件环境为 ep7312/CPU,8Mbyte/flash,16 Mbyte/RAM.在一个由 Microwindow 构成的图形系统中,针对其系统基本工具集 busybox,tinylogin,图形服务器 nano-X 以及其上面的一个小型浏览器对系统库做动态小型化,前后库中各数据对比见表 2.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 %.

表 2 系统库小型化前后对照

	小型化前	小型化后
目标文件(个)	1 183	544
符号(个)	9 118	4 861
显性依赖(条)	3 819	1 887
间接依赖(条)	120 273	55 425
库大小	1 242 480 byte	685 032 byte

参 考 文 献

[1] 毛德操,胡希明. Linux 内核源代码情景分析(第一版). 杭州:浙江大学出版社,2001.