文章编号:2095-476X(2014)05-0056-04

引导工具 GRUB 2 的模块开发分析

黄道颖[1], 连建永[1], 张安琴[2], 陈慧[1], 张安琳[1]

(1. 郑州轻工业学院 计算机与通信工程学院, 河南 郑州 450001;
2. 中国建设银行 江苏省分行, 江苏 南京 210002)

摘要:通过对一个 GRUB 2 模块源代码的编写、编译,到模块的加载、运行和卸载,分析 GRUB 2 的模块结构及开发过程,阐明了 GRUB 2 将功能分布在众多小模块中且在运行时能够动态加载和卸载的模块化设计机制. 结果表明,该机制便于随时扩展功能,进行用户二次开发.

关键词:GRUB 2;模块化设计;可执行连接格式(ELF)文件

中图分类号:TP316 **文献标志码**:A DOI:10.3969/j.issn.2095-476X.2014.05.013

Analysis of module programming for boot-loader GRUB 2

HUANG Dao-ying[1], LIAN Jian-yong[1], ZHANG An-qin[2], CHEN Hui[1], ZHANG An-lin[1]

(1. *College of Computer and Communication Engineering, Zhengzhou University of Light Industry, Zhengzhou* 450001, *China*;
2. *Jiangsu Branch, China Construction Bank, Nanjing* 210002, *China*)

Abstract: By analyzing the process of writing and compiling a simple GRUB 2 module source code, and loading, running, unloading a module, the article described the structure and the process of developing of GRUB 2 module, and clarified the modular design of GRUB 2: it put functions into many small modules, and those modules could be dynamically loaded and unloaded. The analysis showed that GRUB 2 could be redeveloped to consummate its functions and extend its application occasions.

Key words: grand unified boot-loader 2(GRUB 2); modular design; executable and linkable format (ELF) file

0 引言

随着 Linux 操作系统的普及,它的默认引导启动器 GRUB 也越来越受到关注. GRUB 是一个多操作系统引导器,适用于多种体系结构. 在 GRUB Legacy 阶段,它就吸引了众多研究人员,衍生了 GRUB4DOS,GHOST 等产品,目前已由 GRUB Legacy 发展到了 GRUB 2. 经过完全重写的 GRUB 2 相较于 GRUB Legacy,在编码规范和体系结构方面做了重大改进,更加规范、安全、健壮和强大,可以为使用者提供更大的灵活性和性能改进的空间.

与 Linux 的 LKM(loadable kernel module)机制[1]类似,GRUB 2 采用了模块化设计,这大大方便了其功能的扩展,又加上它是基于开源设计的,因而在操作系统的启动管理方面有较好的应用前景. GRUB 2 模块程序的设计与调试是基于类 Linux 的微内核环境[2],虽然 Linux 系统的模块化设计与运行机制已被讨论得很清楚,但有关其默认引导工具

收稿日期:2014-05-27

基金项目:河南省重点科技攻关项目(132102210418);郑州市科技计划资助项目(112PPTGY249-7);郑州轻工业学院研究生科技创新基金项目

作者简介:黄道颖(1967—),男,河南省信阳市人,郑州轻工业学院教授,博士,硕士研究生导师,主要研究方向为计算机网络、分布式计算系统.

GRUB 2 的模块化设计与运行机制的资料却很少,许多程序员对 GRUB 2 模块程序的基本结构不太了解,这对 GRUB 2 的进一步开发造成了诸多不便. 鉴于此,本文通过一个 GRUB 2 实例模块,从源代码的编写、编译,到模块的加载、运行和卸载,分析GRUB 2 模块的开发过程,以期为 GRUB 2 开发提供参考.

1　GRUB 2 模块程序的典型结构

图 1 为 GRUB 2 模块程序的基本结构. 初始化函数用于对功能函数的系统注册,随后功能函数才能以模块命令的形式被调用执行,命名为 GRUB_MOD_INIT( ),结束函数用于对功能函数的注销,释放空间,命名为 GRUB_MOD_FINI( ). 功能函数为程序主体,描述了模块程序完成的功能,通常以 grub_cmd_##name##命名[3].

```
#include<grub/types.h>
#include<grub/misc.h>
#include<grub/mm.h>
#include<grub/err.h>
#include<grub/dl.h>
#include<grub/extcmd.h>                          } 头文件
static grub_err_t
grub_cmd_hello (struct grub_extcmd
*cmd,int argc , char**args)                      } 功能函数
{return 0;}
static grub_extcmd_t cmd;
GRUB_MOD_INIT(hello)
{
cmd=grub_register_extcmd ( ) ;                   } 初始化函数
}
GRUB_MOD_FINI(hello)
{
grub_unregister_extcmd(cmd);                     } 结束函数
}
```

图 1　GRUB 2 模块程序的基本结构

笔者以一个简单的清屏模块的源代码 clear. c 为例,说明模块代码的基本结构.

```
#include < grub/types. h >
#include < grub/misc. h >
#include < grub/mm. h >
#include < grub/err. h >
#include < grub/dl. h >
#include < grub/extcmd. h >
#include < grub/term. h >
static grub_err_t
grub_cmd_clear ( struct grub_extcmd * cmd _attribute_( ( unused) ) ,
    int argc _attribute_( ( unused) ) ,
    char * * args _attribute_( ( unused) ) )
//功能函数,实现清屏操作
{
  grub_cls( ) ; //GRUB 2 中的清屏函数
  grub_printf( " The screen has been cleared! \n" ) ;    //证明执行命令,实现清屏
  return 0;
}
static grub_extcmd_t cmd;
GRUB_MOD_INIT( clear) //初始化函数
{
cmd = grub_register_extcmd( " clear" , grub_cmd_clear, GRUB_COMMAND_FLAG_BOTH," clear" ," clear the screen" ,0) ; //注册 clear 命令
}
GRUB_MOD_FINI( clear) //结束函数
{
//注销 clear 命令
grub_unregister_extcmd ( cmd) ;
}
```

上述清屏模块源代码的功能是实现当前屏幕内容的清除. 其中,grub_cmd_clear( )是完成模块的功能函数,GRUB_MOD_INIT(clear) 和 GRUB_MOD_FINI(clear) 用于初始化和释放.

关于源代码的 2 点说明: 1) "_attribute_ ((unused)) "表示该函数或变量可能不被使用,这个属性可以避免编译器产生警告信息; 2) GRUB 2 具有独立的库函数,但与 Linux 又有相近的表达,如 GRUB 2 中的 grub_printf 与 Linux 中的 printf 有相同的含义,可以在各自的系统实现相同的作用.

2　模块程序的编译

2.1　编译环境

上述模块程序是基于 GRUB 1. 97 beta4 版本编写,将其置于源文件中,与源文件包共同编译. 编译平台是 Ubuntu 13. 04,编译环境为 GCC 4. 1. 3 or later, GNU Make, GNU Bison 2. 3 or later, GNU binutils 2. 9. 1. 0. 23 or later, Other standard GNU/Unix tools, Ruby 1. 6 or later, Autoconf 2. 59 or later.

搭建编译环境后,开始对 GRUB 2 进行预编译[4]:

1) 编写的实例模块程序 clear. c 放进源文件包中,并对配置文件做出相应修改;

2) 运行"./configure",设置好系统参数;

3) 运行"make",编译源文件包;

4)运行“make install”,在系统中在线安装程序和数据.

在运行“make”之后,已经完成对源文件包的编译并生成相应的模块文件.

2.2 模块化程序实例 clear.c 的编译

将清屏程序置于包含这类命令的文件夹../commands/中(也可以放在其他位置),并同时修改../conf/common.mk 文件,增加如下内容:

```
# For clear.mod.
clear_mod_SOURCES = commands/clear.c
clean-module-clear.mod.1:
…
clear_mod_CFLAGS = $(COMMON_CFLAGS)
clear_mod_LDFLAGS = $(COMMON_LDFLAGS)
```

增加的内容是有关如何编译清屏模块的.这些内容是由 command.mk 中与 cat 相关的信息复制而来,简单将 cat 改变为 clear.其中 common.mk 为 makefile[5] 文件,文件中写明了如何编译源代码.如果此时执行至 make 命令,会进行对各个源文件的编译,其中有关 clear.c 的编译信息如下:

步骤 1)gcc-Icommands -I./commands -I. -I./include -I./include -Wall -W -Wall -W -Wshadow -Wpointer-arith -Wmissing-prototypes -Wundef -Wstrict-prototypes -g -Os -falign-jumps=1 -falign-loops=1 -falign-functions=1 -fno-dwarf2-cfi-asm -m32 -fno-stack-protector -mno-stack-arg-probe -fno-builtin -mrtd -mregparm=3 -m32 -MD -c -o clear_mod-commands_clear.o commands/clear.c

步骤 2)rm -f pre-clear.o

步骤 3)gcc -m32 -nostdlib -m32 -Wl,-build-id=none -Wl,-r,-d -o pre-clear.o clear_mod-commands_clear.o

步骤 4)nm -g -defined-only -P -p pre-clear.o | sed 's/^\([^]*\). */\1 clear/' > def-clear.lst

步骤 5)echo 'clear' > und-clear.lst

步骤 6)nm -u -P -p pre-clear.o | cut -f1 -d'' >> und-clear.lst

步骤 7)cat def-clear.lst /dev/null | mawk -f./genmoddep.awk und-clear.lst > moddep.lst

步骤 8)sh./genmodsrc.sh 'clear' moddep.lst > mod-clear.c || (rm -f mod-clear.c; exit 1)

步骤 9)gcc -I. -I./include -I./include -Wall -W -Wall -W -Wshadow -Wpointer-arith -Wmissing-prototypes -Wundef -Wstrict-prototypes -g -Os -falign-jumps=1 -falign-loops=1 -falign-functions=1 -fno-dwarf2-cfi-asm -m32 -fno-stack-protector -mno-stack-arg-probe -fno-builtin -mrtd -mregparm=3 -m32 -c -o mod-clear.o mod-clear.c

步骤 10)rm -f clear.mod

步骤 11)gcc -m32 -nostdlib -m32 -Wl,-build-id=none -Wl,-r,-d -o clear.mod pre-clear.o mod-clear.o

步骤 12)if test ! -z ""; then./clear.mod || (rm -f clear.mod; exit 1); fi

步骤 13)strip -strip-unneeded -K grub_mod_init -K grub_mod_fini -K _grub_mod_init -K _grub_mod_fini -R.note -R.comment clear.mod

最终产生了一个 ELF 文件格式的二进制文件 clear.mod,其文件属性如图 2 所示.

```
clear.nod:ELF 32-bit LSB relocatable,Intel 80386,versiom 1 (SYSV),not strlpped
```

图 2 clear.mod 文件属性

ELF 文件格式[6] 是一种可执行、可链接的文件格式,可用于二进制文件、可执行文件、目标代码等,是 Linux 的主要可执行文件格式.由于 ELF 文件格式的可扩展性和灵活性,GRUB 2 也应用 ELF 作为其标准文件格式.

2.3 模块文件的生成分析

模块程序的编译过程实际上是生成模块文件(.mod)的过程,下面具体对生成 clear.mod 的编译过程进行分析:

步骤 1)对源代码 clear.c 进行编译,得到中间文件 clear_mod-commands_clear.o.

步骤 2)和步骤 3)将上述中间文件进而生成模块的准备文件 pre-clear.o 并确保生成 pre-clear.o 之前没有同名文件.

步骤 4)将 pre-clear.o 定义的符号输出到文件 def-clear.lst.

步骤 5)和步骤 6)将 pre-clear.o 中的符号即模块程序涉及到的函数与 clear 共同生成 und-clear.lst 文件.

步骤 7)将 def-clear.lst 与 und-clear.lst 合成 moddep.lst 文件.生成的 moddep.lst 中,写入的是各个模块之间的依赖关系,其部分内容如图 3 所示.

由上可知,ls 模块依赖 extcmd 和 normal 2 个模块,而 extcmd 模块不依赖任一模块,它随着 GRUB 2 的启动自动加载;normal 模块依赖 boot 模块,而这个模块自动加载,normal 模块也随着系统的启动自

依赖模块	被依赖模块
ls	:extcmd normal
nomal	:boot
boot	:
extcmd	:

图 3　moddep. lst 中模块之间的依赖关系

动加载,从而 ls 模块也自动加载.

步骤 8)执行 genmodsrc. sh,参数是′clear′ moddep. lst,生成 mod-clear. c 文件,它保存了模块的名称和依赖关系.

步骤 9)将生成的 mod-clear. c 编译成格式为 ELF 的二进制文件 mod-clear. o,为生成 clear. mod 做好准备.

步骤 10)和步骤 11)是将之前生成的pre-clear. o 和 mod-clear. o,编译为模块文件 clear. mod.

步骤 12)和步骤 13)去除了 clear. mod 中的一些字符表等信息,生成最终的目标文件 clear. mod.

经过对编译过程的具体分析,可知一个模块文件的生成过程如图 4 所示.

3　模块程序的运行

3.1　模块的安装

将生成的 clear. mod 加载到 GRUB 2 系统有 2 种方法[7]: 1)将编译好的 GRUB 2 版本安装到主引导记录或分区引导记录,覆盖原来的 GRUB 版本,这样添加的实例模块会在新版本的 GRUB 2 中出现; 2)将 clear. mod 文件置于/boot/grub/目录下,并修改相应的配置文件 command. lst,增加 1 行:clear: clear.

command. lst 文件写入的是命令到模块文件的对应关系. 在 1 个模块文件中,可以定义 1 个或多个命令. command. lst 中部分命令与模块的对应关系如图 5 所示,模块 boot 定义了 1 个命令 boot,而模块 bsd 定义了 3 个命令:freebsd_loadenv,freebsd_module,freebsd_module_elf.

3.2　模块的加载、调用和卸载

事实上,用户不能直接访问模块中的函数,而是通过系统调用进入内核空间,然后由内核空间完成相应的工作[8]. 在 GRUB 2 中通常使用内嵌的命令 insmod 和 rmmod 来加载和卸载模块. insmod 开始加载模块过程,通过系统调用将模块二进制文件复制到内核空间,由内核完成加载过程. rmmod 卸载模块的过程与加载过程基本类似.

将模块加载到内核后,内核会维护 2 组有关模块的链表:模块链表与命令链表[9]. 在命令链表中,各命令以特定的数据结构按照名称依次存储命令的信息. 当应用程序去调用这个模块命令时,内核查找模块命令链表,找到与之对应的模块命令,并根据数据结构的各项定义,找到对应完成用户要求的函数,完成函数功能,如查找不到,返回错误信息.

在 GRUB 2 内核中可以用手工加载模块文件 clear. mod. 其方法是:重启系统,进入 GRUB 2 的命令行界面,在提示符使用 insmod 命令 grub > insmod clear 可以加载 clear 模块.

调用、运行 clear 命令:在 GRUB 2 的命令行界面输入:grub > clear. 界面会实现清屏:

The screen has been cleared!

grub >

类似地,如果卸载 clear 模块,则可使用 rmmod 命令:grub > rmmod clear.

4　结论

通过一个在 GRUB 2 中开发清屏模块的完整描述,分析了 GRUB 2 的模块结构及开发过程. 可以得出以下结论:

1)GRUB 2 下的模块程序由头文件、功能函数、初始化函数 GRUB_MOD_INIT( )和结束函数GRUB_MOD_FINI( )组成.

2)GRUB 2 模块功能的实现要经过程序的编译、ELF 文件的解析及命令的加载等过程.

3)当用户加载或者卸载模块时,会通过系统调用内核函数,将模块挂载到内核空间;接着内核解析模块程序,将模块中的命令加载到内核.

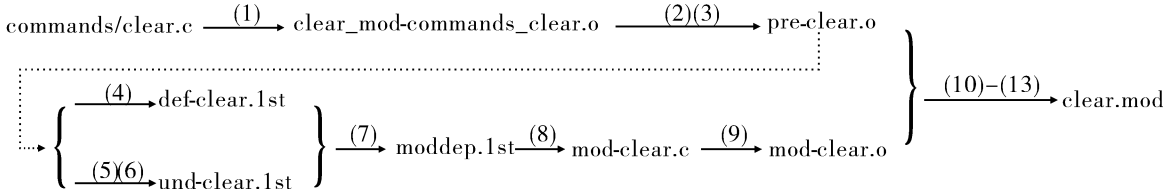

图 4　clear. mod 文件的生成过程

(下转第 67 页)

术将客户端的请求通过应用服务器送至数据库,并将结果返回给客户端,实现了物流管理中的货物管理、车辆调度等基本功能.

4　系统应用

本系统采用 4 台配有 5 块 2 T 容量的 SAS 硬盘的 Linux 服务器构成系统的 MFS 分布式存储系统,通过磁盘映射向数据库服务器映射 1 个分区;在上海实甲公司 SR - 5101 型开发板进行 RFID 超高频读卡器二次开发,其他方面选用了 XCAF - 12L 天线、深圳华为 GTM900GPRS 模块和 ISO18000 - 6C 电子标签. 系统采用 SQL Server 数据库和. NET 前段开发平台,实现了订单管理、货物识别、车辆调度等功能. 在郑州某物流园区近 1 年的应用实践表明,该系统在增强物流在途管理、提高物流效率方面起到了很好的作用.

5　结语

本文设计了一个基于 RFID 的物流在途管理信息系统. 该系统将云存储技术、RFID 技术和 GPRS 技术引入到物流管理中,为中小物流公司搭建了一个提供公共租用服务的物流在途管理平台,使它们能够快速、低成本、安全可靠地拥有企业专属物流在途系统,方便、准确、快速地完成在途货物和车辆的管理,为用户提供在途物品的实时短信通知和短信查询服务,方便用户掌握物品的在途状态. 系统能有效解决中小物流企业所面临的难题,且成本低,易操作.

参考文献:

[1] 徐树民,田心,王绍麟. 基于 RFID 系统的物流安全解决方案[J]. 计算机工程与设计,2011,32(4):1276.

[2] 董淑华. RFID 技术及其在物流中的应用[J]. 物流工程与管理,2012,34(7):50.

[3] 莫凌飞. 物联网 RFID 技术的主要应用[J]. 杭州科技,2010(1):18.

[4] 王丽慧,陈磊. 浅析 RFID 物流信息系统及其发展[J]. 物流工程与管理,2012,34(3):98.

[5] 闫柏睿. RFID 在物流系统中的应用分析及实证[J]. 价值工程,2010,19:39.

[6] 邢峰,徐菱. 基于 RFID 技术的物流管理信息系统的研究[J]. 黑龙江科技信息,2012(35):83.

(上接第 59 页)

命令	模块
boot	:boot
chainloader	:chain
freebsd_loadenv	:bsd
freebsd_module	:bsd
freebsd_module_elf	:bsd

图 5　command. lst 中部分命令与模块的对应关系

4)利用 GRUB 2 的模块化设计,可以进行二次开发,添加某些特殊功能的模块,比如在河南省重点科技攻关项目"单机多操作系统多应用环境版本管理系统"中,我们对整个计算机软件系统的系统环境、现场数据和技术状态实现操作系统级的快速还原备份,对开发关于 GRUB 2 的应用具有一定的借鉴意义.

参考文献:

[1] 刘天华,陈枭,朱宏峰,等. Linux 可加载内核模块机制的研究与应用[J]. 微计算机信息,2007(20):48.

[2] [美]罗德里格斯,费舍尔,斯莫斯基. Linux 内核编程[M]. 北京:人民邮电出版社,2011:194 - 211.

[3] MarcoGerards. Loopbackdevice[EB/OL]. (2005 - 01 - 21)[2014 - 03 - 15]. http://lists. gnu. org/archive/html/grub - devel/2005 - 01/msg00068. html.

[4] 郑强,陈杰,陈贞,等. Linux 驱动开发入门与实战[M]. 2 版. 北京:清华大学出版社,2014:2 - 10.

[5] 王婧怡,应忍冬,周玲玲. 微内核系统直接加载 ELF 文件机制的设计与研究[J]. 信息技术,2009(10):73.

[6] 黄道颖,张安琳,赵昭灵,等. Linux 系统对 SMP 并行处理的支持[J]. 郑州轻工业学院学报:自然科学版,2001,16(4):26.

[7] 朱园. Linux 设备驱动程序的研究与开发[J]. 仪表技术,2008(2):32.

[8] 高源. 浅谈 Linux 系统下 GRUB 的配置与研究[J]. 计算机光盘软件与应用,2012(15):126.

[9] 张学雷. 基于 GRUB 的多重系统启动的应用安全研究[J]. 计算机光盘软件与应用,2010(9):33.