Linux与自由软件资源丛书

Linux 程序设计权威指南

于明俭 陈向阳 方汉 编著

附赠 CD-ROM

机械工业出版社
China Machine Press

Linux与自由软件资源丛书

Linux程序设计权威指南

于明俭 陈向阳 方 汉 编著

机械工业出版社
China Machine Press

本书为在Linux上进行应用开发提供了详细的参考资料。内容包括：Linux开发环境、Linux编程的入门知识、系统和网络编程、多线程程序设计、控制台编程、X窗口系统编程、国际化编程知识等。本书内容丰富、实用性强，附带光盘包含书中样例代码，方便读者使用。

图书在版编目（CIP）数据

Linux 程序设计权威指南/于明俭，陈向阳，方汉编著. －北京：机械工业出版社，2001.4

(Linux 与自由软件资源丛书)

ISBN 7-111-08695-3

I. L… II. ①于… ②陈… ③方… III. Linux 操作系统 - 程序设计 IV.TP 316.89

中国版本图书馆CIP数据核字（2001）第06317号

机械工业出版社（北京市西城区百万庄大街22号 邮政编码100037）

责任编辑：金 锋 吴 怡

北京昌平奔腾印刷厂印刷 · 新华书店北京发行所发行

2001年4月第1版第1次印刷

787mm × 1092mm1/16 · 45印张

印数：0 001-6 000册

定价：79.00元（附光盘）

前　言

Linux 是什么？按照 Linux 开发者的说法，Linux 是一个遵循 POSIX（Portable Operating System Interface）标准的免费操作系统，具有 BSD 和 SYSV 的扩展特性（表明其外表和性能同常见的 UNIX 非常相像，但是所有系统核心代码已经全部被重新编写了）。它的版权所有者是芬兰籍的 Linus B. Torvalds 先生（torvalds@transmeta.com）和其他开发人员，并且遵循 GPL 声明（GNU General Public License）。

目前，Linux 的热潮已经掀起，Linux的普及比起我们写作第一本书《Linux 实用大全》的时候已经不可同日而语了。但是，目前在 Linux 上编程方面的书籍还很少，本书将试图在这方面填补一些空白。

本书写作的目的是为 Linux 上的开发者提供一本比较全面的参考书，不仅介绍在 Linux 上开发的入门知识，还将详细讲述对中国的开发者最重要的 X Window/GTK/QT 编程和国际化编程知识。由于本书内容覆盖面比较广，因此，有些章节将只能起到“师傅领进门，修行在个人”的作用。

本书章节内容

第1章 Linux程序设计基础。主要讲述如何建立开发环境，如何编写 Makefile，如何使用调试工具，如何使用CVS，如何使用集成开发环境等等 Linux 编程的入门知识。

第2章 系统和网络编程。讲述系统和网络编程的基本知识及如何利用 POSIX 线程函数库进行多线程程序设计。

第3章 控制台编程。介绍了如何利用 curses、newt、SVGA等函数库和 FrameBuffer 技术进行基于控制台的编程。

第4章 X 窗口系统编程。介绍基于 X 窗口系统的编程，包含 Xlib 编程、GTK/Gnome 编程、QT/KDE 编程和 Motif/LessTif 编程等方面的知识和技巧。本章特别注意着重介绍在 X 窗口系统下进行国际化和中文化编程的原理和方法。

第5章 脚本语言编程。介绍 Linux 系统上常用的一些脚本语言的编程技巧。

第6章 国际化和中文化编程。介绍 Linux 系统上国际化和中文化的基本概念，Clocale 的基本原理和基于 locale 的国际化编程方法。本章同时介绍了在程序中实现中文打印和使用中文 TrueType 字库的基本原理和方法。

本书的读者对象

本书假设读者已经能够熟练使用Linux或UNIX系统，并且至少掌握一门编程语言，如C语言，并且希望在Linux下开发自己的应用程序。

本书的阅读方法

本书既可以一章一章按步就班地阅读，也可以直接跳到相应章节查阅。本书的目的就是要给用户提供一本简易的编程参考手册。

鸣谢和致歉

在本书的写作过程中，得到了下列同志的帮助，没有他们，这本书也许不会出现，在这里表示衷心的感谢。他们是：利启诚、李振春、王剑、胡鹏飞、马军、机械工业出版社华章公司的老师还有中国大陆和台湾的Linux Hacker们。

此外还要感谢我们的家人，如果没有他们在生活上的照顾，我们也许不会有更多精力去写作和Hacking。他们是：胡擘、王书梅和老于可爱的儿子于晨。

由于成书仓促，这本书还有很多不完善的地方，希望大家原谅，此外我们还参考了网上的不少资料，由于篇幅所限，就不再列出来源，希望作者原谅。读者如有问题请访问：http://www.opencjk.org/CLPG/ mailto:book@opencjk.org。

谨以此书献给Linux和我们的七年Linux Hacking生涯。

Happy Hacking!

方　汉

2000年12月

目　录

第1章 Linux 程序设计基础

1.1 编程风格

Linux作为GNU家族的一员，其源代码数以万计，而在阅读这些源代码时我们会发现，不同的源代码的美观程度和编程风格都不尽相同，有些代码，如以下glibc的代码:

```
static void
release_libc_mem (void)
{
  /* Only call the free function if we still are running in mtrace mode.  */
  if (mallstream != NULL)
    __libc_freeres ();
}
```

或者Linux核心的代码:

```
static int do_linuxrc(void * shell)
{
        static char *argv[] = { "linuxrc", NULL, };
        close(0);close(1);close(2);
        setsid();
        (void) open("/dev/console",O_RDWR,0);
        (void) dup(0);
        (void) dup(0);
        return execve(shell, argv, envp_init);
}
```

看起来令人赏心悦目，而其他有些程序员写的程序则让人看起来直皱眉头。写出干净美观的代码，不仅仅使得代码更容易阅读，还使得代码能够成为一件艺术品。同微软的匈牙利命名法一样，Linux上的编程主要有两种编程风格: GNU风格和Linux核心风格，下面将分别介绍。

1.1.1 GNU编程风格

下面几条是基于GNU开放源代码方面的要求:

1) 在任何情况下都不要引用有版权的源代码。

2) 善意接受别人给你的程序添加的代码，但请记住检查其合法性，即是否也是 GNU 的。

3) 编写日志文件(Changelog)，这将使你的代码更容易维护。

下面是GNU对C程序的风格要求:

1) 函数的开头的左花括号放到最左边，避免把任何其他的左花括号、左括号或者左方括号放到最左边。对于函数定义来说，把函数名的起始字符放到最左边也同样重要。这有助于寻找

函数定义，并且可帮助某些工具识别它们。因此，正确的格式应该是：

```
static char *
concat (s1, s2)         /* Name starts in column zero here */
     char *s1, *s2;
{                       /* Open brace in column zero here */
  ...
}
```

或者，如果希望使用标准C，定义的格式是：

```
static char *
concat (char *s1, char *s2)
{
  ...
}
```

如果参数不能够美观地放在一行中，按照下面的方式把它们分开：

```
int
lots_of_args (int an_integer, long a_long, short a_short,
              double a_double, float a_float)
...
```

对于函数体，我们希望按照如下方式排版：

```
if (x < foo (y, z))
  haha = bar[4] + 5;
else
  {
   while (z)
     {
       haha += foo (z, z);
       z--;
     }
   return ++x + bar ();
}
```

在左括号之前以及逗号之后添加空格将使程序更加容易阅读（尤其是在逗号之后添加空格）。当我们把一个表达式分成多行的时候，应在操作符之前（而不是之后）分割。例如：

```
if (foo_this_is_long && bar > win (x, y, z)
    && remaining_condition)
```

2) 尽力避免让两个不同优先级的操作符出现在相同的对齐方式中。例如，不要像下面那样写：

```
mode = (inmode[j] == VOIDmode
        || GET_MODE_SIZE (outmode[j]) > GET_MODE_SIZE (inmode[j])
        ? outmode[j] : inmode[j]);
```

应该附加额外的括号以使得文本缩进可以表示出嵌套，如下所示：

```
mode = ((inmode[j] == VOIDmode
         || (GET_MODE_SIZE (outmode[j]) > GET_MODE_SIZE (inmode[j])))
         ? outmode[j] : inmode[j]);
```

3) 按照如下方式排版do-while语句:

```
do
  {
    a = foo (a);
  }
while (a > 0);
```

4) 每个程序都应该以一段简短地说明其功能的注释开头。例如: “fmt - filter for simple filling of text”。

5) 请为每个函数书写注释以说明函数做了些什么，需要哪些种类的参数，参数可能值的含义以及用途。如果按照常见的方式使用C语言类型，就没有必要逐字重写C参数声明的含义。如果它使用了任何非标准的东西，或者是可能导致函数不能工作的任何可能的值（例如，不能保证正确处理一个包含了新行的字符串），应对它们进行说明。如果存在重要的返回值，也需要对其进行解释。

6) 不要在声明多个变量时跨行。在每一行中都以一个新的声明开头。例如，不应该:

```
int foo,
    bar;
```

而应该:

```
int foo, bar;
```

或者:

```
int foo;
int bar;
```

如果它们是全局变量，在它们之中的每一个之前都应该添加一条注释。

7) 当在一个if语句中嵌套了另一个if-else语句时，应用花括号把if-else括起来。因此，不要写:

```
if (foo)
  if (bar)
    win ();
  else
    lose ();
```

而要写:

```
if (foo)
  {
    if (bar)
      win ();
    else
      lose ();
  }
```

如果在else语句中嵌套了一个if语句，即可以像下面那样写else if:

```
if (foo)
  ...
```

```
else if (bar)
  ...
```

按照与then那部分代码相同的缩进方式缩进else if的then部分代码，也可以在花括号中像下面那样把if嵌套起来:

```
if (foo)
  ...
else
  {
    if (bar)
      ...
  }
```

8) 要在同一个声明中同时说明结构标识和变量或者结构标识和类型定义（typedef)。单独地说明结构标识，而后用它定义变量或者定义类型。

9) 尽力避免在if的条件中进行赋值。例如，不要写:

```
if ((foo = (char *) malloc (sizeof *foo)) == 0)
  fatal ("virtual memory exhausted");
```

而要写:

```
foo = (char *) malloc (sizeof *foo);
if (foo == 0)
  fatal ("virtual memory exhausted");
```

10) 请在名字中使用下划线以分隔单词，尽量使用小写；把大写字母留给宏和枚举常量，以及根据统一的惯例使用的前缀。例如，应该使用类似ignore_space_change_flag的名字；不要使用类似iCantReadThis的名字。

11) 用于标明一个命令行选项是否被给出的变量应该在选项含义的说明之后，而不是选项字符之后被命名。一条注释既应该说明选项的精确含义，又应该说明选项的字母。例如:

```
/* Ignore changes in horizontal whitespace (-b).  */
int ignore_space_change_flag;
```

1.1.2 Linux 内核编程风格

下面是 Linux 内核所要求的编程风格:

1) Linux内核的缩进格式是8个字符。

2) Linux内核的风格采用K&R标准，将开始的大括号放在一行的最后，而将结束大括号放在一行的第一位，如下所示:

```
if (x is true) {
we do y
}
```

然而，还有一种特殊的情况:命名函数时，开始的括号是放在下一行的第一位，如下所示:

```
int function(int x)
{
```

```
body of function
}
```

需要注意的是结束的括号在它所占的那一行是空的，除了它跟随着同一条语句的继续符号。如while在do-while循环中，或者else在if语句中。如下所示:

```
do {
body of do-loop
} while (condition);
```

以及

```
if (x == y) {
..
} else if (x > y) {
...
} else {
....
}
```

3) 命名尽量简洁。不应使用诸如 ThisVariableIsATemporaryCounter 之类的命名方式。一个C语言的程序员会将它命名为tmp，这很容易书写，且并不是那么难理解。但是命名全局变量时，就像全局函数一样，需要有描述性的命名方式。假如有一个函数用来计算活动用户的数量，应该这样命名: “count_active_users()”，或另外的相近形式，不应命名为 “cntusr()”。本地变量的命名应该尽量避免过长。

4) 函数最好要短小精悍，一般来说如果这个函数中定义的变量超过10个，那么这个函数肯定是过于复杂了。

5) 通常情况下，注释说明代码的功能，而不是说明其实现原理。而且，要试图避免将注释插在一个函数体里，假如这个函数确实很复杂，需要在其中有部分的注释，可以写些简短的注释来注明或警告那些重要的部分，但是必须要避免过多。应该尽量将注释写在函数前面，说明其功能。

看了上面两种风格的介绍，读者是不是觉得有些太难了?不要紧，Linux的好处就是会有很多的工具来帮助我们，首先我们来看一下在Linux上使用广泛的Emacs中的设置。

Linux核心编程风格的.emacs配置文件(该文件放置在用户的home目录下):

```
(defun linux-c-mode ()
"C mode with adjusted defaults for use with the Linux kernel."
(interactive)
(c-mode)
(c-set-style "K&R")
(setq c-basic-offset 8))
(setq auto-mode-alist (cons '("/usr/src/linux.*/.*\\.[ch]$" . linux-c-mode) auto-
mode-alist))
```

最后一行使得一旦编辑/usr/src/linux目录下面的c文件时将自动启动这种编程模式。也可以在Emacs中输入M-x linux-c-mode来启动它。由于Emacs是GNU的产品，因此Emacs缺省的编程风格就是GNU编程风格。在VIM中，用户可以编辑自己的.vimrc文件，加上下面这些配置，就可

以在VIM中实现Linux核心编程风格。

```
set ts=8
if !exists("autocommands_loaded")
  let autocommands_loaded = 1
  augroup C
      autocmd BufRead *.c set cindent
  augroup END
endif
```

其实除了VIM和Emacs以外，还有一个非常有意思的小工具indent可以帮我们做这件事情，indent实际上就是一个C/C++源代码美容师，例如我们有这样一段C程序：

```
test.c:
int __xstat (int vers, const char *name, struct stat *buf)
{
struct kernel_stat kbuf;
int result;

if (vers == _STAT_VER_KERNEL)
return INLINE_SYSCALL (stat, 2, CHECK_STRING (name), CHECK_1 ((struct kernel_stat *)
buf));

result = INLINE_SYSCALL (stat, 2, CHECK_STRING (name), __ptrvalue (&kbuf));
if (result == 0)
result = xstat_conv (vers, &kbuf, buf);

return result;
}
```

那么我们用下面这条命令将其转变为GNU编程风格：

```
indent -gnu test.c
```

test.c现在的格式是：

```
int
__xstat (int vers, const char *name, struct stat *buf)
{
  struct kernel_stat kbuf;
  int result;

  if (vers == _STAT_VER_KERNEL)
    return INLINE_SYSCALL (stat, 2, CHECK_STRING (name),
                           CHECK_1 ((struct kernel_stat *) buf));

  result = INLINE_SYSCALL (stat, 2, CHECK_STRING (name), __ptrvalue (&kbuf));
  if (result == 0)
    result = xstat_conv (vers, &kbuf, buf);

  return result;
}
```

如果要变成Linux核心风格，则输入如下命令：

```
indent -kr -i8 test.c
```

现在的格式是:

```
int __xstat(int vers, const char *name, struct stat *buf)
{
        struct kernel_stat kbuf;
        int result;

        if (vers == _STAT_VER_KERNEL)
                return INLINE_SYSCALL(stat, 2, CHECK_STRING(name),
                                      CHECK_1((struct kernel_stat *) buf));

        result =
            INLINE_SYSCALL(stat, 2, CHECK_STRING(name), __ptrvalue(&kbuf));
        if (result == 0)
                result = xstat_conv(vers, &kbuf, buf);

        return result;
}
```

其中-gnu参数就是采用GNU风格，-kr参数是采用K&R风格，-kr -i8参数就是Linux内核编程的风格。而且indent有几十个参数，将这些参数好好搭配，利用indent这个工具，相信读者也能写出漂亮的代码来。至于其他各种语言的风格，如perl、python、tcl/tk、php3等，最简单的方法就是参看原作者编写的例子代码就能知道正确的代码风格。读者如有兴趣，可参阅下面的有关文献:

1) Linux kernel coding style
   /usr/src/linux/Documentation/CodingStyle
   http://www.people.fas.harvard.edu/~zotto/cs50/linus_style.txt
2) GNU Coding Standards
   http://www.gnu.org/prep/standards_toc.html
3) GNOME Programming Guideline
   http://developer.gnome.org/doc/guides/programming-guidelines/code-style.html
4) Java Code Style
   http://developer.netscape.com/docs/technote/java/codestyle.html
5) Perl Style
   http://www.perl.com/pub/language/style/slide-index.html

1.2 如何使用gcc开发应用程序

在Linux中使用最多的编程语言还是C和C++,而如何使用gcc和编写Makefile,是每一个初学者最头疼的事情,下面我们就来看看如何使用gcc和编写Makefile。

1.2.1 如何使用gcc

1. 使用gcc编译和连接

用gcc编译C程序生成可执行文件有时候看起来像是一步就完成了，但实际上它要经历如下

的四个步骤:

1) 预处理 (Preprocessing): 这一步需要分析各种命令, 如#define、#include、#if等。gcc调用cpp程序来进行预处理工作。

2) 编译 (Compilation): 这一阶段根据输入文件产生汇编语言, 由于通常是立即调用汇编程序, 所以其输出一般不保存在文件中。gcc调用cc1进行编译工作。

3) 汇编 (Assembly): 这一步将汇编语言用作输入, 产生具有.o扩展名的目标文件。gcc调用as进行汇编工作。

4) 连接 (Linking): 这一阶段中, 各目标文件.o被放在可执行文件的适当位置上, 该程序引用的函数也放在可执行文件中 (对使用共享库的程序稍有不同)。gcc调用连接程序ld来完成最终的任务。

gcc的基本用法是:

```
% gcc -o prog main.c subr1.c subr2.c subr3.c
```

"-o prog" 指定输出的可执行文件名为prog。如果没有指定-o参数, gcc就使用缺省的可执行文件名a.out。如果想单独编译每一个源文件, 最后再进行连接, 可以如下进行:

```
% gcc -c  main.c
% gcc -c  subr1.c
% gcc -c  subr2.c
% gcc -c  subr3.c
% gcc -o prog main.o subr1.o subr2.o subr3.o
```

其中的-c选项表示编译产生目标文件, 但不连接。最后将所有目标文件组合在一起, 构成可执行文件。由于最后一个命令的输入都是目标文件, 不需要编译和汇编, 所以gcc就只调用连接程序。

gcc如何处理出现在命令行的文件取决于文件的名字, 确切地说, 取决于文件的扩展名。在每一种情况下, gcc都要将该文件送到相应的程序去进行预处理, 编译或汇编, 然后将产生的所有目标模块连接在一起, 产生一个可执行文件。下面是gcc如何处理不同类型的文件的列表:

```
file.c:    C源文件, 被gcc预处理和编译
file.C:    C++源文件, 被gcc预处理和编译
file.cc:   C++源文件, 被gcc预处理和编译
file.cxx:  C++源文件, 被gcc预处理和编译
file.m:    Objective C源文件, 被gcc预处理、编译和汇编
file.i:    预处理后的C源文件, 被gcc编译
file.ii:   预处理后的C++源文件, 被gcc编译
file.s:    汇编语言源文件, 被as汇编
file.S:    汇编语言源文件, 被as预处理和汇编
file.o:    编译后的目标文件, 传送给ld
file.a:    目标文件库, 传送给ld
```

gcc在命令行上经常使用的几个选项是:

-c 只预处理、编译和汇编源程序, 不进行连接。编译器对每一个源程序产生一个目标文件。

-o file 确定输出文件为file。如果没有用-o选项, 缺省的可执行文件的输出是a.out, 目标文件和汇编文件的输出对source.suffix分别是source.o和source.s, 预处理的C源程序的输出是标准

输出stdout。

-Dmacro或-Dmacro=defn 其作用类似于源程序里的#define。例如:

```
% gcc -c -DHAVE_GDBM -DHELP_FILE=\"help\" cdict.c
```

其中第一个-D选项定义宏HAVE_GDBM，在程序里可以用#ifdef去检查它是否被设置。第二个-D选项将宏HELP_FILE定义为字符串“help”（由于反斜线的作用，引号实际上已成为该宏定义的一部分），这对于控制程序打开哪个文件是很有用的。

-Umacro 某些宏是被编译程序自动定义的。这些宏通常可以指定在其中进行编译的计算机系统类型的符号，用户可以在编译某程序时加上-v选项以查看gcc缺省定义了哪些宏。如果用户想取消其中某个宏定义，用-Umacro选项，这相当于把#undef macro放在要编译的源文件的开头。

-Idir 将dir目录加到搜寻头文件的目录列表中去，并优先于在gcc缺省的搜索目录。在有多个-I选项的情况下，按命令行上-I选项的前后顺序搜索。dir可使用相对路径，如-I../inc等。

-O 对程序编译进行优化，编译程序试图减少被编译程序的长度和执行时间，但其编译速度比不做优化慢，而且要求较多的内存。

-O2 允许比-O更好的优化，编译速度较慢，但结果程序的执行速度较快。

-g 产生一张用于调试和排错的扩展符号表。-g选项使程序可以用GNU的调试程序GDB进行调试。优化和调试通常不兼容，同时使用-g和-O（-O2）选项经常会使程序产生奇怪的运行结果。所以不要同时使用-g和-O（-O2）选项。

-fpic或-fPIC 产生位置无关的目标代码，可用于构造共享函数库。

以上是gcc的编译选项。gcc的命令行上还可以使用连接选项。事实上，gcc将所有不能识别的选项传递给连接程序ld。连接程序ld将几个目标文件和库程序组合成一个可执行文件，它要解决对外部变量、外部过程、库程序等的引用。但我们永远不必要显式地调用ld。利用gcc命令去连接各个文件是很简单的，即使在命令行里没有列出库程序，gcc也能保证某些库程序以正确的次序出现。

gcc的常用连接选项有下列几个:

-Ldir 将dir目录加到搜寻-l选项指定的函数库文件的目录列表中去，并优先于gcc缺省的搜索目录。在有多个-L选项的情况下，按命令行上-L选项的前后顺序搜索。dir可使用相对路径。如-L../lib等。

-lname 在连接时使用函数库libname.a，连接程序在-Ldir选项指定的目录下和/lib，/usr/lib目录下寻找该库文件。在没有使用-static选项时，如果发现共享函数库libname.so，则使用libname.so进行动态连接。

-static 禁止与共享函数库连接。

-shared 尽量与共享函数库连接。这是Linux上连接程序的缺省选项。

下面是一个使用gcc进行连接的例子:

```
% gcc -o prog main.o subr.o -L../lib -lany -lm
```

2. 创建函数库

C语言中有一些函数不需要进行编译，有一些函数也可以在多个程序中使用。一般来说，这些函数都会执行一些标准任务，如数据库输入/输出操作或屏幕控制等。可以事先对这些函数进