第二章　Linux 系统下 C 语言程序设计

目前 Linux 已经被广泛的使用，因此有必要简单介绍一下，在 Linux 系统下如何进行 C 语言程序设计。首先介绍在 Linux 下如何编辑 C 语言源程序，接下来介绍如何编译 C 语言源程序，最好介绍如何调试与运行 C 语言源程序。由于不是所有用户的 Linux 系统中都安装了 GCC 系统，因此有必要先介绍一下 GCC 的安装。

2.1　GCC 的安装

Linux 与 Unix 类似，完全由 C 语言编写而成，因此天生的支持 C 语言。在目前主流的 Linux 发行版本中都包含了 GNU 的 C 语言编译器（简称 GNU C，或称为 GCC）。如果当前的 Linux 系统中没有安装 GCC，可以访问下面的站点搜索所需的安装文件：

http://www.gnu.org/

或直接访问

ftp://ftp.gnu.org/gcc

然后进行安装即可。

在安装之前，需要下载 GCC 编译器、所需的库文件和联机帮助文件，这些文件一般以压缩文件格式（.tar 或.zip）提供，需要解压缩后使用。目前 GCC 的版本为 3.2.2，下载其相关的文件即可。

根据具体的情况，安装 GCC 有如下三种方法：

1) 升级现有 Linux 系统：适用于具有 Linux 安装光盘，但是当前的系统没有安装 GCC 系统。
2) 通过 RPM 安装：适用于具有 Linux 安装光盘，并且包含相关的安装文件。
3) 从 Internet 下载：适用于没有 Linux 安装光盘，但是可以接入互联网（WWW）。

第一种方法基本上是升级当前的 Linux 系统，因此需要如下的步骤：

1) 备份当前系统的重要文件；
2) 插入 Linux 安装光盘。执行系统安装。比较简单的方式是从光盘引导系统；
3) 选择常规模式，并进入安装过程；
4) 选择升级模式；
5) 选择相关的升级包，主要包括 Develpoment/Debuggers、Develpoment/Languages、Develpoment\Libraries、Develpoment\Tools；
6) 等待安装结束即可。

第二种方法类似安装应用软件，利用了 Redhat Package Manger（RPM）技术。基本思路是先在 Linux 系统或互联网（WWW）中查找相关的 RPM 安装包，并下载到本机。安装过程包括如下的步骤：

1) 启动 X-Windows；
2) 插入 Linux 安装光盘；
3) 单击 CD-ROM 图标；
4) 单击 RPM 图标，启动 RPM；
5) 选择 Fil 菜单的下的 Open 命令；

6) 选择/mnt/cdrom 路径，并找到 PRMS 子目录；
7) 选择相应的安装包，对于不同版本的 Linux，其安装包可能不相同。例如可能是 gcc-c++.rpm 或 egcc-c++.rpm.；
8) 双击安装包，然后在提示对话框中选择 Install 按钮；

如果不启动 X-Windows，可以执行如下的命名：

1) 以管理员身份登录
2) 装载光盘 mout /mnt/cdrom
3) 进入 PRMS 子目录
4) 安装相应的安装包。
5) rpm –i gcc-c++ -dev

第三种方法是通过互联网（WWW）下载相关的安装文件，这时获得最新版本的最快方法。基本步骤如下：

1) 下载相关文件，假设为 gcc-c++_3_2_2.tar.gz；
2) 将文件复制一个空目录后，执行如下的命令，即可；

```
gunzip gcc-c++_3_2_2.tar.gz
tar -tf gcc-c++_3_2_2.tar
tar -xvf gcc-c++_3_2_2.tar
```

2.2 C 语言源程序的编辑

GCC 并不是一个完整的集成开发环境，因为其不提供程序代码的编辑器。C 语言程序的编辑需要通过的其他应用软件来完成。一般的 Linux 系统都提供了文本编辑器软件——Vi，下面主要介绍 Vi 编辑器。在 X-Windows 下还有许多的软件可以完成 C 语言程序的编辑工作，这里不一一介绍。

2.2.1 vi 简介

vi 使用了两种状态，一是命令状态（Command Mode），另一是插入状态（Insert Mode）。当 vi 处于命令状态时，输入的内容将作为命令来解释；另一方面，当 vi 处于插入状态时，就可以插入字符。大多数 vi 命令是单字符，由插入状态改变为命令态，指 Esc 键；而由命令状态转为插入状态，则可以输入相应的插入命令，直接输入，无需再按回车键。

命令	说明
i	在光标前插入正文
I	在当前行开始处插入正文
a	在光标后插入正文
A	在当前行末尾插入正文
o	在当前行后插入一新行
O	在当前行前插入一新行

表 2-1 插入命令列表

注意，在插入状态，不能输入命令，必需先按下 Esc 键，返回命令状态。在任何的状态下，按下 Esc 键，都会返回命令状态。

2.2.2 vi 启动

在控制台下输入

$ vi

或

$ vi 文件名

即可启动 vi 编辑器。如果文件是一个新文件，就会在屏幕底部现实一个信息，说明用户正在创建新文件。

2.2.3 建立新文件

在控制台下输入

$ vi

或

$ vi 新文件名

即可建立新文件，这时可以输入相关的文本即可。其中“vi”的作用是启动 vi 环境并建立一个空文档。

“vi 新文件名”的作用是建立以“新文件名”的空文件。

2.2.4 打开文件

在控制台下输入

$ vi 文件名

即可启动 vi 的同时，打开名为“文件名”的文件。如果文件早已存在，vi 则会显示文件的首 24 行。

2.2.5 保存与退出

完成文件的修改之后，必须完成文档的保存，以及退出当前的文件。在 vi 系统中，在进行保存或退出时，必须首先回到命令状态。如果屏幕的左下方出现冒号（:），表示 vi 进入命令状态，可以进行存档或退出等工作。

在 vi 环境下，按 Esc 键退出编辑环境，进入命令环境，输入“:q!”或“:wq”命令。其中“:q!”命令表示放弃当前的修改，并退出系统。“:wq”命令表示保存当前的修改，并退出系统。如果当前文件还没有命名需要使用:w <filename>命令，例如

:w test.c

将当前的内容保存为名为 test.c 的文件。下面是一些常用的命令列表。

命令	说明
:q!	放弃任何改动并退出 vi 系统
:w <filename>	保存当前的文件
:wq	保存当前的文件，并退出 vi 系统
:x	与 wq 类似
:zz	与 wq 的工作类似

表 2-2　保存与退出命令

2.2.6　文件的编辑

文件的所有编辑的操作必须在插入状态下进行。从命令状态切换到插入状态的命令参见表 1 。下面是一个经 vi 打开的一个文件

This is a test

～

～

最后一行开始处的波折号（～）表示文件的结尾。打开文件之后，即可适用对文件进行编辑操作。

1.插入文本

当 vi 处于插入状态时，才可以插入文本到当前的位置。输入新的文本，只需从键盘输入字符即可。参考表 1 中所列命令。

2.删除文本

删除单个字符可以使用 delete 键或 backspace 键。对于复杂的删除命令，可以参考如下的命令列表：

命令	说明
x	删除光标处字符（Character）
nx	删除光标处后 n 个字符
nX	删除光标处前 n 个字符
ndw	删除光标处下 n 个单词（word）
dd	删除整行
d$或 D	删除由光标至该行最末
U	恢复前一次所做的删除

表 3 2　删除文本命令

3.替换文本

当使用 vi 提供丰富的文本替换命令，实现文本的快速修改。表 4 给出了操作命令：

命令	说明
r char	由 char 代替光标处的字符
Rtext〈Esc〉	由 text 代替光标处的字符
cw　text〈Esc〉	由 text 取代光标处的单词
C text〈Esc〉	由 text 取代光标处至该行结尾处
Cc	使整行空白，但保留光标位置，让你开始输入

表 2-4　替换文本命令

与删除命令一样，在指令前输入次数，表示执行该指令多少次。

4.文本搜索

vi 还提供了丰富的文本搜索命令，方便用户搜索特定字符串。Vi 将搜索整个文件，直至找到与搜索字符串相匹配的文本出现。文本搜索可以通过表 5 所示的一组命令来实现。

命令	说明
／str〈Return〉	向前搜寻 str 直至文件结尾处
?str〈Return〉	往后搜寻 str 直至文件开首处

n	同一方向上重复检索
N	相反方向上重复检索

表 2- 5 文本搜索命令

2.3 C 语言源程序的编译与链接

C 语言源程序的编译与链接由 GCC 编译器来完成。 gcc 命令的基本用法格式如下：

gcc [options] [filenames]

其中 filenames 为文件名称；options 为编译选项，说明针对当前文件的编译与链接选项。

2.3.1 GCC 基本选项

GCC 提供的编译选项很多，其中一些选项可能永远都不会用到, 但一些主要的选项可能会频繁使用。 由于很多的 GCC 选项包括一个以上的字符，因此必须为每个选项指定各自的连字符，并且就象大多数 Linux 命令一样你不能在一个单独的连字符后跟一组选项。例如, 下面的两个命令是不同的:

gcc -p -g hello.c

gcc -pg hello.c

第一条命令要求 GCC 编译 test.c 时为 prof 命令建立剖析(profile)信息并且把调试信息加入到可执行的文件中。 第二条命令只要求 GCC 为 gprof 命令建立剖析信息.。

当你不用任何选项编译一个程序时，如果编译成功，GCC 将会自动建立一个名为 a.out 的可执行文件。 例如，下面的命令将在当前目录下产生一个叫 a.out 的文件:

gcc hello.c

但是如果使用-o 编译选项，则可以为将产生的可执行文件指定一个文件名来代替 a.out.。例如，将一个名为 Test.c 的 C 程序编译为名叫 Test 的可执行文件，你将输入下面的命令：

gcc -o Test Test.c

注意：当你使用 -o 选项时, -o 后面必须跟一个文件名。

GCC 同样有指定编译器编译步骤的选项。例如： -c 选项要求 GCC 仅将源代码编译为目标代码，而跳过汇编和链接的步骤。缺省时 GCC 将生成的一个 扩展名为.o 的目标代码文件。

1) -S 选项要求 GCC 在将 C 程序翻译为汇编语言文件后停止编译。GCC 产生的汇编语言文件的缺省扩展名是 .s.。
2) -E 选项要求 GCC 仅对输入文件进行预处理。当这个选项被使用时, 预处理器的输出被送到标准输出而不是储存在文件。

2.3.2 GCC 优化选项

当你用 GCC 编译 C 代码时, 它会试着用最少的时间完成编译并且使编译后的代码易于调试， 易于调试意味着编译后的代码与源代码有同样的执行次序，编译后的代码没有经过优化。有很多选项可用于告诉 GCC 在耗费更多编译时间和牺牲易调试性的基础上产生更小更快的可执行文件. 这些选项中最典型的是-O 和 -O2 选项。

1) -O 选项告诉 GCC 对源代码进行基本优化。这些优化在大多数情况下都会使程序执行的更快。 -O2 选项告诉 GCC 产生尽可能小和尽可能快的代码。
2) -O2 选项将使编译的速度比使用 -O 时慢. 但通常产生的代码执行速度会更快。

除了 -O 和 -O2 优化选项外，还有一些低级选项用于产生更快的代码。 这些选项非常的特殊，而且最好只有当你完全理解这些选项将会对编译后的代码产生什么样的效果时再去使用。这些选项的详细描述, 请参考 GCC 的联机帮助， 在命令行上键入 man gcc 即可。

假设我们有下面一个非常简单的源程序(hello.c):

```
#include “stdio.h”
void main()
{
    printf("Welcome to Linux C World!");
}
```

要编译这个程序，只要在命令行下执行：

```
$ gcc -o hello hello.c
```

输入如下的命令，即可运行程序并看到结果

```
$ ./hello
```

运行结果为

```
Welcome to Linux C World!
```

2.3.3 GCC 调试选项

GCC 支持多种调试和剖析选项，其中最常用是 -g 选项。-g 选项告诉 GCC 产生能被 GNU 调试器使用的调试信息以便调试你的程序。关于调试 C 程序的更多信息请看下一节用 gdb 调试 C 程序。

2.4 C 语言源程序的编译调试

虽然 GCC 提供了调试选项，但是本身不能用于调试。Linux 提供了一个名为 gdb 的 GNU 调试程序。gdb 是一个用来调试 C 和 C++ 程序的调试器。它使你能在程序运行时观察程序的内部结构和内存的使用情况。以下是 gdb 所提供的一些功能:

1) 它使你能监视你程序中变量的值；
2) 它使你能设置断点以使程序在指定的代码行上停止执行；
3) 它使你能一行行的执行你的代码。

2.4.1 GDB 启动

在命令行上键入 gdb 并按回车键就可以运行 gdb 了，如下：

```
$    gdb
```

启动成功，将在屏幕上显示如下类似的内容：

```
GNU gdb 19991004
Copyright 1998 Free Software Foundation, Inc.
GDB is free software, covered by the GNU General Public License, and you are
```

welcome to change it and/or distribute copies of it under certain conditions.
Type "show copying" to see the conditions.
There is absolutely no warranty for GDB. Type "show warranty" for details.
This GDB was configured as "i386-redhat-linux".
(gdb)
当启动 gdb 之后, 即可在命令行上输入命令进行相关的调试操作。

也可以以下面的方式来启动 gdb :

$ gdb <filename>

这种方式启动 gdb ，直接将指定调试的程序文件装载到调试环境中。也就是让 gdb 装入名为 filename 的可执行文件，从而准备调试。

gdb 还提供了其他的启动选项，请参考 gdb 的联机帮助。在命令行上键入 man gdb 并回车即可，如下。

$ man gdb

为了能够进行调试,当前调试的程序文件中必须包含调试信息。其中调试信息包含程序中的每个变量的类型和其在可执行文件里的地址映射以及源代码的行号，gdb 利用这些信息使源代码和机器码相关联。因此在使用 gcc 编译源程序的时候必须使用 -g 选项，以便将调试信息包含在可执行文件中。

例如，编译上述的 hello.c 的过程如下：

$ gcc -o hello hello.c

2.4.2 GDB 基本命令

gdb 支持很多的命令以实现不同的功能，从简单的文件装入到检查所调用的堆栈内容。表 6 列出了在用 gdb 调试时经常用到的一些命令。

命令	说明
file	装入想要调试的可执行文件
kill	终止正在调试的程序
list	列出产生执行文件的源代码的一部分
next	执行一行源代码但不进入函数内部
step	执行一行源代码而且进入函数内部
run	执行当前被调试的程序
quit	退出 gdb
watch	使你能监视一个变量的值而不管它何时被改变
break	在代码里设置断点，这将使程序执行到这里时被挂起
make	使你能不退出 gdb 就可以重新产生可执行文件
shell	使你能不离开 gdb 就执行 UNIX shell 命令

表 2-6 基本 gdb 命令

2.5 应用举例

本节用一个简单的实例讲解应用 gdb 调试程序的基本步骤。

首先利用 vi 编辑建立名为 test.c 的程序文件。命令如下：

$ vi test.c

此时在当前的目录下建立名为 test.c 的文本文件。接下来，在 vi 编辑环境中使用 i 命令输入如下的程序代码：

```
#include "stdio.h"
#include "string.h"
void GetTitle(char *pszText)
{
    char szText[]="This is a Test C Program!\n";
    strcpy(pszText,szText);
}
void PrintTitle(char *pszText)
{
    unsigned int n,i;
    char a;
    n=strlen(pszText);
    for(i=0;i<n;i++)
    {
    a=pszText[i];
    printf("%c",a);
    }
}
int   main()
{
  int a;
  char szTitle[255];
  a=20;
  GetTitle(szTitle);
  PrintTitle(szTitle);
  printf("A=%d",a);
  return 0;
}
```

在程序编写完成之后，vi 编辑环境中按下 Esc 键，并输入:wq 命令，将当前的修改存盘并退出 vi 编辑环境。

用下面的命令编译 test.c，以便生成调试信息，准备调试。执行如下的命令

$gcc -g -o test test.c

启动 gdb 并加载 test

$gdb test

系统将显示如下的信息

```
GNU gdb 19991004
Copyright 1998 Free Software Foundation, Inc.
GDB is free software, covered by the GNU General Public License, and you are
welcome to change it and/or distribute copies of it under certain conditions.
Type "show copying" to see the conditions.
```

There is absolutely no warranty for GDB. Type "show warranty" for details.
This GDB was configured as "i386-redhat-linux"...

首先运行一下程序，大致看一下结果：

(gdb) run

系统将显示如下的信息

Starting program: /a.out
This is a Test C Program!
A=20
Program exited normally.

接下来执行 list 命令，查看程序的代码，主要是查看代码的行号。

(gdb) list 0

系统将显示如下的信息

```
1    #include "stdio.h"
2    #include "string.h"
3    void GetTitle(char *pszText)
4    {
5          char szText[]="This is a Test C Program!\n";
6          strcpy(pszText,szText);
7    }
8    void PrintTitle(char *pszText)
9    {
       unsigned int n,i;
```

由于屏幕有限，因此仅仅显示了 10 行代码，查看下面的 10 行代码，在此输入 list 命令即可。

(gdb) list

系统将显示如下的信息

```
11         char a;
12         n=strlen(pszText);
13               for(i=0;i<n;i++)
14         {
15         a=pszText[i];
16         printf("%c",a);
17         }
18   }
19   int    main()
{
```

接下来，再次输入 list 命令，显示剩余的代码。

(gdb) list

系统将显示如下的信息：

```
21   int a;
22   char szTitle[255];
23   a=20;
24   GetTitle(szTitle);
25   PrintTitle(szTitle);
```

```
26    printf("A=%d",a);
27    return 0;
}
```

为了查看在程序在 PrintTitle 函数中的循环体的运行情况，将断点设置在 15 行，以便观察变量 a 的变化。因此输入命令

```
(gdb) break 15
```

系统将显示如下的信息

```
Breakpoint 1 at 0x80484a0: file test.c, line 15.
```

接下来使用 run 命令运行程序，

```
(gdb) run
```

系统将显示如下的信息

```
Starting program: /a.out

Breakpoint 1, PrintTitle (pszText=0xbffff8b4 "This is a Test C Program!\n")
    at test.c:15
a=pszText[i];
```

提示信息说明程序运行到断点位置停了下来，此时可以通过 watch 命令查看变量的内容。下面查看变量 a 的内容的变化情况，输入如下的命令：

```
(gdb) watch a
```

系统将显示如下的设置成功信息

```
Hardware watchpoint 2: a
```

输入下面的指令，运行下一步。

```
(gdb) next
```

系统将显示变量 a 的变化情况

```
Hardware watchpoint 2: a

Old value = 111 'o'
New value = 84 'T'
PrintTitle (pszText=0xbffff8b4 "This is a Test C Program!\n") at test.c:16
printf("%c",a);
```

再次执行 next 命令查看下一步的结果。

```
(gdb) next
```

系统将显示如下信息

```
#0    PrintTitle (pszText=0xbffff8b4 "This is a Test C Program!\n") at test.c:16
16          printf("%c",a);
13              for(i=0;i<n;i++)
```

再次执行 next 命令。

```
(gdb) next
```

系统将显示如下信息

```
#0    PrintTitle (pszText=0xbffff8b4 "This is a Test C Program!\n") at test.c:13
13              for(i=0;i<n;i++)
```

Breakpoint 1, PrintTitle (pszText=0xbffff8b4 "This is a Test C Program!\n")
    at test.c:15
a=pszText[i];

结束调试，输入如下 quit 命令

(gdb) quit

系统提示如下的信息。

The program is running. Exit anyway? (y or n)

由于当前的程序正在运行，所以提示用户是否要退出，输入 y 后，即可强制结束调试过程，并退出 gdb 调试环境。

由于程序已经没有问题，执行如下命令建立不包括调试信息的发行版本的可执行文件。命令如下：

$gcc -o test test.c

形成名为 test 的可执行文件，此文件不包括调试信息，因此本身比较小，并且代码的执行效率比较高。

Linux 系统下如何编辑、编译/链接、调试、运行程序的基本步骤就简单的介绍完了。下一步的工作是熟练掌握基本的命令，并参考联机帮助进一步的学习。下面总结一下如何获得相关的帮助。在 Linux 系统某个命令的联机帮助是通过如下命令获得：

$ man 命令。

例如，获得上述几个命令的联机帮助的命令如下：

$man gcc ……获得 gcc 的相关帮助可以使用如下的命令；

$man gdb ……获得 gdb 的相关帮助可以使用如下的命令；

$man vi ……获得 vi 的相关帮助可以使用如下的命令。

