Linux 下面如何进行 C 语言编程技术教程

1、源程序的编译

在 Linux 下面,如果要编译一个 C 语言源程序，我们要使用 GNU 的 gcc 编译器。下面我们以一个实例来说明如何使用 gcc 编译器。
假设我们有下面一个非常简单的源程序(hello.c):

```
int main(int argc,char **argv)
{
printf("Hello Linux\n");
}
```

要编译这个程序,我们只要在命令行下执行：

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

gcc 编译器就会为我们生成一个 hello 的可执行文件，执行./hello 就可以看到程序的输出结果了。命令行中 gcc 表示我们是用 gcc 来编译我们的源程序，-o 选项表示我们要求编译器给我们输出的可执行文件名为 hello 而 hello.c 是我们的源程序文件。

gcc 编译器有许多选项，一般来说我们只要知道其中的几个就够了， -o 选项我们已经知道了，表示我们要求输出的可执行文件名。 -c 选项表示我们只要求编译器输出目标代码，而不必要输出可执行文件。 -g 选项表示我们要求编译器在编译的时候提供我们以后对程序进行调试的信息。

知道了这三个选项，我们就可以编译我们自己所写的简单的源程序了，如果你想要知道更多的选项，可以查看 gcc 的帮助文档，那里有着许多对其它选项的详细说明。

2、Makefile 的编写

假设我们有下面这样的一个程序,源代码如下:

```
/*   main.c */
#include "mytool1.h"
#include "mytool2.h"
int main(int argc,char **argv)
```

```
{
mytool1_print("hello");
mytool2_print("hello");
}
/*   mytool1.h   */
#ifndef _MYTOOL_1_H
#define _MYTOOL_1_H
void mytool1_print(char *print_str);
#endif
/*   mytool1.c   */
#include "mytool1.h"
void mytool1_print(char *print_str)
{
printf("This is mytool1 print %s\n",print_str);
}
/* mytool2.h */
#ifndef _MYTOOL_2_H
#define _MYTOOL_2_H
void mytool2_print(char *print_str);
#endif
/*   mytool2.c   */
#include "mytool2.h"
void mytool2_print(char *print_str)
{
printf("This is mytool2 print %s\n",print_str);
}
```

当然由于这个程序是很短的我们可以这样来编译

```
gcc -c main.c
gcc -c mytool1.c
gcc -c mytool2.c
gcc -o main main.o mytool1.o mytool2.o
```

这样的话我们也可以产生 main 程序,而且也不时很麻烦，但是如果我们考虑一下如果有一天我们修改了其中的一个文件(比如说 mytool1.c)那么我们难道还要重新输入上面的命令？写一个 SHELL 脚本，让她帮我去完成不就可以了。是的对于这个程序来说，是可以起到作用

的，但是当我们把事情想的更复杂一点，如果我们的程序有几百个源程序的时候，难道也要编译器重新一个一个的去编译？

为此，聪明的程序员们想出了一个很好的工具来做这件事情，这就是 make。我们只要执行以下 make，就可以把上面的问题解决掉。在我们执行 make 之前,我们要先编写一个非常重要的文件，--Makefile。对于上面的那个程序来说，可能的一个 Makefile 的文件是：

```
#   这是上面那个程序的 Makefile 文件

main:main.o mytool1.o mytool2.o
gcc -o main main.o mytool1.o mytool2.o
main.o:main.c mytool1.h mytool2.h
gcc -c main.c
mytool1.o:mytool1.c mytool1.h
gcc -c mytool1.c
mytool2.o:mytool2.c mytool2.h
gcc -c mytool2.c
```

有了这个 Makefile 文件，不过我们什么时候修改了源程序当中的什么文件，我们只要执行 make 命令，我们的编译器都只会去编译和我们修改的文件有关的文件，其它的文件她连理都不想去理的。

下面我们学习 Makefile 是如何编写的，在 Makefile 中也#开始的行都是注释行.Makefile 中最重要的是描述文件的依赖关系的说明.一般的格式是：

```
target: components
TAB rule
```

第一行表示的是依赖关系，第二行是规则。
比如说我们上面的那个 Makefile 文件的第二行

```
main:main.o mytool1.o mytool2.o
```

表示我们的目标(target)main 的依赖对象(components)是 main.o mytool1.o mytool2.o 当倚赖的对象在目标修改后修改的话,就要去执行规则一行所指定的命令，就象我们的上面那个 Makefile 第三行所说的一样要执行 gcc -o main main.o mytool1.o mytool2.o 注意规则一行中的 TAB 表示那里是一个 TAB 键

Makefile 有三个非常有用的变量.分别是$@,,<代表的意义分别是：

$@--目标文件，$^--所有的依赖文件，$<--第一个依赖文件。

如果我们使用上面三个变量,那么我们可以简化我们的 Makefile 文件为：

```
# 这是简化后的 Makefile
main:main.o mytool1.o mytool2.o
gcc -o $@ $^
main.o:main.c mytool1.h mytool2.h
gcc -c $<
mytool1.o:mytool1.c mytool1.h
gcc -c $<
mytool2.o:mytool2.c mytool2.h
gcc -c $<
```

经过简化后我们的 Makefile 是简单了一点，不过人们有时候还想简单一点，这里我们学习一个 Makefile 的缺省规则

```
.c.o:
gcc -c $<
```

这个规则表示所有的.o 文件都是依赖与相应的.c 文件的，例如 mytool.o 依赖于 mytool.c 这样 Makefile 还可以变为:

```
# 这是再一次简化后的 Makefile
main:main.o mytool1.o mytool2.o
gcc -o $@ $^
.c.o:
gcc -c $<
```

我们的 Makefile 也差不多了，如果想知道更多的关于 Makefile 规则可以查看相应的文档。

3、程序库的链接

试着编译下面这个程序

```
/* temp.c */
#include
int main(int argc,char **argv)
{
double value;
printf("Value:%f\n",value);
}
```

这个程序相当简单，但是当我们用 gcc -o temp temp.c 编译时会出现下面所示的错误。

```
/tmp/cc33Kydu.o: In function `main':
/tmp/cc33Kydu.o(.text+0xe): undefined reference to `log'
collect2: ld returned 1 exit status
```

出现这个错误是因为编译器找不到 log 的具体实现，虽然我们包括了正确的头文件，但是我们在编译的时候还是要连接确定的库。在 Linux 下，为了使用数学函数，我们必须和数学库连接，为此我们要加入 -lm 选项。gcc -o temp temp.c -lm 这样才能够正确的编译，也许有人要问,前面我们用 printf 函数的时候怎么没有连接库呢？是这样的，对于一些常用的函数的实现，gcc 编译器会自动去连接一些常用库，这样我们就没有必要自己去指定了。有时候我们在编译程序的时候还要指定库的路径，这个时候我们要用到编译器的 -L 选项指定路径。比如说我们有一个库在 /home/hoyt/mylib 下，这样我们编译的时候还要加上 -L/home/hoyt/mylib，对于一些标准库来说，我们没有必要指出路径。只要它们在起缺省库的路径下就可以了，系统的缺省库的路径/lib /usr/lib /usr/local/lib 在这三个路径下面的库，我们可以不指定路径。

还有一个问题,有时候我们使用了某个函数,但是我们不知道库的名字,这个时候怎么办呢？对于这个问题我也不知道答案，我只有一个傻办法，首先我到标准库路径下面去找看看有没有和我用的函数相关的库，我就这样找到了线程(thread)函数的库文件(libpthread.a)。 当然如果找不到，只有一个笨方法，比如我要找 sin 这个函数所在的库，就只好用 nm -o /lib/*.so|grep sin>~/sin 命令，然后看~/sin 文件,到那里面去找了，在 sin 文件当中,我会找到这样的一行 libm-2.1.2.so:00009fa0 W sin 这样我就知道了 sin 在 libm-2.1.2.so 库里面，我用 -lm 选项就可以了(去掉前面的 lib 和后面的版本标志,就剩下 m 了所以是 -lm)。

4、程序的调试

我们编写的程序不太可能一次性就会成功的，在我们的程序当中，会出现许许多多我们想不到的错误，这个时候我们就要对我们的程序进行调试了。

最常用的调试软件是 gdb.如果你想在图形界面下调试程序，那么你现在可以选择 xxgdb，记得要在编译的时候加入 -g 选项，关于 gdb 的使用可以看 gdb 的帮助文件。由于我没有用过这个软件，所以我也不能够说出如何使用。不过我不喜欢用 gdb，跟踪一个程序是很烦的事情，我一般用在程序当中输出中间变量的值来调试程序的。当然你可以选择自己的办法，没有必要去学别人的，现在有了许多 IDE 环境，里面已经自己带了调试器。

5、头文件和系统求助

有时候我们只知道一个函数的大概形式，不记得确切的表达式，或者是不记得着函数在那个头文件进行了说明，这个时候我们可以求助系统。

比如说我们想知道 fread 这个函数的确切形式，我们只要执行 man fread 系统就会输出着函数的详细解释的，和这个函数所在的头文件说明了。如果我们要 write 这个函数的说明，当我们执行 man write 时，输出的结果却不是我们所需要的。因为我们要的是 write 这个函数的说明,可是出来的却是 write 这个命令的说明。为了得到 write 的函数说明我们要用 man 2 write，2 表示我们用的 write 这个函数是系统调用函数，还有一个我们常用的是 3 表示函数是 C 的库函数。