Linux 操作系统

C 编程

Linux GCC/G++编译器与调试器

- 编译器
  - 将易于编写、阅读和维护的高级计算机语言翻译为计算机能解读、运行的低级机器语言的程序
- 调试器
  - 用于查找源代码中的错误，测试源代码和可执行文件的工具
- GNU 项目提供了 GCC 编译器、G++ 编译器和 GDB 调试器，这些程序是在 Linux 系统上使用 C 和 C++ 语言进行开发的重要工具

GCC/G++编译器

- GCC 是 GNU 项目中的一个子项目，最初是用于编译 C 语言的编译器
- 目前 GCC 编译器已经能编译C、C++、Ada、Object C 和 Java 等语言的 GNU 编译器家族，同时还可执行跨硬件平台的交叉编译工作
- G++是专门用来编写 C 和 C++语言的编译器
- 为了保持兼容程序语言的最新特性，通常选择 GCC 编译 C 语言源代码，选择 G++ 编译 C++ 源代码

安装 GCC/G++ 编译器

- 可以从 GNU 项目官网 www.gnu.org 下载安装
- 使用 yum 安装方式(视频：35 安装make和gcc)：
  - yum install make
  - yum install gcc
  - yum install gcc-c++
  - 如果安装过程中提示需要选择编译器版本，可根据当前硬件平台选择最新发布的版本。另外，如果提示需要安装其它相关软件包，请一并安装

GCC/G++ 编译命令

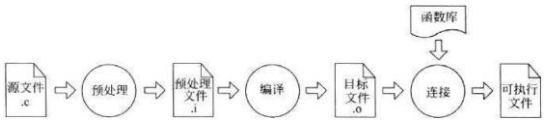
- 格式：
  - gcc [-选项1] [-选项2]… [-选项n] <源文件名>
  - g++ [-选项1] [-选项2]… [-选项n] <源文件名>
- 命令名、选项和原文件名之间用空格分隔
- 一行命令中可以有多个选项，也可以只有一个选项
- 文件名可以包含绝对路径，也可以使用相对路径
- 如果文件名中不包含路径，则源文件被视为存在于工作目录中
- 如果命令中不包含输出的可执行文件名，默认情况下将在工作目录中生成后缀为".out"的可执行文件

GCC/G++ 的常用编译选项

编 译 选 项	说　明
-c	只进行预处理、编译和汇编，生成“.o”文件
-S	只进行预处理和编译，生成“.s”文件
-E	只进行预处理，产生预处理后的结果到标准输出
-C	预处理时不删除注释信息，常与-E 同时使用
-o	指定目标名称，常与-c、-S 同时使用，默认是“.out”
-include file	插入一个文件，功能等同源代码中的#include
-Dmacro[=defval]	定义一个宏，功能等同源代码中的#define macro [defval]
-Umacro	取消宏的定义，功能等同源代码中的#undef macro
-Idir	优先在选项后的目录中查找包含的头文件
-lname	链接后缀为“.so”的动态链接库来编译程序
-Ldir	指定编译搜索库的路径
-O[0-3]	编译器优化，数值越大优化级别越高，0 没有优化
-g	编译器编译时加入 debug 信息
-pg	编译器加入信息给 gprof
-share	使用动态库
-static	禁止使用动态库

GCC/G++ 编译器执行过程

- GCC/G++ 编译器执行过程可总结为 4 步：预处理、编译、汇编、连接



示例

- hello.c 源代码

```
#include <stdio.h>
int main()
{
   printf("Hello World!\n");
   return 0;
}
```

- 方法1：默认生成 a.out (视频：36 默认选项编译和运行)
  - 编译：gcc hello.c
  - 运行：./a.out

```
[root@localhost ~]# gcc hello.c
[root@localhost ~]# ls
anaconda-ks.cfg  a.out  hello.c  install.log  install.log.syslog
[root@localhost ~]# ./a.out
Hello World!
```

- 方法2：生成指定的可执行文件(hello)
  - 编译：gcc -o hello hello.c
  - 运行：./hello

```
[root@localhost ~]# gcc -o hello hello.c
[root@localhost ~]# ls
anaconda-ks.cfg  a.out  hello  hello.c  install.log  install.log.syslog
[root@localhost ~]# ./hello
Hello World!
```


GDB 调试器

- 调试器是帮助程序员修改错误的工具，常用的断点、单步跟踪等功能可快速找到故障点
- Linux 程序员最常用的调试工具是 GDB，它是 GNU 项目的子项目，该程序提供了所有常用调试功能，是 Linux系统中最为简单快捷的

安装 GDB 调试器

- 通常在 Linux 桌面版的软件开发包集合中已包含 GDB 调试器
- 安装或更新 GDB 调试器(视频：37 安装gdb)
  - yum install gdb

GDB 常用调试命令

命　令	解　释
file <文件名>	在 GDB 中打开执行文件
break	设置断点，支持如下形式：break 行号、break 函数名称、break 行号/函数名称 if 条件
info	查看和可执行程序相关的各种信息
kill	终止正在调试的程序
print	显示变量或表达式的值
set args	设置调试程序的运行参数
delete	删除设置的某个断点或观测点，与 break 操作相似
clear	删除设置在指定行号或函数上的断点
continue	从断点处继续执行程序
list	列出 GDB 中打开的可执行文件代码
watch	在程序中设置观测点
run	运行打开的可执行文件
next	单步执行程序
step	进入所调用的函数内部，查看执行情况
whatis	查看变量或函数类型，调用格式为“whatis 变量名/函数名”
ptype	显示数据结构定义情况
make	编译程序
quit	退出 GDB

GDB 调试器(续)

- 使用 GCC/G++编译器编译源代码时，必须加上 -g 选项才能使目标可执行文件包含可被调试的信息

例(视频：38 使用gdb进行调试) helloworld.c

- helloworld.c 源代码

```
#include <stdio.h>
int main()
{
    const char *c;
    c="hello world!";
    printf("%s\n",c);
    return 0;
}
```

- gcc -g -o helloworld helloworld.c // 编译连接，使包含调试信息

```
[root@localhost ~]# gcc -g -o helloworld helloworld.c
```


例(续)：在 GDB 下运行程序

- 打开可执行文件后，可根据需要在程序中加入断点或观察点，并运行程序
- 例：在变量赋值前加入断点，并运行程序
  - gdb helloworld // 用调试器打开并调试helloworld可执行文件
    - (gdb) break 5 // 在源代码第5行，即变量c赋值处加入断点
    - (gdb) run // 运行程序

```
[root@localhost ~]# gdb helloworld
GNU gdb (GDB) Red Hat Enterprise Linux (7.2-92.el6)
Copyright (C) 2010 Free Software Foundation, Inc.
License GPLv3+: GNU GPL version 3 or later <http://gnu.org/licenses/gpl.html>
This is free software: you are free to change and redistribute it.
There is NO WARRANTY, to the extent permitted by law.  Type "show copying"
and "show warranty" for details.
This GDB was configured as "x86_64-redhat-linux-gnu".
For bug reporting instructions, please see:
<http://www.gnu.org/software/gdb/bugs/>...
Reading symbols from /root/helloworld...done.
(gdb) break 5
Breakpoint 1 at 0x4004cc: file helloworld.c, line 5.
(gdb) run
Starting program: /root/helloworld

Breakpoint 1, main () at helloworld.c:5
5           c="hello world!";
Missing separate debuginfos, use: debuginfo-install glibc-2.12-1.212.el6.x86_64
```


例(续)：检查数据

- 在程序中加入断点后，程序运行时会在断点处暂时停止，以便检查程序中的数据
- 通过检查数据可判断出许多中错误所在
- 例：
  - 检查常量 c 的值可输入命令 print c：

```
(gdb) print c
$1 = 0x0
```

    - 结果 $1=0x0 标明 c 所指向的地址为空，没有任何内容
  - 输入命令 next 继续单步执行

```
(gdb) next
6           printf("%s\n",c);
```

  - 输入 print c 再检查变量 c 的值：

```
(gdb) print c
$2 = 0x4005e8 "hello world!"
```

  - 结果 $2=0x4005e8 “hello world!” 标明 c 指向了地址 0x4005e8，该地址的内容转换为 ASCII 码结果为 hello world!
  - 键入 quit 退出调试

```
(gdb) quit
A debugging session is active.

        Inferior 1 [process 1427] will be killed.

Quit anyway? (y or n) y
```


关于 C 工程

- 需要阅读有关的 “GNU make” 手册