编译原理实验报告

实验二、 编译器认知实验报告

姓　　名：刘宇诺

班　　级：计算机 20-1

学　　号：20201210207

任课教师：韩连金

目录

一、实验目的

了解工业界常用的编译器 GCC，熟悉编译器的安装和使用过程，观察编译器工作过程中生成的中间文件的格式和内容，了解编译器的优化效果，为编译器的学习和构造奠定良好的基础。

二、实验内容

1.查看 GCC 编译器的版本

2.使用 GCC 编译器编译单个 C 程序文件，并运行生成的可执行文件

3.查看 GCC 编译器的预处理结果

4.查看 GCC 编译器生成的目标代码

5.比较 GCC 编译器在不同优化等级-O0 与-O2 下生成的目标代码的区别

三、实验环境

Linux 系统（Ubuntu、Debian、Gentoo 等系统均可）

gcc 7.3.0 或更高版本

四、数据准备

本次实验需要进行编译的文件均在 file 文件夹下。其中，需要进行编译的文件为 prime.c。

五、实验过程描述

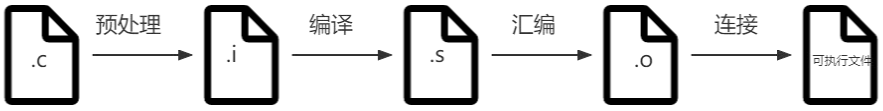
本次实验是在 Linux Ubuntu20.04 环境下进行。

1. 先进行查看 GCC 编译器的版本，在 Linux 终端通过 gcc --version 命令进行查看。

2. 用 gcc 编译器编译 C 文件，过程是 prime.c 源文件通过预处理生成 prime.i 文件，然后通过编译生成 prime.s 文件，再通过汇编生成 prime.o 二进制文件，机器可以直接运行二进制文件,然后在命令行通过 ./prime.o 可以运行生成的二进制文件。

预处理命令 gcc -E prime.c -o prime.i
编译命令 gcc -S prime.i -o prime.s
汇编命令 gcc -C prime.s -o prime.o

该过程可以通过如图解释：



3. 查看预处理的结果，在终端中通过 vim prime.i 命名查看 c 文件与处理后的结果。

4. 查看 GCC 编译器生成的目标代码，为了方便比较，分别生成-O0 和-O2 的两种编译后的代码

-O0 编译代码 gcc -O0 -S prime.i -o prime0.s
-O2 编译代码 gcc -O2 -S prime.i -o prime2.s

六、实验结果及分析

1.查看 GCC 编译器版本

版本为 9.3.0

```
lyn@iZhp31j9z6lt1wi9pdwbo7Z:~$ gcc --version
gcc (Ubuntu 9.3.0-10ubuntu2) 9.3.0
Copyright (C) 2019 Free Software Foundation, Inc.
This is free software; see the source for copying conditions.  There is NO
warranty; not even for MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.

lyn@iZhp31j9z6lt1wi9pdwbo7Z:~$
```

2.用 GCC 编译器编译 C 程序

1)进行预处理生成 prime.i 文件

```
lyn@iZhp31j9z6lt1wi9pdwbo7Z:~/xjuexp$ ls
prime.c
lyn@iZhp31j9z6lt1wi9pdwbo7Z:~/xjuexp$ gcc -E prime.c -o prime.i
lyn@iZhp31j9z6lt1wi9pdwbo7Z:~/xjuexp$ la
prime.c  prime.i
```

2)进行编译生成 prime.s 文件

```
lyn@iZhp31j9z6lt1wi9pdwbo7Z:~/xjuexp$ gcc -S prime.i -o prime.s
lyn@iZhp31j9z6lt1wi9pdwbo7Z:~/xjuexp$ ls
prime.c  prime.i  prime.s
lyn@iZhp31j9z6lt1wi9pdwbo7Z:~/xjuexp$
```

3)进行汇编生成 prime.o 文件

```
lyn@iZhp31j9z6lt1wi9pdwbo7Z:~/xjuexp$ gcc -C prime.s -o prime.o
lyn@iZhp31j9z6lt1wi9pdwbo7Z:~/xjuexp$ ls
prime.c  prime.i  prime.o  prime.s
lyn@iZhp31j9z6lt1wi9pdwbo7Z:~/xjuexp$ _
```

4)执行，prime 的功能是求 0-100 之间的质数。进行汇编后生成的.o 二进制文件，计算机是可以直接运行的。

```
lyn@iZhp31j9z6lt1wi9pdwbo7Z:~/xjuexp$ ls
prime.c  prime.i  prime.o  prime.s
lyn@iZhp31j9z6lt1wi9pdwbo7Z:~/xjuexp$ ./prime.o
2 3 5 7 11 13 17 19 23 29 31 37 41 43 47 53 59 61 67 71 73 79 83 89 97
lyn@iZhp31j9z6lt1wi9pdwbo7Z:~/xjuexp$
```

3.查看 GCC 编译器的预处理结果

可知预处理文件一共 1842 行

```
  1 # 1 "prime.c"
  2 # 1 "<built-in>"
  3 # 1 "<command-line>"
  4 # 31 "<command-line>"
  5 # 1 "/usr/include/stdc-predef.h" 1 3 4
  6 # 32 "<command-line>" 2
  7 # 1 "prime.c"
  8 # 1 "/usr/include/stdio.h" 1 3 4
  9 # 27 "/usr/include/stdio.h" 3 4
 10 # 1 "/usr/include/x86_64-linux-gnu/bits/libc-header-start.h" 1 3 4
 11 # 33 "/usr/include/x86_64-linux-gnu/bits/libc-header-start.h" 3 4
 12 # 1 "/usr/include/features.h" 1 3 4
 13 # 461 "/usr/include/features.h" 3 4
 14 # 1 "/usr/include/x86_64-linux-gnu/sys/cdefs.h" 1 3 4
 15 # 452 "/usr/include/x86_64-linux-gnu/sys/cdefs.h" 3 4
 16 # 1 "/usr/include/x86_64-linux-gnu/bits/wordsize.h" 1 3 4
 17 # 453 "/usr/include/x86_64-linux-gnu/sys/cdefs.h" 2 3 4
 18 # 1 "/usr/include/x86_64-linux-gnu/bits/long-double.h" 1 3 4
 19 # 454 "/usr/include/x86_64-linux-gnu/sys/cdefs.h" 2 3 4
 20 # 462 "/usr/include/features.h" 2 3 4
 21 # 485 "/usr/include/features.h" 3 4
 22 # 1 "/usr/include/x86_64-linux-gnu/gnu/stubs.h" 1 3 4
 23 # 10 "/usr/include/x86_64-linux-gnu/gnu/stubs.h" 3 4
 24 # 1 "/usr/include/x86_64-linux-gnu/gnu/stubs-64.h" 1 3 4
 25 # 11 "/usr/include/x86_64-linux-gnu/gnu/stubs.h" 2 3 4
 26 # 486 "/usr/include/features.h" 2 3 4
 27 # 34 "/usr/include/x86_64-linux-gnu/bits/libc-header-start.h" 2 3 4
 28 # 28 "/usr/include/stdio.h" 2 3 4
 29
 30
 31
 32
 33
 34 # 1 "/usr/lib/gcc/x86_64-linux-gnu/9/include/stddef.h" 1 3 4
 35 # 209 "/usr/lib/gcc/x86_64-linux-gnu/9/include/stddef.h" 3 4
 36
 37 # 209 "/usr/lib/gcc/x86_64-linux-gnu/9/include/stddef.h" 3 4
prime.i
```

```
1826 }
1827
1828 int main() {
1829     int *num = (int*) calloc(100, sizeof(int));
1830     int i, j;
1831     for(i = 2; i < 100; ++i) {
1832         if(num[i] != 0) continue;
1833         for(j = 2; multiply(i, j) < 100; ++j) {
1834             num[multiply(i, j)] = -1;
1835         }
1836     }
1837     for(i = 2; i < 100; ++i) {
1838         if(num[i] == 0) printf("%d ", i);
1839     }
1840     printf("\n");
1841     return 0;
1842 }
prime.i
```

4.查看 GCC 编译器生成的目标代码并与 O2 优化后的目标代码进行比较

```
lyn@iZhp31j9z6lt1wi9pdwbo7Z:~/xjuexp$ gcc -O0 -S prime.i -o prime0.s
lyn@iZhp31j9z6lt1wi9pdwbo7Z:~/xjuexp$ gcc -O2 -S prime.i -o prime2.s
lyn@iZhp31j9z6lt1wi9pdwbo7Z:~/xjuexp$ ls
prime0.s  prime2.s  prime.c  prime.i  prime.o  prime.s
lyn@iZhp31j9z6lt1wi9pdwbo7Z:~/xjuexp$
 0@iZhp31j9z6lt1wi9p
```

没有进行优化的 prime0.s 文件有 132 行

```
115 >---.ident>-"GCC: (Ubuntu 9.3.0-10ubuntu2) 9.3.0"
116 >---.section>---.note.GNU-stack,"",@progbits
117 >---.section>---.note.gnu.property,"a"
118 >---.align 8
119 >---.long>-- 1f - 0f
120 >---.long>-- 4f - 1f
121 >---.long>-- 5
122 0:
123 >---.string> "GNU"
124 1:
125 >---.align 8
126 >---.long>-- 0xc0000002
127 >---.long>-- 3f - 2f
128 2:
129 >---.long>-- 0x3
130 3:
131 >---.align 8
132 4:
prime0.s                                                    132,1         Bot
"prime0.s" 132L, 2004C
```

进行-O2 优化的 prime2.s 文件有 120 行

```
106 >---.align 8
107 >---.long>-- 1f - 0f
108 >---.long>-- 4f - 1f
109 >---.long>-- 5
110 0:
111 >---.string> "GNU"
112 1:
113 >---.align 8
114 >---.long>-- 0xc0000002
115 >---.long>-- 3f - 2f
116 2:
117 >---.long>-- 0x3
118 3:
119 >---.align 8
120 4:
prime2.s                                                    120,2         Bot
"prime2.s" 120L, 1787C
```

经过 O2 优化的编译后文件比不经过优化的编译后文件更加小，行数更少。

七、实验心得体会

通过此次试验，我理解了一个 C 文件是如果一步一步变成可执行文件的，一个 C 文件需要通过预处理、编译、汇编，最后连接生成一个可执行文件。

了解了如何通过 gcc 编译器一步步生成各个阶段的文件，通过实验也对这些命令使用进行了实践。

学会使用编译器的一些常用选项，以及一些参数优化，这些优化可以减少目标代码的执行时间和目标代码量等等。