PPL Homework —— Local or global

3170102492 夏豪诚

> Modified Date: 2019.11.12
>
> 编译工具：gcc (Ubuntu 9.2.1-9ubuntu2) 9.2.1 20191008
>
> 编译环境：Ubuntu 19.10

由于本次作业内容相对较少，在展示完成实验的过程和基本思路后，就课堂上提及的和自己查阅得到的与编译相关的知识进行了一定的展开，如有错误之处，望见谅。

1 实验过程

Step0 编译给出的c语言源文件，运行效果如下：

对照源程序内容，可以看到输出的为局部变量x的值。

```
//源程序 File：a.c
#include<stdio.h>

int x=3;

int main(void)
{
  int x=5;
  printf("x = %d\n", x);
}
```

Step1 对包含了C语言源代码的文件 *a.c* 进行预处理操作和编译操作。

此过程可以分步使用如下命令实现；

```
gcc -E a.c -o a.i #将a.c预处理输出a.i文件
gcc -S a.i #将预处理输出文件a.i汇编成a.s文件
```

也可以直接使用`gcc -S a.c -o a.s`,使得整个编译过程只执行到生成汇编文件。

```
percy@ubuntu ~/test vim a.c
percy@ubuntu ~/test gcc -S a.c -o a.s
```

Step2 查看汇编文件a.s的内容并对其中内容进行理解。

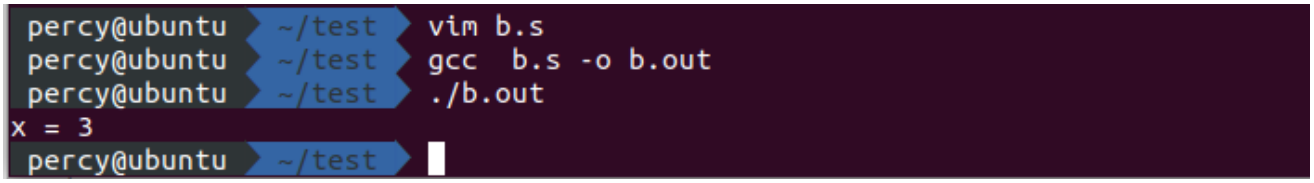
a.s 文件的具体内容如下（针对部分语句用 #开头字串 进行注释）：

```
    .file   "a.c"
    .text
    .globl  x #globl将x声明为外部可访问的标签
    .data
    .align 4 #指定数据的对齐方式,
    .type   x, @object #x的类型为对象
    .size   x, 4
x:
    .long   3         #将long类型的数据，内容为3，存储在以标号x:为起始地址的地方
    .section    .rodata #定义数据段，段名为 “.rodata”
.LC0:
    .string "x = %d\n"  #将字符串数据，内容为 x = %d\n,存储在以标号.LC0为起始地址的地方
    .text           #代码段
    .globl  main
    .type   main, @function #用来指定main符号的类型是函数类型
main:
.LFB0:
    .cfi_startproc #函数开始，用与.cfi_endproc相配套使用
    endbr64
    pushq   %rbp #扩展基址指针寄存器ebq的64位扩展，push保存保证函数能够正确返回
    .cfi_def_cfa_offset 16
    .cfi_offset 6, -16
    movq    %rsp, %rbp #把寄存器rsp的值赋给rbp，即用原先的栈顶作为新的栈底
    .cfi_def_cfa_register 6
    subq    $16, %rsp #把栈顶减去16
    movl    $5, -4(%rbp) #把数据5存到-4(%rbp)位置，即rbq地址减去4
    movl    -4(%rbp), %eax #把数据5传给eax寄存器
    movl    %eax, %esi #数据5再由eax传给esi
    leaq    .LC0(%rip), %rdi #载入字符串
    movl    $0, %eax
    call    printf@PLT #调用函数实现输出
    movl    $0, %eax
    leave
    .cfi_def_cfa 7, 8
    ret
    .cfi_endproc #函数结束，用与.cfi_startproc相配套使用
.LFE0: #编译信息相关
    .size   main, .-main
    .ident  "GCC: (Ubuntu 9.2.1-9ubuntu2) 9.2.1 20191008" #本次使用的编译工具及其版本
    .section    .note.GNU-stack,"",@progbits
    .section    .note.gnu.property,"a"
    .align 8
    .long   1f - 0f
    .long   4f - 1f

    .long   5
```

```
0:
    .string  "GNU" #GCC由GUN开发
1:
    .align 8
    .long    0xc0000002
    .long    3f - 2f
2:
    .long    0x3
```

Step3 修改汇编文件 *a.s* 得到 *b.s* , 实现输出从局部变量5，变为全局变量3。



如下为修改后得到的 *b.s* 文件的内容，我们只需要修改极少部分即可实现自己的目的：

```
    .file   "a.c"
    .text
    .globl  x
    .data
    .align 4
    .type   x, @object
    .size   x, 4
x:
    .long   3
    .section    .rodata
.LC0:
    .string "x = %d\n"
    .text
    .globl  main
    .type   main, @function
main:
.LFB0:
    .cfi_startproc
    endbr64
    pushq   %rbp
    .cfi_def_cfa_offset 16
    .cfi_offset 6, -16
    movq    %rsp, %rbp
    .cfi_def_cfa_register 6
    subq    $16, %rsp
    # modified part #
    movl    x(%rip), %eax #我们只需要修改此处传给eax寄存器的值变为代表全局变量x的符号x的值即可
    # modified part #
    movl    %eax, %esi
    leaq    .LC0(%rip), %rdi
    movl    $0, %eax
    call    printf@PLT
    movl    $0, %eax

    leave
```

```
    .cfi_def_cfa 7, 8
    ret
    .cfi_endproc
.LFE0:
    .size   main, .-main
    .ident  "GCC: (Ubuntu 9.2.1-9ubuntu2) 9.2.1 20191008"
    .section    .note.GNU-stack,"",@progbits
    .section    .note.gnu.property,"a"
    .align 8
    .long   1f - 0f
    .long   4f - 1f
    .long   5
0:
    .string  "GNU"
1:
    .align 8
    .long   0xc0000002
    .long   3f - 2f
2:
    .long   0x3
3:
    .align 8
4:
```

在进行上述修改之前，我还进行了如下的尝试：

```
    movl    $5, -4(%rbp) #把数据5存到-4(%rbp)位置，即rbq地址减去4
    movl    -4(%rbp), %eax #把数据5传给eax寄存器
    ==>修改为
    movl    x(%rip), -4(%rbp) #把数据5存到-4(%rbp)位置，即rbq地址减去4
    movl    -4(%rbp), %eax #把数据5传给eax寄存器
```

不加思考似乎还算合理，但是在生成可执行文件时就报了如下的错误：

```
percy@ubuntu ~/test gcc  b.s -o b.out
b.s: Assembler messages:
b.s:26: Error: too many memory references for `mov'
```

Error信息为：too many references for 'mov'，这是为什么呢？我们知道mov使用的规范中rbp，rip都可以作为偏移量被使用，但是在同一语句中同时出现就会出现问题。换言之"*You can't reference two memory locations in a single* `mov` *instruction*"。因此更改为`movl x(%rip), %eax`就可以避免这种问题。

2 内容相关

2.1 编译流程与相关概念

gcc的编译流程主要分为以下四步：

1. **预处理**：

   此步骤中C 编译器对各种预处理命令进行处理，包括头文件包含、宏定义的扩展、条件编译的选择等，在gcc中此时文件的后缀名为i。

2. **编译**

   将预处理得到的源代码文件，进行“翻译转换”，产生出机器语言的目标程序，得到机器语言的汇编文件，此时文件后缀名为s。

3. **汇编**

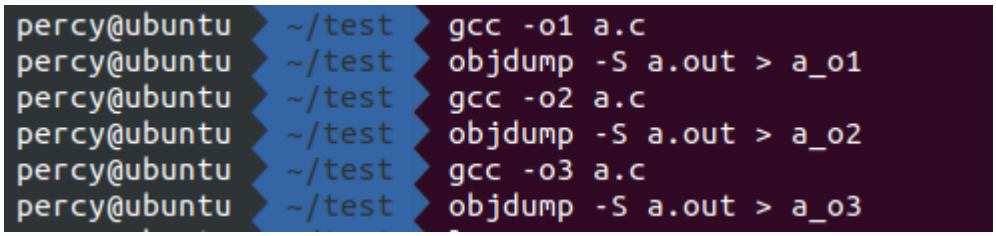
   将汇编代码通过汇编器翻译成了机器码，但是还不可以运行，此时后缀名为o。

4. **链接**，利用链接器（ld），处理可重定位文件，把各种符号引用和符号定义转换成为可执行文件中的合适信息，通常是虚拟地址，最后得到可执行的out文件。

2.2 GCC优化选项尝试

由于本次作业包含了观察程序汇编码，同时我们知道gcc有着编译优化选项，因此在此尝试对源程序采用不同优化选项后得到的汇编码有什么不同。

执行如下命令：



我们可以得到三个文件a_o1,a_o2和a_o3，从低到高对应着不同的优化等级，详细汇编码见附录，此处仅作简单比较,事实上，这三者的文件内容是完全一直的，但是我们可以看到相比于a.s中将基址寄存器减去16，在此处的代码中符合了实际上只有一个参数的情况，只是将基址寄存器减去了4。

```
0000000000001149 <main>:
    1149:   f3 0f 1e fa             endbr64
    114d:   55                      push   %rbp
    114e:   48 89 e5                mov    %rsp,%rbp
    1151:   48 83 ec 10             sub    $0x10,%rsp
    1155:   c7 45 fc 05 00 00 00    movl   $0x5,-0x4(%rbp)
    115c:   8b 45 fc                mov    -0x4(%rbp),%eax
    115f:   89 c6                   mov    %eax,%esi
    1161:   48 8d 3d 9c 0e 00 00    lea    0xe9c(%rip),%rdi        # 2004 <_IO_stdin_used+0x4>
    1168:   b8 00 00 00 00          mov    $0x0,%eax
    116d:   e8 de fe ff ff          callq  1050 <printf@plt>
    1172:   b8 00 00 00 00          mov    $0x0,%eax
    1177:   c9                      leaveq
    1178:   c3                      retq
    1179:   0f 1f 80 00 00 00 00    nopl   0x0(%rax)
```

附：a_o1 & a_o2 & a_o3内容

```
a.out:     file format elf64-x86-64


Disassembly of section .init:

0000000000001000 <_init>:
    1000:	f3 0f 1e fa           	endbr64
    1004:	48 83 ec 08           	sub    $0x8,%rsp
    1008:	48 8b 05 d9 2f 00 00  	mov    0x2fd9(%rip),%rax        # 3fe8 <__gmon_start__>
    100f:	48 85 c0              	test   %rax,%rax
    1012:	74 02                 	je     1016 <_init+0x16>
    1014:	ff d0                 	callq  *%rax
    1016:	48 83 c4 08           	add    $0x8,%rsp
    101a:	c3                    	retq

Disassembly of section .plt:

0000000000001020 <.plt>:
    1020:	ff 35 9a 2f 00 00     	pushq  0x2f9a(%rip)        # 3fc0 <_GLOBAL_OFFSET_TABLE_+0x8>
    1026:	f2 ff 25 9b 2f 00 00  	bnd jmpq *0x2f9b(%rip)        # 3fc8 <_GLOBAL_OFFSET_TABLE_+0x10>
    102d:	0f 1f 00              	nopl   (%rax)
    1030:	f3 0f 1e fa           	endbr64
    1034:	68 00 00 00 00        	pushq  $0x0
    1039:	f2 e9 e1 ff ff ff     	bnd jmpq 1020 <.plt>
    103f:	90                    	nop

Disassembly of section .plt.got:

0000000000001040 <__cxa_finalize@plt>:
    1040:	f3 0f 1e fa           	endbr64
    1044:	f2 ff 25 ad 2f 00 00  	bnd jmpq *0x2fad(%rip)        # 3ff8 <__cxa_finalize@GLIBC_2.2.5>
    104b:	0f 1f 44 00 00        	nopl   0x0(%rax,%rax,1)

Disassembly of section .plt.sec:

0000000000001050 <printf@plt>:
    1050:	f3 0f 1e fa           	endbr64
    1054:	f2 ff 25 75 2f 00 00  	bnd jmpq *0x2f75(%rip)        # 3fd0 <printf@GLIBC_2.2.5>
    105b:	0f 1f 44 00 00        	nopl   0x0(%rax,%rax,1)

Disassembly of section .text:

0000000000001060 <_start>:
    1060:	f3 0f 1e fa           	endbr64
    1064:	31 ed                 	xor    %ebp,%ebp
    1066:	49 89 d1              	mov    %rdx,%r9
    1069:	5e                    	pop    %rsi
    106a:	48 89 e2              	mov    %rsp,%rdx
    106d:	48 83 e4 f0           	and    $0xfffffffffffffff0,%rsp
    1071:	50                    	push   %rax
    1072:	54                    	push   %rsp
    1073:	4c 8d 05 76 01 00 00  	lea    0x176(%rip),%r8        # 11f0 <__libc_csu_fini>
```

```
    107a:	48 8d 0d ff 00 00 00    	lea    0xff(%rip),%rcx        # 1180 <__libc_csu_init>
    1081:	48 8d 3d c1 00 00 00    	lea    0xc1(%rip),%rdi        # 1149 <main>
    1088:	ff 15 52 2f 00 00       	callq  *0x2f52(%rip)        # 3fe0 <__libc_start_main@GLIBC_2.2.5>
    108e:	f4                      	hlt
    108f:	90                      	nop

0000000000001090 <deregister_tm_clones>:
    1090:	48 8d 3d 81 2f 00 00    	lea    0x2f81(%rip),%rdi        # 4018 <__TMC_END__>
    1097:	48 8d 05 7a 2f 00 00    	lea    0x2f7a(%rip),%rax        # 4018 <__TMC_END__>
    109e:	48 39 f8                	cmp    %rdi,%rax
    10a1:	74 15                   	je     10b8 <deregister_tm_clones+0x28>
    10a3:	48 8b 05 2e 2f 00 00    	mov    0x2f2e(%rip),%rax        # 3fd8 <_ITM_deregisterTMCloneTable>
    10aa:	48 85 c0                	test   %rax,%rax
    10ad:	74 09                   	je     10b8 <deregister_tm_clones+0x28>
    10af:	ff e0                   	jmpq   *%rax
    10b1:	0f 1f 80 00 00 00 00    	nopl   0x0(%rax)
    10b8:	c3                      	retq
    10b9:	0f 1f 80 00 00 00 00    	nopl   0x0(%rax)

00000000000010c0 <register_tm_clones>:
    10c0:	48 8d 3d 51 2f 00 00    	lea    0x2f51(%rip),%rdi        # 4018 <__TMC_END__>
    10c7:	48 8d 35 4a 2f 00 00    	lea    0x2f4a(%rip),%rsi        # 4018 <__TMC_END__>
    10ce:	48 29 fe                	sub    %rdi,%rsi
    10d1:	48 89 f0                	mov    %rsi,%rax
    10d4:	48 c1 ee 3f             	shr    $0x3f,%rsi
    10d8:	48 c1 f8 03             	sar    $0x3,%rax
    10dc:	48 01 c6                	add    %rax,%rsi
    10df:	48 d1 fe                	sar    %rsi
    10e2:	74 14                   	je     10f8 <register_tm_clones+0x38>
    10e4:	48 8b 05 05 2f 00 00    	mov    0x2f05(%rip),%rax        # 3ff0 <_ITM_registerTMCloneTable>
    10eb:	48 85 c0                	test   %rax,%rax
    10ee:	74 08                   	je     10f8 <register_tm_clones+0x38>
    10f0:	ff e0                   	jmpq   *%rax
    10f2:	66 0f 1f 44 00 00       	nopw   0x0(%rax,%rax,1)
    10f8:	c3                      	retq
    10f9:	0f 1f 80 00 00 00 00    	nopl   0x0(%rax)

0000000000001100 <__do_global_dtors_aux>:
    1100:	f3 0f 1e fa             	endbr64
    1104:	80 3d 09 2f 00 00 00    	cmpb   $0x0,0x2f09(%rip)        # 4014 <_edata>
    110b:	75 2b                   	jne    1138 <__do_global_dtors_aux+0x38>
    110d:	55                      	push   %rbp
    110e:	48 83 3d e2 2e 00 00    	cmpq   $0x0,0x2ee2(%rip)        # 3ff8 <__cxa_finalize@GLIBC_2.2.5>
    1115:	00
    1116:	48 89 e5                	mov    %rsp,%rbp
    1119:	74 0c                   	je     1127 <__do_global_dtors_aux+0x27>
    111b:	48 8b 3d e6 2e 00 00    	mov    0x2ee6(%rip),%rdi        # 4008 <__dso_handle>
    1122:	e8 19 ff ff ff          	callq  1040 <__cxa_finalize@plt>
    1127:	e8 64 ff ff ff          	callq  1090 <deregister_tm_clones>
    112c:	c6 05 e1 2e 00 00 01    	movb   $0x1,0x2ee1(%rip)        # 4014 <_edata>
    1133:	5d                      	pop    %rbp
    1134:	c3                      	retq

    1135:	0f 1f 00                	nopl   (%rax)
```

```
    1138:   c3                      retq
    1139:   0f 1f 80 00 00 00 00    nopl   0x0(%rax)

0000000000001140 <frame_dummy>:
    1140:   f3 0f 1e fa             endbr64
    1144:   e9 77 ff ff ff          jmpq   10c0 <register_tm_clones>

0000000000001149 <main>:
    1149:   f3 0f 1e fa             endbr64
    114d:   55                      push   %rbp
    114e:   48 89 e5                mov    %rsp,%rbp
    1151:   48 83 ec 10             sub    $0x10,%rsp
    1155:   c7 45 fc 05 00 00 00    movl   $0x5,-0x4(%rbp)
    115c:   8b 45 fc                mov    -0x4(%rbp),%eax
    115f:   89 c6                   mov    %eax,%esi
    1161:   48 8d 3d 9c 0e 00 00    lea    0xe9c(%rip),%rdi        # 2004 <_IO_stdin_used+0x4>
    1168:   b8 00 00 00 00          mov    $0x0,%eax
    116d:   e8 de fe ff ff          callq  1050 <printf@plt>
    1172:   b8 00 00 00 00          mov    $0x0,%eax
    1177:   c9                      leaveq
    1178:   c3                      retq
    1179:   0f 1f 80 00 00 00 00    nopl   0x0(%rax)

0000000000001180 <__libc_csu_init>:
    1180:   f3 0f 1e fa             endbr64
    1184:   41 57                   push   %r15
    1186:   4c 8d 3d 2b 2c 00 00    lea    0x2c2b(%rip),%r15        # 3db8 <__frame_dummy_init_array_entry>
    118d:   41 56                   push   %r14
    118f:   49 89 d6                mov    %rdx,%r14
    1192:   41 55                   push   %r13
    1194:   49 89 f5                mov    %rsi,%r13
    1197:   41 54                   push   %r12
    1199:   41 89 fc                mov    %edi,%r12d
    119c:   55                      push   %rbp
    119d:   48 8d 2d 1c 2c 00 00    lea    0x2c1c(%rip),%rbp        # 3dc0 <__init_array_end>
    11a4:   53                      push   %rbx
    11a5:   4c 29 fd                sub    %r15,%rbp
    11a8:   48 83 ec 08             sub    $0x8,%rsp
    11ac:   e8 4f fe ff ff          callq  1000 <_init>
    11b1:   48 c1 fd 03             sar    $0x3,%rbp
    11b5:   74 1f                   je     11d6 <__libc_csu_init+0x56>
    11b7:   31 db                   xor    %ebx,%ebx
    11b9:   0f 1f 80 00 00 00 00    nopl   0x0(%rax)
    11c0:   4c 89 f2                mov    %r14,%rdx
    11c3:   4c 89 ee                mov    %r13,%rsi
    11c6:   44 89 e7                mov    %r12d,%edi
    11c9:   41 ff 14 df             callq  *(%r15,%rbx,8)
    11cd:   48 83 c3 01             add    $0x1,%rbx
    11d1:   48 39 dd                cmp    %rbx,%rbp
    11d4:   75 ea                   jne    11c0 <__libc_csu_init+0x40>
    11d6:   48 83 c4 08             add    $0x8,%rsp
    11da:   5b                      pop    %rbx

    11db:   5d                      pop    %rbp
```

```
    11dc:   41 5c                   pop    %r12
    11de:   41 5d                   pop    %r13
    11e0:   41 5e                   pop    %r14
    11e2:   41 5f                   pop    %r15
    11e4:   c3                      retq
    11e5:   66 66 2e 0f 1f 84 00    data16 nopw %cs:0x0(%rax,%rax,1)
    11ec:   00 00 00 00

00000000000011f0 <__libc_csu_fini>:
    11f0:   f3 0f 1e fa             endbr64
    11f4:   c3                      retq

Disassembly of section .fini:

00000000000011f8 <_fini>:
    11f8:   f3 0f 1e fa             endbr64
    11fc:   48 83 ec 08             sub    $0x8,%rsp
    1200:   48 83 c4 08             add    $0x8,%rsp
    1204:   c3                      retq
```