分析报告

文红兵 20214590

在Linux系统中使用GCC编译器，使用以下C代码编写一个简单的“Hello　World!”程序：

```c
#include <stdio.h>

int main() {
  printf("Hello world!\n");
  return 0;
}
```

保存文件并将其命名为`hello.c` 。使用以下命令将其编译为ELF格式的可执行程序hello：

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

可以使用以下命令查看汇编代码：

```
objdump -d hello
```

以下是终端输出

```

hello:     文件格式 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__@Base>
    100f:   48 85 c0                test   %rax,%rax
    1012:   74 02                   je     1016 <_init+0x16>
    1014:   ff d0                   call   *%rax
    1016:   48 83 c4 08             add    $0x8,%rsp
    101a:   c3                      ret

Disassembly of section .plt:

0000000000001020 <.plt>:                                            # 3fc0 <_GLOBAL_OFFSET_TABLE_+0x8>
    1020:   ff 35 9a 2f 00 00       push   0x2f9a(%rip)                # 3fc8 <_GLOBAL_OFFSET_TABLE_+0x10>
    1026:   f2 ff 25 9b 2f 00 00    bnd jmp *0x2f9b(%rip)
    102d:   0f 1f 00                nopl   (%rax)
    1030:   f3 0f 1e fa             endbr64
    1034:   68 00 00 00 00          push   $0x0
    1039:   f2 e9 e1 ff ff ff       bnd jmp 1020 <_init+0x20>
    103f:   90                      nop

Disassembly of section .plt.got:

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

Disassembly of section .plt.sec:

0000000000001050  <puts@plt>:                                     # 3fd0 <puts@GLIBC_2.2.5>
    1050:   f3 0f 1e fa             endbr64
    1054:   f2 ff 25 75 2f 00 00    bnd jmp *0x2f75(%rip)
    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:   45 31 c0                xor    %r8d,%r8d
    1076:   31 c9                   xor    %ecx,%ecx
    1078:   48 8d 3d ca 00 00 00    lea    0xca(%rip),%rdi        # 1149 <main>
    107f:   ff 15 53 2f 00 00       call   *0x2f53(%rip)        # 3fd8 <__libc_start_main@GLIBC_2.34>
    1085:   f4                      hlt
    1086:   66 2e 0f 1f 84 00 00    cs nopw 0x0(%rax,%rax,1)
    108d:   00 00 00

0000000000001090  <deregister_tm_clones>:
    1090:   48 8d 3d 79 2f 00 00    lea    0x2f79(%rip),%rdi        # 4010 <__TMC_END__>
    1097:   48 8d 05 72 2f 00 00    lea    0x2f72(%rip),%rax        # 4010 <__TMC_END__>
    109e:   48 39 f8                cmp    %rdi,%rax
    10a1:   74 15                   je     10b8 <deregister_tm_clones+0x28>
    10a3:   48 8b 05 36 2f 00 00    mov    0x2f36(%rip),%rax        # 3fe0
<_ITM_deregisterTMCloneTable@Base>
    10aa:   48 85 c0                test   %rax,%rax
    10ad:   74 09                   je     10b8 <deregister_tm_clones+0x28>
    10af:   ff e0                   jmp    *%rax
    10b1:   0f 1f 80 00 00 00 00    nopl   0x0(%rax)
    10b8:   c3                      ret
    10b9:   0f 1f 80 00 00 00 00    nopl   0x0(%rax)

00000000000010c0  <register_tm_clones>:
    10c0:   48 8d 3d 49 2f 00 00    lea    0x2f49(%rip),%rdi        # 4010 <__TMC_END__>
    10c7:   48 8d 35 42 2f 00 00    lea    0x2f42(%rip),%rsi        # 4010 <__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@Base>
    10eb:   48 85 c0                test   %rax,%rax
    10ee:   74 08                   je     10f8 <register_tm_clones+0x38>
    10f0:   ff e0                   jmp    *%rax
    10f2:   66 0f 1f 44 00 00       nopw   0x0(%rax,%rax,1)
    10f8:   c3                      ret
    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 05 2f 00 00 00    cmpb   $0x0,0x2f05(%rip)        # 4010 <__TMC_END__>
    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          call   1040 <__cxa_finalize@plt>
    1127:   e8 64 ff ff ff          call   1090 <deregister_tm_clones>
    112c:   c6 05 dd 2e 00 00 01    movb   $0x1,0x2edd(%rip)        # 4010 <__TMC_END__>
    1133:   5d                      pop    %rbp
    1134:   c3                      ret
    1135:   0f 1f 00                nopl   (%rax)
    1138:   c3                      ret
    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          jmp    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 8d 05 ac 0e 00 00    lea    0xeac(%rip),%rax        # 2004 <_IO_stdin_used+0x4>
    1158:   48 89 c7                mov    %rax,%rdi
    115b:   e8 f0 fe ff ff          call   1050 <puts@plt>
    1160:   b8 00 00 00 00          mov    $0x0,%eax
    1165:   5d                      pop    %rbp
    1166:   c3                      ret

Disassembly of section .fini:
```

```
0000000000001168 <_fini>:
    1168:   f3 0f 1e fa             endbr64
    116c:   48 83 ec 08             sub    $0x8,%rsp
    1170:   48 83 c4 08             add    $0x8,%rsp
    1174:   c3                      ret
```

该输出包含以下部分：

- `.init`：函数 `_init` 的汇编代码。这是在程序开始时执行的函数，主要是加载共享库等操作。
- `.plt`：过程链接表（Procedure Linkage Table）的汇编代码。它用于动态链接，以便在运行时解析符号。
- `.text`：主要的代码段，包括`_start`和`main`函数，以及其他自定义函数。
- `.fini`：函数 `_fini` 的汇编代码。这是在程序结束时执行的函数，主要是卸载共享库等操作。

可执行程序的基本结构如下：

- .ELF头部：文件格式、操作系统、程序入口地址等信息。
- .text段：程序的代码段，包含了程序的指令。
- .data段：程序的数据段，包含了程序中定义的全局变量和静态变量。
- .bss段：程序的未初始化数据段，包含了程序中定义的未初始化的全局变量和静态变量。
- .rodata段：只读数据段，包含了程序中定义的只读变量和常量。
- .symtab段：符号表，包含了程序中定义和引用的符号（函数名、变量名等）的信息。
- .strtab段：字符串表，包含了程序中使用的字符串。
- .rela.text段：重定位表，包含了程序中需要重定位的地址信息。
- .plt段：过程链接表，包含了程序中需要动态链接的函数的信息。
- .got段：全局偏移表，包含了程序中需要动态链接的函数的地址信息。
- .dynamic段：动态链接信息表，包含了程序中需要动态链接的库的信息。
- .comment段：注释信息，包含了程序的版本、作者等信息。
- .debug段：调试信息，包含了程序的调试信息。
- .eh_frame段：异常处理信息，包含了程序的异常处理信息。