《Linux 和 Windows 可执行程序的基本结构》分析报告

一、任务目标

1.请在 Linux/ Windows 系统中使用 GCC 工具链，使用 C 语言写一个简单输出“hello world!”的程序，并编译为 ELF(.o)格式的可执行程序

2. 观察汇编代码的格式，分析可执行程序的基本结构；

二、实现步骤

在指定的路径创建一个新的 c 语言文件，内容如下：

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

然后将其保存，编译运行，在命令行中，执行以下语句：

```
objdump -d D:\zjw\demo\cpp\output\helloworld.exe > hellow
orld.asm
```

三、运行结果

终端输出：

Hello, world!

得到的汇编语言程序：

```
 .file "hello.cpp"
 .def ___main; .scl 2; .type 32; .endef
 .section .rdata,"dr"
LC0:
 .ascii "Hello, world!\0"
 .text
 .globl _main
 .def _main; .scl 2; .type 32; .endef
_main:
LFB12:
 .cfi_startproc
 pushl %ebp
 .cfi_def_cfa_offset 8
 .cfi_offset 5, -8
```

```
    movl %esp, %ebp
    .cfi_def_cfa_register 5
    andl $-16, %esp
    subl $16, %esp
    call ___main
    movl $LC0, (%esp)
    call _puts
    movl $0, %eax
    leave
    .cfi_restore 5
    .cfi_def_cfa 4, 4
    ret
    .cfi_endproc
LFE12:
    .ident "GCC: (MinGW.org GCC-6.3.0-1) 6.3.0"
    .def _puts; .scl 2; .type 32; .endef
```

四、结果分析

代码中，"Hello, world!\0"字段用..section .rdata 定义，是一个只读数据段，其中包含了字符串 ，使用标签 LC0 进行引用。.text 指示接下来的指令属于代码段。

在代码段中，_main 标签表示程序的入口点，即 main 函数。

在 _main 函数内部，首先进行了一系列的准备工作：

pushl %ebp：将栈底指针保存到栈上。

movl %esp, %ebp：将栈顶指针保存到基址指针中。

andl $-16, %esp：将栈顶指针向下对齐到 16 字节边界。

subl $16, %esp：为局部变量在栈上分配空间。

call ___main：调用 ___main 函数，通常用于初始化 C++ 运行时环境。

接下来，将字符串 "Hello, world!\0" 的地址推送到栈上。

调用 _puts 函数，该函数通常用于输出字符串到标准输出设备。

leave 指令等效于 mov %ebp, %esp 后的 pop %ebp，用于恢复栈指针和栈底指针。

ret 指令用于返回到调用者。

综合以上，对于一个可执行程序，可执行程序的基本结构可以总结如下：

1.文件信息：包括源文件的名称和其他可能的元数据，例如编译器版本信息等。

2.数据段：存放程序中的静态数据，如字符串常量等。

3.代码段：包含程序的实际执行代码。其中包括程序的入口点，通常是 main 函数的汇编版本，包括对栈的管理（如保存和恢复栈帧）、调用其他函数以及执行程序逻辑的指令序列。

4.其他信息：可能包括编译器版本信息、符号定义等元数据。

其中，代码段是程序执行的核心，数据段存放着程序需要的静态数据，而文件信息和其他信息则提供了一些辅助性的信息