在深入了解这个项目之前我们需要先了解一些简单的储备知识。

**储备知识**

**简述页式内存机制**

当需要将一定内容段存储在内存中时，因为段的长短不定，在分配内存时，可能会发生内存中的空闲区域小于要加载的段，或者空闲区域远远大于要加载的段。在前一种情况下，我们就需要额外的需求空闲区域；在最后一种情况下，即使我们分配成功，也会浪费一大片内存空间。为了解决这一问题，目前很多处理器已经引入了分页机制。分页功能从整体上来说，是用长度固定的页来代替长度不一定的段，借此来解决因为段长度不同而带来的内存空间管理问题。

一旦决定采用页式内存管理，就应当把4GB内存分成大小相同的页。页的最小单位是4KB，也就是4096字节，用十六进制表示的话就是0x1000.因此第一个页的物理地址就是0x00000000，第二个页的物理地址是0x00001000，第三个就是0x00000000……最后一个页就是0xfffff000。这样4GB内存就被划分为1048576（0x100000）个页。

由于32为CPU的寻址能力为4GB，但是有些用户的物理内存达不到这个值。于是操作系统和CPU的内存机制共同作用为用户提供虚拟的内存管理机制，采用的就是分页机制实现。这样用户就会感觉自己好像在使用4GB的内存。

**PE文件**

PE文件的全称是Portable Executable，也就是可移植的可执行文件，我们平时常常看到的诸如EXE，DLLs，COM，对象代码 以及一些其他被用在32位和64位Windows操作系统中的文件都是PE文件，换句话说，PE文件是微软Windows操作系统上的程序文件，它其实是一种数据结构， 里面包含了Windows 操作系统加载器管理可执行代码所需要的必要信息。PE文件使用的是一个平面地址空间，所有代码和数据都被合并在一起，组成一个很大的结构。

PE文件是Unix COFF文件的变更版本，它继承了很多COFF文件的特点，所以想了解PE文件就得稍微了解COFF文件。COFF文件-通用对象文件格式（Common Object File Format），是一种很流行的对象文件格式（主意：这里不说它是“目标”文件是为了和编译器产生的文件相区别，因为这种格式不只用于目标文件，库文件、可执行文件也经常是这种格式）。COFF文件一共有8中数据结构，自上而下分别为文件头（File Header）、可选头（Optional Header）、段落头（Section Header）、段落数据（Section Data）、重定位表（Relocation Directives）、行号表（Line Numbers）、符号表（Symbol Table）、字符串表（String Table）。这些数据结构被PE文件很好地继承，并且同时进行了适当的扩展，下面将着重介绍PE文件的层次结构。

**PE文件层次结构**

**DOS 头结构**

PE文件格式以DOS头这一结构开始，它既是头文件也是一段可执行代码，它存在的主要目的只是为了兼容早期的DOS操作系统，DOS头是由IMAGE\_DOS\_HEADER结构体来定义，以下为其结构定义代码：

|  |  |
| --- | --- |
| Struct IMAGE\_DOS\_HEADER { | |
|  | public UInt16 e\_magic; // 魔术数字 |
|  | public UInt16 e\_cblp; // 文件最后页的字节数 |
|  | public UInt16 e\_cp; // 文件页数 |
|  | public UInt16 e\_crlc; // 重定义元素个数 |
|  | public UInt16 e\_cparhdr; // 头部尺寸，以段落为单位 |
|  | public UInt16 e\_minalloc; // 所需的最小附加段 |
|  | public UInt16 e\_maxalloc; // 所需的最大附加段 |
|  | public UInt16 e\_ss; // 初始的SS值（相对偏移量） |
|  | public UInt16 e\_sp; // 初始的SP值 |
|  | public UInt16 e\_csum; // 校验和 |
|  | public UInt16 e\_ip; // 初始的IP值 |
|  | public UInt16 e\_cs; // 初始的CS值（相对偏移量） |
|  | public UInt16 e\_lfarlc; // 重分配表文件地址 |
|  | public UInt16 e\_ovno; // 覆盖号 |
|  | public UInt16[] e\_res; // 保留字，8字节 |
|  | public UInt16 e\_oemid; // OEM标识符（相对e\_oeminfo） |
|  | public UInt16 e\_oeminfo; // OEM信息 |
|  | public UInt16[] e\_res2; // 保留字，20字节 |
|  | public UInt32 e\_lfanew; // 新exe头部的文件地址 |

此结构体中有两个字段需要特别注意，一个是e\_magic，一个是e\_lfanew字段。e\_magic字段是一个DOS可执行文件的标识符，分别为4Dh和5Ah，它占用两个字节，保存着的字符是”MZ“，并且它在Winnt.h头文件中有一个宏定义，定义内容如下：

#define IMAGE\_DOS\_SIGNATURE 0x5A4D //MZ

正因为这样，有时候也会称呼DOS头为DOS MZ Header。至于e\_lfanew，它是一个双字数据，为PE头的离文件头部的偏移量。Windows加载器可以通过它跳过DOS Stub部分直接找到PE头。DOS头后面还会跟着一个DOS Stub的数据，是链接器链接执行文件的时候加入的部分数据，一般情况下，如果在DOS系统中执行PE文件，会根据这部分内容显示：”This program must be run under Microsoft Windows“。这个可以通过修改链接器的设置来修改成自定义的数据。————TO DO Later