А.В. Макаров, С.Ю. Скоробогатов, А.М. Чеповский

Учебный курс “CIL и системное программирование в Microsoft .NET”

Лекция 4.

Формат исполняемых файлов

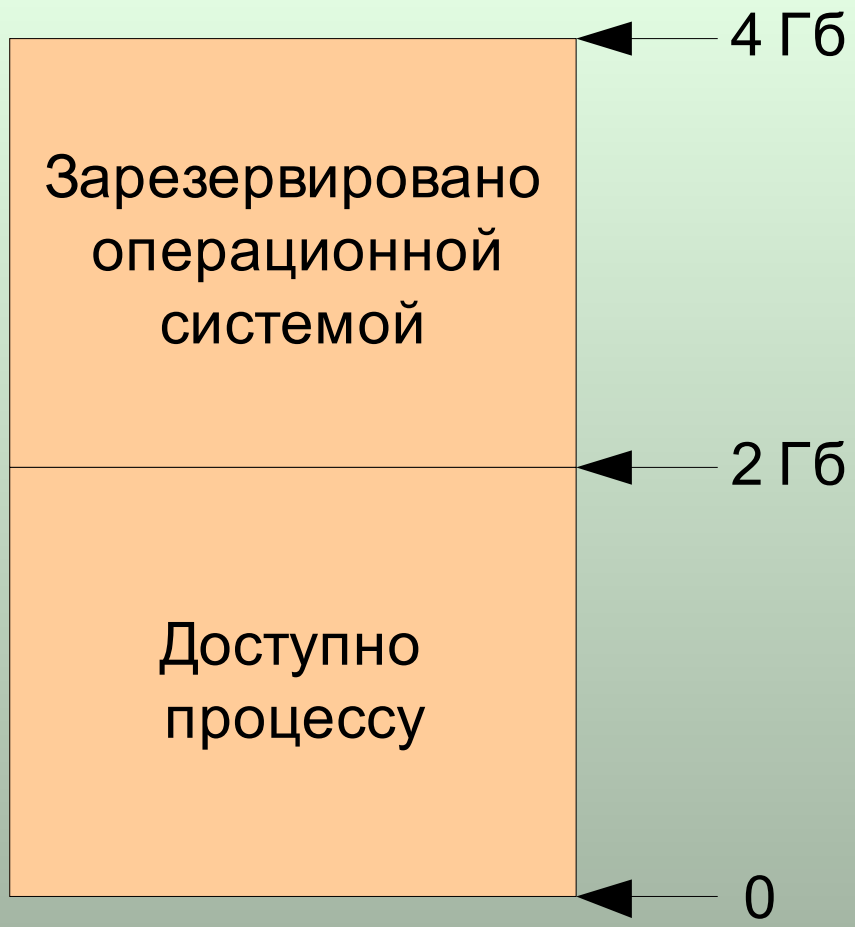
Введение

- Исполняемый файл – это файл, который может быть загружен в память загрузчиком операционной системы и затем исполнен
- Формат Portable Executable (PE) – основной формат для хранения исполняемых файлов в операционной системе Windows
- Большая часть разработчиков Windows NT пришли из DEC, поэтому PE – развитие формата COFF (Common Object File Format) из VAX/VMS
- Другие форматы – New Executable (NE) и Linear Executable (LE)
- Аппаратные платформы, на которых использовался формат PE: Intel x86, MIPS R4000, DEC Alpha, PowerPC
- PE обобщен для использования на 64-разрядных платформах (PE32+)

4.1. Управление памятью в Windows

- Управление памятью в Windows NT/2k/XP/2k3 осуществляет менеджер виртуальной памяти (virtual-memory manager)
- Физическая память делится на физические страницы размером в 4096 байт (8192 байта на 64-разрядных архитектурах)
- Физические страницы могут вытесняться в один или несколько файлов подкачки (page files). Вытесненные на жесткий диск страницы затем могут быть загружены обратно в память, если возникнет необходимость

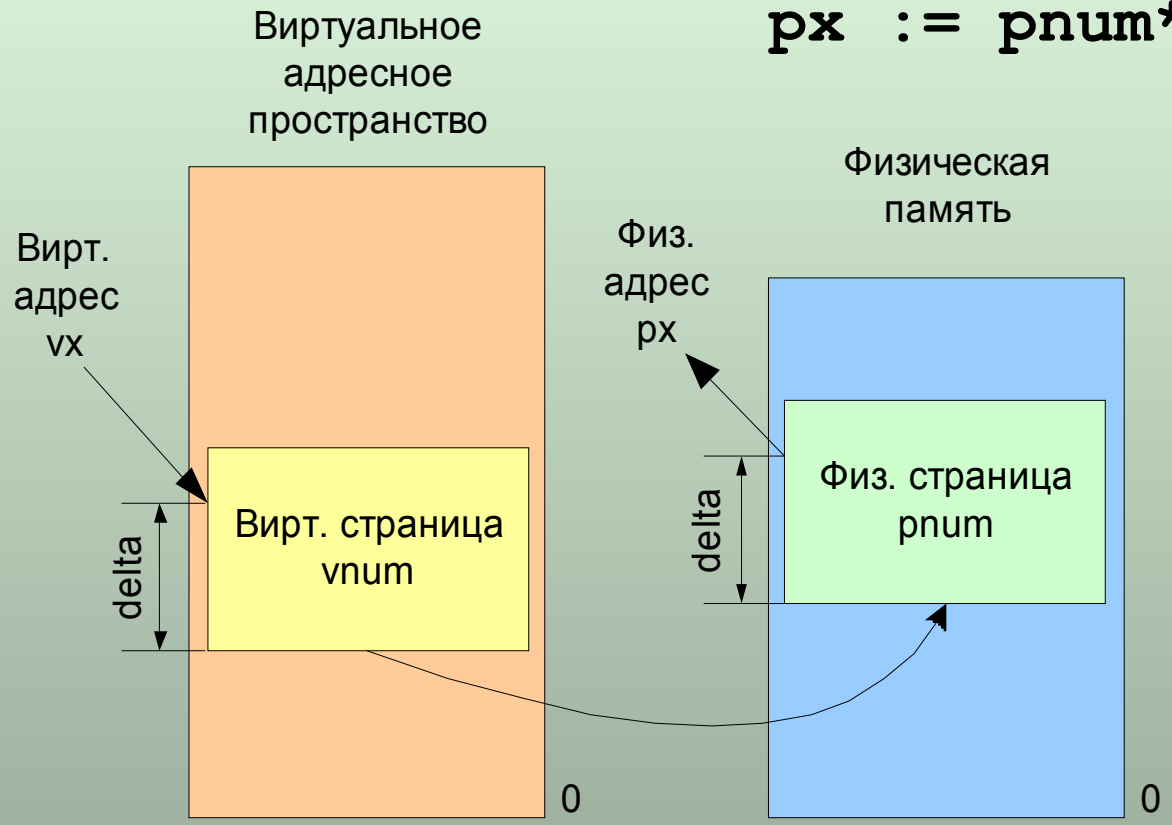
4.1.1. Виртуальное адресное пространство процесса



- Каждый процесс запускается в своем виртуальном адресном пространстве размером в 4 Гб
- Виртуальное адресное пространство делится на виртуальные страницы размером в 4096 байт (или 8192 байта на 64-разр. арх.)
- Процесс для работы с памятью использует не реальные адреса физической памяти, а так называемые виртуальные адреса

Перевод виртуального адреса в физический адрес

```
vnum := vx div 4096;
delta := vx mod 4096;
px := pnum*4096+delta;
```



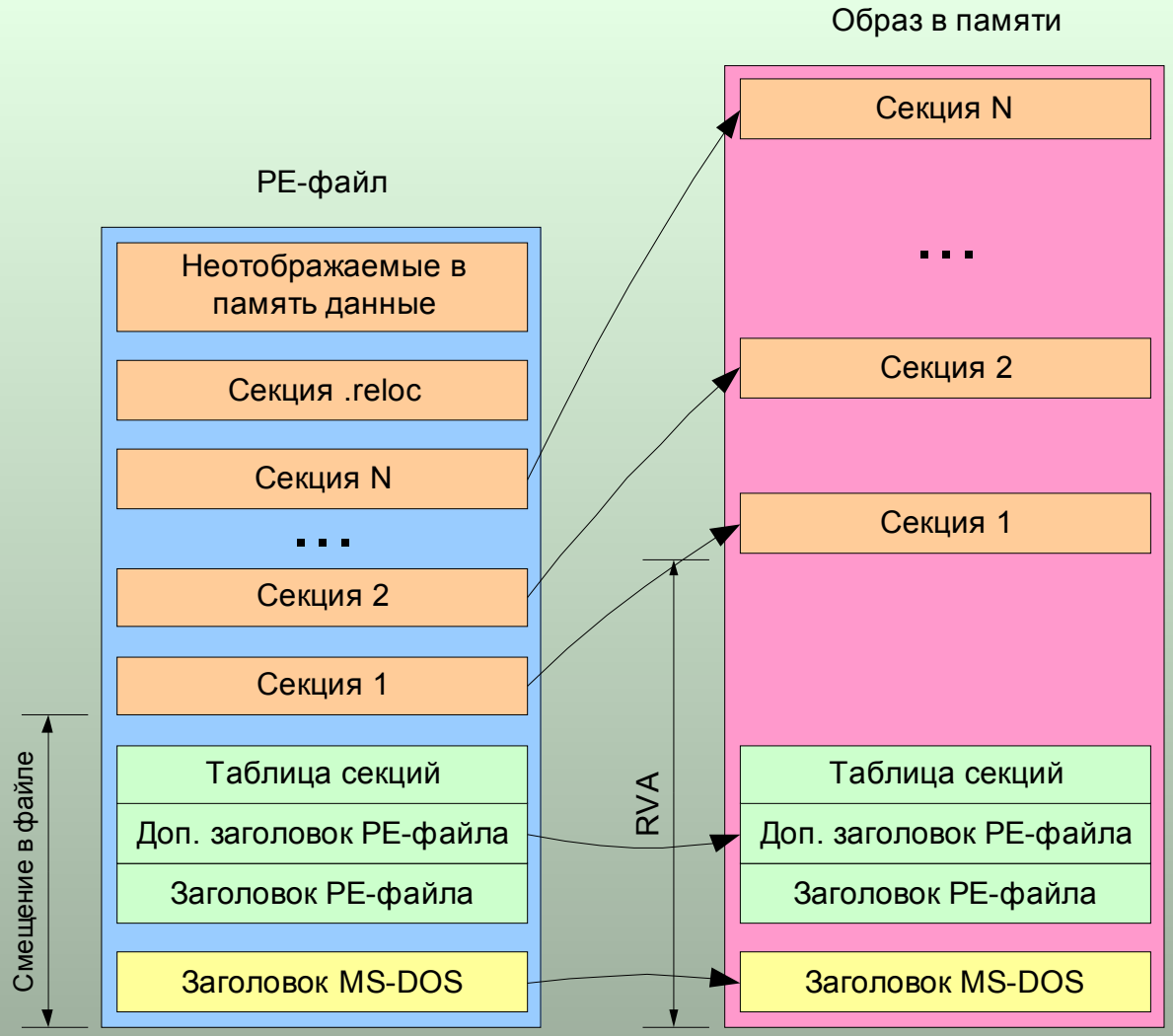
Выводы

- Процессы изолированы друг от друга. Один процесс не может обратиться к памяти другого процесса
- Передача виртуальных адресов между процессами совершенно бессмысленна. Один и тот же виртуальный адрес в адресных пространствах разных процессов соответствует разным физическим адресам
- Процессы используют преимущества плоской адресации памяти. Виртуальный адрес представляет собой 32-разрядное целое значение, что позволяет легкую реализацию адресной арифметики

4.1.2. Отображаемые в память файлы

- Отображаемые в память файлы (memory-mapped files) – это мощная возможность ОС, позволяющая приложениям осуществлять доступ к файлам на диске тем же самым способом, каким осуществляется доступ к динамической памяти, то есть через указатели
- Смысл отображения файла в память заключается в том, что содержимое файла (или часть содержимого) отображается в некоторый диапазон виртуального адресного пространства процесса, после чего обращение по какому-либо адресу из этого диапазона означает обращение к файлу на диске

4.2. Обзор структуры PE-файла



- PE-файл, загруженный в оперативную память для исполнения, почти ничем не отличается от своего представления на диске

RVA и смещения в PE-файле

- RVA некоторого элемента PE-файла – это разность виртуального адреса этого элемента и базового адреса, по которому PE-файл загружен в память
  - Например, если файл загружен по адресу 0x400000, и некоторый элемент в нем располагается по адресу 0x402000, то RVA этого элемента равен (0x402000 – 0x400000) = 0x2000
- Смещение элемента в файле представляет собой количество байт, которое надо отсчитать от начала файла, чтобы попасть на начало элемента

4.2.1. Секции

- Секция в PE-файле представляет либо код, либо некоторые данные (глобальные переменные, таблицы импорта и экспорта, ресурсы, таблица релокаций)
- Исполняемый файл всегда содержит, по крайней мере, одну секцию, в которой помещен исполняемый код
- Выравнивание секций в исполняемом файле на диске и в образе файла в памяти чаще всего отличается

4.2.2. Выбор базового адреса образа PE-файла в памяти

- В заголовках файла присутствует поле ImageBase, содержащее предпочтительное значение базового адреса. Exe-файлы всегда загружаются по этому адресу
- Динамическая библиотека загружается в адресное пространство уже существующего процесса, и хотя dll-файл тоже содержит некоторое значение в поле ImageBase, очень часто может так получиться, что этот адрес уже занят чем-то другим
- Загрузчик исправляет абсолютные адреса в образе PE-файла в соответствии с таблицей релокаций

4.2.3. Импорт функций

- Секция ".idata" описывает функции, который PE-файл импортирует из динамических библиотек
- В процессе загрузки программы осуществляется связывание (binding) импортируемых функций
- Сборки .NET, как правило, импортируют функцию _CorExeMain (или _CorDllMain) из динамической библиотеки mscoree.dll. При запуске сборки .NET управление сразу же передается этой функции, которая запускает Common Language Runtime, осуществляющий JIT-компиляцию программы и контролирующий в дальнейшем ее выполнение

4.2.4. Экспорт функций

- Необходимость в экспорте функций возникает только тогда, когда сборка .NET должна использоваться обычной программой Windows, код которой не управляется средой выполнения .NET
- Информация об экспортируемых функциях хранится внутри PE-файла в специальной секции ".edata". При этом каждой функции присваивается уникальный номер, и с этим номером связывается RVA тела функции, и, возможно, имя функции

4.3. Заголовки

- Заголовок MS-DOS
- Заголовок PE-файла
- Дополнительный заголовок PE-файла
- Массив заголовков секций

4.3.1. Заголовок MS-DOS

- Небольшая (128 байт) программа, записанная в формате исполняемых файлов MS-DOS. Выводит на экран сообщение "This program cannot be run in DOS mode".
- Заголовок начинается с сигнатуры "MZ" (инициалы Марка Збиковски). Ни одна инструкция процессоров семейства Intel x86 с не начинается с "MZ"
- Сразу после заголовка MS-DOS следует сигнатура PE-файла, состоящая из четырех байт: 0x50, 0x45, 0x00 и 0x00 (в строковом представлении она выглядит как "PE\0\0")

4.3.2. Заголовок PE-файла

- Основные поля:

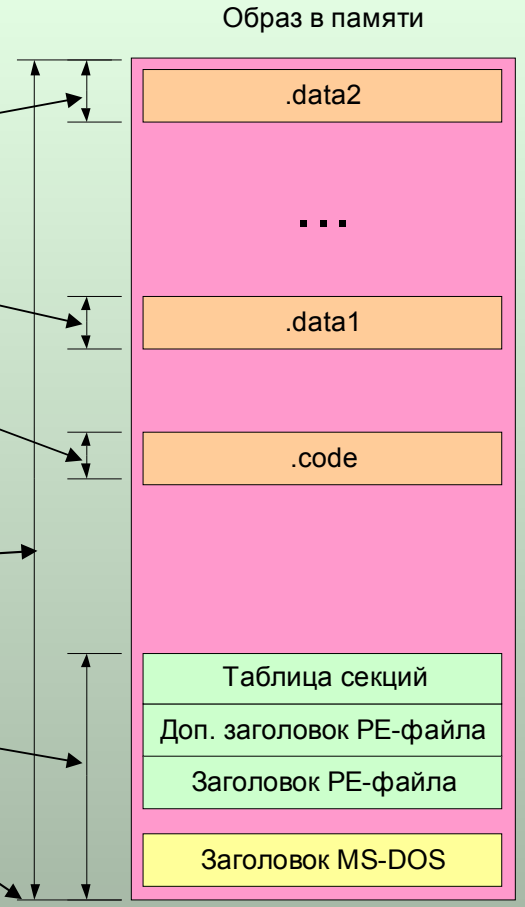
```
struct PeHeader
{
  short Machine; // 0x14c
  short NumberOfSections;
  long TimeDateStamp;
    ...
  short OptionalHeaderSize;
  short Characteristics;
};
```

4.3.3. Дополнительный заголовок PE-файла

```
struct PeOptionalHeader
{
    /* Standard fields */
    ...
    long CodeSize;
    long InitializedDataSize;
    long UninitializedDataSize;
    long EntryPointRVA;
    ...

    /* NT-Specific Fields */
    long ImageBase;
    long SectionAlignment;
    long FileAlignment;
    ...
    long ImageSize;
    long HeaderSize;
    ...

    /* Data directories */
    struct DataDirectory dirs[16];
};
```



Директории данных

- Местонахождение некоторых важных структур данных в образе PE-файла задается в директориях данных (Data Directories). Каждая директория содержит RVA и размер соответствующей структуры
- Для сборок .NET важны 4 из 16 директорий данных:
  - Директория импорта (номер 2). Указывает на данные об импортируемых из динамических библиотек функциях (другими словами, указывает на секцию ".idata").
  - Директория релокаций (номер 6). Указывает на таблицу релокаций.
  - Директория таблицы адресов импорта (номер 13). В некотором смысле дублирует директорию импорта, указывая на таблицу адресов импорта.
  - Директория заголовка CLI (номер 15). Указывает на заголовок, описывающий метаданные сборки .NET.

4.3.4. Заголовки секций

- Заголовок секции содержит следующие поля:

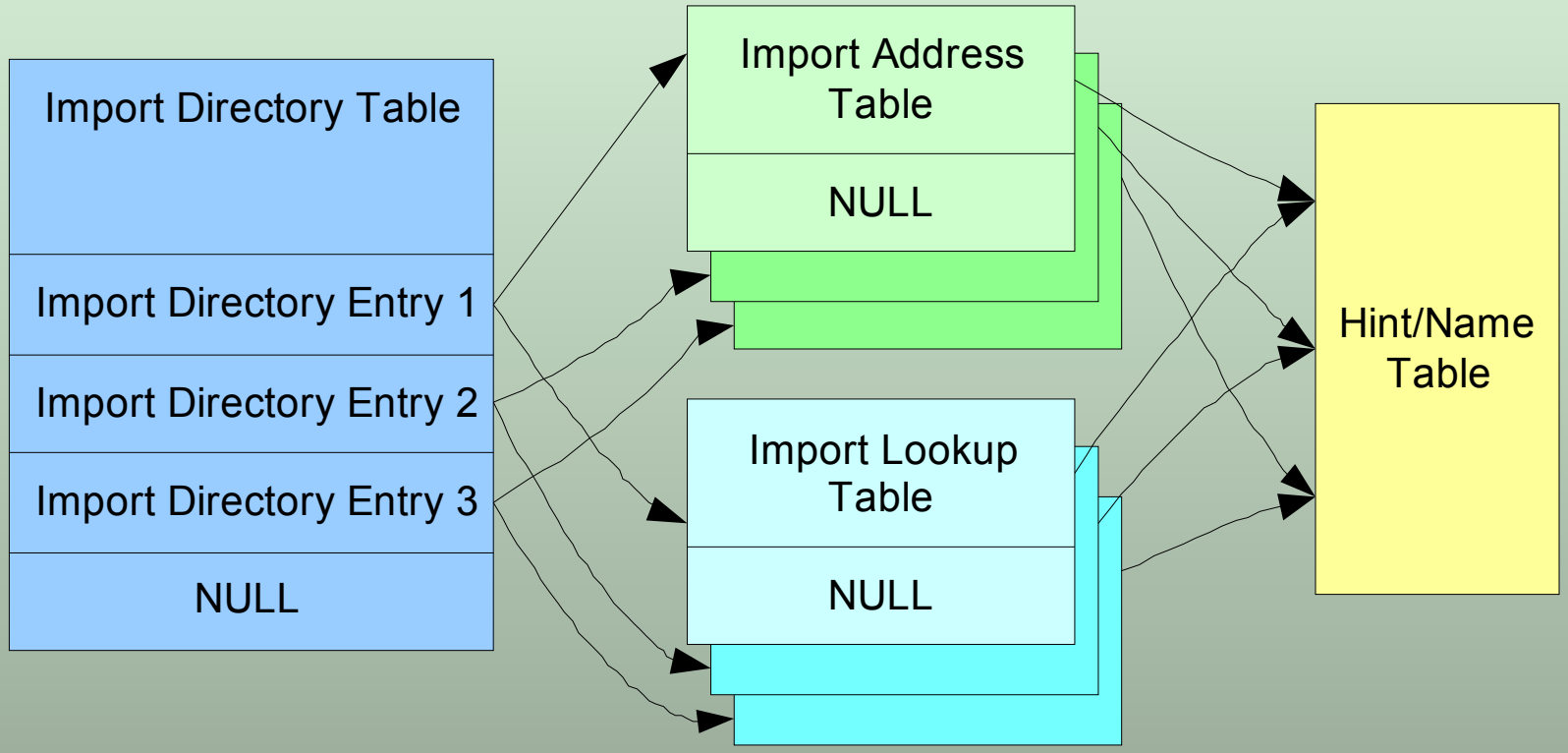
```
struct SectionHeader
{
    char Name[8];
    long VirtualSize;
    long VirtualAddress;
    long SizeOfRawData;
    long PointerToRawData;
    ....
    long Characteristics;
};
```

4.4. Особые секции PE-файла

- Секции PE-файла, как правило, содержат исполняемый код и данные, которые не имеют специального смысла для загрузчика
- Также бывают особые секции:
  - Секция импорта (“.idata”)
  - Секция релокаций (“.reloc”)
  - И некоторые другие…

4.4.1. Секция импорта

- В секции импорта перечисляются все dll-файлы, используемые программой, а также все символы, импортируемые из этих файлов



Элемент таблицы импорта

- Содержит следующие поля:

```
struct ImportDirectoryEntry
{
    long ImportLookupTableRVA;
    …
    long NameRVA;
    long ImportAddressTableRVA;
};
```

Структура Hint/Name

- Структура Hint/Name состоит из трех полей:
  - `short Hint;`
    - Это поле является подсказкой для загрузчика. Оно содержит предполагаемый номер импортируемого символа. Загрузчик сначала ищет этот символ по указанному номеру. В случае неудачи он выполняет бинарный поиск символа по имени
  - `char Name[x];`
    - Имя импортируемого символа в виде ASCIIZ-строки
  - `char Pad;`
    - Это поле служит для выравнивания структуры по четной границе, то есть оно присутствует только тогда, когда структура имеет нечетный размер. Всегда равно нулю

4.4.2. Секция релокаций

- В секции релокаций (".reloc") содержится таблица исправлений, в которой перечислены все абсолютные адреса в PE-файле, которые надо исправить, если файл загружается по адресу, отличному от указанного в поле ImageBase
- Таблица исправлений разбита на блоки. Каждый блок описывает исправления, которые нужно внести в определенную страницу (4K байт) загруженного в память PE-файла. Каждый блок должен начинаться на 32-битовой границе

Структура блока таблицы исправлений

- В начале каждого блока располагается заголовок, состоящий из следующих полей:
  - `long PageRVA;`
    - Это поле содержит RVA страницы PE-файла, исправления в которой описываются данным блоком.
  - `long BlockSize;`
    - Суммарный размер блока в байтах, включая заголовок.
- После заголовка следует массив 16-разрядных слов, каждое из которых описывает одно исправление. При этом старшие четыре бита каждого из этих слов задают тип исправления, а остальные 12 бит обозначают смещение относительно начала страницы, соответствующей данному блоку.

4.5. Заголовок CLI

- Содержит следующие поля:

```
struct CLIHeader
{
    long Cb;
    ...
    struct { long RVA, Size; } Metadata;
    long Flags;
    long EntryPointToken;
    ...
};
```