6-可执行文件的装载与进程

进程的建立

- 创建一个独立的虚拟地址空间
- 读取可执行文件头，建立虚拟空间和可执行文件的映射关系
  - 映射关系保存在进程中的VMA（虚拟内存区域中）
  - **每个进程都有一个由VMA组成的VMA链表**，该链表描述了进程的整个虚拟地址空间。每个VMA区域代表了进程中一段**被映射的虚拟内存，可以是代码段、数据段、堆、栈等。**

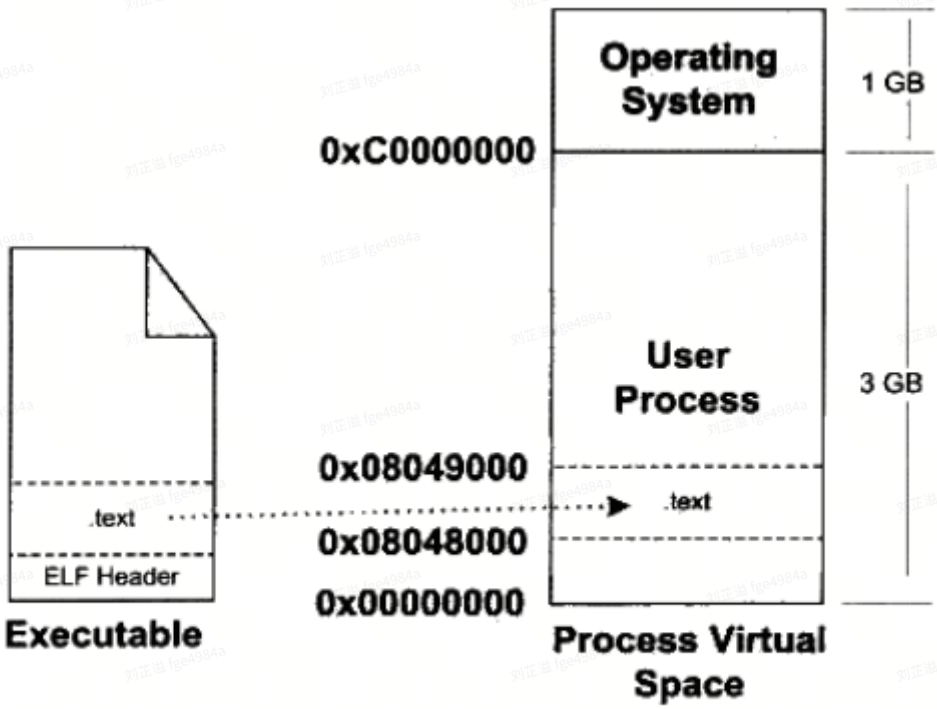


图 6-5　可执行文件与进程虚拟空间

- 将CPU指令寄存器设置成可执行文件入口，启动运行

页错误

- 刚刚只是建立了映射关系，指令和数据并没有真正加载
- 如果允许执行，发生page fault；OS将查询VMA，计算页面在可执行文件中的偏移，在物理内存中分配页

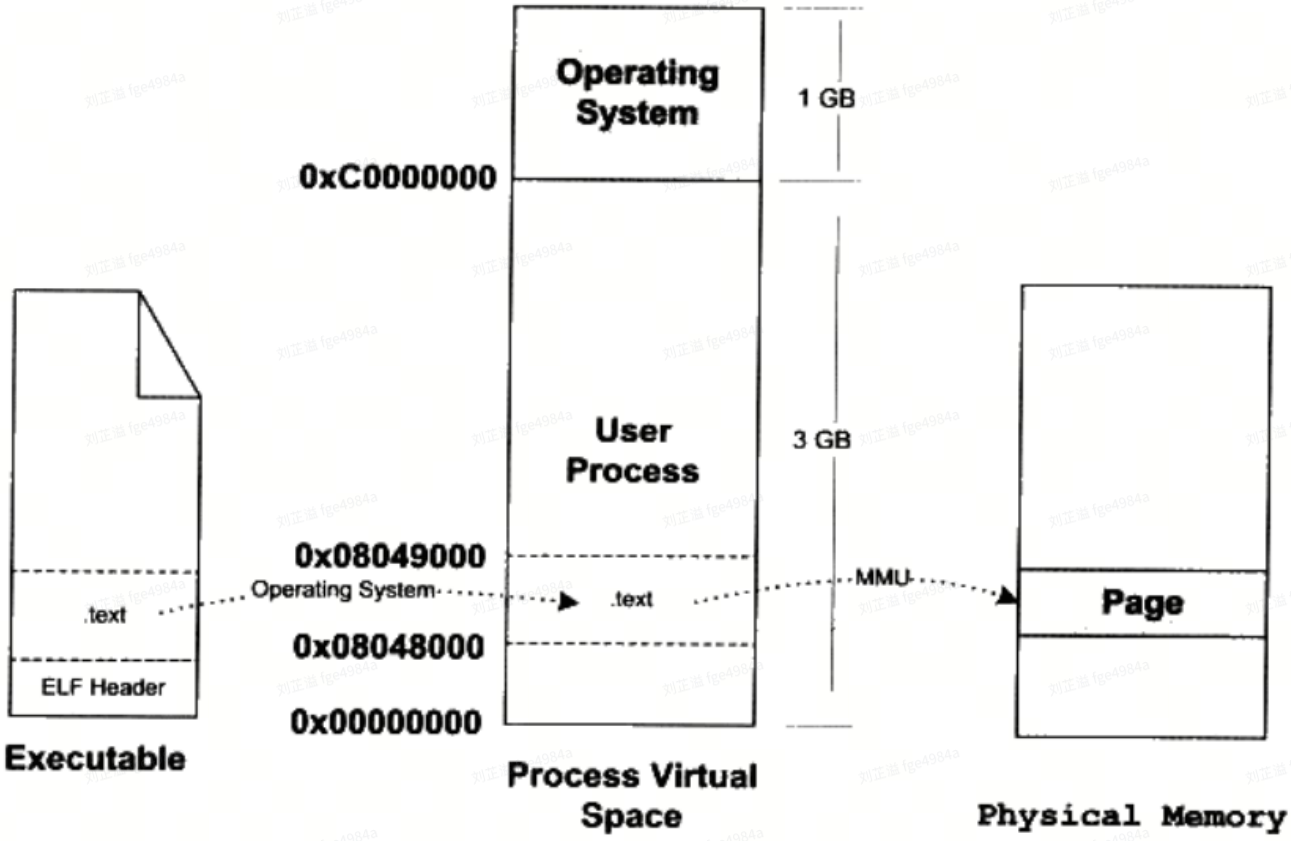


图 6-6　页错误

进程虚存空间分布

- 对于相同权限的段，把段合并到一起当作一个段进行映射
- 一个Segment包含一个或多个属性类似的Section

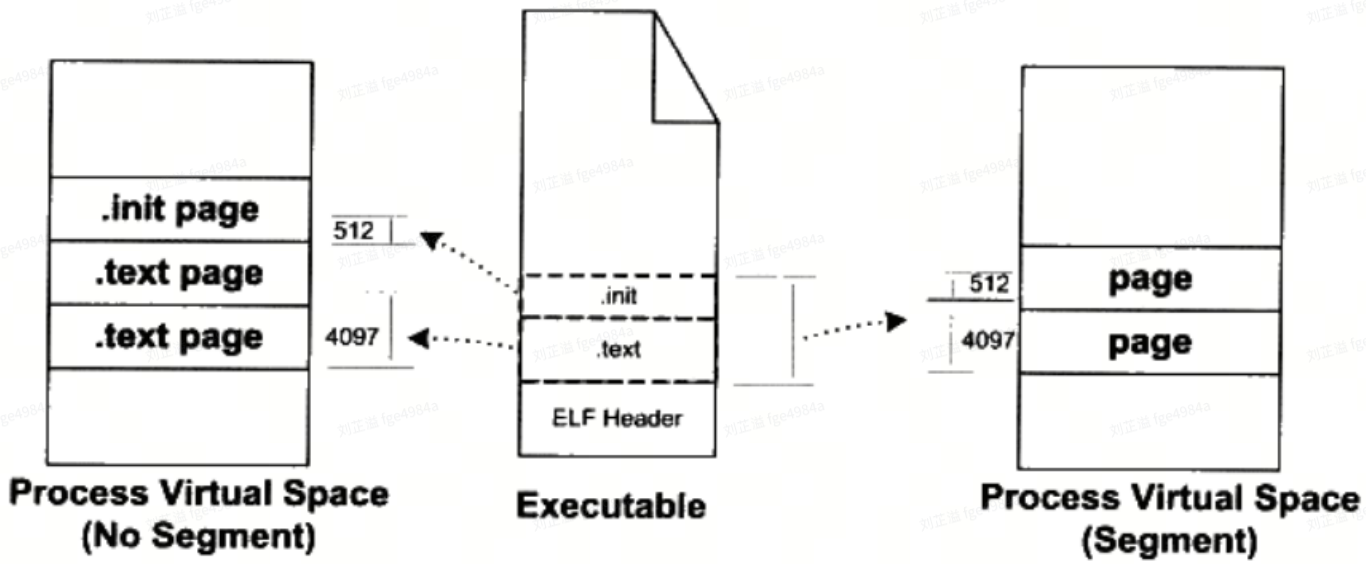
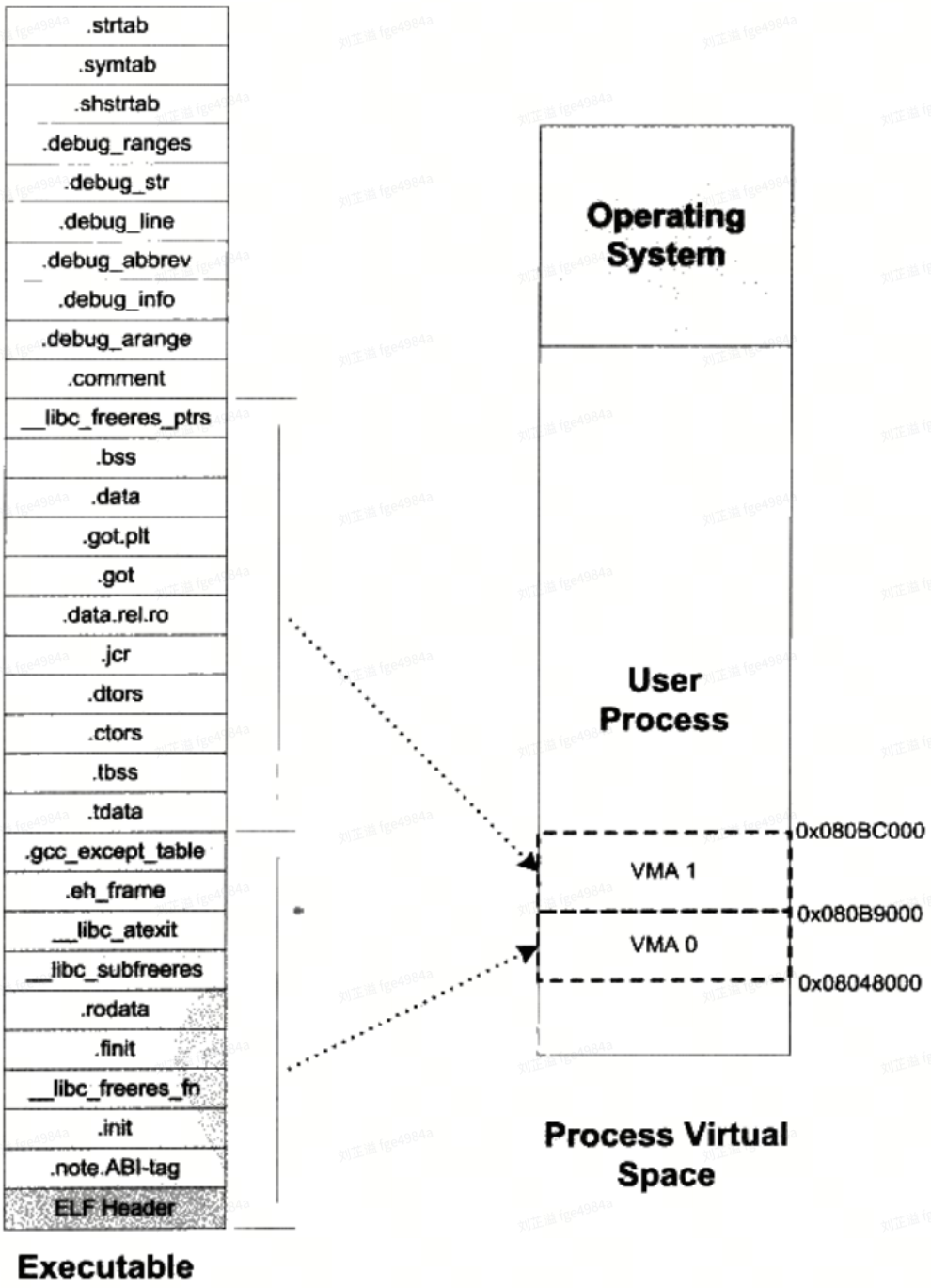


图 6-7　ELF Segment

Section和Segment

- **Section是链接视图**
- **Segment是执行视图**

所以总的来说，“Segment”和“Section”是从不同的角度来划分同一个 ELF 文件。这个在 ELF 中被称为不同的**视图**（View），从“Section”的角度来看 ELF 文件就是**链接视图**（Linking View），从“Segment”的角度来看就是**执行视图**（Execution View）。当我们在谈到 ELF 装载时，“段”专门指“Segment”；而在其他的情况下，“段”指的是“Section”。



- **ELF可执行文件中有程序头表**，保存Segment信息；而ELF目标文件不需要装载所有没有

```
1   // Program header tabletypedef
2   struct {
3       uint32_t   p_type;
```

```
    uint32_t   p_flags;
    Elf64_Off  p_offset;    // Segment在文件中偏移
    Elf64_Addr p_vaddr;     // Segment第一个字节virtual addr
    Elf64_Addr p_paddr;     // Segment第一个字节physical addr
    uint64_t   p_filesz;    // Segment在ELF文件中length
    uint64_t   p_memsz;     // Segment在虚存中length
    uint64_t   p_align;     // Segment对齐属性
} Elf64_Phdr;
```

表 6-2

成员	含义
p_type	“Segment”的类型，基本上我们在这里只关注“LOAD”类型的“Segment”。“LOAD”类型的常量为 1。还有几个类型诸如“DYNAMIC”、“INTERP”等我们在介绍 ELF 动态链接时还会碰到
p_offset	“Segment”在文件中的偏移
p_vaddr	“Segment”的第一个字节在进程虚拟地址空间的起始位置。整个程序头表中，所有“LOAD”类型的元素按照 p_vaddr 从小到大排列
p_paddr	“Segment”的物理装载地址，我们在本书的第 2 部分已经碰到过一个叫做 LMA（Load Memory Address）的概念，这个物理装载地址就是 LMA。p_paddr 的值在一般情况下跟 p_vaddr 是一样的
p_filesz	“Segment”在 ELF 文件中所占空间的长度。它的值可能是 0，因为有可能这个“Segment”在 ELF 文件中不存在内容
p_memse	“Segment”在进程虚拟地址空间中所占用的长度。它的值也可能是 0
p_flags	“Segment”的权限属性，比如可读“R”、可写“W”和可执行“X”
p_align	“Segment”的对齐属性。实际对齐字节等于 2 的 p_align 次。比如 p_align 等于 10，那么实际的对齐属性就是 2 的 10 次方，即 1 024 字节

```
lzy@lzy:~/Workspace/Linker_Lib/static_lib$ readelf -l a.out

Elf file type is EXEC (Executable file)
Entry point 0x401b90
There are 10 program headers, starting at offset 64

Program Headers:
  Type           Offset             VirtAddr           PhysAddr
                 FileSiz            MemSiz              Flags  Align
  LOAD           0x0000000000000000 0x0000000000400000 0x0000000000400000
                 0x0000000000000518 0x0000000000000518  R      0x1000
  LOAD           0x0000000000001000 0x0000000000401000 0x0000000000401000
                 0x0000000000093791 0x0000000000093791  R E    0x1000
  LOAD           0x0000000000095000 0x0000000000495000 0x0000000000495000
                 0x000000000002662d 0x000000000002662d  R      0x1000
  LOAD           0x00000000000bc0c0 0x00000000004bd0c0 0x00000000004bd0c0
                 0x0000000000005150 0x000000000000068c0  RW     0x1000
  NOTE           0x0000000000000270 0x0000000000400270 0x0000000000400270
                 0x0000000000000020 0x0000000000000020  R      0x8
  NOTE           0x0000000000000290 0x0000000000400290 0x0000000000400290
                 0x0000000000000044 0x0000000000000044  R      0x4
  TLS            0x00000000000bc0c0 0x00000000004bd0c0 0x00000000004bd0c0
                 0x0000000000000020 0x0000000000000060  R      0x8
  GNU_PROPERTY   0x0000000000000270 0x0000000000400270 0x0000000000400270
                 0x0000000000000020 0x0000000000000020  R      0x8
  GNU_STACK      0x0000000000000000 0x0000000000000000 0x0000000000000000
                 0x0000000000000000 0x0000000000000000  RW     0x10
  GNU_RELRO      0x00000000000bc0c0 0x00000000004bd0c0 0x00000000004bd0c0
                 0x0000000000002f40 0x0000000000002f40  R      0x1
```

```
 Section to Segment mapping:
  Segment Sections...
   00     .note.gnu.property .note.gnu.build-id .note.ABI-tag .rela.plt
   01     .init .plt .text __libc_freeres_fn .fini
   02     .rodata .stapsdt.base .eh_frame .gcc_except_table
   03     .tdata .init_array .fini_array .data.rel.ro .got .got.plt .data __
libc_subfreeres __libc_IO_vtables __libc_atexit .bss __libc_freeres_ptrs
   04     .note.gnu.property
   05     .note.gnu.build-id .note.ABI-tag
   06     .tdata .tbss
   07     .note.gnu.property
   08
   09     .tdata .init_array .fini_array .data.rel.ro .got
```

- ELF头部表

```
1  typedef struct
2  {
3      unsigned char e_ident[EI_NIDENT]; /*Magic number and other info */
```

```
    Elf64_Half   e_type;         /*Object file type */
    Elf64_Half   e_machine;      /*Architecture */
    Elf64_Word   e_version;      /*Object file version */
    Elf64_Addr   e_entry;    /*Entry point virtual address */
    Elf64_Off   e_phoff;    /*Program header table file offset */
    Elf64_Off   e_shoff;    /*Section header table file offset */
    Elf64_Word   e_flags;    /* Processor-specific flags */
    Elf64_Half   e_ehsize;       /*ELF header size in bytes */
    Elf64_Half   e_phentsize;    /*Program header table entry size */
    Elf64_Half   e_phnum;    /*Program header table entry count */
    Elf64_Half   e_shentsize;    /*Section header table entry size */
    Elf64_Half   e_shnum;    /*Section header table entry count */
    Elf64_Half   e_shstrndx;     /*Section header string table index */
} Elf64_Ehdr;
```

```
lzy@lzy:~/Workspace/Linker_Lib/static_lib$ readelf -h a.out
ELF Header:
  Magic:   7f 45 4c 46 02 01 01 03 00 00 00 00 00 00 00 00
  Class:                             ELF64
  Data:                              2's complement, little endian
  Version:                           1 (current)
  OS/ABI:                            UNIX - GNU
  ABI Version:                       0
  Type:                              EXEC (Executable file)
  Machine:                           Advanced Micro Devices X86-64
  Version:                           0x1
  Entry point address:               0x401b90
  Start of program headers:          64 (bytes into file)
  Start of section headers:          869784 (bytes into file)
  Flags:                             0x0
  Size of this header:               64 (bytes)
  Size of program headers:           56 (bytes)
  Number of program headers:         10
  Size of section headers:           64 (bytes)
  Number of section headers:         32
  Section header string table index: 31
```

- 节头部表

```
/* Section header.  */
typedef struct
{
    Elf32_Word   sh_name;       /* Section name (string tbl index) */
    Elf32_Word   sh_type;       /* Section type */
    Elf32_Word   sh_flags;      /* Section flags */
```

```
    Elf32_Addr  sh_addr;       /*Section virtual addr at execution */
    Elf32_Off   sh_offset;    /* Section file offset */
    Elf32_Word  sh_size;      /* Section size in bytes */
    Elf32_Word  sh_link;      /* Link to another section */
    Elf32_Word  sh_info;      /* Additional section information */
    Elf32_Word  sh_addralign;     /* Section alignment */
    Elf32_Word  sh_entsize;   /* Entry size if section holds table */
} Elf32_Shdr;
```

```
lzy@lzy:~/Workspace/Linker_Lib/static_lib$ readelf -S a.out
There are 32 section headers, starting at offset 0xd4598:

Section Headers:
  [Nr] Name              Type            Address          Offset
       Size              EntSize         Flags  Link  Info  Align
  [ 0]                   NULL            0000000000000000  00000000
       0000000000000000  0000000000000000           0     0     0
  [ 1] .note.gnu.propert NOTE            0000000000400270  00000270
       0000000000000020  0000000000000000   A       0     0     8
  [ 2] .note.gnu.build-i NOTE            0000000000400290  00000290
       0000000000000024  0000000000000000   A       0     0     4
  [ 3] .note.ABI-tag     NOTE            00000000004002b4  000002b4
       0000000000000020  0000000000000000   A       0     0     4
  [ 4] .rela.plt         RELA            00000000004002d8  000002d8
       0000000000000240  0000000000000018  AI       0    20     8
  [ 5] .init             PROGBITS        0000000000401000  00001000
       000000000000001b  0000000000000000  AX       0     0     4
  [ 6] .plt              PROGBITS        0000000000401020  00001020
       0000000000000180  0000000000000000  AX       0     0     16
  [ 7] .text             PROGBITS        00000000004011a0  000011a0
       0000000000091950  0000000000000000  AX       0     0     16
  [ 8] __libc_freeres_fn PROGBITS        0000000000492af0  00092af0
       0000000000001c94  0000000000000000  AX       0     0     16
  [ 9] .fini             PROGBITS        0000000000494784  00094784
       000000000000000d  0000000000000000  AX       0     0     4
  [10] .rodata           PROGBITS        0000000000495000  00095000
       000000000001bfac  0000000000000000   A       0     0     32
```

堆和栈

- linux下可以查看“/proc”来查看进程的虚拟空间分布

```
lzy@lzy:/proc$ cat 70338/maps
55cd03883000-55cd03884000 r--p 00000000 08:20 270326                     /home/lzy/Workspace/Linker_Lib/static_lib/a.out
55cd03884000-55cd03885000 r-xp 00001000 08:20 270326                     /home/lzy/Workspace/Linker_Lib/static_lib/a.out
55cd03885000-55cd03886000 r--p 00002000 08:20 270326                     /home/lzy/Workspace/Linker_Lib/static_lib/a.out
55cd03886000-55cd03887000 r--p 00002000 08:20 270326                     /home/lzy/Workspace/Linker_Lib/static_lib/a.out
55cd03887000-55cd03888000 rw-p 00003000 08:20 270326                     /home/lzy/Workspace/Linker_Lib/static_lib/a.out
55cd0540b000-55cd0542c000 rw-p 00000000 00:00 0                          [heap]
7f4c83f49000-7f4c83f6b000 r--p 00000000 08:20 29539                      /usr/lib/x86_64-linux-gnu/libc-2.31.so
7f4c83f6b000-7f4c840e3000 r-xp 00022000 08:20 29539                      /usr/lib/x86_64-linux-gnu/libc-2.31.so
7f4c840e3000-7f4c84131000 r--p 0019a000 08:20 29539                      /usr/lib/x86_64-linux-gnu/libc-2.31.so
7f4c84131000-7f4c84135000 r--p 001e7000 08:20 29539                      /usr/lib/x86_64-linux-gnu/libc-2.31.so
7f4c84135000-7f4c84137000 rw-p 001eb000 08:20 29539                      /usr/lib/x86_64-linux-gnu/libc-2.31.so
7f4c84137000-7f4c8413d000 rw-p 00000000 00:00 0
7f4c84154000-7f4c84155000 r--p 00000000 08:20 29534                      /usr/lib/x86_64-linux-gnu/ld-2.31.so
7f4c84155000-7f4c84178000 r-xp 00001000 08:20 29534                      /usr/lib/x86_64-linux-gnu/ld-2.31.so
7f4c84178000-7f4c84180000 r--p 00024000 08:20 29534                      /usr/lib/x86_64-linux-gnu/ld-2.31.so
7f4c84181000-7f4c84182000 r--p 0002c000 08:20 29534                      /usr/lib/x86_64-linux-gnu/ld-2.31.so
7f4c84182000-7f4c84183000 rw-p 0002d000 08:20 29534                      /usr/lib/x86_64-linux-gnu/ld-2.31.so
7f4c84183000-7f4c84184000 rw-p 00000000 00:00 0
7ffe71d71000-7ffe71d92000 rw-p 00000000 00:00 0                          [stack]
7ffe71deb000-7ffe71def000 r--p 00000000 00:00 0                          [vvar]
7ffe71def000-7ffe71df1000 r-xp 00000000 00:00 0                          [vdso]
```

- **堆和栈中间，还存在共享库的映射**

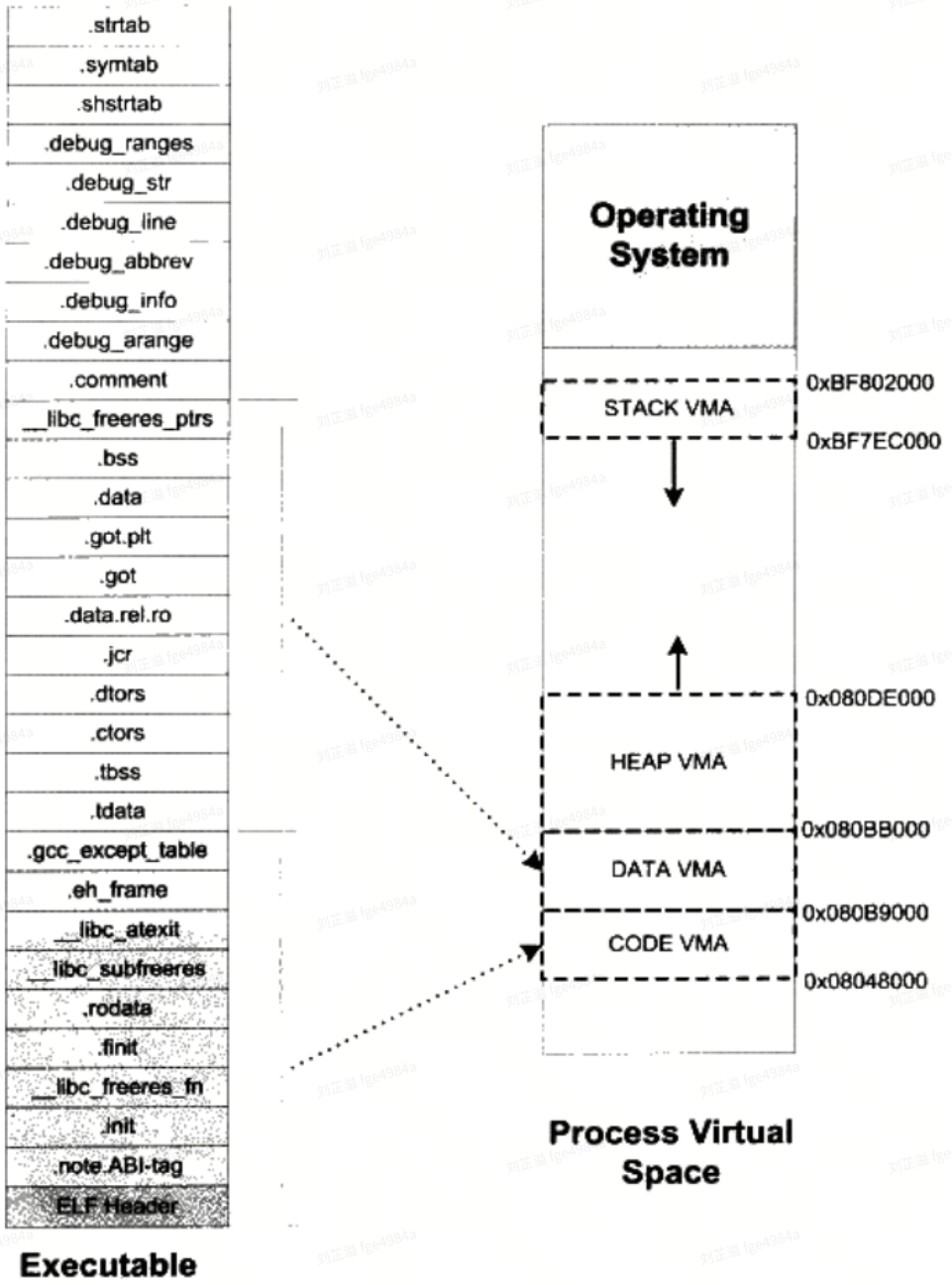


图 6-9　ELF 与 Linux 进程虚拟空间映射关系

段地址对齐

- 连续存放，**但是两个段共享一页的段需要映射两次**

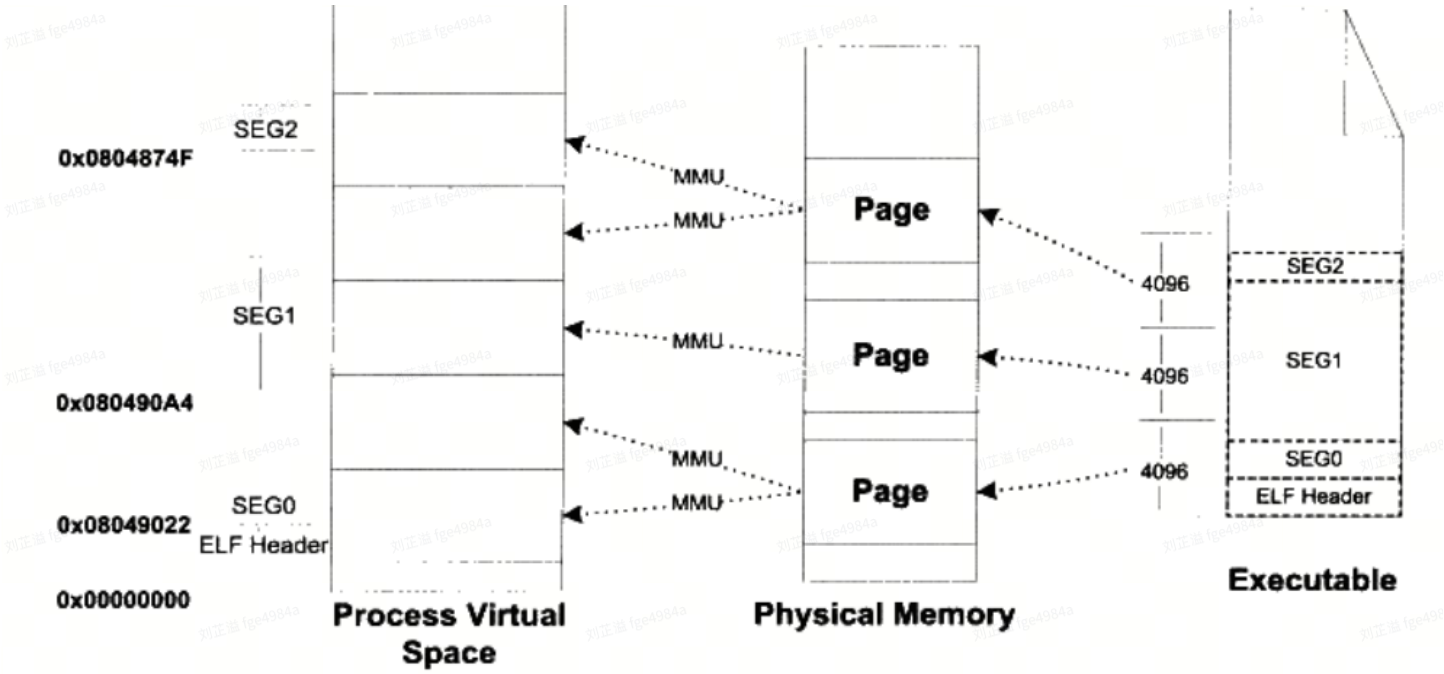


图 6-11　ELF 文件段合并情况