# C3.10 ELF文件格式

赵磊 副教授
武汉大学国家网络安全学院

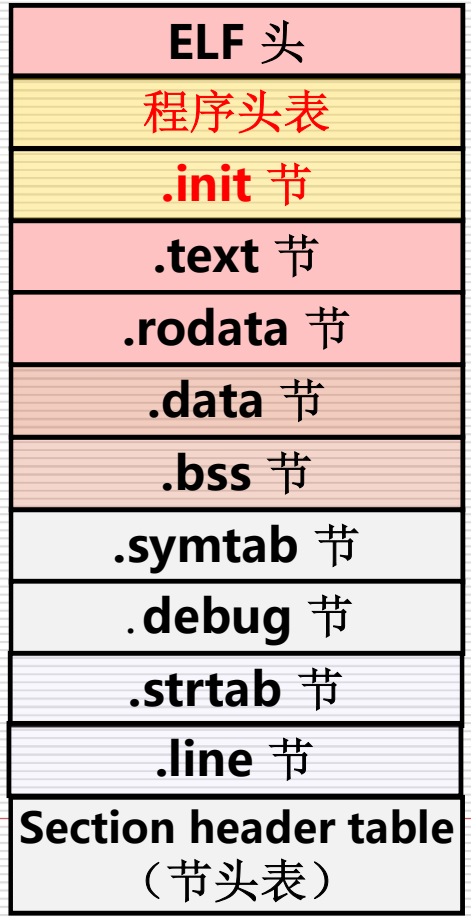
# 可执行文件格式

## 标准的几种可执行文件格式

- DOS操作系统（最简单）：COM格式，文件中仅包含代码和数据，且被加载到固定位置
- System V UNIX早期版本：COFF格式，文件中不仅包含代码和数据，还包含重定位信息、调试信息、符号表等其他信息，由一组严格定义的数据结构序列组成
- Windows： PE格式（COFF的变种），称为可移植可执行（Portable Executable，简称PE）
- Linux等类UNIX：ELF格式（COFF的变种），称为可执行可链接（Executable and Linkable Format，简称ELF）

# ELF程序的结构

- ELF文件格式
  - 头信息
  - 代码节
  - 数据节
  - 重定位节
  - 符号表
  - 调试信息等



# 1. ELF头（ELF Header）

- 文件结构说明信息
  - 32位和64位版本
- 右边是32位系统对应的数据结构

```
#define EI_NIDENT     16
typedef struct {
        unsigned char   e_ident[EI_NIDENT];
        Elf32_Half      e_type;
        Elf32_Half      e_machine;
        Elf32_Word      e_version;
        Elf32_Addr      e_entry;
        Elf32_Off       e_phoff;
        Elf32_Off       e_shoff;
        Elf32_Word      e_flags;
        Elf32_Half      e_ehsize;
        Elf32_Half      e_phentsize;
        Elf32_Half      e_phnum;
        Elf32_Half      e_shentsize;
        Elf32_Half      e_shnum;
        Elf32_Half      e_shstrndx;
} Elf32_Ehdr;
```

# 1. ELF头（ELF Header）

- ELF魔数、版本、小端/大端、操作系统平台、目标文件的类型、机器结构类型、程序执行的入口地址、程序头表（段头表）的起始位置和长度、节头表的起始位置和长度等
- 魔数：文件开头几个字节通常用来确定文件的类型或格式

```
#define EI_NIDENT     16
typedef struct {
        unsigned char   e_ident[EI_NIDENT];
        Elf32_Half      e_type;
        Elf32_Half      e_machine;
        Elf32_Word      e_version;
        Elf32_Addr      e_entry;
        Elf32_Off       e_phoff;
        Elf32_Off       e_shoff;
        Elf32_Word      e_flags;
        Elf32_Half      e_ehsize;
        Elf32_Half      e_phentsize;
        Elf32_Half      e_phnum;
        Elf32_Half      e_shentsize;
        Elf32_Half      e_shnum;
        Elf32_Half      e_shstrndx;
} Elf32_Ehdr;
```

# ELF头信息举例

```
$ readelf -h main
ELF Header:
  Magic:   7f 45 4c 46 01 01 01 00 00 00 00 00 00 00 00 00
  Class:     ELF32
  Data:      2's complement, little endian
  Version:  1 (current)
  OS/ABI:    UNIX - System V
  ABI Version:     0
  Type:    EXEC (Executable file)
  Machine:   Intel 80386
  Version:    0x1
  Entry point address:    x8048580
  Start of program headers:  52 (bytes into file)
  Start of section headers:    3232 (bytes into file)
```

# 2. 程序头表

- 程序头表描述可执行文件中的节与虚拟空间中的存储段之间的映射关系
- 一个表项（32B）说明虚拟地址空间中一个连续的段或一个特殊的节

```
typedef struct {
        Elf32_Word    p_type;
        Elf32_Off     p_offset;
        Elf32_Addr    p_vaddr;
        Elf32_Addr    p_paddr;
        Elf32_Word    p_filesz;
        Elf32_Word    p_memsz;
        Elf32_Word    p_flags;
        Elf32_Word    p_align;
} Elf32_Phdr;
```

# 2. 程序头表

```
Program Headers:
  Type           Offset   VirtAddr   PhysAddr   FileSiz MemSiz  Flg Align
  PHDR           0x000034 0x08048034 0x08048034 0x00100 0x00100 R E 0x4
  INTERP         0x000134 0x08048134 0x08048134 0x00013 0x00013 R   0x1
      [Requesting program interpreter: /lib/ld-linux.so.2]
  LOAD           0x000000 0x08048000 0x08048000 0x004d4 0x004d4 R E 0x1000
  LOAD           0x000f0c 0x08049f0c 0x08049f0c 0x00108 0x00110 RW  0x1000
  DYNAMIC        0x000f20 0x08049f20 0x08049f20 0x000d0 0x000d0 RW  0x4
  NOTE           0x000148 0x08048148 0x08048148 0x00044 0x00044 R   0x4
  GNU_STACK      0x000000 0x00000000 0x00000000 0x00000 0x00000 RW  0x4
  GNU_RELRO      0x000f0c 0x08049f0c 0x08049f0c 0x000f4 0x000f4 R   0x1
```

- 有8个表项，其中两个为可装入段（即Type=LOAD）

# 2. 程序头表

```
LOAD           0x000000 0x08048000 0x08048000 0x004d4 0x004d4 R E 0x1000
LOAD           0x000f0c 0x08049f0c 0x08049f0c 0x00108 0x00110 RW  0x1000
```

第一可装入段：第0x00000~0x004d3字节（包括ELF头、程序头表、.init、.text和.rodata节），映射到虚拟地址0x8048000开始长度为0x4d4字节的区域，按0x1000对齐，具有只读/执行权限（Flg=RE），是只读代码段。

第二可装入段：第0x000f0c开始长度为0x108字节的.data节，映射到虚拟地址0x8049f0c开始长度为0x110字节的存储区域，按0x1000对齐，具有可读可写权限（Flg=RW），是可读写数据段。

# 3. 代码节和数据节

.text 节

- ✓ 编译后的代码部分

.rodata 节

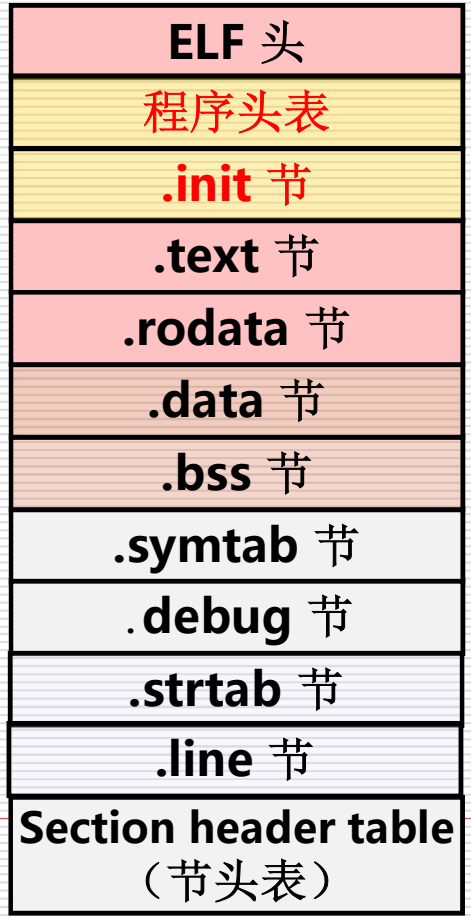
- ✓ 只读数据，如 printf 格式串、switch 跳转表等

.data 节

- ✓ 已初始化的全局变量

.bss 节

- ✓ 未初始化全局变量



# 4. 节头表

- 节头表是ELF可重定位目标文件中重要部分
- 描述每个节的节名、在文件中的偏移、大小、访问属性、对齐方式等

# 4. 节头表

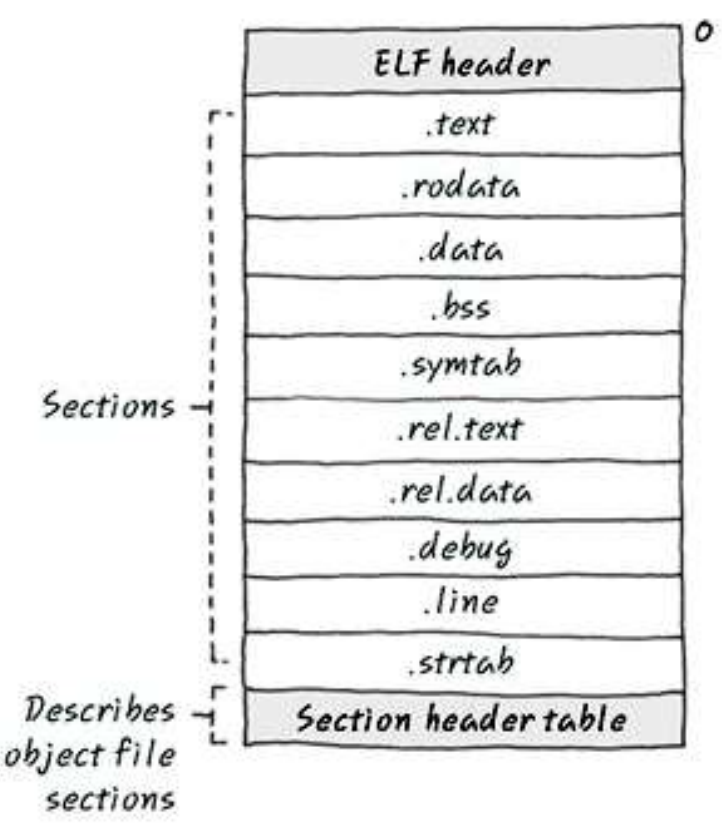
```
/* Section header.  */

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;
```

# 4. 节头表

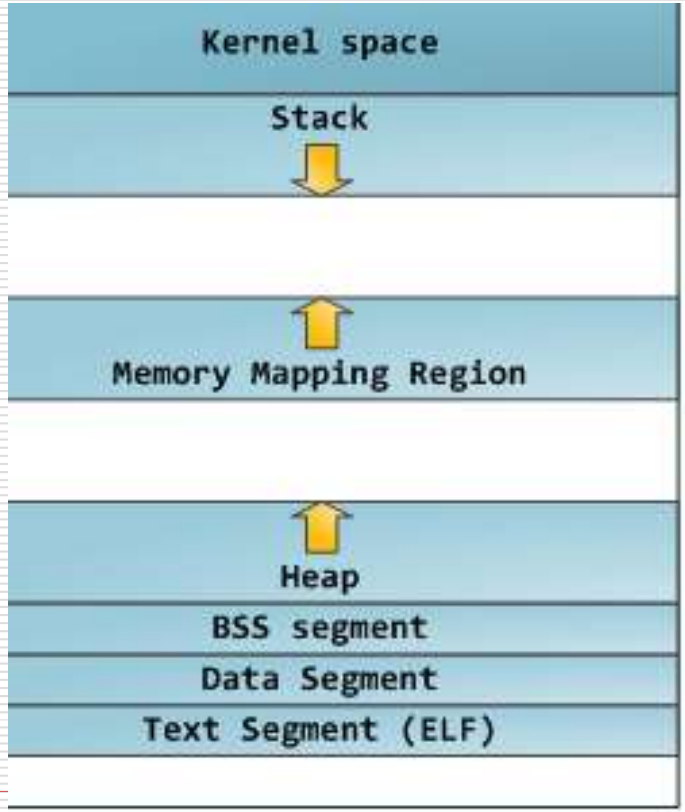
一共是31个段表 20135322

```
Section Headers:
  [Nr] Name              Type            Addr     Off    Size   ES Flg Lk Inf Al
  [ 0]                   NULL            00000000 000000 000000 00      0   0  0
  [ 1] .interp           PROGBITS        08048154 000154 000013 00   A  0   0  1
  [ 2] .note.ABI-tag     NOTE            08048168 000168 000020 00   A  0   0  4
  [ 3] .note.gnu.build-i NOTE            08048188 000188 000024 00   A  0   0  4
  [ 4] .gnu.hash         GNU_HASH        080481ac 0001ac 000020 04   A  5   0  4
  [ 5] .dynsym           DYNSYM          080481cc 0001cc 000050 10   A  6   1  4
  [ 6] .dynstr           STRTAB          0804821c 00021c 00004a 00   A  0   0  1
  [ 7] .gnu.version      VERSYM          08048266 000266 00000a 02   A  5   0  2
  [ 8] .gnu.version_r    VERNEED         08048270 000270 000020 00   A  6   1  4
  [ 9] .rel.dyn          REL             08048290 000290 000008 08   A  5   0  4
  [10] .rel.plt          REL             08048298 000298 000010 08  AI  5  24  4
  [11] .init             PROGBITS        080482a8 0002a8 000023 00  AX  0   0  4
  [12] .plt              PROGBITS        080482d0 0002d0 000030 04  AX  0   0 16
  [13] .plt.got          PROGBITS        08048300 000300 000008 00  AX  0   0  8
  [14] .text             PROGBITS        08048310 000310 000192 00  AX  0   0 16
  [15] .fini             PROGBITS        080484a4 0004a4 000014 00  AX  0   0  4
  [16] .rodata           PROGBITS        080484b8 0004b8 00000e 00   A  0   0  4
  [17] .eh_frame_hdr     PROGBITS        080484c8 0004c8 00002c 00   A  0   0  4
  [18] .eh_frame         PROGBITS        080484f4 0004f4 0000cc 00   A  0   0  4
  [19] .init_array       INIT_ARRAY      08049f08 000f08 000004 00  WA  0   0  4
  [20] .fini_array       FINI_ARRAY      08049f0c 000f0c 000004 00  WA  0   0  4
  [21] .jcr              PROGBITS        08049f10 000f10 000004 00  WA  0   0  4
  [22] .dynamic          DYNAMIC         08049f14 000f14 0000e8 08  WA  6   0  4
  [23] .got              PROGBITS        08049ffc 000ffc 000004 04  WA  0   0  4
  [24] .got.plt          PROGBITS        0804a000 001000 000014 04  WA  0   0  4
  [25] .data             PROGBITS        0804a014 001014 000008 00  WA  0   0  4
  [26] .bss              NOBITS          0804a01c 00101c 000004 00  WA  0   0  1
  [27] .comment          PROGBITS        00000000 00101c 00002d 01  MS  0   0  1
  [28] .shstrtab         STRTAB          00000000 0016c9 00010a 00      0   0  1
  [29] .symtab           SYMTAB          00000000 00104c 000450 10     30  47  4
  [30] .strtab           STRTAB          00000000 00149c 00022d 00      0   0  1
```

# 5. 程序的装入

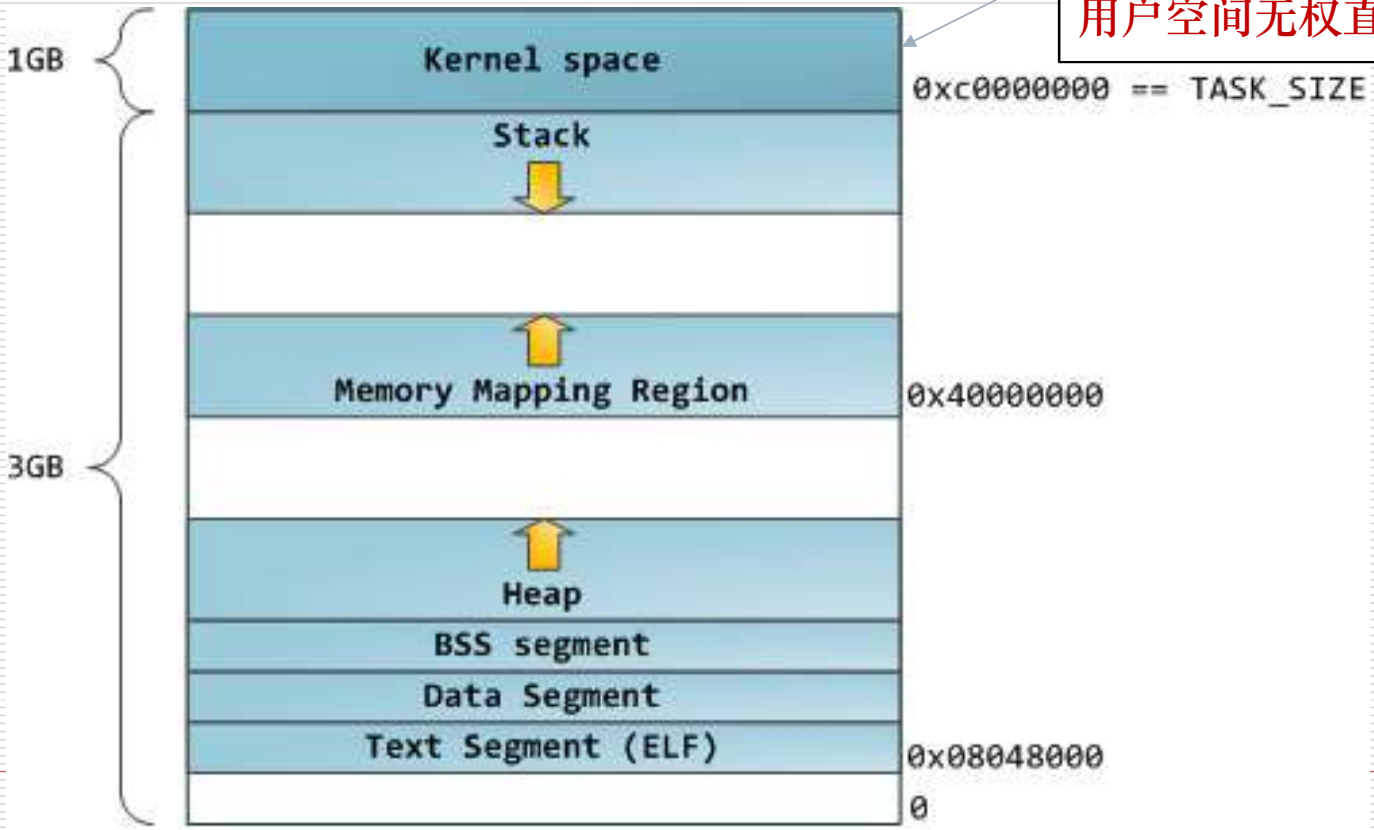

0
ELF header
.text
.rodata
.data
.bss
.symtab
.rel.text
.rel.data
.debug
.line
.strtab
Section header table
Sections
Describes object file sections

内存映射

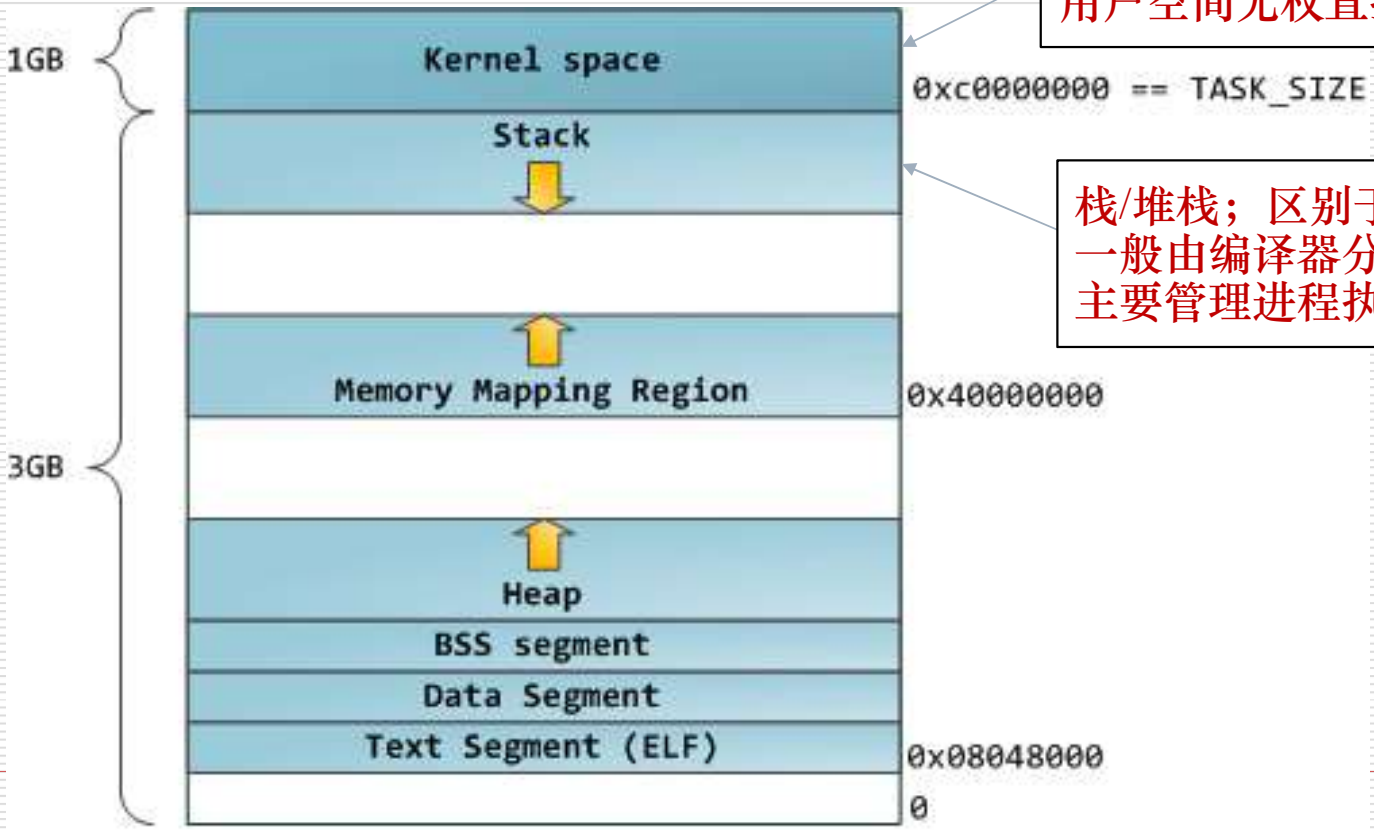

Kernel space
Stack
Memory Mapping Region
Heap
BSS segment
Data Segment
Text Segment (ELF)

# 逻辑地址空间布局

内核空间、多进程共享、
用户空间无权直接访问



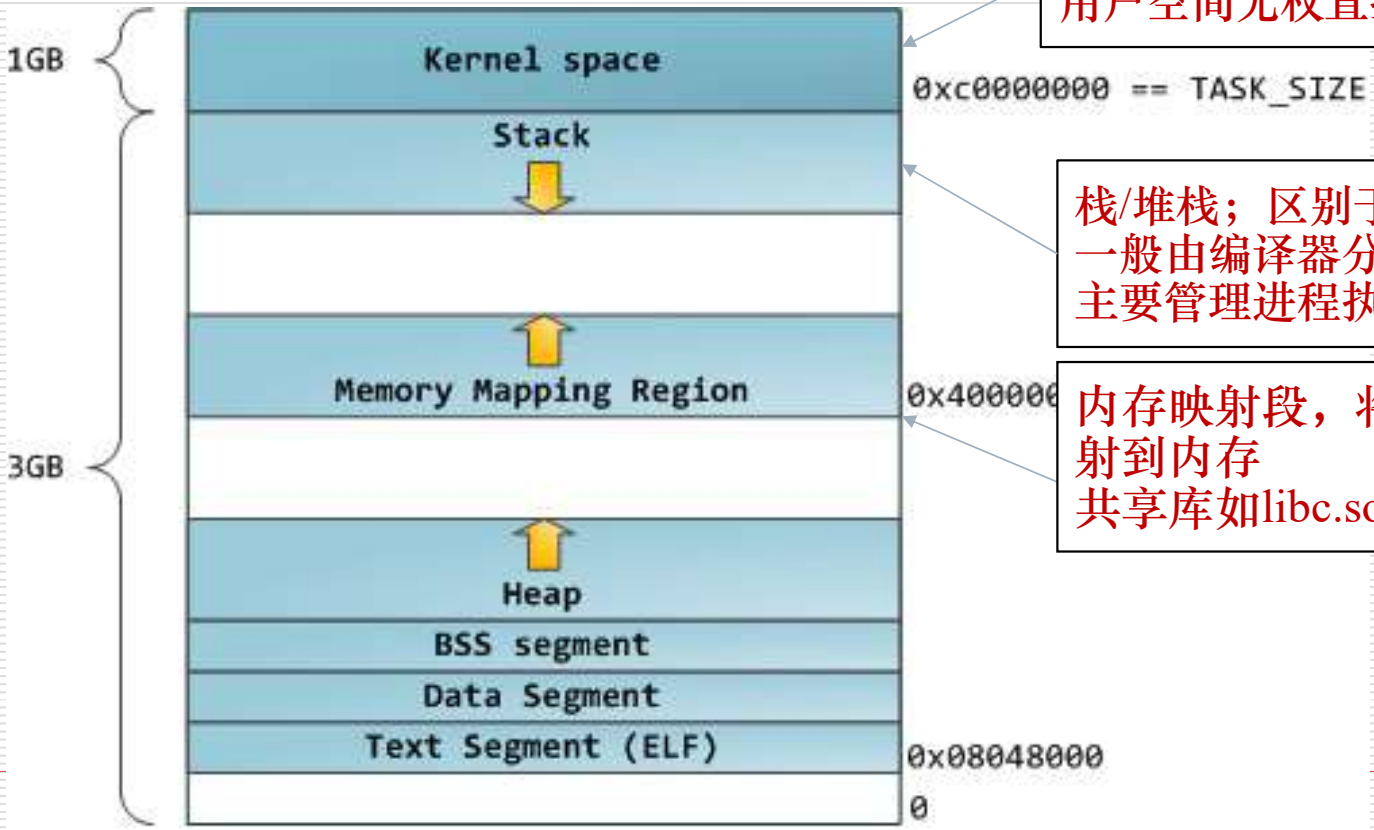
# 逻辑地址空间布局




内核空间、多进程共享、
用户空间无权直接访问

栈/堆栈；区别于系统栈
一般由编译器分配
主要管理进程执行

# 逻辑地址空间布局

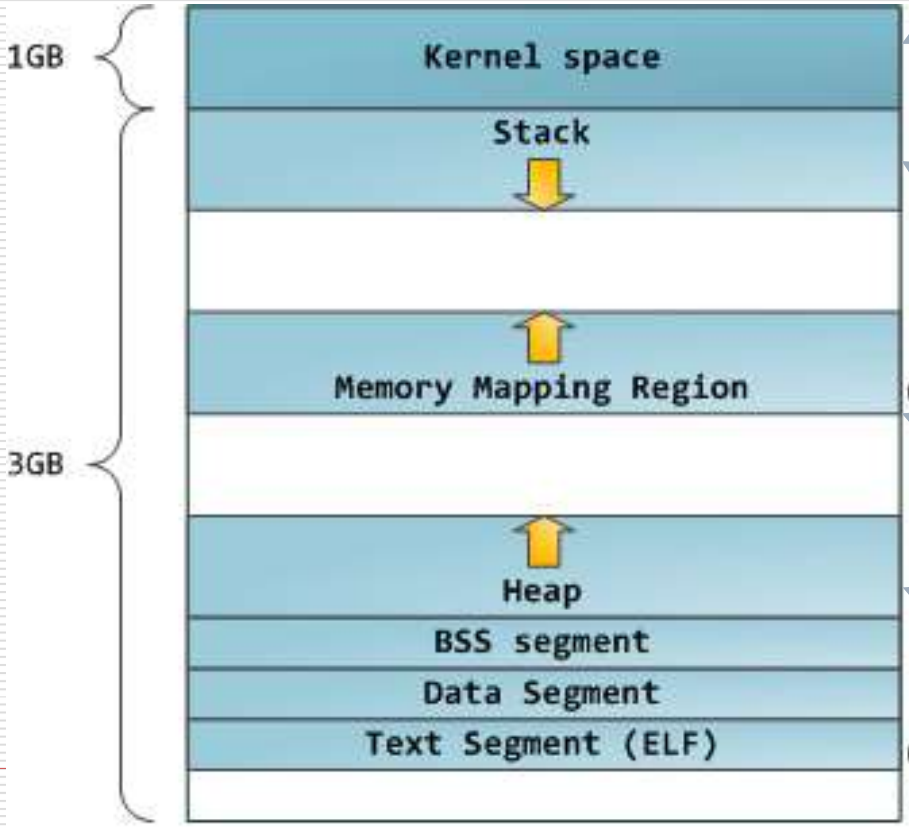



内核空间、多进程共享、
用户空间无权直接访问

栈/堆栈；区别于系统栈
一般由编译器分配
主要管理进程执行

内存映射段，将文件映
射到内存
共享库如libc.so

# 逻辑地址空间布局



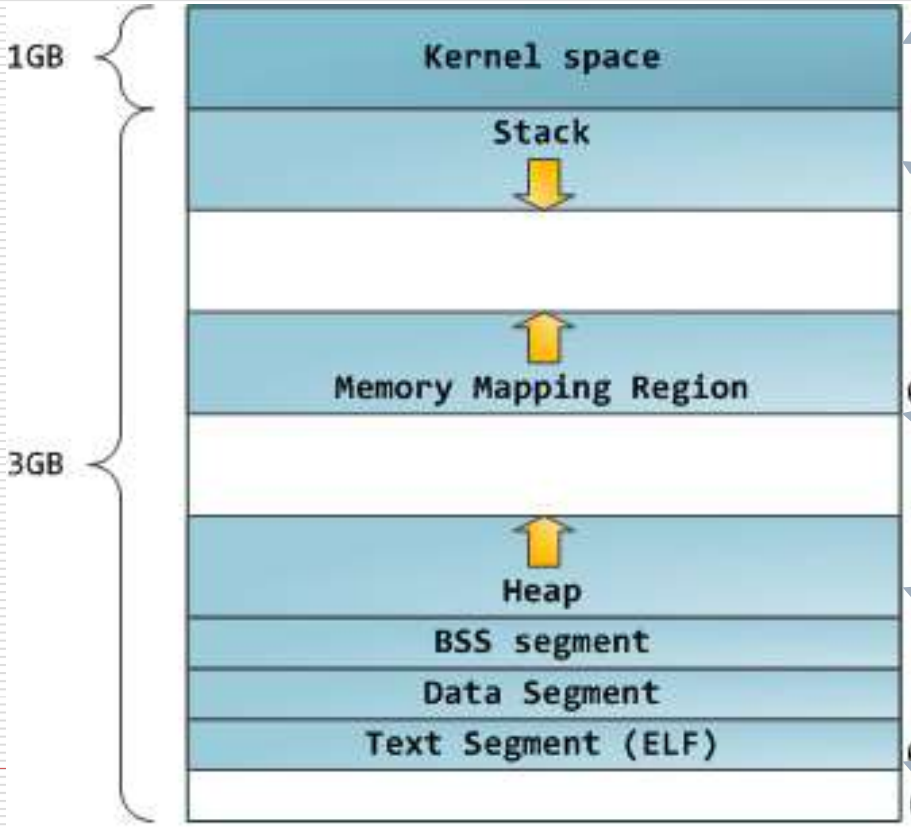

内核空间、多进程共享、
用户空间无权直接访问

栈/堆栈；区别于系统栈
一般由编译器分配
主要管理进程执行

内存映射段，将文件映
射到内存
共享库如libc.so

堆，由程序员动态分配

# 逻辑地址空间布局




内核空间、多进程共享、
用户空间无权直接访问

栈/堆栈；区别于系统栈
一般由编译器分配
主要管理进程执行

内存映射段，将文件映
射到内存
共享库如libc.so

堆，由程序员动态分配

代码段、数据段等

# 课后练习题

- ELF文件
  - 目标文件(.o)
  - 可执行文件(.out)
- 练习
  - 利用readelf等工具，分析观察目标文件与可执行文件格式的区别