目标文件格式概述

南京大学
计算机科学与技术系
袁春风
email: cfyuan@nju.edu.cn
2015.6

回顾：一个C语言程序举例

main.c

```
int buf[2] = {1, 2};
void swap();

int main()
{
  swap();
  return 0;
}
```

swap.c

```
extern int buf[];

int *bufp0 = &buf[0];
static int *bufp1;

void swap()
{
   int temp;
   bufp1 = &buf[1];
   temp = *bufp0;
   *bufp0 = *bufp1;
   *bufp1 = temp;
}
```

每个模块有自己的代码、数据（初始化全局变量、未初始化全局变量，静态变量、局部变量）

局部变量temp分配在栈中，不会在过程外被引用，因此不是符号定义

回顾：可执行文件的生成

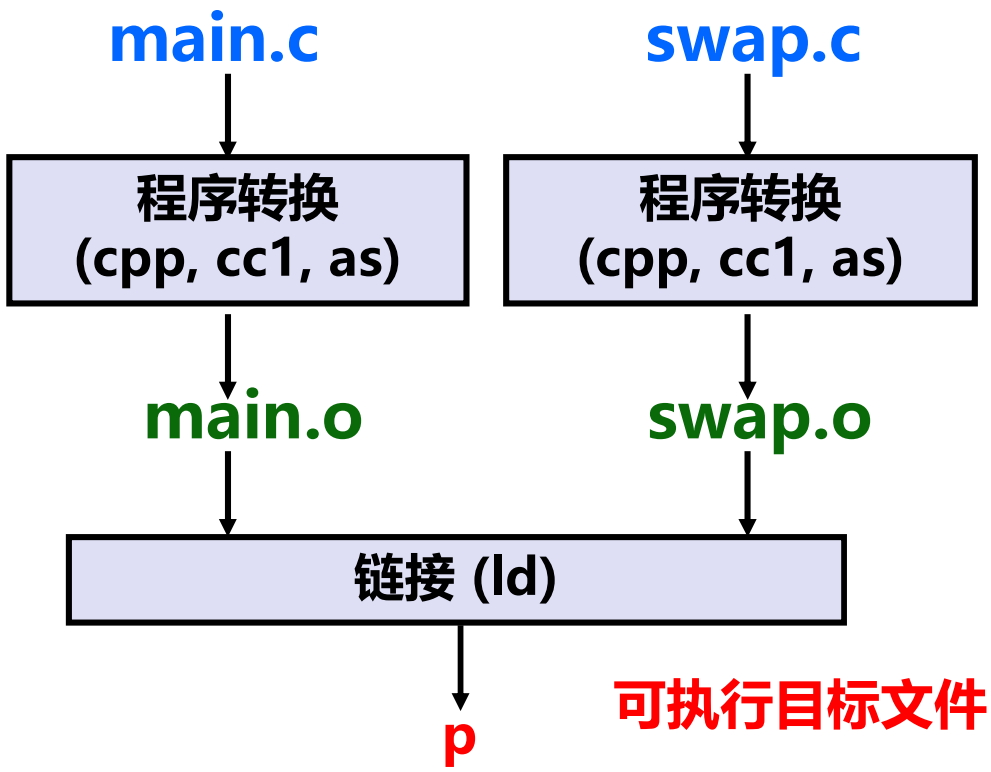
- 使用GCC编译器编译并链接生成可执行程序P:
  - $ gcc -O2 -g -o p main.c swap.c
  - $ ./p

-O2：2级优化

-g：生成调试信息

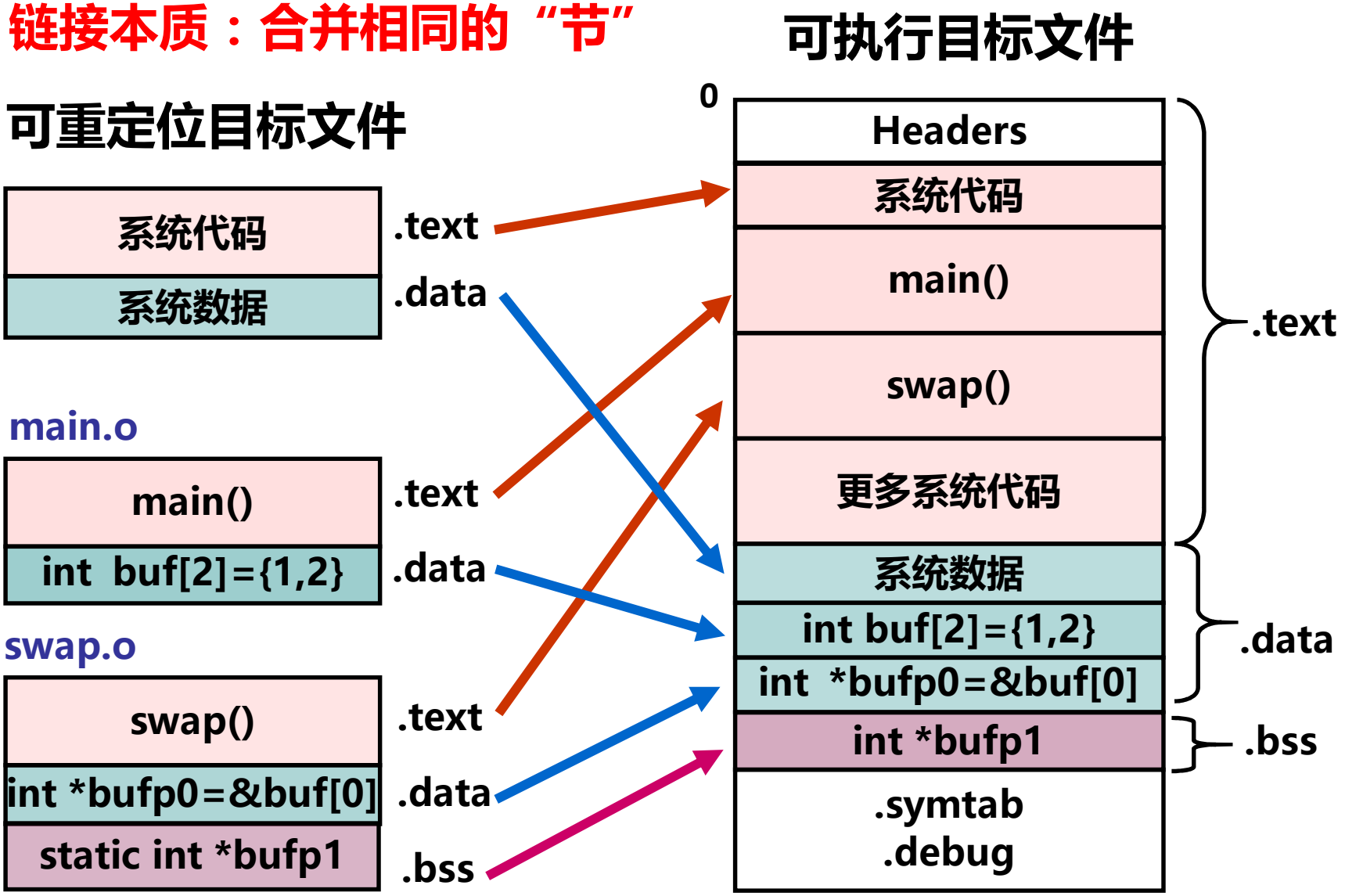
-o：目标文件名

GCC编译器的静态链接过程



回顾：链接过程的本质

链接本质：合并相同的“节”



链接操作的步骤

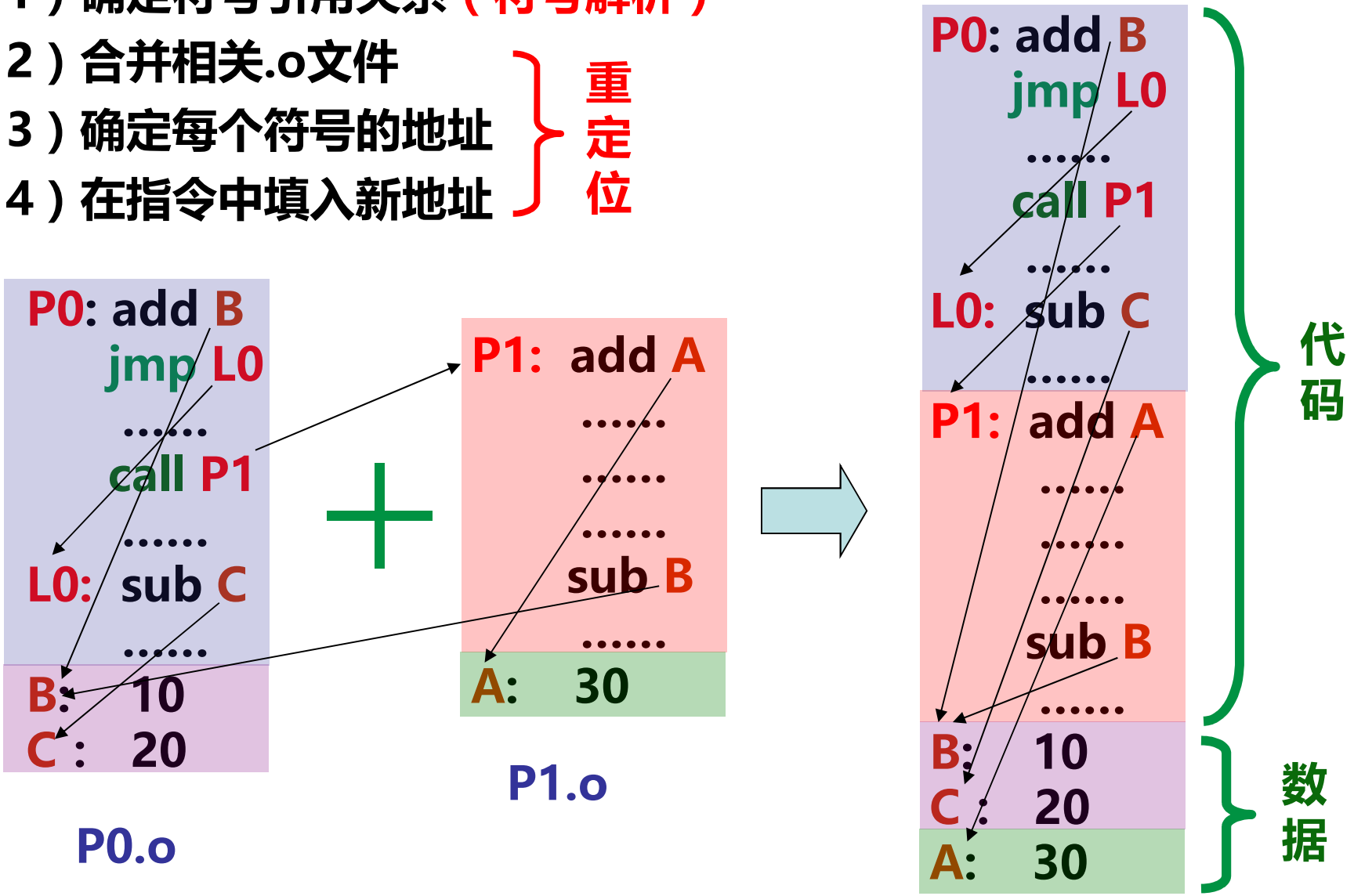
1）确定符号引用关系（符号解析）

2）合并相关.o文件

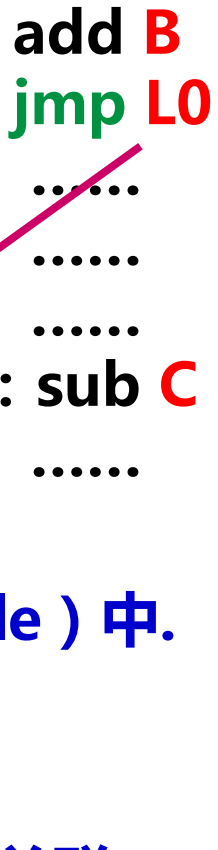
3）确定每个符号的地址

4）在指令中填入新地址

重定位



链接操作的步骤



- Step 1. 符号解析（Symbol resolution）
  - 程序中有定义和引用的符号 (包括变量和函数等)
    - void swap() {…} /* 定义符号swap */
    - swap(); /* 引用符号swap */
    - int *xp = &x; /* 定义符号 xp, 引用符号 x */
  - 编译器将定义的符号存放在一个符号表（ symbol table ）中.
    - 符号表是一个结构数组
    - 每个表项包含符号名、长度和位置等信息
  - 链接器将每个符号的引用都与一个确定的符号定义建立关联
- Step 2. 重定位
  - 将多个代码段与数据段分别合并为一个单独的代码段和数据段
  - 计算每个定义的符号在虚拟地址空间中的绝对地址
  - 将可执行文件中符号引用处的地址修改为重定位后的地址信息

三类目标文件

- 可重定位目标文件 (.o)
  - 其代码和数据可和其他可重定位文件合并为可执行文件
    - 每个.o 文件由对应的.c文件生成
    - 每个.o文件代码和数据地址都从0开始
- 可执行目标文件 (默认为a.out)
  - 包含的代码和数据可以被直接复制到内存并被执行
  - 代码和数据地址为虚拟地址空间中的地址
- 共享的目标文件 (.so)
  - 特殊的可重定位目标文件，能在装入或运行时被装入到内存并自动被链接，称为共享库文件
  - Windows 中称其为 *Dynamic Link Libraries* (DLLs)

目标文件

```
/* main.c */
int add(int, int);
int main( )
{
   return add(20, 13);
}
```

```
/* test.c */
int add(int i, int j)
{
    int x = i + j;
    return x;
}
```

objdump -d test.o

```
00000000 <add>:
       0:   55              push  %ebp
       1:   89 e5           mov   %esp, %ebp
       3:   83 ec 10        sub   $0x10, %esp
       6:   8b 45 0c        mov    0xc(%ebp), %eax
       9:   8b 55 08        mov    0x8(%ebp), %edx
       c:   8d 04 02        lea    (%edx,%eax,1), %eax
       f:   89 45 fc        mov    %eax, -0x4(%ebp)
      12:   8b 45 fc        mov    -0x4(%ebp), %eax
      15:   c9              leave
      16:   c3              ret
```

objdump -d test

```
080483d4 <add>:
 80483d4:   55              push  %ebp
 80483d5:   89 e5           mov   %esp, %ebp
 80483d7:   83 ec 10        sub   $0x10, %esp
 80483da:   8b 45 0c        mov    0xc(%ebp), %eax
 80483dd:   8b 55 08        mov    0x8(%ebp), %edx
 80483e0:   8d 04 02        lea    (%edx,%eax,1), %eax
 80483e3:   89 45 fc        mov    %eax, -0x4(%ebp)
 80483e6:   8b 45 fc        mov    -0x4(%ebp), %eax
 80483e9:   c9              leave
 80483ea:   c3              ret
```

目标文件的格式

- **目标代码（Object Code）指编译器和汇编器处理源代码后所生成的机器语言目标代码**
- **目标文件（Object File）指包含目标代码的文件**
- **最早的目标文件格式是自有格式，非标准的**
- **标准的几种目标文件格式**
  - **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）**

Executable and Linkable Format (ELF)

- **两种视图**
  - **链接视图（被链接）：可重定位目标文件 (Relocatable object files)**
  - **执行视图（被执行）：可执行目标文件（Executable object files）**

链接视图

节（section）是ELF 文件中具有相同特征的最小可处理单位

.text节: 代码

.data节: 数据

.rodata: 只读数据

.bss: 未初始化数据

执行视图

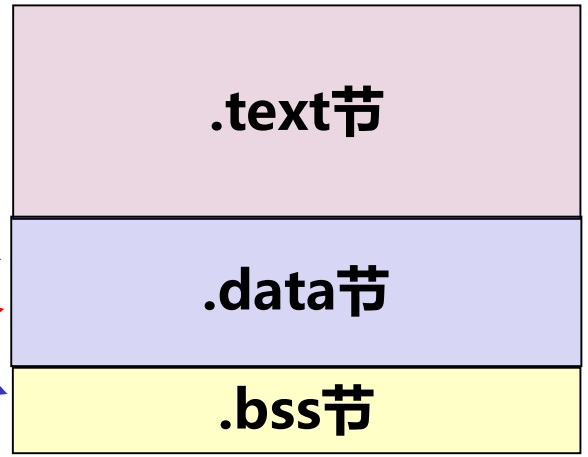
由不同的段（segment）组成，描述节如何映射到存储段中，可多个节映射到同一段，如：可合并.data节和.bss节,并映射到一个可读可写数据段中

链接视图—可重定位目标文件

- 可被链接（合并）生成可执行文件或**共享目标文件**
- **静态链接库文件**由若干个可重定位目标文件组成
- 包含代码、数据（已初始化.data和未初始化.bss）
- 包含**重定位信息**（指出哪些符号引用处需要重定位）
- 文件扩展名为.o（相当于Windows中的 .obj文件）

```
int x=100;
int y;
void prn(int n)
{
   printf( "%d\n" ,n);
}

void main( )
{
    static int a=1;
    static int b;
    int i=200,j;
    prn(x+a+i);
}
```



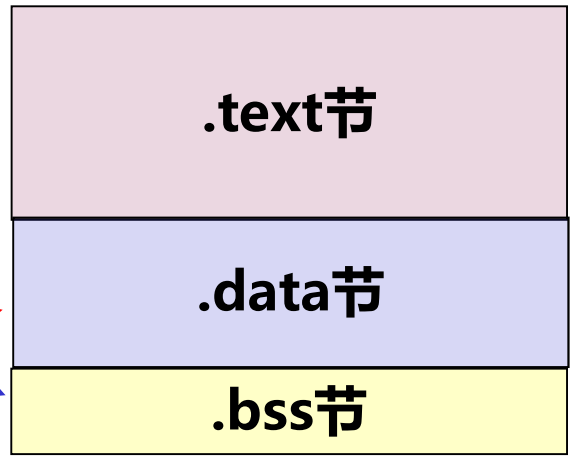
为了**进行链接**，还需要其他许多信息，如符号表、重定位信息等许多其他的节（Section）

执行视图—可执行目标文件

- 包含代码、数据（已初始化.data和未初始化.bss）
- 定义的所有变量和函数已有确定地址（虚拟地址空间中的地址）
- 符号引用处已被重定位，以指向所引用的定义符号
- 没有文件扩展名或默认为a.out（相当于Windows中的 .exe文件）
- 可被CPU直接执行，指令地址和指令给出的操作数地址都是虚拟地址

```
int x=100;
int y;
void prn(int n)
{
   printf( "%d\n" ,n);
}

void main( )
{
    static int a=1;
    static int b;
    int i=200,j;
    prn(x+a+i);
}
```



为了能执行，还需将具相同访问属性的节合并成段（Segment），并说明每个段的属性，如：在可执行文件中的位移、大小、在虚拟空间中的位置、对齐方式、访问属性等