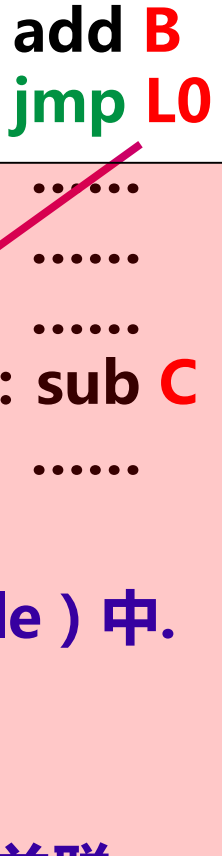
静态链接和符号解析

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：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

如何划分模块？

静态链接对象：

多个可重定位目标模块 + 静态库（标准库、自定义库）

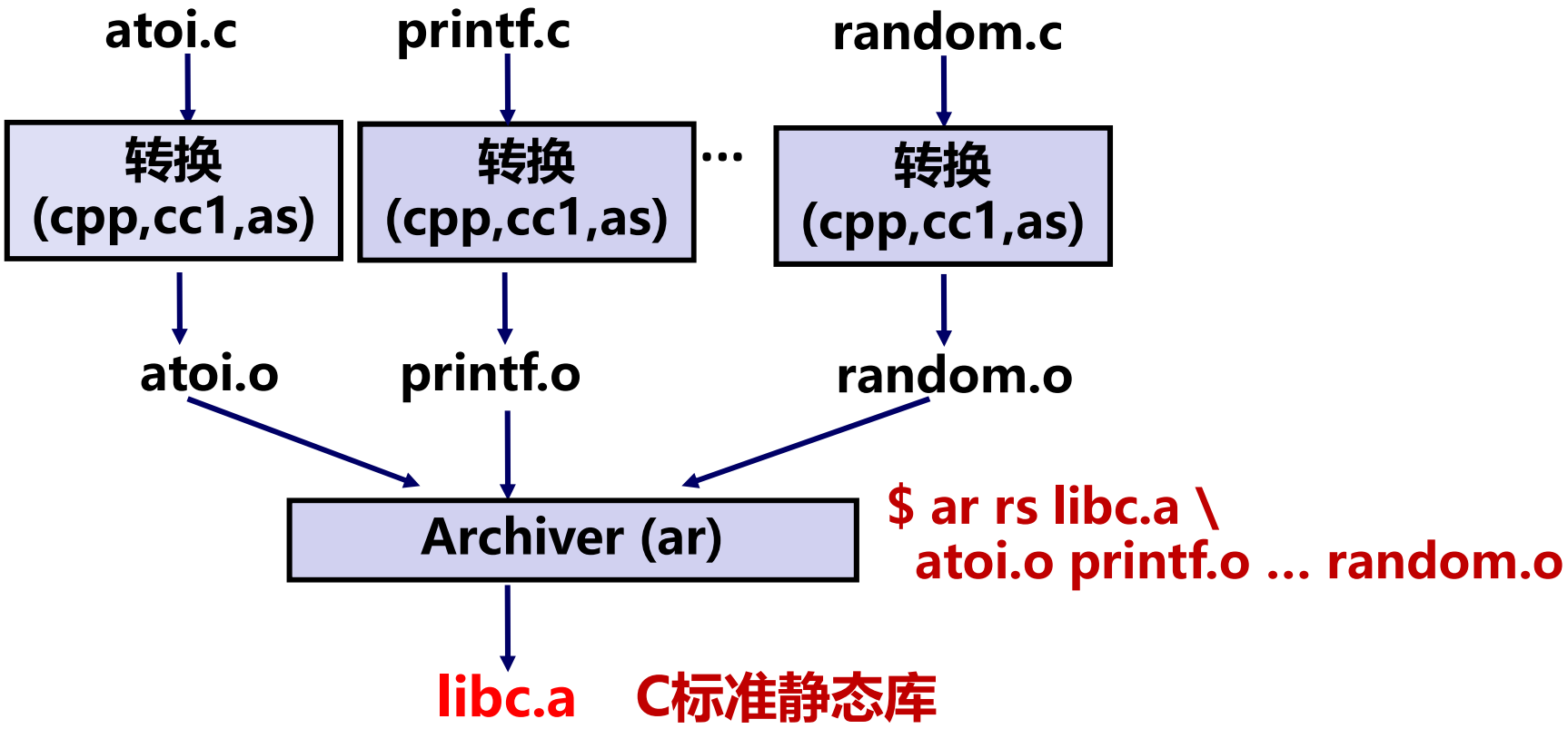
（.o文件）　　（.a文件，其中包含多个.o模块）

- **库函数模块：许多函数无需自己写，可使用共享的库函数**
  - **如数学库, 输入/输出库, 存储管理库，字符串处理等**
- **对于自定义模块，避免以下两种极端做法**
  - **将所有函数都放在一个源文件中**
    - **修改一个函数需要对所有函数重新编译**
    - **时间和空间两方面的效率都不高**
  - **一个源文件中仅包含一个函数**
    - **需要程序员显式地进行链接**
    - **效率高，但模块太多，故太繁琐**

静态共享库

- **静态库 (.a archive files)**
  - **将所有相关的目标模块（.o）打包为一个单独的库文件（.a），称为静态库文件 ，也称存档文件（archive）**
  - **使用静态库，可增强链接器功能，使其能通过查找一个或多个库文件中定义的符号来解析符号**
  - **在构建可执行文件时，只需指定库文件名，链接器会自动到库中寻找那些应用程序用到的目标模块，并且只把用到的模块从库中拷贝出来**
  - **在gcc命令行中无需明显指定C标准库libc.a(默认库)**

静态库的创建



- Archiver（归档器）允许增量更新，只要重新编译需修改的源码并将其.o文件替换到静态库中。

常用静态库

libc.a（C标准库）

- 1392个目标文件（大约8 MB）
- 包含I/O、存储分配、信号处理、字符串处理、时间和日期、随机数生成、定点整数算术运算

libm.a (the C math library)

- 401 个目标文件（大约 1 MB）
- 浮点数算术运算(如sin, cos, tan, log, exp, sqrt, ...)

```
% ar -t /usr/lib/libc.a | sort
...
fork.o
...
fprintf.o
fpu_control.o
fputc.o
freopen.o
fscanf.o
fseek.o
fstab.o
...
```

```
% ar -t /usr/lib/libm.a | sort
...
e_acos.o
e_acosf.o
e_acosh.o
e_acoshf.o
e_acoshl.o
e_acosl.o
e_asin.o
e_asinf.o
e_asinl.o
...
```

自定义一个静态库文件

举例：将myproc1.o和myproc2.o打包生成mylib.a

myproc1.c

```
# include <stdio.h>
void myfunc1() {
    printf("This is myfunc1!\n");
}
```

myproc2.c

```
# include <stdio.h>
void myfunc2() {
    printf("This is myfunc2\n");
}
```

```
$ gcc –c myproc1.c myproc2.c
$ ar rcs mylib.a myproc1.o myproc2.o
```

main.c

```
void myfunc1(viod);
int main()
{
   myfunc1();
   return 0;
}
```

```
$ gcc –c main.c
$ gcc –static –o myproc main.o ./mylib.a
```

libc.a无需明显指出！

调用关系：main→myfunc1→printf

问题：如何进行符号解析？

链接器中符号解析的全过程

```
$ gcc –c main.c
$ gcc –static –o myproc main.o ./mylib.a
```

libc.a无需明显指出！

调用关系：main→myfunc1→printf

E 将被合并以组成可执行文件的所有目标文件集合
U 当前所有未解析的引用符号的集合
D 当前所有定义符号的集合

main.c

```
void myfunc1(viod);
int main()
{
   myfunc1();
   return 0;
}
```

开始E、U、D为空，首先扫描main.o，把它加入E，同时把myfun1加入U，main加入D。接着扫描到mylib.a，将U中所有符号（本例中为myfunc1）与mylib.a中所有目标模块（myproc1.o和myproc2.o）依次匹配，发现在myproc1.o中定义了myfunc1，故myproc1.o加入E，myfunc1从U转移到D。在myproc1.o中发现还有未解析符号printf，将其加到U。不断在mylib.a的各模块上进行迭代以匹配U中的符号，直到U、D都不再变化。此时U中只有一个未解析符号printf，而D中有main和myfunc1。因为模块myproc2.o没有被加入E中，因而它被丢弃。

接着，扫描默认的库文件libc.a，发现其目标模块printf.o定义了printf，于是printf也从U移到D，并将printf.o加入E，同时把它定义的所有符号加入D，而所有未解析符号加入U。处理完libc.a时，U一定是空的。

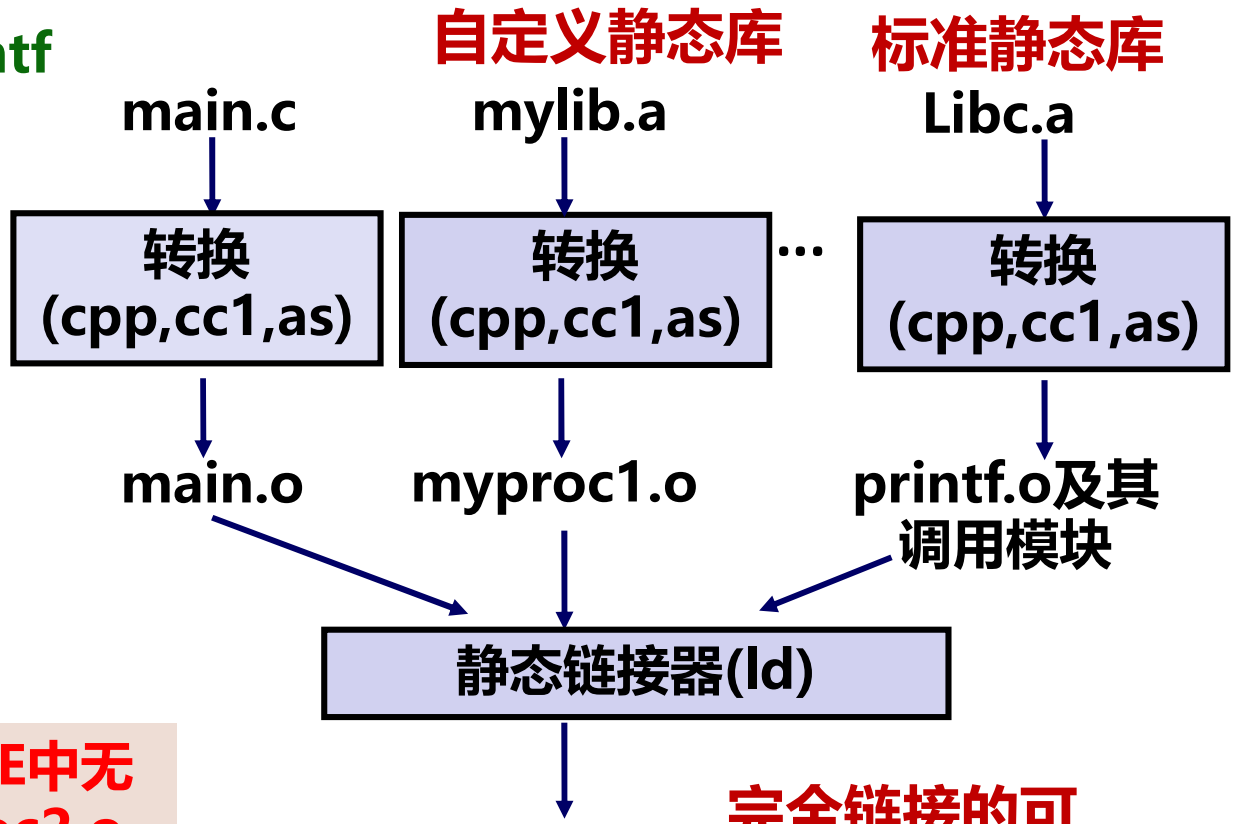
链接器中符号解析的全过程

$ gcc –static –o myproc main.o ./mylib.a

main→myfunc1→printf

main.c

```
void myfunc1(viod);
int main()
{
   myfunc1();
   return 0;
}
```



注意：E中无myproc2.o

解析结果：

E中有main.o、myproc1.o、printf.o及其调用的模块

D中有main、myproc1、printf及其引用的符号

链接器中符号解析的全过程

main.c

```
void myfunc1(viod);
int main()
{
   myfunc1();
   return 0;
}
```

$ gcc –static –o myproc main.o ./mylib.a

解析结果：

E中有main.o、myproc1.o、printf.o及其调用的模块

D中有main、myproc1、printf及其引用符号

main→myfunc1→printf

被链接模块应按调用顺序指定！

若命令为：$ gcc –static –o myproc ./mylib.a main.o， 结果怎样？

首先，扫描mylib，因是静态库，应根据其中是否存在U中未解析符号对应的定义符号来确定哪个.o被加入E。因为开始U为空，故其中两个.o模块都不被加入E中而被丢弃。

然后，扫描main.o，将myfunc1加入U，直到最后它都不能被解析。Why？

因此，出现链接错误！

它只能用mylib.a中符号来解析，而mylib中两个.o模块都已被丢弃！

使用静态库

- 链接器对外部引用的解析算法要点如下:
  - 按照命令行给出的顺序扫描.o 和.a 文件
  - 扫描期间将当前未解析的引用记录到一个列表U中
  - 每遇到一个新的.o 或 .a 中的模块，都试图用其来解析U中的符号
  - 如果扫描到最后，U中还有未被解析的符号，则发生错误
- 问题和对策
  - 能否正确解析与命令行给出的顺序有关
  - 好的做法：将静态库放在命令行的最后

libmine.a 是静态库

假设调用关系：libtest.o→libfun.o(在libmine.a中）

-lxxx=libxxx.a (main) →(libfun)

```
$ gcc -L. libtest.o -lmine
$ gcc -L. -lmine libtest.o
libtest.o: In function `main':
libtest.o(.text+0x4): undefined reference to `libfun'
```

扫描libtest.o，将libfun送U，扫描到libmine.a时，用其定义的libfun来解析

说明在libtest.o中的main调用了libfun这个在库libmine中的函数，
所以，在命令行中，应该将libtest.o放在前面，像第一行中那样！

链接顺序问题

- 假设调用关系如下：

  func.o → libx.a 和 liby.a 中的函数

  libx.a → libz.a 中的函数

  libx.a 和 liby.a 之间、liby.a 和 libz.a 相互独立

  则以下几个命令行都是可行的：

  - gcc -static –o myfunc func.o libx.a liby.a libz.a
  - gcc -static –o myfunc func.o liby.a libx.a libz.a
  - gcc -static –o myfunc func.o libx.a libz.a liby.a

- 假设调用关系如下：

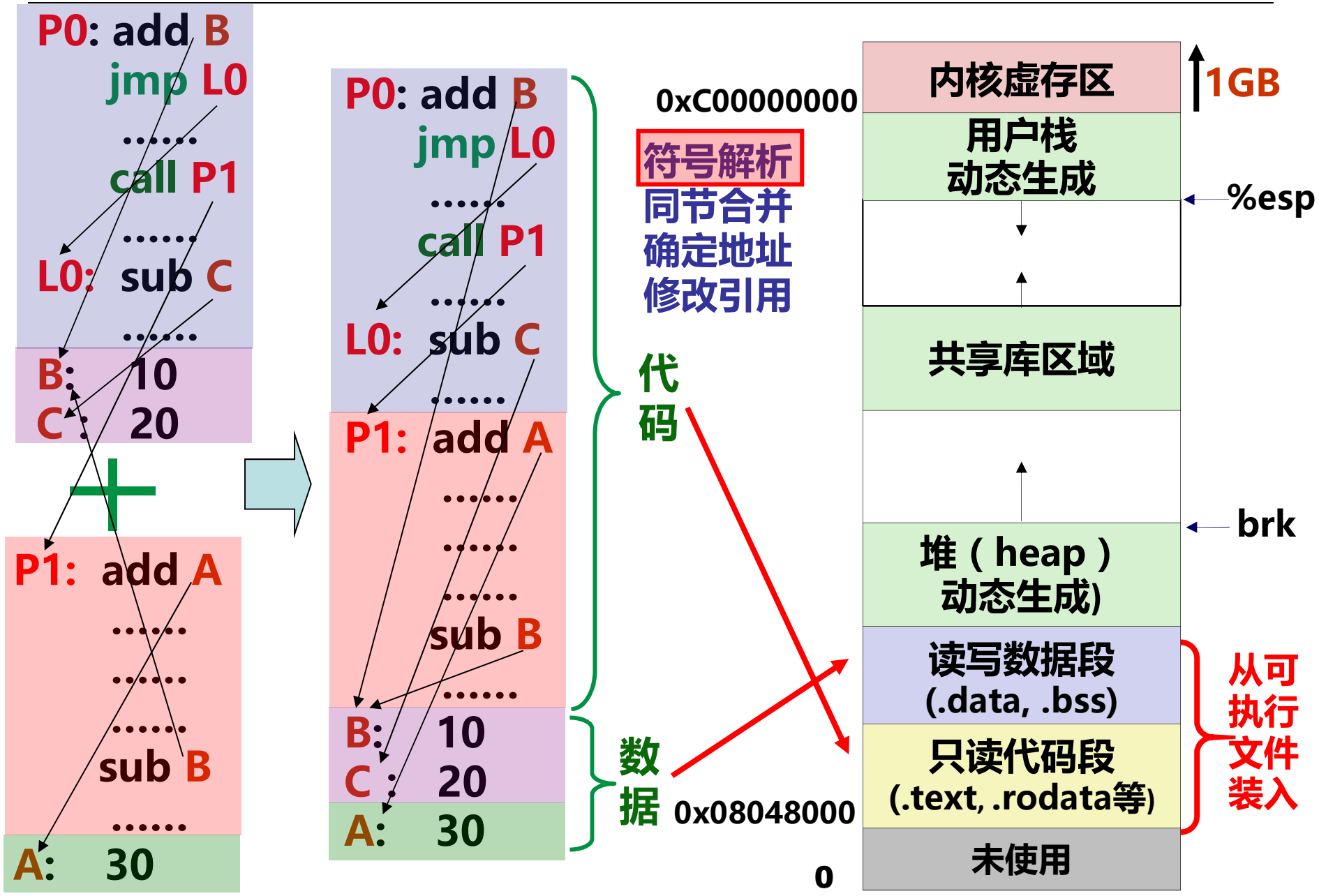
  func.o → libx.a 和 liby.a 中的函数

  libx.a → liby.a 同时 liby.a → libx.a

  则以下命令行可行：

  - gcc -static –o myfunc func.o libx.a liby.a libx.a

链接操作的步骤


P0: add B
jmp L0
.....
call P1
.....
L0: sub C
.....
B: 10
C: 20
P1: add A
.....
.....
.....
sub B
.....
A: 30
P0: add B
jmp L0
.....
call P1
.....
L0: sub C
.....
P1: add A
.....
.....
.....
sub B
.....
B: 10
C: 20
A: 30
符号解析
同节合并
确定地址
修改引用
代码
数据
0xC0000000
0x08048000
0
内核虚存区
用户栈
动态生成
%esp
共享库区域
堆（heap）
动态生成)
brk
读写数据段
(.data, .bss)
只读代码段
(.text, .rodata等)
未使用
1GB
从可执行文件装入