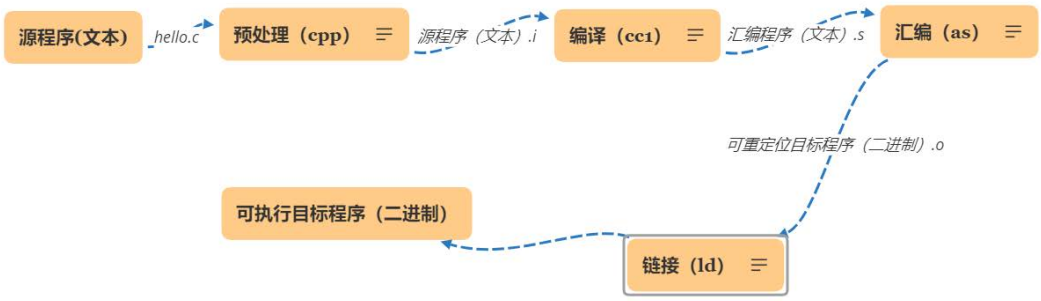
1. 程序转换处理过程



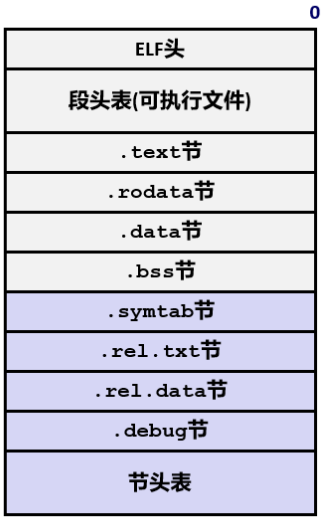
2. 链接器两个任务

1) 符号解析：
   a) 程序中有定义和引用的符号（包括变量和函数）
   b) 编译器将定义的符号存放在一个符号表中
      i. 符号表是一个结构数组，在.symtab 节中
      ii. 每个表项包含符号名、长度和位置等信息
      iii. 编译器将符号的引用存放在重定位节（.rel.text 和.rel.data）中
   c) 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
2) 重定位：
   a) 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
   b) 计算每个定义的符号在虚拟地址空间中的绝对地址
   c) 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

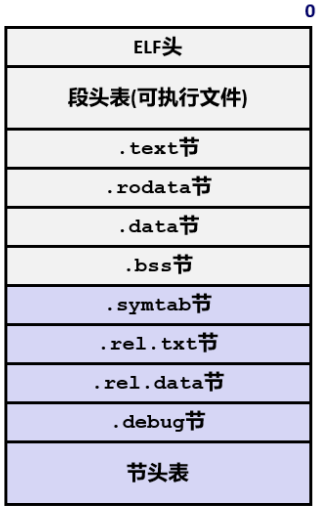
3. 目标文件格式

1) 三种目标文件：
   a) 可重定位文件（.o）
   b) 可执行目标文件（a.out、*.exe）
   c) 共享的目标文件（*.so）
2) 目标文件格式：
   a) 可重定位目标文件（具体详见 ppt12 页）
   b) 可执行目标文件（注意与可重定位目标文件的区别）（ppt29 页）

- **ELF 头**
  - 字大小、字节顺序、文件类型(.o，exec，.so)，机器类型（如IA-32），节头表的偏移、节头表的表项大小以及表项个数等等
- **段头表/程序头表（可执行文件）**
  - 页面大小，虚拟地址内存段(节)，段大小
- **节头表Section header table**
  - 每个节的节名、偏移量和大小
- **.text 节（代码）**
- **.rodata 节（只读数据）**
  - 只读数据：printf格式串、 跳转表, ...
- **.data 节 （数据/可读写）**
  - 已初始化全局变量
- **.bss 节 （未初始化全局变量）**
  - 未初始化的全局变量，仅是占位符，不占据任何实际磁盘空间。

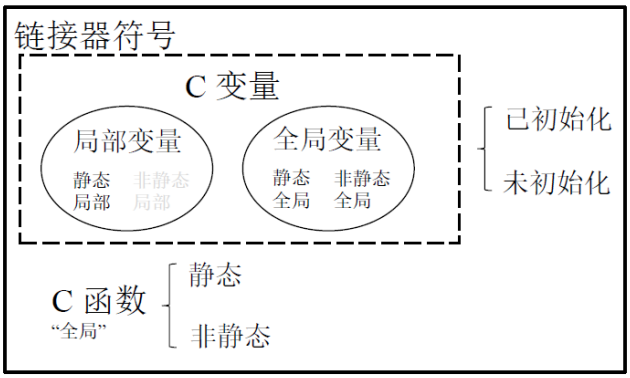


- **.symtab 节（符号表）**
  - 符号表
  - 函数和静态变量名
  - 节名称和位置
- **.rel.text 节（可重定位代码）**
  - **.text 节的可重定位信息**
  - 在可执行文件中需要修改的指令地址
  - 需修改的指令
- **.rel.data 节（可重定位数据）**
  - .data 节的可重定位信息
  - 在合并后的可执行文件中需要修改的指针数据的地址
- **.debug 节（调试）**
  - 为符号调试的信息(gcc -g)
- **.strtab节**
  - 包含symtab和debug节中的符号及节名



4. 符号和符号表

1) 三种符号：
   a) 全局符号（非 static）：由模块 m 定义并能被其他模块引用的符号。例如，非 static C 函数和非 static 的 C 全局变量
      i. 强弱特性：函数名和已初始化的全局变量名是强符号。未初始化的全局变量名是弱符号。
      ii. 符号处理规则：
         1. 强符号不能多次定义。
         2. 若一个符号被定义为一次强符号和多次弱符号，则按强定义为准。
         3. 若有多个弱符号定义，则任选其中一个
         4. 符号解析时只能有一个确定的定义（即每个符号仅占一处存储空间）。
   b) 外部符号
   c) 局部符号（带 static）：仅有模块 m 定义和引用的本地符号，带 static 的。不是局部变量。



5. 静态链接

1) 链接对象：多个可重定位目标模块（.o 文件）+静态库（标准库、自定义库）（.a 文件，其中包含多个.o 模块）
2) 静态库：
   a) 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）
   b) 可将经常使用的函数模块放到静态函数库中，以供调用
   c) 使用静态库，可增强链接器功能，使其通过查找一个或多个库文件中定义的符号来解析符号
   d) 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，链接到可执行文件中
3) 符号解析过程（ppt29 页）

6. 重定位

1) 符号解析完成后，可进行重定位工作，分三步：
   a) 合并相同的节
      i. 将集合 E 的所有目标模块中相同的节合并成新节。例如，所有.text 节合并作为可执行文件中的.text 节
   b) 对集合 D 中的定义符号进行重定位（确定地址）
      i. 确定新节中所有定义符号在虚拟地址空间中的地址。例如，为函数确定首地址，进而确定每条指令的地址，为变量确定首地址
      ii. 完成这一步后，每条指令和每个变量的地址都可确定
   c) 对引用符号进行重定位（确定地址）
      i. 修改.text 节和.data 节中对每个符号的引用（地址）（需要用到在.rel_data 和.rel_text 节中保存的重定位信息）

ii. 重定位方式有多种，只有绝对地址方式才是将引用处的地址修改为与之关联（绑定）的定义处的首地址，而其他重定位方式就不一定是这样的。例如 PC 相对地址方式，引用处填写的是一个相对地址。

2) 重定位信息：汇编器遇到引用时，生成一个重定位条目（定义不会生成重定位条目）（具体计算过程详见 ppt26）
   a) 数据引用的重定位条目在.rel_data 节中
   b) 指令（movl 这些）中引用的重定位条目在.rel_text 节中
   c) ELF 重定位条目格式
      i. PC32：PC 相对地址，PC 值通常是下一条指令在内存中的地址。
      ii. 32：绝对地址。

7. 动态链接

1) 共享库：
   a) 共享库是包含目标模块的文件，每个模块包含有代码和数据
   b) 从程序中分离出来，磁盘和内存都只有一个备份
   c) 可以在装入时或运行时动态地被加载并链接
2) 两种方式：
   a) 第一次加载运行时进行：
      i. 在 Linux 中，通常由动态链接器（ld-linux.so）自动处理
      ii. 标准 C 库（lib.so）通常按这种方式动态被链接
      iii. 思路：
         1. 当创建可执行文件时，静态执行一些链接，然后在程序加载时，动态完成链接过程。
         2. 加载 myproc 时，加载器发现在可执行文件的程序头表中有.interp 段，其中包含了动态链接器路径名 ld-linux.so，因而加载器根据指定路径加载并启动动态链接器运行。动态链接器完成相应的重定位工作后，再把控制权交给 myproc，启动器第一条指令执行。
   b) 在已经运行后进行：
      i. 在 Linux 中，通过调用 dlopen()等接口来实现（分发软件包、构建高性能 Web 服务器等）
      ii. 思路：将每个生成动态内容的函数打包在共享库中。服务器动态地加载和链接适当的函数，然后直接调用它，而不是使用 fork 和 execve 在子进程的上下文中运行函数。函数会一直缓存在服务器的地址空间中，所以只要一个简单的函数调用的开销就可以处理随后的请求了。进一步说，在运行时无需停止服务器，就可以更新已存在的函数，以及添加新的函数。
      可通过动态链接器接口提供的函数在运行时进行动态链接。