**c语言编译过程详解**

       C语言的编译链接过程要把我们编写的一个c程序（源代码）转换成可以在硬件上运行的程序（可执行代码），需要进行编译和链接。编译就是把文本形式源代码翻译为机器语言形式的目标文件的过程。链接是把目标文件、操作系统的启动代码和用到的库文件进行组织形成最终生成可执行代码的过程。过程图解如下：

       从图上可以看到，整个代码的编译过程分为编译和链接两个过程，编译对应图中的大括号括起的部分，其余则为链接过程。

**编译过程**

编译过程又可以分成两个阶段：编译和会汇编。

**编译**

   编译是读取源程序（字符流），对之进行词法和语法的分析，将高级语言指令转换为功能等效的汇编代码，源文件的编译过程包含两个主要阶段：

   第一个阶段是预处理阶段，在正式的编译阶段之前进行。预处理阶段将根据已放置在文件中的预处理指令来修改源文件的内容。如#include指令就是一个预处理指令，它把头文件的内容添加到.cpp文件中。这个在编译之前修改源文件的方式提供了很大的灵活性，以适应不同的计算机和操作系统环境的限制。一个环境需要的代码跟另一个环境所需的代码可能有所不同，因为可用的硬件或操作系统是不同的。在许多情况下，可以把用于不同环境的代码放在同一个文件中，再在预处理阶段修改代码，使之适应当前的环境。

主要是以下几方面的处理：

（1）宏定义指令，如 #define a   b  
对于这种伪指令，预编译所要做的是将程序中的所有a用b替换，但作为字符串常量的 a则不被替换。还有 #undef，则将取消对某个宏的定义，使以后该串的出现不再被替换。

（2）条件编译指令，如#ifdef，#ifndef，#else，#elif，#endif等。  
这些伪指令的引入使得程序员可以通过定义不同的宏来决定编译程序对哪些代码进行处理。预编译程序将根据有关的文件，将那些不必要的代码过滤掉  
  
（3） 头文件包含指令，如#include "FileName"或者#include <FileName>等。  
在头文件中一般用伪指令#define定义了大量的宏（最常见的是字符常量），同时包含有各种外部符号的声明。采用头文件的目的主要是为了使某些定义可以供多个不同的C源程序使用。因为在需要用到这些定义的C源程序中，只需加上一条#include语句即可，而不必再在此文件中将这些定义重复一遍。预编译程序将把头文件中的定义统统都加入到它所产生的输出文件中，以供编译程序对之进行处理。包含到c源程序中的头文件可以是系统提供的，这些头文件一般被放在/usr/include目录下。在程序中#include它们要使用尖括号（<>）。另外开发人员也可以定义自己的头文件，这些文件一般与c源程序放在同一目录下，此时在#include中要用双引号（""）。  
  
（4）特殊符号，预编译程序可以识别一些特殊的符号。  
例如在源程序中出现的LINE标识将被解释为当前行号（十进制数），FILE则被解释为当前被编译的C源程序的名称。预编译程序对于在源程序中出现的这些串将用合适的值进行替换。  
  
   预编译程序所完成的基本上是对源程序的“替代”工作。经过此种替代，生成一个没有宏定义、没有条件编译指令、没有特殊符号的输出文件。这个文件的含义同没有经过预处理的源文件是相同的，但内容有所不同。下一步，此输出文件将作为编译程序的输出而被翻译成为机器指令。  
  
   第二个阶段编译、优化阶段，经过预编译得到的输出文件中，只有常量；如数字、字符串、变量的定义，以及C语言的关键字，如main,if,else,for,while,{,}, +,-,\*,\等等。  
  
   编译程序所要作得工作就是通过词法分析和语法分析，在确认所有的指令都符合语法规则之后，将其翻译成等价的中间代码表示或汇编代码。  
  
   优化处理是编译系统中一项比较艰深的技术。它涉及到的问题不仅同编译技术本身有关，而且同机器的硬件环境也有很大的关系。优化一部分是对中间代码的优化。这种优化不依赖于具体的计算机。另一种优化则主要针对目标代码的生成而进行的。  
  
   对于前一种优化，主要的工作是删除公共表达式、循环优化（代码外提、强度削弱、变换循环控制条件、已知量的合并等）、复写传播，以及无用赋值的删除，等等。

    后一种类型的优化同机器的硬件结构密切相关，最主要的是考虑是如何充分利用机器的各个硬件寄存器存放的有关变量的值，以减少对于内存的访问次数。另外，如何根据机器硬件执行指令的特点（如流水线、RISC、CISC、VLIW等）而对指令进行一些调整使目标代码比较短，执行的效率比较高，也是一个重要的研究课题。  
  
**汇编**  
  
   汇编实际上指把汇编语言代码翻译成目标机器指令的过程。对于被翻译系统处理的每一个C语言源程序，都将最终经过这一处理而得到相应的目标文件。目标文件中所存放的也就是与源程序等效的目标的机器语言代码。目标文件由段组成。通常一个目标文件中至少有两个段：

代码段：该段中所包含的主要是程序的指令。该段一般是可读和可执行的，但一般却不可写。

数据段：主要存放程序中要用到的各种全局变量或静态的数据。一般数据段都是可读，可写，可执行的。

UNIX环境下主要有三种类型的目标文件：

（1）可重定位文件  
其中包含有适合于其它目标文件链接来创建一个可执行的或者共享的目标文件的代码和数据。

（2）共享的目标文件

这种文件存放了适合于在两种上下文里链接的代码和数据。第一种是链接程序可把它与其它可重定位文件及共享的目标文件一起处理来创建另一个 目标文件；  
第二种是动态链接程序将它与另一个可执行文件及其它的共享目标文件结合到一起，创建一个进程映象。

（3）可执行文件  
  
   它包含了一个可以被操作系统创建一个进程来执行之的文件。汇编程序生成的实际上是第一种类型的目标文件。对于后两种还需要其他的一些处理方能得到，这个就是链接程序的工作了。

**链接过程**

   由汇编程序生成的目标文件并不能立即就被执行，其中可能还有许多没有解决的问题。  
  
   例如，某个源文件中的函数可能引用了另一个源文件中定义的某个符号（如变量或者函数调用等）；在程序中可能调用了某个库文件中的函数，等等。所有的这些问题，都需要经链接程序的处理方能得以解决。  
  
   链接程序的主要工作就是将有关的目标文件彼此相连接，也即将在一个文件中引用的符号同该符号在另外一个文件中的定义连接起来，使得所有的这些目标文件成为一个能够诶操作系统装入执行的统一整体。  
  
   根据开发人员指定的同库函数的链接方式的不同，链接处理可分为两种：

（1）静态链接  
       在这种链接方式下，函数的代码将从其所在地静态链接库中被拷贝到最终的可执行程序中。这样该程序在被执行时这些代码将被装入到该进程的虚拟地址空间中。静态链接库实际上是一个目标文件的集合，其中的每个文件含有库中的一个或者一组相关函数的代码。

（2） 动态链接  
  
   在此种方式下，函数的代码被放到称作是动态链接库或共享对象的某个目标文件中。链接程序此时所作的只是在最终的可执行程序中记录下共享对象的名字以及其它少量的登记信息。在此可执行文件被执行时，动态链接库的全部内容将被映射到运行时相应进程的虚地址空间。动态链接程序将根据可执行程序中记录的信息找到相应的函数代码。  
  
   对于可执行文件中的函数调用，可分别采用动态链接或静态链接的方法。使用动态链接能够使最终的可执行文件比较短小，并且当共享对象被多个进程使用时能节约一些内存，因为在内存中只需要保存一份此共享对象的代码。但并不是使用动态链接就一定比使用静态链接要优越。在某些情况下动态链接可能带来一些性能上损害。   
  
   我们在linux使用的gcc编译器便是把以上的几个过程进行捆绑，使用户只使用一次命令就把编译工作完成，这的确方便了编译工作，但对于初学者了解编译过程就很不利了，下图便是gcc代理的编译过程：

http://student.csdn.net/image/zoom.gif

从上图可以看到：

**预编译**将.c 文件转化成 .i文件  
使用的gcc命令是：gcc –E  
对应于预处理命令cpp

**编译**将.c/.h文件转换成.s文件  
使用的gcc命令是：gcc –S  
对应于编译命令 cc –S

**汇编**将.s 文件转化成 .o文件  
使用的gcc 命令是：gcc –c  
对应于汇编命令是   as

**链接**将.o文件转化成可执行程序  
使用的gcc 命令是： gcc  
对应于链接命令是   ld

   总结起来编译过程就上面的四个过程：预编译、编译、汇编、链接。Lia了解这四个过程中所做的工作，对我们理解头文件、库等的工作过程是有帮助的，而且清楚的了解编译链接过程还对我们在编程时定位错误，以及编程时尽量调动编译器的检测错误会有很大的帮助的。