Conceptos Básicos Relacionados con la Traducción

1. Compilación, enlace y carga

Compilación, enlace y carga son las fases básicas que hay que seguir para que un ordenador ejecute la interpretación de un texto escrito mediante la utilización de un lenguaje de alto nivel.

Por regla general, el compilador no produce directamente un fichero ejecutable, sino que el código generado se estructura en módulos que se almacenan en un fichero objeto. Los ficheros objeto poseen información relativa tanto al código máquina como a una tabla de símbolos que almacena la estructura de las variables y tipos utilizados por el programa fuente. La ***Figura 1*** muestra el resultado real que produce un compilador.

***Figura 1*.** Entrada y salida de un compilador.

Pero, ¿por qué no se genera directamente un fichero ejecutable? Sencillamente, para permitir la compilación separada, de manera que varios programadores puedan desarrollar simultáneamente las partes de un programa más grande y, lo que es más importante, puedan compilarlos independientemente y realizar una depuración en paralelo. Una vez que cada una de estas partes ha generado su correspondiente fichero objeto, estos deben fusionarse para generar un solo ejecutable.

Como se ha comentado, un fichero objeto posee una estructura de módulos también llamados registros. Estos registros tienen longitudes diferentes dependiendo de su tipo y cometido. Ciertos tipos de estos registros almacenan código máquina, otros poseen información sobre las variables globales, y otros incluyen información sobre los objetos externos (p. ej, variables que se supone que están declaradas en otro ficheros. El lenguaje C permite explícitamente esta situación mediante el modificador “extern”).

Durante la fase de enlace, el enlazador o linker resuelve las referencias cruzadas, (así se llama a la utilización de objetos externos), que pueden estar declarados en otros ficheros objeto, o en librerías (ficheros con extensión “lib” o “dll”), engloba en un único bloque los distintos registros que almacenan código máquina, estructura el bloque de memoria destinado a almacenar las variables en tiempo de ejecución y genera el ejecutable final incorporando algunas rutinas

adicionales procedentes de librerías, como por ejemplo las que implementan funciones matemáticas o de E/S básicas. La ***Figura 2*** ilustra este mecanismo de funcionamiento.

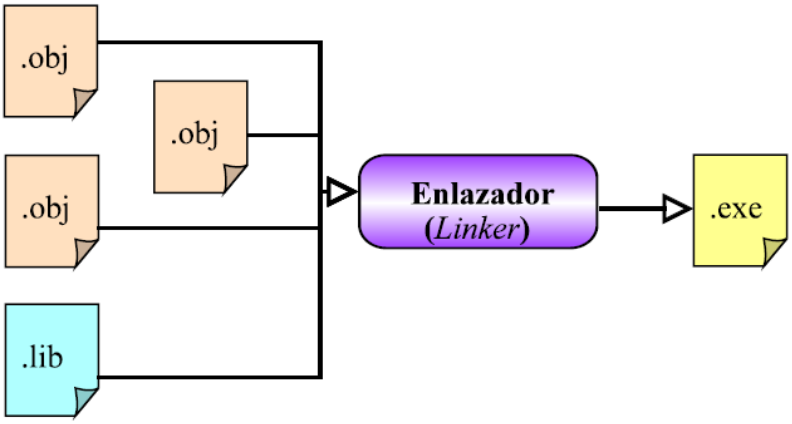

Figura 2. Funcionamiento de un enlazador.

De esta manera, el bloque de código máquina contenido en el fichero ejecutable es un código reubicable, es decir, un código que en su momento se podrá ejecutar en diferentes posiciones de memoria, según la situación de la misma en el momento de la ejecución. Según el modelo de estructuración de la memoria del microprocesador, este código se estructura de diferentes formas. Lo más usual es que el fichero ejecutable esté dividido en segmentos: de código, de datos, de pila de datos, etc.

Cuando el enlazador construye el fichero ejecutable, asume que cada segmento va a ser colocado en la dirección 0 de la memoria. Como el programa va a estar dividido en segmentos, las direcciones a que hacen referencia las instrucciones dentro de cada segmento (instrucciones de cambio de control de flujo, de acceso a datos, etc.), no se tratan como absolutas, sino que son direcciones relativas a partir de la dirección base en que sea colocado cada segmento en el momento de la ejecución. El cargador carga el fichero .exe, coloca sus diferentes segmentos en memoria (donde el sistema operativo le diga que hay memoria libre para ello) y asigna los registros base a sus posiciones correctas, de manera que las direcciones relativas funcionen correctamente. La ***Figura 3*** ilustra el trabajo que realiza un cargador de programas.

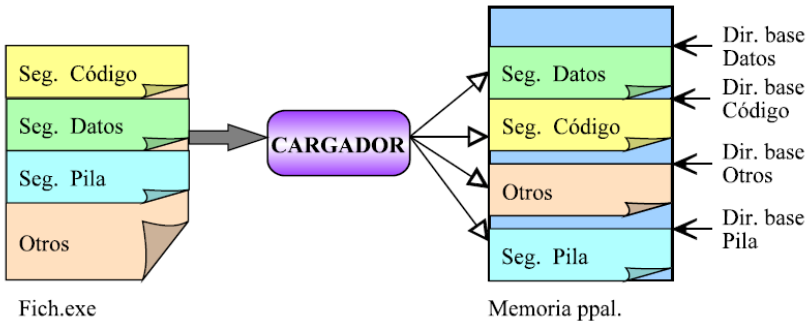

Figura 3. Tarea realizada por el cargador.

2. Pasadas de compilación

Es el número de veces que un compilador debe leer el programa fuente para generar el código. Hay algunas situaciones en las que, para realizar la compilación, no es suficiente con leer el fichero fuente una sola vez. Por ejemplo, en situaciones en las que existe recursión indirecta (una función A llama a otra B y la B llama a la A). Cuando se lee el cuerpo de A, no se sabe si B está declarada más adelante o se le ha olvidado al programador; o si lo está, si los parámetros reales coinciden en número y tipo con los formales o no. Es más, aunque todo estuviera correcto, aún no se ha generado el código para B, luego no es posible generar el código máquina correspondiente a la invocación de B puesto que no se sabe su dirección de comienzo en el segmento de código. Por todo esto, en una pasada posterior hay que controlar los errores y rellenar los datos que faltan.

Actualmente, cuando un lenguaje necesita hacer varias pasadas de compilación, suele colocar en memoria una representación abstracta del fichero fuente, de manera que las pasadas de compilación se realizan sobre dicha representación en lugar de sobre el fichero de entrada, lo que soluciona el problema de la ineficiencia debido a operaciones de e/s.

3. Compilación incremental

Cuando se desarrolla un programa fuente, éste se recompila varias veces hasta obtener una versión definitiva libre de errores. Pues bien, en una compilación incremental sólo se recompilan las modificaciones realizadas desde la última compilación. Lo ideal es que sólo se recompilen aquellas partes que contenían los errores o que, en general, hayan sido modificadas, y que el código generado se reinserte con cuidado en el fichero objeto generado en la última compilación. Sin embargo esto es muy difícil de conseguir y no suele ahorrar tiempo de compilación más que en casos muy concretos.

La compilación incremental se puede llevar a cabo con distintos grados de afinación. Por ejemplo, si se olvida un ‘;’ en una sentencia, se podría generar un fichero objeto transitorio parcial. Si se corrige el error y se añade el ‘;’ que falta y se recompila, un compilador incremental puede funcionar a varios niveles

• A nivel de carácter: se recompila el ‘;’ y se inserta en el fichero objeto la sentencia que faltaba.
• A nivel de sentencia: si el ‘;’ faltaba en la línea 100 sólo se compila la línea 100 y se actualiza el fichero objeto.
• A nivel de bloque: si el ‘;’ faltaba en un procedimiento o bloque sólo se compila ese bloque y se actualiza el fichero objeto.
• A nivel de fichero fuente: si la aplicación completa consta de 15 ficheros fuente, y solo se modifica 1(al que le faltaba el ‘;’), sólo se compila aquél al que se le ha añadido el ‘;’, se genera por completo su fichero objeto y luego se enlazan todos juntos para obtener el ejecutable.

Lo ideal es que se hiciese eficientemente a nivel de sentencia, pero lo normal es encontrarlo a nivel de fichero. La mayoría de los compiladores actuales realizan una compilación incremental a este nivel.

4. Autocompilador

Es un compilador escrito en el mismo lenguaje que compila (o parecido). Normalmente, cuando se extiende entre muchas máquinas diferentes el uso de un compilador, y éste se desea mejorar, el nuevo compilador se escribe utilizando el lenguaje del antiguo, de manera que pueda ser compilado por todas esas máquinas diferentes, y dé como resultado un compilador más potente de ese mismo lenguaje.

5. Metacompilador

Este es uno de los conceptos más importantes con los que vamos a trabajar. Un metacompilador es un compilador de compiladores. Se trata de un programa que acepta como entrada la descripción de un lenguaje y produce el compilador de dicho lenguaje. Hoy por hoy no existen metacompiladores completos, pero sí parciales en los que se acepta como entrada una gramática de un lenguaje y se genera un autómata que reconoce cualquier sentencia del lenguaje. A este autómata podemos añadirle código para completar el resto del compilador. Ejemplos de metacompiladores son: Lex, YACC, FLex, Bison, JavaCC, JLex, Cup, PCCTS, MEDISE.

6. Descompilador

Un descompilador realiza una labor de traducción inversa, esto es, pasa de un código máquina (programa de salida) al equivalente escrito en el lenguaje que lo generó (programa fuente). Cada descompilador trabaja con un lenguaje de alto nivel concreto.

Los descompiladores se utilizan especialmente cuando el código máquina ha sido generado con opciones de depuración, y contiene información adicional de ayuda al descubrimiento de errores (puntos de ruptura, seguimiento de trazas, opciones de visualización de variables, etc.).