Cómo compilar programas en C y Assembler

Organización del Computador II

Departamento de Computación
Facultad de Ciencias Exactas y Naturales
Universidad de Buenos Aires

Segundo cuatrimestre - 2006

1. Desde los fuentes al ejecutable

Un archivo *ejecutable* o "un binario" está compuesto por numerosas partes y es el resultado de uno o varios archivos *fuente*. Los archivos fuente pueden ser de diversos tipos, lenguajes, etc. Por ello, es fundamental conocer cómo se compila, ensambla y linkea[1] un ejecutable, más aún, cuando hay que realizar esa labor manualmente.

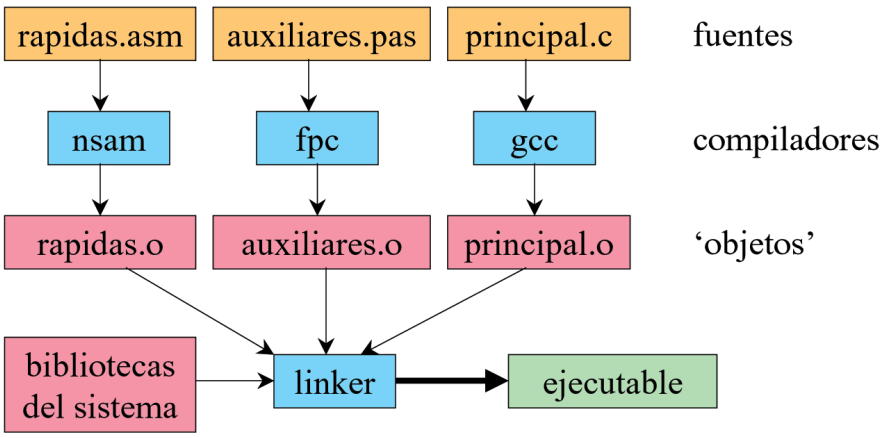

Figura 1: Diagrama del proceso de compilación y ensamblaje

Básicamente, la tarea se divide en dos partes, aunque a veces no lo notemos. La primer parte es la obtención un archivo objeto, a partir de un archivo fuente, la cual se denomina *compilación*[2] o *ensamblaje*[3], según el caso. La segunda parte consiste en la combinación de todos los archivos objeto en un archivo ejecutable, la cual se denomina *linkeo*. Además, en esa combinación, se define el punto de entrada del programa, que será la primer instrucción que se ejecutará del programa final.

[1] "*linkear*" es la castellanización del verbo en inglés *to link*, la cual usaremos vulgarmente como si fuera adecuado.

[2] *compilar*. Preparar un programa en el lenguaje máquina a partir de otro programa de ordenador escrito en otro lenguaje.

[3] *ensamblar*. Preparar un programa en lenguaje máquina a partir de un programa en lenguaje simbólico.

2. Compilando con `gcc`

Para compilar un archivo escrito en lenguaje C y obtener el archivo objeto utilizaremos el `gcc` de la siguiente manera:

```
gcc -c -o principal.o principal.c
```

donde `principal.c`[4] es nuestro código fuente en lenguaje C, `principal.o` es el archivo objeto de destino y `-c` indica que estamos compilando. En general, el comando `-o` ***nombre*** indica que el archivo de destino es ***nombre***.

Si nuestro programa está compuesto de más de un archivo en lenguaje C, deberemos ejecutar una llamada `gcc` por cada archivo y obtener así un objeto para cada uno de ellos.

3. Ensamblando con `nasm`

Para ensamblar un archivo en assembler con `nasm` debemos ejecutar la siguiente instrucción:

```
nasm -felf rapidas.asm
```

donde `rapidas.asm` es nuestro archivo assembler en cuestión y la opción `-felf` indica que el formato del archivo objeto será ELF (el más común para Linux). Aquí también se podría haber utilizado la opción `-o rapidas.o` pero `nasm` por omisión utiliza ese archivo como destino.

Nuevamente, si tuviéramos varios archivos en lenguaje ensamblador que utilizan funciones entre ellos, debemos ensamblarlos por separado y obtener así un archivo objeto para cada.

4. Linkeando todo para obtener el ejecutable

Existen varios *linkers* pero aquí mostraremos dos de ellos: el `ld` y el propio `gcc`. El linker toma todos los archivos objeto y los combina para formar el ejecutable. Como cada archivo objeto fue compilado por separado, podría haber llamadas a funciones que no están en el propio archivo objeto. Por ejemplo, si nuestro `principal.c` tiene una llamada a una función que fue programada en lenguaje ensamblador porque necesitaba ser rápida, la cual está definida en `rapidas.asm`, el `gcc` compila el código que está en `principal.c` y anota en alguna parte del archivo objeto que necesita una función con un determinado nombre. Será tarea del linker encontrar en qué archivo objeto fue definida una función con ese nombre y efectivamente enlazar esa llamada con la función correcta.

De igual modo, en el archivo en lenguaje ensamblador puede haber llamadas a funciones ajenas a ese archivo, las cuales pueden estar en lenguaje ensablador en otros archivos o bien en lenguaje C.

En un archivo en lenguaje C, para indicar que la función no está en el archivo actual simplemente se debe incluir el prototipo de la función, pero no el cuerpo. De este modo, el `gcc` sabrá qué parámetros tiene esa función y sabrá además que estará definida en otro archivo.

En un archivo en lenguaje ensamblador, para indicar que una función estará definida en otro archivo basta con incluir una línea que diga `extern nombreDeLaFuncion`. Como el lenguaje ensamblador no entiende de parámetros a funciones (*en* el lenguaje), basta con dar el nombre de la función, la cual se entenderá como una etiqueta más. Por otra parte, para decir expresamente que una función (o etiqueta) determinada puede ser utilizada desde fuera de ese archivo se debe

[4] Es destacable notar que la extensión `.c`, en minúscula, denota que el lenguaje es C y no C++

incluir la línea `global miFuncion`. Si no incluimos esta línea, el `nasm` no incluirá la dirección y el nombre de esa función en el archivo objeto y el linker no podrá encontrarla.

De este modo, podremos mezclar funciones escritas en distintos lenguajes y linkearlas todas juntas.

4.1. Linkeando con `ld`

Hasta el momento no hemos hablado mucho del punto de entrada del programa. El punto de entrada es una función que tiene un nombre fijo, definido por el linker, que tácitamente se entiende como tal. En el caso del `ld`, el punto de entrada es la función o etiqueta `_start`. Para que el linker la reconozca, en el caso de haber sido escrita en lenguaje ensamblador, debemos colocar `global _start` en nuestro fuente.

La sintaxis resumida es la siguiente:

```
ld -s -o ejecutable principal.o rapidas.o auxiliares.o
```

Nuevamente, la opción `-o` indica que el archivo de destino será *`ejecutable`*. Los archivos que linkeará irán a continuación, pudiendo ser uno o muchos.

4.2. Linkeando con `gcc`

A veces, además de las funciones propias escritas en otros lenguajes, utilizamos funciones que vienen en la biblioteca estandar de C, como `printf()`, `malloc`, etc. En estos casos, si intentamos linkear con `ld` notaremos que no encontrará la definición de estas funciones. Una solución simple para ello es utilizar el `gcc` como linker pues este conoce además las funciones de la biblioteca estandar de C.

Otra diferencia destacable con `ld` es que el punto de entrada que pondrá `gcc` será la función `main`, ya sea definida en un archivo `.c` o bien con un `global main` desde un archivo en lenguaje ensamblador.

La sintaxis es la siguiente:

```
gcc -o ejecutable principal.o rapidas.o
```

5. Extras

Una herramienta útil cuando no tenemos claro qué funciones está exportando un archivo y qué funciones ajenas a él está usando, es el `nm`. Esta utilidad nos muestra la tabla de nombres ajenos usados por un archivo objeto y los nombres definidos en ese archivo y exportados. La sintaxis es muy simple, se debe colocar `nm` *`archivo_objeto.o`* en la consola de Linux y veremos la tabla de nombres. Con una letra `U` estarán las funciones ajenas y con una letra `T` las funciones definidas en ese archivo objeto.

Por otra parte, si se desea utilizar `gcc` para linkear, podemos ahorrarnos los pasos de compilar cada fuente en C y obtener un ejecutable a partir de los archivos en C y los demás archivos objeto que estamos usando:

```
gcc -o ejecutable principal.c rapidas.o
```

Esto demorará un poco más, pues debe compilar cada archivo `.c` y requiere recompilar todos los archivos `.c` al hacer una modificación en sólo uno de ellos, puesto que no obtenemos los archivos objeto intermedios del proceso para reusarlos.