Compilación de manera tradicional un programa C, mediante gcc

Los lenguajes de programación compilados, no necesitan de un intérprete, si no que necesitan compiladores y linkers.

Los lenguajes de programación compilados no requieren que exista un programa que los vaya leyendo, interprete, si no que, antes de ejecutarse, deben ser traducidos a un lenguaje que la máquina sea capaz de entender directamente, sin intermediarios. En ese proceso se llama al compilador y al enlazador o linker.

- Al compilador se le pasan los archivos de código fuente y los convierte a un archivo que la máquina sea capaz de entender. Es decir, convierte un programa legible por el humano a uno legible por la máquina. Estos nuevos archivos se llaman objetos, y, en linux, suelen tener la extensión .o o .obj dependiendo del compilador.
- El linker toma los objetos generados por el compilador y los une para crear ejecutables. También busca las librerías que se quieran añadir, y las añade en el caso de que sean librerías estáticas.

El compilador GCC (GNU C Compiler) es capaz de hacer ambas cosas e internamente hará ambas si no le especificamos lo contrario. En los proyectos un poco complicados (que tengan más de un archivo o usen librerías) así deberemos hacerlo, compilando primero y linkando después desde la consola.

```
gcc -c suid.c        # Crea el objeto testsuid.o, -c indica sólo compilar generará el fichero suid.o
gcc suid.o           # Crea el ejecutable (por defecto a.out)
./a.out              # Ejecutar el binario recién creado
```

fichero suid.c

```
#include <stdio.h>
#include <unistd.h>
int main(){
    char a;
    printf("UID:%d,EUID: %d,GID:%d,EGID;%d\ n",getuid(),geteuid(),getgid(),getegid());
    scanf("%c",&a);
}
```

make makefile

Cuando se tiene que compilar programas grandes, en los que hay muchos ficheros fuente, ficheros .c, y muchos ficheros de headers, ficheros .h, librerías, y además repartidos por varios directorios, hacer esto por cada uno de ellos y luego linkar todos los .o en un único ejecutable puede muy tedioso y en ocasiones ingobernable. Mediante el comando make podemos hacerlo de forma más eficiente. Make nos permite mediante ficheros llamados makefile definir el orden en el que se compilarán los ficheros .c y linkarán los ficheros .o. Una vez definidos estos archivos con lanzar la orden *make* él mismo sabrá como debe actuar, es decir, es capaz de saber qué cosas hay que recompilar. Si cuando estamos depurando nuestro programa tocamos un fichero fuente, al compilar con **make** sólo se recompilaran aquellos ficheros que dependan del que hemos tocado. Evitando errores y tiempo de compilación.

El Makefile tiene entradas de la siguiente forma:

objetivo: requisitos
regla para conseguirlo

fichero makefile

```
all: a.out

a.out: suid.o
	gcc suid.o		deber de haber una tabulación

suid.o: suid.c
	gcc -c suid.c		deber de haber una tabulación
```

Ejecutamos el comando

```
make
```

Nos muestra la siguiente salida:

```
gcc -c suid.c
gcc suid.o
```

Obteniendo el fichero ejecutable a.out

Ejecutamos el binario a.out recien creado

```
./a.out
```