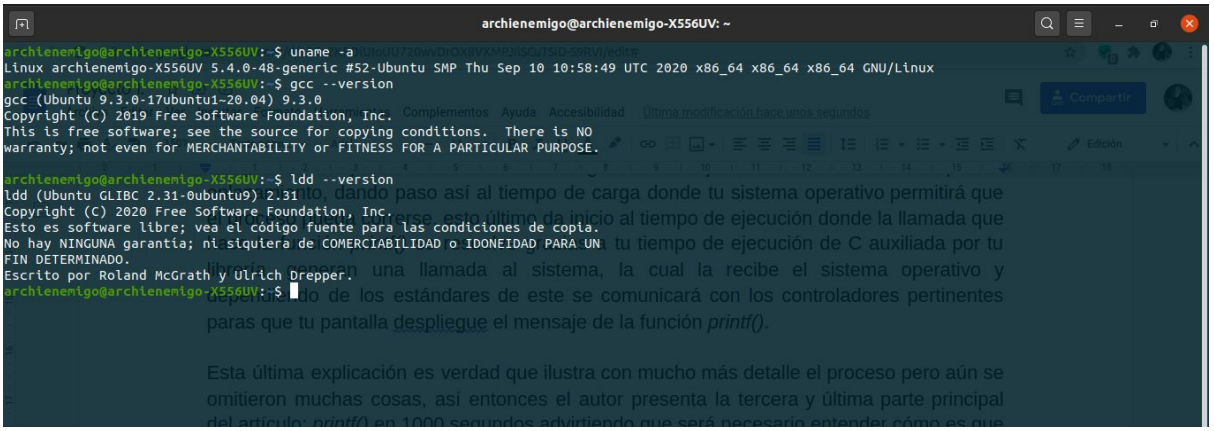
“Tearing apart printf()”: Resúmen

El artículo plantea tres formas de “desmenuzar” la función *printf()*, la primera la desglosa en una explicación que el autor considera se entiende en treinta segundos, explicando a muy grandes rasgos los procesos por los que pasa la ejecución de un programa con esta función *printf()*, segmentada en cuatro partes encargadas de diferentes agentes: tu programa, tu librería en tiempo de ejecución de C, tu sistema operativo y por último tu pantalla. Básicamente explica que una vez que tu compilador/enlazador produce un código binario de tu programa, y el código binario de la función *printf()* es un apuntador en tiempo de carga a tu librería de C, tu tiempo de ejecución de C corrige el formato y envía la cadena al kernel, tu sistem operativo después es el intermedio que permite que tu arreglo acceda a su representación en consola a través de un controlador de dispositivo que permitirá mostrar el texto en pantalla. Esta es la forma más “burda” de explicar lo que sucede, pero el autor del artículo nos muestra una segunda forma de desglosar los pasos necesarios para que la función *printf()* se ejecute.

printf() en noventa segundos es la siguiente forma de explicar lo anterior pero aquí el autor hace referencia a cinco segmentos: Tiempo de diseño, tiempo de compilación, tiempo de enlazamiento, tiempo de carga y tiempo de ejecución. Cada una de estas tiene sus propios componentes. Empezando por el tiempo de diseño cuyo personaje principal es el código fuente que gracias a tu compilador a través de los encabezados (librerías) se transforma en código objeto, durante el tiempo de compilado. Seguido de esto, el enlazador toma el código objeto y lo transforma en código binario ejecutable durante el tiempo de enlazamiento, dando paso así al tiempo de carga donde tu sistema operativo permitirá que el proceso pueda correrse, esto último da inicio al tiempo de ejecución donde la llamada que hace la función *printf()* se resuelve gracias a tu tiempo de ejecución de C auxiliada por tu librería, generan una llamada al sistema, la cual la recibe el sistema operativo y dependiendo de los estándares de este se comunicará con los controladores pertinentes paras que tu pantalla despliegue el mensaje de la función *printf()*.

Esta última explicación es verdad que ilustra con mucho más detalle el proceso pero aún se omitieron muchas cosas, así entonces el autor presenta la tercera y última parte principal del artículo: *printf()* en 1000 segundos advirtiendo que será necesario entender cómo es que el compilador, bibliotecas, sistemas operativos y el hardware funcionan por separado y en conjunto para poder entender verdaderamente cómo es que la función *printf()* funciona.



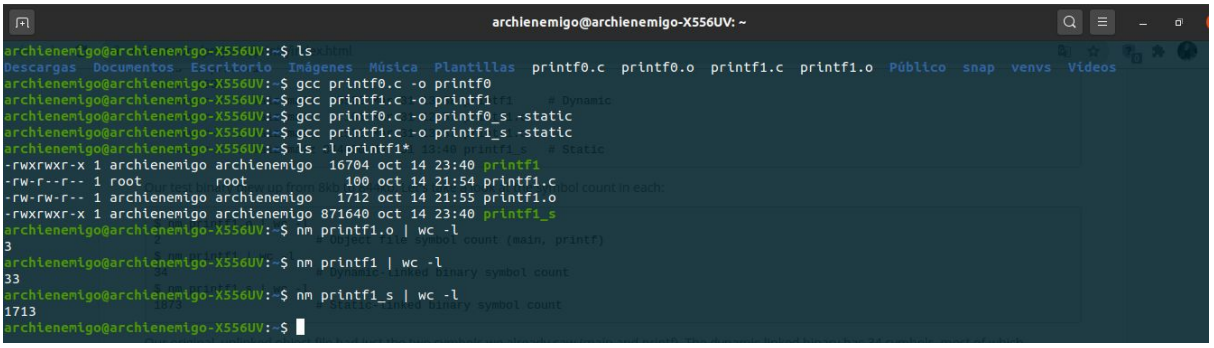
La función *printf()* nace de la idea de una interfaz que permita al programador producir salidas sin entender realmente qué está ocurriendo por detrás. Es decir, el programador llama a la función *printf()* y el sistema se encargaría del resto, y el artículo leído es la prueba de que esta idea de ocultar los detalles de la implementación de hecho funciona.

La biblioteca <stdio.h> incluye los estándares con los que *printf()* funciona, cuya interfaz no ha cambiado mucho, la verdad, pero sí se han incluido muchísimas características más, sobre todo a partir del 2011.

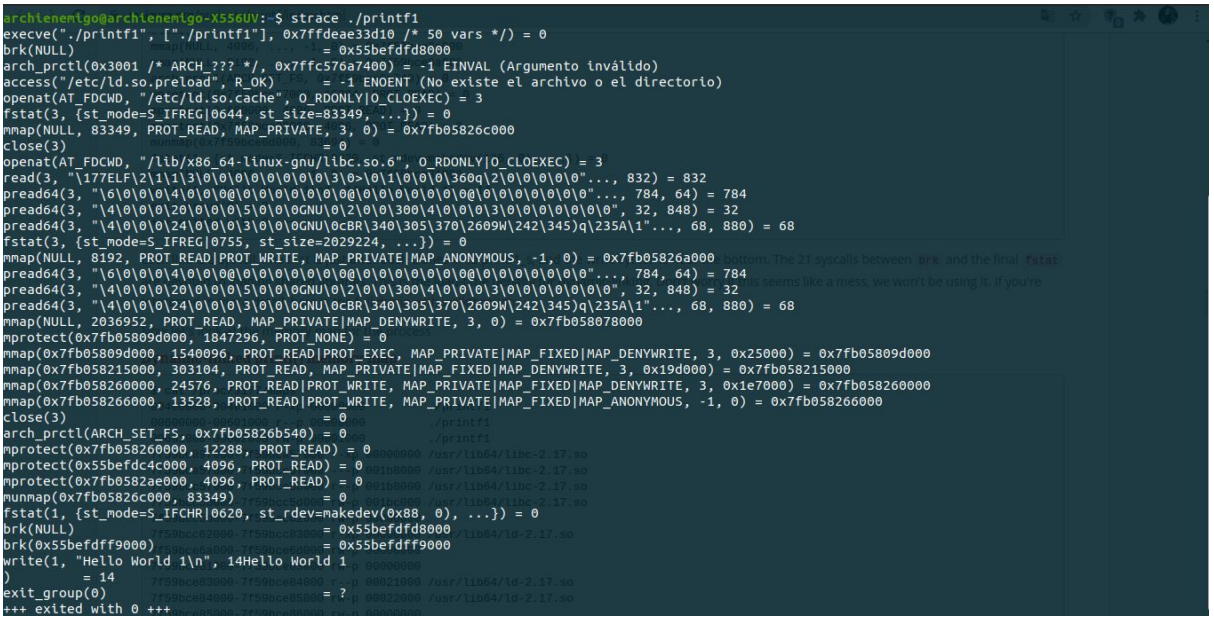
Vale la pena aclarar que no todos los programas que hacen una llamada a la función *printf* usan *printf()*, esto porque si la llamada no contiene formatos entonces el compilador no se molesta ni en usar la función *printf()* completa, sino que usa una función llamada *puts* para mostrar en pantalla tu mensaje, esta es una forma de optimizar el código que ni siquiera depende de ti porque la sustitución ocurre durante el análisis semántico del compilador, no durante la optimización del mismo. Recordando que el compilador no sabe donde *puts()* y *printf()* están, sólo sabe que existen en algún lado gracias a la biblioteca <*stdio.h*>.

```
archienemigo@archienemigo-X556UV:~$ sudo vim printf1.c
archienemigo@archienemigo-X556UV:~$ sudo vim printf0.c
archienemigo@archienemigo-X556UV:~$ gcc printf0.c -c -o printf0.o
archienemigo@archienemigo-X556UV:~$ nm printf0.o
                 U _GLOBAL_OFFSET_TABLE_
0000000000000000 T main
                 U puts
archienemigo@archienemigo-X556UV:~$ nm printf1.o
                 U _GLOBAL_OFFSET_TABLE_
0000000000000000 T main
                 U printf
archienemigo@archienemigo-X556UV:~$
```

El trabajo del enlazador es construir un binario que incluya todo el código en un sólo paquete. Dependiendo de si es estático o dinámico se generarán hasta casi 2000 símbolos y 34 respectivamente, dichos contienen toda la información necesaria que podrían requerir las bibliotecas.



Lo que sigue en el proceso es preparar el ambiente de ejecución que implica encontrar el código en bibliotecas compartidas y mapearlos en memoria, y el comando strace ayuda a visualizarlo:



En esta parte se hacen llamadas al sistema delegadas para cargar las bibliotecas compartidas. Lo que sigue es mapear en memoria el proceso.

Los últimos pasos en resumen son:

La llamada de *printf()* brinca a la biblioteca estándar, después procesar cada parte de la cadena dentro de la función “Hello world”, “%d”, “\n”; seguido de eventos que hacen llamas de sistemas al kernel, para que el controlador escriba la cadena de salida, y la salida de consola en el buffer se actualiza para imprimir “Hello world!” en tu pantalla.