UNIVERSIDAD NACIONAL AUTÓNOMA DE MÉXICO
FACULTAD DE INGENIERÍA
DIVISIÓN DE INGENIERÍA ELÉCTRICA
INGENIERÍA EN COMPUTACIÓN

Firmware

-de BIOS a UEFI-

Elaboró:

- *Santos Albañil Morteo*

17/noviembre/2021

Contenido

Introducción 3
BIOS 3
Funciona en MODO REAL (16 bits) 3
Esquema de particiones MBR 4
Código de arranque de 446 bytes 4
Tamaños de sector de 512 bytes 4
UEFI 4
UEFI y consideraciones erróneas 5
¿Y entonces? 6
Hola mundo 7
Conclusiones 8
Referencias 9
Anexo A. Código de ejemplo 10
Anexo B. Otras referencias usadas 11
Referencia a páginas de interés 12

Introducción

A mediados de la década de los 70´s surgieron los primeros softwares de Firmware tipo "BIOS", su función, de modo muy resumido, era la de hacer de intermediario entre el hardware[1] que empezaba a ser muy heterogéneo y un "primitivo" sistema operativo que se pretendía se instalase y funcionase sin importar eso[2]. Fue tal el éxito en su propósito, que se mantuvo como el paradigma de facto por casi 25 años, Intel comenzó con el desarrollo de un sustituto del BIOS en 1998, de este intento surgió EFI, pensado originalmente para la arquitectura Itanium (lanzadas en 2001 y 2002) y no fue hasta el 2005 que se conformó "The Unified EFI Forum" (Intel, 2021). Aun así, actualmente muchas de las computadoras de uso cotidiano continúan usando BIOS y, la mayoría del hardware que ya no lo tiene, lo emula sobre UEFI con CSM.

El BIOS indudablemente se actualizó, pero sus bases fundamentales se mantuvieron intactas, lo que con el tiempo generó la acumulación de una serie de deficiencias. Además, BIOS podía tener variaciones significativas de un fabricante a otro.

BIOS

Funciona en MODO REAL (16 bits)

BIOS se lanzó como tal con la IBM PC-5150, que tenía un procesador INTEL 8088 de 16 bits, lanzado en 1981 (IBM, www.ibm.com, 2021). Sobre el código de 16 bits original se construyeron las actualizaciones posteriores, esto aunado al hecho de que se mantuviera la retrocompatibilidad para código de 16 bits, incluso en los procesadores más modernos (INTEL, 2021), hizo que fuese innecesario migrar el código del BIOS por mucho tiempo. Funcionar en modo real limita la potencia del hardware en el arranque e incluso después de este, el procesador solo tiene 20 bits para direccionar RAM, por lo que también la RAM está limitada a 1 MB (INTEL, 2021). Existe hardware nuevo mapeado fuera de este rango y, por lo tanto, inaccesible en este modo.

La interfaz de BIOS suele prescindir del ratón y usa normalmente un modo de pantalla VGA de solo texto, esto podría tener varias motivaciones:

1. Las memorias sobre las que estaba guardado BIOS estaban limitadas en tamaño[3]
2. Originalmente estaba programada en ensamblador dificultando su programación
3. Aun usando un lenguaje de alto nivel, añadir interfaz gráfica hace el código más pesado
4. Las pocas funciones que ofrecía no requerían estrictamente una interfaz más compleja
5. Puede asumirse que, todas las tarjetas gráficas traen este modo de solo texto por compatibilidad. Disponer de modos más avanzados no está garantizado y comúnmente requiere drivers específicos

[1] Este texto está enfocado en lo que conocemos popularmente como computadora personal, que comúnmente usan procesadores con set de instrucciones x86, aunque no necesariamente todas han tenido esta característica, y de dichas computadoras existe muchas variantes por lo que la delimitación no es estricta.

[2] CP/M fue el primero en implementar algo que puede llamarse BIOS, posteriormente IBM entro al mercado implementando su propia versión de BIOS, ésta última fue la domino el mercado.

[3] Algunas modernas tienen 32 Mbytes comparadas con los 64 Kbytes de las primeras.

Esquema de particiones MBR

El BIOS funciona con un esquema de particiones conocido popularmente como MBR (***master boot record***), diseñado de tal modo que, el primer sector del disco (512 bytes), contiene toda la información necesaria sobre el particionado, de los cuales 446 bytes contienen código ejecutable, dejando solo 64 bytes para los datos de partición, que son cuatro, por lo tanto, cada una de las particiones tiene solo 16 bytes para definir sus características (Dixit, 2021). Una de las características de partición que se almacena es su tamaño en sectores, para lo cual se utilizan solo 4 bytes (32 bits) para cada dato, en términos llanos, el máximo tamaño de disco es de aproximadamente 2.2 TB[4], cuando en la actualidad, se ofertan discos de incluso 18 TB (westerndigital, 2021).

Código de arranque de 446 bytes

Cuando el BIOS termina su trabajo de "encendido" de la máquina, busca en el disco duro la marca 55AA[5], que le indicaría que es un disco que contiene código ejecutable, carga ese código en memoria y ejecuta un salto a ese código. Este código solo se usa para cargar más código en memoria[6]. El código MBR podía corromperse y añadir virus que eran capaces de eludir cualquier mecanismo del SO, y cuya única forma de limpieza era editando el disco antes del arranque.

Tamaños de sector de 512 bytes

El tamaño de sector de 512 bytes es un valor que se estableció como dominante al ser utilizado por la IBM-PC (no necesariamente originado por él), y que mantenía un buen compromiso de rendimiento, sectores más pequeños implicaban lecturas más lentas y fragmentación de datos, sectores más grandes implicaban desperdicio de espacio al ser el sector la unidad mínima de almacenamiento. Muchas cosas en la computadora funcionan asumiendo 512 bytes por sector, haciéndolo extremadamente difícil de modificar, a tal grado que, los discos modernos con sectores de otros tamaños deben ser múltiplos enteros de 512, e incluso medios de almacenamiento como los SSD que no funcionan con sectores los emulan. Además de los bytes de información del usuario, el disco contiene bytes con propósito de control, haciendo un gasto excesivo de ellos cuando la granularidad es fina, haciendo más costoso cada byte y degradando el rendimiento (Seagate, 30).

En general, BIOS ya no es adecuado para las computadoras modernas, incluso, a pesar de no existir en muchos sistemas, sus efectos se siguen sintiendo en los mecanismos de retrocompatibilidad.

UEFI

UEFI pretende ser la solución a las limitaciones planteadas por BIOS, no como una actualización de BIOS, sino como algo completamente nuevo. A continuación, se presenta una lista de algunas cosas, que ofrece UEFI, iniciando por aquellas que anteriormente se numeraron como problemas en BIOS:

---

[4] Algunos SO son capaces de direccionar discos duros de mayor tamaño, al salirse de este diseño.

[5] En realidad, la palabra (dos bytes) es AA55h, solo que los procesadores Intel son del tipo *little-endian*, lo que significa que, cuando se trata de almacenar más de un byte, se almacena primero la parte menos significativa y después la más significativa, al revisar el MBR con un editor hexadecimal aparecerá como AA 55.

[6] Algunas versiones de Windows también lo usan para verificar si existe un módulo TPM

- UEFI funciona directamente en modos de 32 y 64 bits (dependiendo de la implementación y arquitectura de procesador), lo que da acceso completo a las funciones del procesador y a toda la memoria disponible, además permite acceso al hardware en general[7]
- Se cambia a un esquema de particiones "GUID Partition Table (GPT)", que elimina el límite de 4 particiones, la implementación de Windows, por ejemplo, admite 128, la de Linux 256, este límite está establecido por el SO y no por UEFI, que tiene un campo de 32 bits para el número de particiones y un tope teórico de 9.4 ZB[8] para el tamaño de disco y de partición, que supera por mucho las necesidades actuales (UEFI, 2017).
- UEFI "entiende" la tabla de particiones GPT (también MBR y El Torito, ISO-9660), sistemas de archivos FAT12, FAT16 y FAT32[9] (definido también en UEFI), por lo que es capaz de cargar ejecutables (en formato .efi PE) de los medios de almacenamiento disponibles, y no están limitados en tamaño, salvo las limitaciones del sistema de archivos.
- Por compatibilidad, se puede informar un tamaño de bloque de 512 bytes, pero UEFI admite directamente tamaños de bloque de 4096, o el que admita el hardware.
- UEFI ofrece una capa de abstracción más avanzada, que permite el diseño de entornos pre-SO con capacidades amplias como: uso de interfaces gráficas, accesos a red cableada e inalámbrica, uso de mouse, entre otras. Existen framework´s que permiten programar estas características en un lenguaje de alto nivel.

UEFI puede ampliarse y recibir funcionalidad cargando módulos adicionales[10] programados en lenguajes de alto nivel, por ejemplo, C[11]. UEFI soporta múltiples arquitecturas, las diferencias entre ellas es manejada por el Firmware especifico de cada una, permitiendo ofrecer la misma interfaz (UEFI) a pesar de las diferencias.

UEFI y consideraciones erróneas

A pesar de que ya pasaron más de 15 años desde que la especificación UEFI fue lanzada (2006), aún existen varias confusiones respecto a lo que UEFI es y lo que puede hacer. Con el reciente lanzamiento de Windows 11, que tiene como requisitos para poder ser instalado que el sistema cuente con Secure Boot y modulo TPM (microsoft, 2021), la situación se tornó más confusa con cientos de páginas publicando información errónea o poco rigurosa al respecto.

- UEFI es un estándar de Interfaz de Firmware creado y adoptado por muchas empresas, y no es algo que se le ocurriese a Microsoft exclusivamente, de hecho, aunque muchas empresas veían un problema en el modelo BIOS, fue Intel quien dio los primeros pasos exitosos[12].

[7] Con los controladores adecuados
[8] Al parecer al cálculo se hace asumiendo un tamaño de bloque de 512 bytes (LBA), que se mantiene por compatibilidad, pero tanto a nivel físico como lógico, hay unidades que funcionan con LBA de 4096 bytes, pudiendo ser el límite teórico de 75ZB aproximadamente, o inclusive mayor cambiando el tamaño de bloque.
[9] Basados en el FAT de Microsoft, pero redefinidos y documentados (estandarizados para UEFI)
[10] El firmware en sí y las aplicaciones que corren sobre él
[11] Es el único lenguaje en el que logré compilar una aplicación
[12] En el siguiente enlace pueden verse los participantes: https://uefi.org/members

- Secure Boot está definido dentro de UEFI como un mecanismo para evitar la ejecución de código no autorizado por el usuario, dicho mecanismo es necesario para prevenir la infección del código de arranque (o cualquier otro), como sucedía con MBR. La confusión y la crítica surgen porque es Microsoft quien funge como autenticador del código, y porque las firmas para arrancar Windows vienen preinstaladas en la implementación de la mayoría de los fabricantes de placas base para computadoras. Microsoft solicita que Secure Boot esté activado, pero también que se pueda deshabilitar, es el usuario, en última instancia, quien decide si activarlo o no, salvo que quien implementa UEFI decida lo contrario. Adicionalmente, las claves de arranque seguro pueden eliminarse o se pueden añadirse nuevas[13]. Secure Boot tiene el potencial de bloquear cualquier software (tiene lista negra), pero por ahora, solo es eso (UEFI, 2017).
- TPM es un estándar para un cripto-procesador, por extensión se llama modulo TPM a las implementaciones, es independiente del procesador central y puede ser usado por el Firmware, el SO o cualquier otra aplicación para tareas de seguridad del software. También, puede trabajar en conjunto con Secure Boot para garantizar un arranque con software autorizado, pero no es su única finalidad, en algunas placas base puede deshabilitarse y en otras incluso puede retirarse. Es una capa adicional de seguridad y no es parte de UEFI.
- UEFI no es algo que se puede deshabilitar o habilitar, viene grabado en una memoria integrada directamente en la placa base, y se ejecuta siempre (en las placas más modernas que lo traen).
- UEFI no es BIOS, aunque muchos siguen llamándolo así, e incluso algunos fabricantes la nombran “UEFI BIOS”, esto es incorrecto, algunas implementaciones pueden parecerse a la interfaz del viejo BIOS, pero no lo es.
- Algunas (actualmente casi todas) implementaciones UEFI pueden emular el comportamiento de BIOS, pero no implica que la placa tenga ambas cosas.
- Lo anterior debería dejar claro que, en una máquina con UEFI, puede instalarse software que no sea Windows o que no esté firmado por Microsoft[14].
- UEFI, como su nombre lo dice, es una especificación de la interfaz firmware-SO y no el Firmware en sí, aunque por extensión se llame UEFI a las implementaciones de Firmware que ofrecen esta interfaz

¿Y entonces?

Ahora que tenemos una perspectiva más amplia de lo que es UEFI, sabemos que en los sistemas donde está implementado es capaz de acceder a todas las funciones de la máquina. Además, se pueden cargar aplicaciones para prácticamente cualquier propósito, esto puede llevar a pensar que UEFI es un SO, pero está limitado en muchos aspectos porque su finalidad no es sustituir al SO (sí a BIOS).

- Solo se cuenta con los módulos o aplicaciones que se cargan específicamente, y no viene con las funcionalidades completas o uniformes que ofrece un SO.

[13] Si UEFI está correctamente implementado

[14] Hablando de arquitecturas x86-x64, en algunas implementaciones UEFI, de arquitecturas como ARM, Secure Boot no se puede deshabilitar.

- Facilita al SO o a cualquier software[15] sus tareas relacionadas al hardware, pero no lo administra de ninguna manera, ni ofrece ninguna clase de protección, solo una **interfaz** para accederlo.
- No tiene directamente mecanismos para multitarea.
- Aunque UEFI puede permitir la creación de GUI muy avanzadas, no las ofrece como tal, cada característica debe ser implementada.
- Facilita la carga de aplicaciones y el acceso al hardware, pero no ofrece ninguna de estas cosas como algo listo para usar
- UEFI presenta una máquina abstracta, más sencilla y estructurada, facilitando en gran medida el trabajo de programación, permitiendo incluso eliminar el SO para tareas sencillas.
- Se debe hacer énfasis en que, aunque UEFI contiene controladores para el hardware, dichos controladores no están pensados para sustituir los que vienen con un SO completo (UEFI, 2017) (seccion1.1), son controladores generales enfocados en "puertos", por ejemplo, ofrece una interfaz USB, eso no quiere decir que podrás usar directamente los dispositivos conectados a este puerto, para esto se requiere un controlador especifico.

Hola mundo

A pesar de ser un estándar, dependiendo de la plataforma, el mecanismo para la creación y ejecución de programas para UEFI puede variar, algunas de las razones son:

- La correcta implementación de la plataforma
- Las pequeñas variaciones de una versión a otra de la implementación
- El entorno de programación

Escribir un "hola mundo" es relativamente sencillo, manipular las complejidades del hardware no, adicionalmente, salvo la documentación oficial, no existe mucho material de apoyo (y es nulo en español).

El framework utilizado es EDK II (varios, 2021) (varios, 2021), no es el más recomendable por su tamaño y complejidad[16], pero está bien documentado, además es multiplataforma y su salida también lo es.

EDK II cuenta con bibliotecas de código para facilitar el trabajo de programación de firmware y de aplicaciones que correrán sobre este, es una desventaja si solo se busca programar aplicaciones.

Las páginas de documentación de EDK II contienen los pasos necesarios para configurar el entorno y escribir una aplicación[17], para todo lo demás, se asume que el lector sabe programar al menos a un nivel que le permita usar características del lenguaje C, como son: estructuras, apuntadores y tipos de datos.

Para la ejecución del código generado, EDK II viene con una máquina virtual que implementa UEFI, en el caso de hardware real, los detalles sobre cómo ejecutar pueden variar.

[15] Hace referencia a la posibilidad de cargar software que no sea el SO.
[16] Porque está pensado para escribir Firmware y aplicaciones
[17] https://github.com/tianocore/tianocore.github.io/wiki/Getting-Started-with-EDK-II

Algunas implementaciones incluyen un Shell[18], desde la cual puede ser lanzada cualquier otra aplicación tipo .efi, cuando esto no es así, podemos recurrir a una argucia: las implementaciones UEFI suelen cargar en automático el ejecutable ubicado en el directorio EFI->boot->bootx64.efi, del dispositivo seleccionado (UEFI, 2017). Puede crear el directorio en una memoria USB (formateada con FAT32), descargar el Shell, renombrarlo como bootx64.efi, y ubicarlo en dicho directorio; ahí mismo pueden guardarse las aplicaciones (o en cualquier otro directorio) para ser lanzadas por el Shell.

La forma de arrancar desde un dispositivo especifico también es diferente entre plataformas.

El código sigue la sintaxis de C, pero tiene algunas particularidades a considerar:

1. Se verifica que las variables declaradas estén referenciadas, solo declararlas se considera un error
2. Es casi obligatorio el uso de estructuras, funciones y apuntadores para acceder a las funcionalidades de UEFI, tener estructuras tan anidadas hace que el código se torne largo y obliga a usar alias para simplificarlo.

Se anexa código de ejemplo con funciones básicas como: entrada y salida, ciclos, bloques condicionales, definición de variables y uso de funciones.

Conclusiones

Es de suma importancia conocer las características de la máquina sobre la que se ejecuta el SO, esto que parece obvio, muchas veces se ignora. Entender las características más relevantes del hardware nos llevará a comprender el porqué de ciertos problemas en el desarrollo de SO y las formas de abordarlos, siendo el firmware el intermediario entre estos, tener una noción básica del papel que desarrolla resulta fundamental.

UEFI sin duda es una enorme mejoría respecto a BIOS, pero no es perfecto, la interfaz solo simplifica el proceso de arranque, al menos es así en entornos domésticos (Microsoft, 2021), para los SO es más sencillo usar, lo que tenían previamente desarrollado, que empezar de cero con UEFI. Lo que sí es verdad, es que ahora la mayoría de las interfaces de usuario de configuración de arranque se “ven un poco mejor”, si logras verlas…[19]

UEFI proporciona algunas facilidades para la ejecución de aplicaciones previas al SO, pero queda muy limitado en cuanto a la posibilidad de control de dispositivos, lo anterior no se debe específicamente a UEFI, más bien, a los fabricantes que solo proveen controladores específicos y fuertemente acoplados para algunos SO[20]. Incluso, SO maduros sufren este problema. Eso no debería verse como algo malo, UEFI no está diseñado para sustituir al SO, y no tendría por qué traer controladores específicos.

Lo anterior no debe desanimar a nadie a escribir aplicaciones fuera del SO, al contrario, UEFI da una excelente forma de experimentar y aprender el funcionamiento de la máquina y de los SO.

[18] Que no es otra cosa que una aplicación UEFI

[19] Las políticas de muchas implementaciones hacen complicado el acceso a la configuración, muchas de ellas implementan el acceso por medio del SO (Windows->Inicio Avanzado).

[20] Tal vez sea buena idea desacoplar del SO los controladores de hardware (utopía)

Referencias

Dixit, A. (27 de 10 de 2021). *knowitlikepro.com.* Obtenido de https://knowitlikepro.com/understanding-master-boot-record-mbr/

IBM. (26 de 10 de 2021). *www.ibm.com*. Obtenido de https://www.ibm.com: https://www.ibm.com/ibm/history/exhibits/pc25/pc25_press.html

IBM. (26 de 10 de 2021). *www.ibm.com*. Obtenido de https://www.ibm.com: https://www.ibm.com/ibm/history/exhibits/pc25/pc25_birth.html

INTEL. (25 de 10 de 2021). *Intel® 64 and IA-32 Architectures Software Developer's Manual,Volume 3, Chapter 9,"PROCESSOR MANAGEMENT AND INITIALIZATION".* Obtenido de www.intel.com: https://www.intel.com/content/www/us/en/developer/articles/technical/intel-sdm.html

Intel. (25 de 10 de 2021). *www.intel.la*. Obtenido de https://www.intel.la: https://www.intel.la/content/www/xl/es/architecture-and-technology/unified-extensible-firmware-interface/efi-homepage-general-technology.html

Microsoft. (05 de 11 de 2021). *https://docs.microsoft.com*. Obtenido de https://docs.microsoft.com: https://docs.microsoft.com/es-es/windows/client-management/img-boot-sequence

microsoft. (30 de 10 de 2021). *www.microsoft.com.* Obtenido de https://www.microsoft.com/en-us/windows/windows-11-specifications

seagate. (24 de 10 de 30). *www.seagate.com.* Obtenido de https://www.seagate.com/: https://www.seagate.com/la/es/tech-insights/advanced-format-4k-sector-hard-drives-master-ti/

UEFI. (2017). *Unified Extensible Firmware Interface (UEFI) Specification Version 2.6 Errata C.*

varios. (01 de 11 de 2021). Obtenido de https://github.com/tianocore/edk2

varios. (01 de 11 de 2021). Obtenido de https://www.tianocore.org/

westerndigital. (24 de 10 de 2021). *www.westerndigital.com.* Obtenido de https://www.westerndigital.com/products/internal-drives/wd-red-pro-sata-hdd#WD181KFGX

Anexo A. Código de ejemplo

```
#include <Uefi.h> //Para usar tipos base como EFI_TIME
#include <Library/UefiLib.h> //Para usar Print
#include <Protocol/SimpleTextIn.h>
EFI_STATUS  //Propisito desconocido pero son necesarios
EFIAPI
UefiMain (
 IN EFI_HANDLE      ImageHandle,  //identifica a la aplicacion
 IN EFI_SYSTEM_TABLE  *SystemTable  //un apuntador a una tabla con las funciones de UEFI
 )
{
        EFI_STATUS Status;    //declaracion de variable
        EFI_TIME Time;                  //no rerquieren inicializacion

        //es una buena practica verificar con la variable EFI_STATUS(Status)
        //la correcta ejecucion de las funciones
        Status = SystemTable->RuntimeServices->GetTime (&Time, NULL); //obtener la fecha del sistema

        //ejemplo if-else para verificar si se obtuvo la fecha
        if (EFI_ERROR (Status)) {
        Status=SystemTable->ConOut->OutputString(SystemTable->ConOut, L"Fallo en obtener la fecha\r\n");
        }
        else{
        Status=SystemTable->ConOut->OutputString(SystemTable->ConOut, L"Exito al obtener la fecha\r\n");
        }
        Print(L"Tiempo del sistema [%d-%d-%d %d-%d-%d-%d]\n", //usar la funcion Print provista en UefiLib.h
        Time.Year,
        Time.Month,
        Time.Day,
        Time.Hour,
        Time.Minute,
        Time.Second,
        Time.Nanosecond);
        INT32 Index;                    //declaracion de variables
        Index = 0;              //requieren inicializacion
        for(Index = 0; Index < 8; Index ++) //ejemplo de ciclo
        {
        //UEFI usa cadenas con caracteres de 16 bits L le indica eso al compilador
        //Salida usando UEFI          mi propia consola,    la cadena para imprimir
        Status=SystemTable->ConOut->OutputString(SystemTable->ConOut, L"Hello World\r\n");
        Print(L"%d\n",Index); //es mas complicado usar la salida directa para variables
        }
        UINTN       Index2=0;
        EFI_INPUT_KEY  Key;
        for(Index = 0; Index < 30; Index ++)
        {
                Status = SystemTable->BootServices->WaitForEvent(1, &SystemTable->ConIn->WaitForKey, &Index2);
                if (EFI_ERROR(Status)) {
                Print(L"WaitKey: WaitForEvent Error!\n");
                }
                Status = SystemTable->ConIn->ReadKeyStroke (SystemTable->ConIn, &Key);
                if (EFI_ERROR(Status)) {
                Print(L"WaitKey: ReadKeyStroke Error!\n");
                }else{
                        Print(L"ScanCode [%d]",Key.ScanCode);
                        Print(L"UnicodeChar [%c]\n",Key.UnicodeChar);
                }
        }
        //indica terminacion exitosa
        return EFI_SUCCESS;
}
```

Anexo B. Otras referencias usadas

Referencias adicionales consultadas

https://github.com/zhenghuadai/uefi-programming/blob/master/book/Event/TestEvent.c

Las líneas 45 a 56 de este código fueron usadas tal cual en el código de ejemplo del anexo anterior, no son de mi autoría.

http://www.lab-z.com/stugt-2/

De esta página vienen las líneas que obtienen la fecha del sistema y la muestran en pantalla

https://thestarman.pcministry.com/asm/mbr/200MBR.htm

https://thestarman.pcministry.com/asm/mbr/W7MBR.htm

https://thestarman.pcministry.com/asm/mbr/W8VBR.htm

https://thestarman.pcministry.com/asm/mbr/CompV-7-8_VBR.htm

Estas páginas contienen un análisis detallado del funcionamiento del MBR, el primero de DOS 2.00, el segundo de Windows 7 y el tercero de Windows 8. El cuarto ofrece una comparativa y puede verse que no existen cambios mayores entre ellos.

https://qastack.mx/programming/22054578/how-to-run-a-program-without-an-operating-system

http://x86asm.net/articles/uefi-programming-first-steps/index.html

Aquí alguien se tomó el tiempo de hacer un tutorial sobre algunas particularidades para -hola mundo-

https://wiki.debian.org/SecureBoot

Una explicación de qué es UEFI y Secure Boot, dónde también le quitan un poco de culpa a Microsoft

https://www.scirp.org/journal/paperinformation.aspx?paperid=71259

Un análisis comparativo entre DOS y CP/M, solo de interés histórico

https://wiki.osdev.org/UEFI

Explica algunas cuestiones sobre BIOS y UEFI, tiene recursos adicionales sobre Sistemas Operativos

https://wiki.archlinux.org/title/Unified_Extensible_Firmware_Interface

Mas información sobre UEFI y los conflictos con dual boot junto a Windows.

Referencia a páginas de interés

https://www.guidgen.com/ Generador de identificadores GUID

https://thestarman.pcministry.com/ Detalles técnicos del funcionamiento del arranque del PC con BIOS, entre otras cosas

https://uefi.org/ Página de UEFI

https://www.youtube.com/c/UEFIForum Canal de UEFI

https://www.intel.com/content/www/us/en/developer/articles/tool/unified-extensible-firmware-interface.html

Información y recursos sobre UEFI