Estudiante: Joshua Solís Fuentes
Carné:2023064637

Introducción:

En las lecciones en clase hemos introducido conceptos clave como es el UEFI (Unified Extensible Firmware Interface) es una especificación que define una interfaz de software entre el sistema operativo y el firmware de la plataforma. Lo hemos visto mucho en teoría pero es necesario comprenderlo en la práctica.
La presente tarea aborda la necesidad de realmente comprender el comportamiento de estos programas de una manera pre sistema operativo, específicamente a través la implementación de un programa que recibe un prompt y lo traduce a código morse.
La herramienta está desarrollada en nasm64 y este programa implementa una aplicación UEFI. La aplicación lee caracteres del teclado mediante el protocolo ConIn de UEFI, los almacena en un búfer, y al presionar enter, convierte toda la línea ingresada a código Morse utilizando el protocolo ConOut para la salida. El programa se ensambla utilizando NASM para Win64 (dado que UEFI utiliza código de 64 bits) y se enlaza con lld-link para generar un ejecutable EFI:

```
nasm -f win64 programa.asm -o BOOTX64.obj
lld-link /subsystem:efi_application /entry:efi_main /machine:x64
/nodefaultlib /out:BOOTX64.efi BOOTX64.obj
```

El archivo BOOTX64.efi resultante debe copiarse a una unidad USB formateada en FAT32. Luego, se puede arrancar desde el firmware UEFI seleccionando la unidad USB como dispositivo de arranque.

Ambiente de desarrollo:

- **Sistema Operativo:** Linux (Ubuntu)
- **Arquitectura:** x86_64
- **Lenguaje de programación:** Nasm64 (2.16.01)
- **Editor/IDE:** Text editor y VScode
- **Bibliotecas principales:**
  El programa no utiliza bibliotecas externas, ya que interactúa directamente con los protocolos UEFI (ConIn y ConOut) proporcionados por el firmware.
- **Herramientas:**

- NASM: Ensamblador para compilar el código fuente.
- lld-link: Enlazador de LLVM para generar el ejecutable EFI.

Estructuras de datos:

- **búfer_tecla:** Reserva espacio para 2 palabras (4 bytes) para almacenar la estructura que contiene:
  - ScanCode (UINT16): Código de escaneo de la tecla
  - UnicodeChar (UINT16): Carácter Unicode de la tecla
- **búfer_línea:** Búfer de tamaño MAX_BUF (512 caracteres UTF-16, 1024 bytes) para almacenar la línea de texto ingresada por el usuario. Cada carácter se almacena en formato UTF-16 (2 bytes por carácter).
- **Cadenas UTF-16:**
  - cadena_inicio: Mensaje de bienvenida "Palabra morse!!:"
  - caracter_punto: Carácter . para representar puntos Morse
  - caracter_raya: Carácter - para representar rayas Morse
  - caracter_espacio: Carácter ' ' para separar símbolos Morse
  - cadena_prompt: Prompt "// " para indicar entrada
  - cadena_nueva_linea: Secuencia de retorno de carro y nueva línea

---

Funciones principales:

- **efi_main:** Función principal de entrada punto del programa UEFI. Coordina todo el proceso de:
  - Inicialización de protocolos UEFI (ConIn y ConOut)
  - Configuración de consolas de entrada y salida
  - Gestión del bucle principal de lectura de teclas
  - Procesamiento de líneas completas
  - Conversión a código Morse
  - Gestión de la salida del programa
- **Subrutinas principales dentro de efi_main:**
  - .prompt: Muestra el prompt "// " para indicar al usuario que puede ingresar texto.
  - .bucle_lectura: Lee teclas del servicio de entrada UEFI (ConIn) utilizando la función ReadKeyStroke. Maneja el ENTER y otros caracteres imprimibles

- .procesar_linea: Procesa una línea completa cuando se presiona ENTER:
  - Añade el terminador nulo a la cadena
  - Imprime la línea ingresada para retroalimentación
  - Llama al proceso de conversión a Morse
- .bucle_iteracion: Itera sobre cada carácter en el búfer de línea y llama a la rutina de conversión Morse correspondiente.

Macros especiales:

- **REPRODUCIR_PUNTO:** Macro que imprime un punto (.) utilizando el protocolo de salida UEFI
- **REPRODUCIR_RAYA:** Macro que imprime una raya (-) utilizando el protocolo de salida UEFI

Ejecución del programa:

Compilación:

```
nasm -f win64 $1 -o BOOTX64.obj
```

Enlace del ejecutable EFI:

```
lld-link /subsystem:efi_application /entry:efi_main /machine:x64 /nodefau
```

Preparación USB:

```
sudo mkdir -p $2/EFI/BOOT
sudo cp BOOTX64.efi $2/EFI/BOOT
```

Configuración de BIOS/UEFI:

1. Conectar la unidad USB al sistema objetivo
2. Acceder a la configuración del firmware (UEFI) durante el arranque
3. Desactivar Secure Boot si está habilitado
4. Seleccionar la unidad USB como dispositivo de arranque prioritario

Ejecución:

Al arrancar desde la unidad USB, el firmware UEFI cargará automáticamente BOOTX64.efi.
El programa mostrará el mensaje "Palabra morse!!".
El usuario puede escribir texto y presionar ENTER para convertirlo a Morse.

Bitácora del estudiante:

Estudiante	Actividad	Horas	Fecha
Josh Lis	Desarrollo de programa nasm 64 para linux para tener una idea de como ejecutar el programa	4	5/09/2025
Josh Lis	Desarrollo inicial de UEFI	3	6/09/2025
Josh Lis	Investigación de como bootear desde una llave	4	8/09/2025
Josh Lis	Desarrollo del programa en UEFI	6	8/09/2025
Josh Lis	Refactor y limpieza	2	9/09/2025
Josh Lis	Documentación	3	9/09/2025

Total: 22 horas

Autoevaluación:

Estado final del programa:

El programa UEFI de conversión a Morse funciona correctamente como aplicación independiente sin sistema operativo, pero presenta algunas limitaciones en su funcionalidad, como el de no poder reproducir sonido.

Funcionalidades implementadas:

- Interfaz UEFI nativa
- Lectura de entrada por teclado
- Soporte para caracteres alfanuméricos (A-Z, a-z, 0-9)
- Manejo de búfer de entrada (512 caracteres)
- Visualización de Morse con puntos y rayas
- Salida con protocolo ConOut

Problemas encontrados:

- Falta de sonido real
- Soporte limitado de caracteres (sin signos de puntuación)
- Dependencia de firmware UEFI
- Solo funciona en sistemas con UEFI x86-64
- Requiere desactivar Secure Boot
- Visualización básica
- No hay persistencia de datos
- Tamaño de búfer limitado

Log de github:

```
commit 98095bd6c4ef5e3db9c17638f99d9c5d3f5c9ac4
Author: josh_lis <josh.solisf@gmail.com>
Date:   Mon Sep 15 10:37:42 2025 -0600

    final changes

commit f8c32163274e6904d93dee1faaa2199a768f680e
Author: josh_lis <josh.solisf@gmail.com>
Date:   Tue Sep 9 16:48:47 2025 -0600

    fix

commit 29bd2eb8370362d1347825c0b2fc41a72d9610d3
Author: josh_lis <josh.solisf@gmail.com>
Date:   Tue Sep 9 16:31:29 2025 -0600

    puntuation

commit 206d5a273ca7cf9dd742335c9545526e49513e9e
Author: josh_lis <josh.solisf@gmail.com>
Date:   Tue Sep 9 16:16:20 2025 -0600

    Documentation

commit adb23e345fb9bdcdf6154ff65ae439e39b7889be
Author: josh_lis <josh.solisf@gmail.com>
Date:   Tue Sep 9 11:38:16 2025 -0600

    code refactor

commit b85322f6bad870a759341db4c8572d7c1ffe8e3d
```

```
Author: josh_lis <josh.solisf@gmail.com>
Date:   Tue Sep 9 11:11:24 2025 -0600

    Code refactor

commit e4f9d0b38b7027392f736745183d4d6674b4c7d7
Author: josh_lis <josh.solisf@gmail.com>
Date:   Tue Sep 9 10:27:00 2025 -0600

    comments for better understading

commit a7a530294ae86d6b716255c130468447ff243d54
Author: josh_lis <josh.solisf@gmail.com>
Date:   Tue Sep 9 09:20:09 2025 -0600

    added functionality to boot

commit d09996f7dcfd8e2fd073dc8bdb040434a3935e7f
Author: josh_lis <josh.solisf@gmail.com>
Date:   Sat Sep 6 22:13:51 2025 -0600

    early implementation of uefi

commit fbfdaaa87ffad0a98a22cad9282664d5be297ac7
Author: josh_lis <josh.solisf@gmail.com>
Date:   Sat Sep 6 11:32:41 2025 -0600

    code transalation to adapt calls from uefi instead of syscalls

commit f2d2997d26d023a0ea9dcebf636e7200e8a8881c
Author: josh_lis <josh.solisf@gmail.com>
Date:   Sat Sep 6 10:43:10 2025 -0600

    added functionality to write more letters

commit 782170943705c890286f094561d37b72447692b0
Author: josh_lis <josh.solisf@gmail.com>
Date:   Sat Sep 6 10:22:51 2025 -0600

    initial code with basic functionalinity and no sound
```

```
commit 92c594064f4e2ede0c6c3680e8c7aef6cf82b1e7
Author: josh_lis <josh.solisf@gmail.com>
Date:   Fri Sep 5 19:56:26 2025 -0600

    initial idea for the program structure

commit a19f5208abef0d2b0f32dc59465079dbe80a9838
Author: josh_lis <josh.solisf@gmail.com>
Date:   Fri Sep 5 18:08:23 2025 -0600

    Initial commit to test out nasm functionality

commit ac210636aeda20369d543081b43c9a9db763587d
Author: josh_lis <josh.solisf@gmail.com>
Date:   Fri Sep 5 16:56:26 2025 -0600

    first commit

commit 62ecd2e99eec09dc88e5d7013cb6a9e75fe6eaef
Author: josh_lis <josh.solisf@gmail.com>
Date:   Fri Sep 5 16:55:32 2025 -0600

    first commit
```

Rúbrica:

Sector de Arranque: 30 /30.

Morse: 40/50.

Documentación: 20/20

Lecciones aprendidas:

1. El desarrollo de aplicaciones UEFI requiere conocimiento de bajo nivel y documentación específica.
2. NASM x86-64 difiere de ensamblador para sistemas operativos.
3. Los servicios UEFI son útiles pero limitados frente a APIs de un SO completo.
4. La gestión de memoria en UEFI es más restrictiva.

5. El proceso de compilación y despliegue difiere del de aplicaciones convencionales.

Recomendaciones a futuros estudiantes:

- Estudiar UEFI y sus servicios antes de programar.
- Conocer convenciones de llamadas en x86-64.
- Empezar con ejemplos simples.
- Probar en QEMU y hardware real.

Bibliografía:

- 12. Protocols — Console Support — UEFI Specification 2.9A documentation." https://uefi.org/specs/UEFI/2.9_A/12_Protocols_Console_Support.html
- "uefi - Rust." https://docs.rs/uefi/latest/uefi/
- "@depletionmode - 2 of 1; half a nybble of another - Understanding modern UEFI-based platform boot." https://depletionmode.com/uefi-boot.html
- "*-unknown-uefi - The rustc book." https://doc.rust-lang.org/beta/rustc/platform-support/unknown-uefi.html
- ptrace man page
- "How to create an UEFI bootable USB stick from an ISO," Super User, Nov. 01, 2012. https://superuser.com/questions/497672/how-to-create-an-uefi-bootable-usb-stick-from-an-iso
- H. O. Store, "What is UEFI? A complete guide to BIOS vs UEFI | HP® Tech Takes - Hong Kong," hp, Feb. 14, 2025. https://www.hp.com/hk-en/shop/tech-takes/post/what-is-uefi
- Wikipedia contributors, "UTF-16," Wikipedia, Aug. 27, 2025. https://en.wikipedia.org/wiki/UTF-16