Informe - Actividad 1
Boot de Micro S.O. en QEMU

Benjamin Aballay y Elliot Acevedo
Ingeniería Civil en Ciencia de Datos

Mayo 2025

1. Introducción

Esta actividad tiene como propósito introducir en los fundamentos del arranque de un sistema operativo básico, utilizando herramientas de bajo nivel como el lenguaje ensamblador, NASM y el emulador QEMU.

El objetivo principal fue comprender cómo se inicia un sistema computacional desde su fase más básica, analizando el funcionamiento del código en ensamblador y los componentes mínimos que permiten una arquitectura funcional.

2. Herramientas utilizadas

Para el desarrollo de esta actividad se utilizaron las siguientes herramientas:

- **NASM (Netwide Assembler):** Ensamblador utilizado para compilar el código fuente en lenguaje ensamblador hacia un archivo binario tipo boot sector.
- **QEMU:** Emulador de sistemas que permite ejecutar el archivo generado como si fuera un disco de arranque en una arquitectura x86.
- **Editor de texto:** Se utilizó un editor como notepad para escribir y editar el código ensamblador.
- **Sistema operativo anfitrión:** Windows

3. Descripción del desarrollo

El código ensamblador desarrollado inicia mostrando un mensaje de bienvenida en pantalla, utilizando la interrupción `int 0x10` con la función `0x0E` para imprimir caracteres en modo texto. El mensaje contiene un menú con dos opciones: “Iniciar sistema” y “Apagar”.

Una vez mostrado el menú, el sistema espera una entrada del usuario mediante `int 0x16`, la cual captura una tecla presionada. Si el usuario presiona '1', el sistema imprime el mensaje "Sistema iniciando..." seguido de "Sistema iniciado exitosamente!". Si se presiona '2', se muestra "Apagando..." seguido de "Apagado exitoso.". En ambos casos, después de mostrar los mensajes correspondientes, el sistema se detiene con un salto infinito (`jmp $`).

El flujo del programa se organiza con etiquetas y subrutinas:

- La etiqueta `start:` inicia el proceso con un puntero (`SI`) al mensaje principal.
- El bucle `.loop:` recorre e imprime el mensaje carácter por carácter.
- La subrutina `imprimir:` permite reutilizar el código para mostrar mensajes.
- La subrutina `delay:` introduce una pausa artificial simulando una operación de espera.
- La etiqueta `halt:` representa el fin del programa.

El código también incluye manejo de salto de línea con caracteres ASCII 10 (Line Feed) y 13 (Carriage Return), para una presentación ordenada del texto.

Esta estructura mínima emula el comportamiento de un sistema básico capaz de interactuar con el usuario en el entorno de BIOS, sin requerir un sistema operativo completo.

4. Proceso de compilación y ejecución

Para convertir el código fuente en un archivo ejecutable por QEMU, se siguieron los siguientes pasos:

1. Se compiló el archivo usando NASM con el siguiente comando:

   ```
   nasm -f bin boot.asm -o boot.img
   ```

2. Luego, se ejecutó el sistema usando QEMU con el comando:

   ```
   qemu-system-i386 -fda boot.img
   ```

Al ejecutar el archivo `boot.img` en QEMU, el sistema muestra el menú inicial y espera la selección de una opción por parte del usuario. Según la tecla presionada, se despliega el mensaje correspondiente y se realiza una pausa antes de finalizar el programa con un salto infinito.

Este proceso permitió simular el comportamiento de un sistema operativo en sus primeras etapas de arranque, desde un entorno controlado y sin requerir hardware físico adicional.

5. Análisis crítico

Durante el desarrollo de esta actividad se logró entender en profundidad cómo funciona el ciclo de inicio (boot) de un sistema a bajo nivel. El uso de interrupciones del BIOS permitió realizar

operaciones esenciales como mostrar texto y capturar entradas del teclado directamente, sin un sistema operativo intermedio.

Se evidenció la importancia del Master Boot Record (MBR) y de cómo un código de apenas 512 bytes puede ejecutar instrucciones básicas que permiten interactuar con el usuario. El manejo de subrutinas y ciclos en ensamblador también fue un aprendizaje fundamental, dado que obliga a pensar en términos de registros, memoria y flujo de control detallado.

Entre las principales dificultades se encontraron:

- La interpretación precisa de los valores ASCII y su traducción visual en pantalla.
- El control de saltos y la segmentación de las rutinas.
- El uso correcto de la instrucción `int` para invocar servicios del BIOS.

A nivel de innovación, se podría agregar una tercera opción al menú, mejorar la estética del mensaje utilizando colores (cambiando atributos de video con `int 0x10`) o incluso crear una animación simple en texto para mostrar el ïnicio"del sistema.

6. Conclusión

Esta actividad fue una buena oportunidad para entender cómo funciona un computador desde que se enciende. Aunque al principio el lenguaje ensamblador puede parecer complicado, ir paso a paso permitió ver cómo se puede controlar directamente el hardware sin usar un sistema operativo.

Crear un menú simple, mostrar mensajes en pantalla y leer lo que el usuario escribe fue más que suficiente para darnos cuenta de todo lo que ocurre cuando una máquina arranca. Además, trabajar con herramientas como NASM y QEMU ayudó a ver cómo probar estos programas sin necesidad de dañar nada real.

En resumen, esta actividad ayudó a aterrizar conceptos que muchas veces solo se ven en teoría. También fue útil para aprender a trabajar con bajo nivel, algo que puede servir mucho si en el futuro nos interesa el desarrollo de sistemas operativos o si queremos entender mejor cómo funciona la arquitectura de los computadores.