UNIVERSIDAD TECNOLÓGICA METROPOLITANA
UTEM del Estado de Chile

INFB6074 - INFRASTRUCTURA PARA CIENCIA DE DATOS

INGENIERÍA CIVIL EN CIENCIA DE DATOS

"Informe Infrastuctura para ciencia de datos"

Joaquín Araya / Welinton Barrera

Mayo del 2025

1. Introducción

El desarrollo de sistemas operativos y la interacción con el hardware a bajo nivel son tareas fundamentales dentro del ámbito de la ingeniería informática y la arquitectura de computadores. Para adentrarse en estos temas, es necesario comprender cómo se comunica el software directamente con la máquina, lo cual implica trabajar con lenguajes de bajo nivel como el ensamblador y comprender los fundamentos del lenguaje máquina. En este contexto, se vuelve indispensable el uso de herramientas específicas que permitan escribir, ensamblar y ejecutar código ensamblador, así como probar dicho código en un entorno controlado.

El propósito de esta pequeña tarea consistió en familiarizarse con el uso de dos herramientas fundamentales: NASM (Netwide Assembler) y QEMU. NASM es un ensamblador de código abierto que permite escribir código para arquitecturas x86 y x86-64, y que se caracteriza por su eficiencia, simplicidad y amplia adopción en entornos académicos y profesionales. Por su parte, QEMU es un emulador de hardware que permite virtualizar diferentes arquitecturas y ejecutar sistemas operativos personalizados sin necesidad de hardware físico adicional. En conjunto, estas herramientas permiten llevar a cabo proyectos de bajo nivel como la creación de sistemas operativos desde cero, pruebas de carga de bootloaders y ejecución de código binario en máquinas virtuales.

Antes de comenzar con la instalación de las herramientas, se realizó una fase de documentación inicial que permitió comprender la función y relevancia de cada una dentro del flujo de trabajo necesario para desarrollar un sistema operativo básico. Esta documentación incluyó conceptos como lenguaje máquina, ciclo de vida de una instrucción en la arquitectura x86, cómo interactuar con el BIOS o UEFI al momento de bootear, y cómo compilar y ensamblar archivos en NASM para generar imágenes ejecutables que puedan ser cargadas en una máquina virtual a través de QEMU.

Este informe tiene como objetivo describir el proceso seguido durante esta fase inicial, detallando las etapas de documentación, descarga, instalación y configuración de las herramientas mencionadas. Además, se mencionan consideraciones importantes para que estas herramientas funcionen correctamente en el entorno del sistema operativo del usuario. El trabajo desarrollado en esta etapa sienta las bases para futuras tareas que involucrarán el diseño y prueba de componentes fundamentales de un sistema operativo, tales como el bootloader, el kernel y los controladores básicos.

2. Documentación Previa

Antes de proceder con la instalación de las herramientas necesarias, se realizó un proceso de documentación para comprender los conceptos fundamentales involucrados. En primer lugar, se investigó sobre el lenguaje máquina y su interacción mediante ensambladores como NASM (Netwide Assembler), el cual es un ensamblador de código abierto ampliamente utilizado para arquitecturas x86 y x86-64.

Posteriormente, se analizó la utilidad de QEMU, una herramienta de virtualización que permite emular una máquina completa. Esta herramienta resulta esencial para probar un sistema operativo desarrollado por el usuario, ya que permite iniciar (bootear) dicho sistema desde una imagen binaria en un entorno controlado.

3. Instalación de Herramientas

Una vez adquiridos los conocimientos teóricos básicos, se procedió con la instalación de las herramientas necesarias en nuestros equipos personales.

3.1 Instalación de NASM

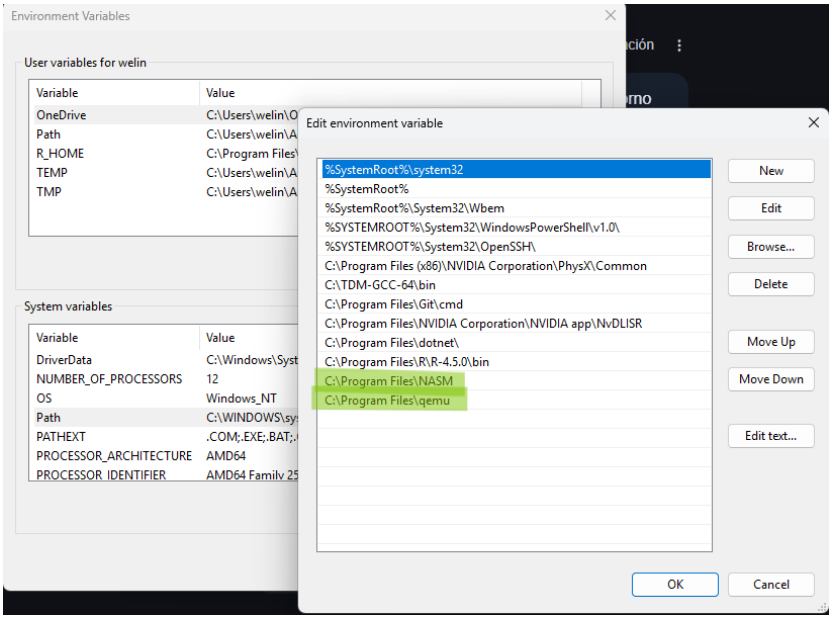
Para instalar NASM, se accedió a su sitio web oficial: https://www.nasm.us/. Desde allí se descargó el instalador correspondiente a la versión x64 compatible con nuestro sistema operativo. La instalación se completó siguiendo los pasos del asistente de instalación.

3.2 Instalación de QEMU

El proceso de instalación de QEMU fue similar. Se visitó su sitio oficial en https://www.qemu.org/, desde donde se descargó la versión x64. Tras completar la descarga, se ejecutó el instalador y se siguieron los pasos indicados por el asistente para finalizar correctamente la instalación.

3.3 Consideraciones Importantes

Una vez instaladas ambas herramientas, fue necesario agregar las rutas de instalación de NASM y QEMU a la variable de entorno `PATH` del sistema. Esta acción es fundamental, ya que permite ejecutar ambos programas desde cualquier terminal o consola sin necesidad de especificar su ruta completa de instalación.



4. Codigo utilizado

Ya al momento de tener nuestras dependencias instaladas, pasamos a nuestro editor de texto, en nuestro caso Visual Studio Code para empezar nuestro codigo.

4.1 System.bat

A continuación se muestra el código utilizado para compilar un archivo de arranque en NASM y ejecutarlo con QEMU:

```
nasm boot.asm -f bin -o os.flp
qemu-system-x86_64 -drive format=raw,file=os.flp
pause
```

Listing 1: Compilación y ejecución de un SO con NASM y QEMU

Explicación de cada línea:

- `nasm boot.asm -f bin -o os.flp`: Utiliza el ensamblador NASM para compilar el archivo fuente `boot.asm` en formato binario plano (`-f bin`) y genera un archivo de salida llamado `os.flp`, que será una imagen de disquete arrancable.
- `qemu-system-x86_64 -drive format=raw,file=os.flp`: Lanza QEMU, un emulador de hardware, utilizando una arquitectura x86_64. Carga la imagen de disco `os.flp` como unidad en formato `raw`, permitiendo así simular el arranque del sistema operativo que contiene.
- `pause`: Detiene la ejecución del script en Windows para mantener la ventana abierta, útil cuando se ejecuta desde un archivo por lotes (`.bat`) para que el usuario pueda ver posibles mensajes antes de que la ventana se cierre.

4.2 boot.asm

A continuación se presenta un código de ejemplo escrito en NASM, el cual define un sector de arranque (boot sector) válido para un sistema operativo mínimo:

```
[org 0x7c00]

start:
    mov si, mensaje

.loop:
    lodsb
    cmp al, 0
    je  halt
    cmp al, 10
    jne .print
        mov ah, 0x0E
        mov al, 13
        int 0x10
        mov al, 10
        int 0x10
        jmp .loop

.print:
    mov ah, 0x0E
    mov bh, 0x00
    mov bl, 0x07
    int 0x10
    jmp .loop

halt:
    jmp halt

mensaje db "␣_␣␣␣_␣_____␣_____␣__␣␣__",10
        db "|␣|␣|␣|_␣␣␣_|␣____|␣␣\/␣␣|",10
        db "|␣|␣|␣|␣|␣|␣|␣␣_|␣|␣|\/|␣|",10
        db "|␣|_|␣|␣|␣|␣|␣|___|␣|␣␣|␣|",10
        db "|_␣_␣_|␣|_|␣|_____|_|␣␣|_|",10
        db "",10
        db "␣Bienvenido␣a␣tu␣OS␣UTEM!",10
        db 0

times 510-($-$$) db 0
dw 0xAA55
```

Listing 2: Bootloader básico en NASM

Aspectos clave:

- `[org 0x7C00]`: Le indica al ensamblador que el código se cargará en la dirección de memoria física 0x7C00, que es donde el BIOS copia el primer sector del disco (512 bytes) durante el arranque.
- El programa comienza en la etiqueta `start`, donde se inicializa el registro `SI` apuntando a la cadena `mensaje`, la cual será impresa en pantalla.
- El ciclo principal lee carácter por carácter usando `LODSB` y muestra cada símbolo usando la interrupción de BIOS `int 0x10` con el modo teletipo (`AH = 0x0E`).
- La última instrucción `times 510-($-$$) db 0` completa el sector hasta los 510 bytes con ceros, y `dw 0xAA55` escribe la firma obligatoria del boot sector, que el BIOS requiere para reconocerlo como válido.

5. Conclusión

Respecto a lo trabajado durante esta tarea se logro entender de forma mas profunda el como trabaja internamente un sistema operativo, el como se debe especificar a que sector de memoria se debe llamar internamente para lograr distintas cosas, como inicializaciones u otras tareas, esto nos hace tambien entender la complejidad de un sistema operativo y como desde su concepción es algo tan complejo pero a su vez interasante.