Instituto Tecnológico de Costa Rica
Escuela de Ingeniería en Computación
Centro Académico de Alajuela

Tarea 1: MRPV

Estudiante: Jesús Cordero Díaz

Curso:
IC-6600: Principios de Sistemas Operativos

Grupo: 20

Profesor: Kevin Moraga Garcia

Fecha de entrega: 18 de marzo del 2025

Semestre: I

Contents

1 Introducción

El arranque de un sistema operativo es el primer y más importante paso que transforma el estado inactivo del hardware en una plataforma operativa capaz de interactuar con el usuario. Este proceso, a pesar de parecer transparente para muchos, implica una serie de etapas críticas en las que se configura y activa el entorno que permitirá el funcionamiento del software.

En esta tarea se aborda el reto de implementar, en lenguaje ensamblador para la arquitectura x86, una aplicación denominada **MRPV**. El problema central consiste en lograr que la computadora realice correctamente el booteo desde una unidad USB y, a partir de ello, cargue un juego usando el alfabeto fonético internacional.

La aplicación genera secuencias de caracteres de manera aleatoria y desafía al usuario a transcribirlas utilizando el alfabeto fonético. De esta forma, se busca no solo comprender el funcionamiento del proceso de arranque, sino también ofrecer una herramienta que combine teoría y práctica en un entorno de aprendizaje interactivo.

Este documento detalla la solución propuesta, desde la configuración del entorno de desarrollo y la descripción de las estructuras implementadas, hasta las instrucciones precisas para la ejecución del programa y el análisis crítico del desempeño alcanzado.

2 Ambiente de Desarrollo

Para la implementación de la tarea se utilizaron las siguientes herramientas:

- **Visual Studio Code:** Editor de código ligero y multiplataforma que ofrece un amplio ecosistema de extensiones para facilitar la programación y la depuración.
- **NASM (versión 2.16.01):** Compilador de lenguaje ensamblador orientado a la arquitectura x86, ampliamente utilizado por su sencillez y eficiencia.
- **QEMU (versión 8.2.2):** Emulador que permite simular entornos de hardware, facilitando la prueba y depuración del código en arquitecturas x86.
- **GitHub:** Plataforma de control de versiones basada en Git que facilita la colaboración y el seguimiento de cambios en el desarrollo del proyecto.
- **Ubuntu 24.04.2 LTS:** Sistema operativo Linux basado en Debian, conocido por su estabilidad y soporte a largo plazo, ideal para entornos de desarrollo.

Adicionalmente, se implementó el alfabeto fonético internacional, el cual se define a continuación:

Letra	**Significado**	**Letra**	**Significado**
A	Alfa	N	November
B	Bravo	O	Oscar
C	Charlie	P	Papa
D	Delta	Q	Quebec
E	Echo	R	Romeo
F	Foxtrot	S	Sierra
G	Golf	T	Tango
H	Hotel	U	Uniform
I	India	V	Victor
J	Juliett	W	Whiskey
K	Kilo	X	X-Ray
L	Lima	Y	Yankee
M	Mike	Z	Zulu

Table 1: Alfabeto Fonético Internacional (Letras)

Número	**Significado**	**Número**	**Significado**
1	One	6	Six
2	Two	7	Seven
3	Three	8	Eight
4	Four	9	Nine
5	Five		

Table 2: Alfabeto Fonético Internacional (Números)

3 Estructuras de Datos y Funciones Fundamentales

El proyecto se implementa en lenguaje ensamblador para la arquitectura x86, y se organiza en torno a funciones clave que gestionan el proceso de booteo, la generación de cadenas aleatorias, la interacción con el usuario, la validación de respuestas y la actualización del puntaje. A continuación se describen las funciones y estructuras más fundamentales.

Rutina de Boot

El proyecto incluye un bootloader que se encarga de cargar el Master Boot Record (MBR) desde el disco duro. Los pasos principales de la rutina de boot son los siguientes:

- **Origen y Configuración:** Se indica al ensamblador que el código se cargará en la dirección `0x7C00` y que se usará el modo de 16 bits. Además, se configuran los segmentos de datos y extra (DS y ES) a cero.
- **Lectura del Sector:** Mediante la interrupción de BIOS `0x13`, se lee el sector 2 del disco duro (primer disco, indicado con `0x80`) y se carga en la dirección `0x7E00`. Este sector contiene el MBR, que es el punto de partida para la ejecución del sistema.
- **Manejo de Errores:** Si ocurre algún error durante la lectura (verificado a través de la bandera de acarreo), se muestra un mensaje de error en pantalla utilizando la función `imprimir` y se detiene la ejecución del sistema.
- **Transferencia de Ejecución:** Una vez leída correctamente la información, se transfiere la ejecución a la dirección `0x7E00`, donde reside el MBR.

Funciones Fundamentales del Juego

- **main:** Función de entrada que inicializa la pantalla y establece el ciclo principal. Dentro de este ciclo se generan cadenas aleatorias, se muestra el menú del juego, se solicita la entrada del usuario y se actualiza el puntaje.
- **crear_cadena_aleatoria:** Genera una cadena de 4 caracteres (letras o números) utilizando el contador de tiempo como semilla. Convierte el residuo de la división para determinar si se debe generar una letra o un número, almacenando cada carácter en un buffer y terminando con un carácter nulo.
- **menu_juego:** Presenta el mensaje de bienvenida y la cadena aleatoria generada, invitando al usuario a participar en el juego.
- **perdir_input_usuario:** Solicita y procesa la entrada del usuario para cada uno de los 4 caracteres de la cadena. Para cada carácter, muestra el valor esperado y llama a las funciones que leen y validan la entrada.
- **validar_input_usuario:** Compara la entrada del usuario con la palabra fonética correspondiente, utilizando una tabla fonética que asocia cada letra y número con

su representación (por ejemplo, "alfa", "bravo", etc.). Si la respuesta es correcta, se incrementa el puntaje.

- **imprimir:** Función genérica para imprimir cadenas en pantalla utilizando la interrupción de BIOS.
- **mostrar_puntaje:** Muestra el puntaje acumulado en pantalla, combinando mensajes informativos y la conversión del valor numérico a caracteres ASCII.

Estructuras de Datos Esenciales

- **Buffers y Variables:**
  - `cadena_aleatoria`: Espacio reservado para almacenar la cadena generada (4 caracteres + terminador nulo).
  - `input_entrada`: Buffer para almacenar la entrada del usuario.
  - `contador_puntos`: Variable de 16 bits que acumula el puntaje del usuario.
- **Tabla Fonética (`tabla_palabras`):** Estructura que asocia cada letra (de 'a' a 'z') y cada número (0-9) con su correspondiente palabra fonética. Cada entrada es un puntero a una cadena terminada en nulo.
- **Mensajes:** Cadenas utilizadas para la interacción con el usuario, tales como:
  - `mensaje_bienvenida`: Saludo e introducción al juego.
  - `mensaje_correcto` y `mensaje_incorrecto`: Indicadores de validación de la entrada.
  - `mensaje_puntaje`: Texto para mostrar el puntaje acumulado.

4 Instrucciones para Ejecutar el Programa

Para ejecutar el programa desarrollado, se deben seguir los siguientes pasos y utilizar los comandos indicados. Cada comando cumple una función específica en el proceso de compilación, creación de la imagen de disco y prueba en un emulador, así como en la escritura de la imagen en una memoria USB.

Pasos para la Ejecución

1. **Compilar el Bootloader:** Se utiliza NASM para compilar el bootloader, generando un archivo binario:

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

2. **Compilar el Programa Principal:** De igual forma, se compila el código principal que contiene la lógica del juego:

   ```
   nasm -f bin -o mrpv.bin mrpv.asm
   ```

3. **Obtener una Imagen Limpia:** Se crea una imagen de disco limpia de 2048 sectores (cada sector de 512 bytes) usando la herramienta `dd`:

   ```
   dd if=/dev/zero of=disk.img bs=512 count=2048
   ```

4. **Crear una Imagen de Disco:** Se incorpora el bootloader y el programa principal a la imagen de disco. Primero se copia el bootloader en el primer sector:

   ```
   dd if=boot.bin of=disk.img bs=512 count=1
   ```

   Luego, se añade el programa principal en el sector siguiente (a partir del sector 1):

   ```
   dd if=mrpv.bin of=disk.img bs=512 seek=1
   ```

5. **Ejecutar y Probar la Imagen:** Se utiliza QEMU para emular la ejecución de la imagen de disco:

   ```
   qemu-system-x86_64 -drive format=raw,file=disk.img
   ```

6. **Escribir la Imagen en la Memoria USB:** Para desplegar el programa en hardware real, se escribe la imagen del bootloader en una memoria USB utilizando `dd`. Es importante verificar la identificación correcta de la unidad (en este ejemplo se usa `/dev/sdb`):

   ```
   sudo dd if=boot.bin of=/dev/sdb bs=512 status=progress
   ```

Cada uno de estos comandos debe ejecutarse en el entorno de desarrollo Linux (Ubuntu 24.04.2 LTS) para garantizar que se realicen las operaciones de compilación y escritura correctamente.

5 Actividades Realizadas por el Estudiante

A continuación se presenta un resumen de las actividades realizadas durante el desarrollo del proyecto, detallando brevemente la tarea, la fecha en que se realizó y el tiempo invertido en cada una. El total de horas invertidas en el proyecto asciende a 22 horas.

Fecha	**Tarea Realizada**	**Horas Invertidas**
04/03/2025	Planificación y análisis de requerimientos.	3
06/03/2025	Configuración del ambiente de desarrollo.	2
08/03/2025	Diseño de la estructura del bootloader.	3
10/03/2025	Programación de generación de cadenas aleatorias y funciones de impresión.	4
12/03/2025	Implementación de la validación de entrada del usuario.	3
14/03/2025	Integración y pruebas en QEMU.	4
16/03/2025	Redacción de la documentación y revisión final del código.	3

Table 3: Bitácora de Actividades

6 Autoevaluación

El proyecto se concluyó satisfactoriamente, implementándose todas las funcionalidades requeridas. Se logró desarrollar tanto el bootloader como el programa principal que genera cadenas aleatorias, valida la entrada del usuario y muestra el puntaje acumulado. No obstante, se detectó un inconveniente relacionado con la ejecución del código en modo MBR en el hardware real, ya que, tras una breve investigación, se concluyó que la laptop utilizada no posee la capacidad de ejecutar código en dicho modo. Este problema se solucionó mediante el uso del emulador QEMU, que permitió validar el funcionamiento del sistema en un entorno controlado.

A continuación se presenta el reporte de commits realizado durante el desarrollo del proyecto:

Table 4: Reporte de Commits

Fecha	**Mensaje del Commit**
04/03/2025	"Inicialización del proyecto: planificación y análisis de requerimientos."
06/03/2025	"Configuración del ambiente de desarrollo y establecimiento de herramientas."
08/03/2025	"Diseño y desarrollo de la estructura del bootloader."
10/03/2025	"Implementación de generación de cadenas aleatorias y funciones de impresión."
12/03/2025	"Implementación de validación de entrada del usuario y corrección de errores menores."
14/03/2025	"Integración y pruebas en QEMU, resolución de problemas en el emulador."
16/03/2025	"Redacción de la documentación y revisión final del código."

Con base en la rúbrica de evaluación, se asigna la siguiente calificación:

- **Sector de Arranque (30%):** 30/30 – El bootloader se implementó correctamente y se integró mediante QEMU, superando la limitación de hardware.
- **Funcionalidad del Juego (50%):** 50/50 – Se implementaron todas las funcionalidades principales.
- **Documentación (20%):** 20/20 – La documentación es completa, detallada y cumple con los requerimientos establecidos.

Calificación Final: 100/100

A pesar de la limitación para ejecutar el código en modo MBR en el hardware real, el uso de emulación permitió validar el correcto funcionamiento del sistema, lo que se refleja en la calificación final obtenida.

7 Lecciones Aprendidas del Proyecto

Durante el desarrollo de este proyecto se han extraído diversas lecciones que pueden ser de gran ayuda para futuros estudiantes del curso. Algunas de las más relevantes son:

- **Planificación y organización:** Una planificación detallada desde el inicio permite definir claramente los requerimientos y establecer un cronograma realista, lo que facilita la gestión del tiempo y la identificación temprana de posibles problemas.
- **Configuración del entorno de desarrollo:** La correcta configuración de herramientas como Visual Studio Code, NASM, QEMU y GitHub es fundamental para garantizar un flujo de trabajo eficiente y minimizar errores durante la compilación y las pruebas.
- **Importancia de la documentación:** Llevar un registro detallado (bitácora o timesheet) y documentar cada etapa del proyecto resulta clave para facilitar la revisión, integración y corrección de errores, además de servir como recurso de aprendizaje para futuros proyectos.
- **Uso de emuladores para pruebas:** Ante limitaciones de hardware, como la imposibilidad de ejecutar código en modo MBR, el uso de emuladores como QEMU permite validar el funcionamiento del sistema en un entorno controlado, asegurando que el software cumpla con los requerimientos establecidos.
- **Depuración y validación continua:** Realizar pruebas constantes y validar cada componente del sistema de forma individual ayuda a detectar y corregir errores de manera oportuna, mejorando la estabilidad y el rendimiento del proyecto.
- **Adaptabilidad ante problemas técnicos:** La experiencia adquirida demuestra que es crucial ser flexible y buscar soluciones alternativas (por ejemplo, el uso de emulación) cuando se enfrentan obstáculos técnicos, manteniendo siempre el progreso del proyecto.

8 Bibliografía

References

[1] Wikipedia, "Alfabeto radiofónico," [Online]. Available: https://es.wikipedia.org/wiki/Alfabeto_radiof%C3%B3nico.

[2] NASM Documentation, "NASM Documentation," [Online]. Available: https://www.nasm.us/doc/.

[3] QEMU Documentation, "QEMU Documentation," [Online]. Available: https://www.qemu.org/documentation/.