Tarea 2: Finding Treasure booteable

Gabriel Alfaro-Herrera, Roger-Andres Valderrama-Ordoñez, Emannuel Esquivel-Lopez
email: gabofaro@estudiantec.cr,roger.andres18@estudiantec.cr,ema11412@estudiantec.cr
CE 4303 — Principios de Sistemas Operativos
Área Académica de Ingeniería en Computadores
Instituto Tecnológico de Costa Rica

***Abstract*—This paper shows a way to create a bootloader program with with x86 assembly lenguaje. Once the bootloader is loaded and running, it will place in memory a "Finding Treasure" game, then the user will be able to play the game. The game was tested with the help of QEMU emulator. With the help of a bootable USB devise, previously loaded with the game and bootloader program, the game will be launch as soon as the computer is turn on.**

***Keywords*—Bootloader, Ensamblador x86, QEMU, USB**

I. Introducción

Cada vez que el computador es encendido o reiniciado el sistema básico de entradas y salidas (BIOS) realiza una revisión básica del sistema, luego transfiere el control al bootloader [1].

El bootloader es un segmento de código pre-programado que que ubica en el disco, dicho código es ejecutado de primero al encender el computador. Este pequeño programa es el encargado de posicionar al Sistema Operativo en memoria [1].

Para poder realizar el código de un bootloader es requerido escribirlo en lenguaje ensamblador, para el desarrollo de esta tarea se empleará lenguaje ensamblador x86. La razón se debe a la amplia información y fuentes de referencias que se pueden encontrar en linea del lenguaje anterior mencionado.

La idea fundamental del juego a realizar es crear una serie de laberintos o obstáculos por los cuales un marciano se puede trasladar, las cuales se desvanecerán si reciben un impacto de poder (bola de fuego). El marciano tira bolas de fuego, el cual es dirigido por el usuario y su función es derribar al jefe de la comunidad (quien tiene el tesoro).

El Jefe de la comunidad estará protegido por paredes de hielo, las cuales podrán ser derribadas por las bolas de fuego. El jefe tiene el poder de colocar bombas de manera aleatoria por todo el campo de juego. En caso de que el marciano choque contra una bomba perderá el juego.

El juego contará con dos niveles de dificultad y el usuario ganará cuando se pase el segundo nivel. Se debe proporcionar alguna animación cuando se haya ganado el juego. Se deberá proveer en todo momento la cantidad de ataques hechos a las paredes y el nivel de juego que se está llevando a cabo. Es importante que se muestren los comandos en pantalla como parte de la guía para el usuario. Cuando el jugador gana el segundo nivel, el juego permitirá la opción de reiniciar o salir, además debe de desplegar un mensaje en pantalla con alguna animación de felicitaciones por haber ganado.

Se podrá reiniciar el juego en cualquier instante, así como finalizarlo. Cabe resaltar que cuando se inicia el juego el usuario debe de aceptar una confirmación para empezar a jugar.

Los movimientos del marciano se realizaran por medio de las flechas arriba, abajo, derecha e izquierda. La barra espaciadora es para disparar, L para pausar y R para reiniciar.

En las siguientes secciones se expone la manera en la que el problema anterior planteado es solucionado.

II. Ambiente de desarrollo

El proyecto fue desarrollado en linux, por lo que mucha de su implementación y elaboración fue con el uso de comandos en la terminal del mismo, esta fue desarrollada en el lenguaje de programación Assembly x86.

Se llevo acabo en una computadora de arquitectura 64 bits, sistema operativo Linux Kali linux 2020.2, 16 Gb memoria, procesador Ryzen 7 2700x, además de la edición de código en VSCode.

Como emulador principal se utilizo QEMU, ya que este nos provee una gran cantidad de funciones para poder ejecutar el binario generado, utilizando Netwide Assembler NASM para compilar el código.

Para montar la imagen se utilizo HDD raw copy, este nos permite el *quemado* de la ISO generada por el programa principal *kernel* y a su vez es copiada a una USB.

III. Detalles del diseño

III-A. Diagrama de arquitectura

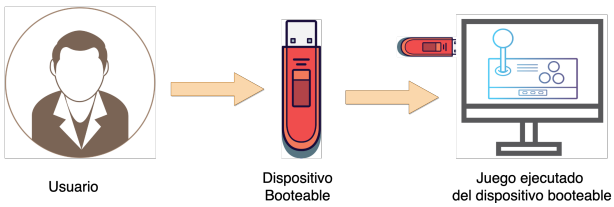

Figura 1. Diagrama de la arquitectura del sistema implementado

En este diagrama podemos ver la interacción general de la aplicación con las entidades, ya que podemos ver el usuario el cual va a utilizar el juego, una USB la cual es la que contara el juego booteable y por ultimo se necesita una computadora la cual sea capaz de elegir el dispositivo de arranque y que este sea la USB.

III-B. Diagrama de paquetes

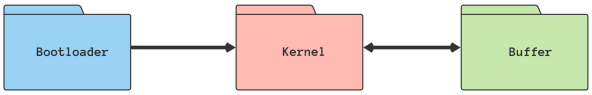

Figura 2. Diagrama de paquetes del sistema implementado

Como se muestra en el diagrama, el sistema se compone de tres paquetes. El primero consiste en el booloader, el cual es el encargado de realizar los ajustes previos al booteo en general, ya que se encarga de direccionar el juego a las direcciones de memoria correspondientes, asi como las interrupciones necesarias, además de esto este mismo es conectado al kernel, el cual es el main del proyecto ya que aquí se gestiona de manera general el juego y este esta en contacto con un buffer el cual se encarga de realizar correcciones gráficas y de memoria para poder presentar de manera adecuada en la pantalla.

III-C. Booteo

```
startGame:
   ;Initialize Registers
   cli                    ; disable interrupts
   xor ax, ax
   mov ds, ax
   mov ss, ax
   mov es, ax
   mov fs, ax
   mov gs, ax
   mov sp, 0x6ef0
   sti                    ; enable interrupts

   mov ah, 0
   int 0x13               ; 0x13 ah=0 dl = drive number
   jc error

   ; Load the kernel into RAM
   mov bx, 0x8000         ; Direction where the kernel starts
   mov al, SECTOR_AMOUNT  ; Number of sectors to read from kernel
   mov ch, 0              ; Disk configuration
   mov dh, 0
   mov cl, 2
   mov ah, 2
   int 0x13
   jc error
   jmp 0x8000
```

Figura 3. Bootloader implementado

En esta sección se puede ve claramente como se inicia el booteo, porque definimos el sector y bytes a utilizar así como el nombre del mismo.

Dato importante es la dirección donde se debe cargar la cual es $0x7c00$ además en los últimos 2 bits 511 o 512 se debe escribir $0x55$ o $0xAA$ esto para que el MBR lo reconozca

Posteriormente se utiliza la interrupción $0x13$ en el BIOS para que se cargue el programa en RAM.

IV. Makefile

```
all: build run

clean:
    cd bin && rm -rf ./*.bin

build:
    nasm -fbin ./src/kernel.asm -o ./bin/kernel.bin
    nasm -fbin ./src/bootloader.asm -o ./bin/bootloader.bin
    cat ./bin/bootloader.bin ./bin/kernel.bin > ./bin/Finding_Treasure_booteable.bin

run:
    qemu-system-i386 ./bin/Finding_Treasure_booteable.bin

```

Figura 4. Makefile implementado

Este es un makefile muy sencillo ya que en este se compila de manera separada el kernel y bootloader, asi como se interconectan para formar el booteable final, ademas se corre el programa en qemu para verificar errores o cosas varias.

V. Atributos

Se puede ver un aumento y refuerzo en el atributo de análisis ingenieril de problemas, ya que el análisis fue bastante grande a comparación con anteriores proyectos, así como se entiende de mejor manera el diseño de dispositivos booteables.

El análisis matemático no fue tan exhausto, ya que son operaciones muy simples lo cuales no reforzaron este atributo.

El aprendizaje continuo fue una parte muy importante ya que continuamente se debía realizar alguna nueva técnica o nuevos elementos añadidos los cuales es necesario tener el conocimiento.

El trabajo en equipo fue fortalecido ya que se trabajo de manera continua todos en el proyecto y fue bastante agradable y ameno el trabajo.

VI. Instrucciones

1. Bootear desde el dispositivo USB
2. Cuando ya haya iniciado, presionar ENTER para iniciar el juego.
3. Para mover el marciano se deben usar las teclas de las flechas con su respectiva dirección.
4. La tecla R, sirve para reiniciar y L pausa.

VII. Tabla de actividades

VII-A. Gabriel Alfaro

Actividad	Horas
Investigacion sobre x86	1
Busqueda de emuladores Assembly	2
Imprementacion del main kernel	9
Creacion de imagenes	1
Creacion del Booteo	5
Documentacion	2
Total	20

VII-B. Emanuel Esquivel

Actividad	Horas
Investigación sobre x86	2
Búsqueda de emuladores Assembly	2
Implementación del main kernel	13
Implementación del Buffer	3
Documentación	3
Total	23

VII-C. Roger Valderrama

Actividad	Horas
Investigación sobre x86	4
Implementación del mapa	10
Conversión de imágenes a bin	4
Documentación	5
Total	23

VIII. Coevaluación

Todo el grupo se siente confiable entre si y sentimos que no hubo unos que trabajaron mas que otros además podíamos ver grandes resultados a la hora del trabajo, a pesar de tener diversas situaciones siempre supimos trabajar en equipo por lo que la evaluación entre nosotros se basara en una escala de 1 - 10 de aportes a la tarea.

VIII-A. Emanuel Esquivel

Integrante	Calificación
Gabriel Alfaro	9.5
Roger Valderrama	10

VIII-B. Roger Valderrama

Integrante	Calificación
Emanuel Esquivel	9.5
Gabriel Alfaro	10

VIII-C. Gabreal Alfaro

Integrante	Calificación
Emanuel Esquivel	9.8
Roger Valderrama	9.8

IX. Conclusiones

Para reconocer una memoria booteable, el BIOS busca un MBR en cierta dirección de memoria predefinida, en la cual se encuentra el kernel

Se profundizo el lenguaje x86 para la implementación de la tarea, así como se indago y aprendió la creación de interfaz gráfica.

X. Sugerencias y recomendaciones

Antes de implementar el bootloader y el juego se recomienda tener un conocimiento intermedio del lenguaje ensamblador para la arquitectura x86 y utilizar NASM para tener un código más compatible con diferentes sistemas operativos.

Se sugiere utilizar un emulador para poder probar el código de una forma mas rápida y aislar problemas relacionados con hardware.

Se recomienda una memoria de tipo flash para bootear el juego en hardware.

Referencias

[1] Rouse, M., (2006). boot loader (boot manager). Recuperado de: https://searchdatacenter.techtarget.com/definition/boot-loader-boot-manager

[2] NASM. (2015). Nasm. Recuperado de: https://www.nasm.us

[3] Leeky. (2017). Realmode assembly-writing bootable stuff-part 1. Recuperadode:https://0x00sec.org/t/realmode-assembly-writing-bootable-stuff-part-1/2901