Juego sobre hardware

Daniel Méndez, Francisco Olivares

***Resumen*—Mostraremos el desarrollo de un juego que se ejecute directamente en la máquina, es decir, en el procesador, sin la intervención de un sistema operativo de por medio. El juego será el famoso Snake [1], que requiere de la implementación de drivers para el uso y administración de los hardwares, como también de las herramientas necesarias para poder iniciar nuestro juego en la máquina, como un bootloader.**

Desarrollaremos entonces algo cercano a un microkernel, el cual se encargará de ejecutar nuestro juego y manejar los inputs y outputs para la interacción con el usuario.

I. Introducción

En este informe presentaremos los pasos a seguir para la construcción de un juego ejecutado en el hardware. Desarrollaremos los conceptos relevantes que se deben entender a la hora de desarrollar algo así, y los pasos que seguimos para intentar lograr el objetivo. Se muestran los resultados obtenidos y los problemas que se nos presentaron a medida que avanzamos en el desarrollo del software. Se adjunta también el código, en el que se pueden ver las diferentes funcionalidades de forma separada.

II. Marco teórico

Para iniciar el desarrollo se requieren de herramientas fundamentales tanto para evitar la mal utilización del tiempo como también la seguridad ante posibles daños a nivel de hardware, perdida de información o código durante el desarrollo, sin dejar de lado la documentación y las herramientas necesarias para probar y correr nuestros códigos. Como principal herramienta de vital importancia tanto para las pruebas como para la presentación de la aplicación es Bochs [2] que permite el booteo de kernels o aplicaciones booteables de arquitectura x86 como de x64, con esta aplicación evitamos las pruebas de manera directa en la maquina física utilizando una maquina virtual lo que nos ahorra tiempo y posibles daños en la maquina.

Para esta aplicación se nos ha pedido que se trabaje en la arquitectura x32 como también el uso de lenguaje en C y de Assembler [3] con GCC [4] de 16bit para el desarrollo de las funciones de interrupción a nivel de hardware como también para el desarrollo de los archivos booteables que se bootearan en bochs. Estas aplicaciones booteables se compilan por medio de pequeños softwares como Dd [5], Nasm [6] y los archivos generados por gcc los que en conjunto nos entregan un producto final .img o extensión booteable.

En cuanto a la documentación y a los backups de el codigo se utiliza Doxygen [7] y el uso de la plataforma Gitlab [8] respectivamente, para que de esta forma se pueda tener una guía de rápida mantención con también un código versionado en cada una de sus etapas para poder tomar las medidas a tiempo en el desarrollo y mantención.

e-mail: daniel.mendezf@mail.udp.cl
e-mail: francisco.olivaresr@mail.udp.cl

III. Desarrollo

El desarrollo de este software parte primero con la implementación de las funciones y medios necesarios para la interacción de usuario hardware por ejemplo la entrega por pantalla del reconocimiento de teclas o el que se imprima por pantalla uno o varios píxeles. El implementar esto requiere de las interrupciones a nivel de hardware ya que sin estas no podremos realizar las operaciones necesarias cuando se implementen las instrucciones para el juego.

Las interrupciones son funciones que nos permitirán interactuar a nivel de hardware las cuales están implementadas en C y Asm, cada una de ella cumple una función especifica, por ejemplo las funciones getcharAS(), printString(), putcharAS(), getch(),drawpixel(), initenvironment(), son funciones que nos permiten las interrupciones tanto para poder esperar teclas, imprimir caracteres, imprimir píxeles, preparar el entorno etc. Generando así las interrupciones y las funciones básicas para la implementación de algoritmos mas complejos que requieren de estas estructuras mas atomizadas.

Finalizada la etapa de interrupciones se prosigue con la implementación de las funciones que dibujaran el entorno del juego (initgraphics()), la administración de los movimientos (mov()) cuyos movimientos de la serpiente son (up(), down(), left(), rigth()), el dibujo de la serpiente por pantalla, la función que administre los movimientos (mov()) y las funciones de hash necesarias para la ubicación aleatoria de la comida (hashi(), hashj()) a buscar de la serpiente y una función que nos permite generar un spinwait en el procesador (eleep()).

Una vez dado a conocer los pasos seguidos debemos dejar claro que se utilizo un código base llamado psycho [9] el cual nos facilito parte del trabajo para el posterior desarrollo del juego, este archivo realizaba una impresión por pantalla a nivel de hardware de una serie de contornos de diferentes colores.

Basados en este código se prosiguió a modificarlo para adaptarlo y generar nuestro juego. Durante el desarrollo del software se requiere de un .img (archivo booteable) para probarlo en la maquina virtual bochs, este archivo se generaba por medio de un makefile que generaba y recibía los archivos .o, .s , .bin necesarios para el armado de la imagen (.img), esta imagen no debía poseer mas de 512 byte para el booteo cuya explicación esta en el anterior informe [10]: Junto al archivo principal .c se necesita un archivo .ld este archivo se llama linker script [11] que indicara a nuestra imagen las instrucciones de código de donde empiezan los archivos y

como están indicados los registros para el booteo, así nuestra imagen sabe donde empezar y si hay mas de un stage para levantar los archivos que son necesarios para nuestro juego, como también indica donde empezar a leer ya que nuestra imagen esta diseñada para ser levantada por medio una imagen floppy véase informe [12].
Para revisar el programa que logramos hacer, dentro de la carpeta código, se debe ejecutar el comando `./run.sh`, el cual abrirá 7 ventanas del emulador bochs. Cada una de las ventanas muestra una funcionalidad por separado, estas son

- La sepiente se mueve hacia la izquierda.
- La sepiente se mueve hacia la derecha.
- La sepiente se mueve hacia la arriba.
- La sepiente se mueve hacia la abajo.
- Se interactua con las teclas W A S D.
- La serpiente comienza a moverse cuando se presiona una tecla.
- Se muestra el hash que genera puntos dispersos para el alimento de la serpiente.

El producto final no se logro en su totalidad, se ha explicado anteriormente los pasos seguidos durante el desarrollo pero muchos de esas etapas se vieron frenadas o limitadas durante el desarrollo, en la sección de dificultades se dan a conocer los principales problemas y el por que no se termino el proyecto.

IV. Dificultades

Durante el desarrollo una de las principales dificultades fue en crear una imagen booteable con los archivos necesarios, esto se soluciono por medio de el proyecto [9] base que utilizamos ya que éste nos proporcionaba gran parte de los archivos básicos para generar un buen .img como los son el .ld, el .c donde se trabaja la parte principal del programa, el bochsrc.txt archivo base para la ejecucion por parte de bochs de la imagen y un makefile que nos genera todo lo necesario incluyendo el .img listo para su uso.

Solucionada la parte y ejecución de un archivo booteable se prosiguió con la implementación de el juego en si, ya que pudiendo bootear la imagen, las funciones que se iban implementando se probaban obteniendo resultados satisfactorios como logrando el movimiento de la serpiente por medio de tener presionada una tecla o el movimiento y borrado de los píxeles impresos por pantalla, como también movimientos verticales y horizontales y delimitando el movimiento de la serpiente.

A medida que se avanzaba se produjeron errores como solapado de registros de memoria, ya que secciones de registros asignados quedaban cortos en espacio asignados provocando errores al momento de querer compilar nuestra imagen y poder seguir codificando el resto de el juego ya que la falta de memoria nos limitaba en generar código para poder probarlo algo que se hizo insostenible a medida que se avanzaba mas. El problema de solapado se debe específicamente a que nuestro archivo .img no debía superar los 512 bytes para su booteo, por lo cual buscando y preguntando se presentaron dos posibles soluciones:

1. Modificación archivo linker script (.ld): Esta solución traía consigo una solución, pero de corto plazo la cual consistía en asignar mas memoria a las diferentes secciones del stack a cargar en los registros, por ejemplo a la sección .data .bss pero se tendría que mover la sección .rodata ya que se solapaban, en el caso de lograrlo sea manual o dinámicamente se presentaba el problema que si nuestro software era mas grande que 512 byte la asignación de mayor o menor memoria no tendría relevancia por el echo que nos se pueden sobrepasar los 512 bytes de booteo.

2. Generar un stage0 y stage1 para el booteo: Esta solución es usada por los kernels y sistemas operativos, la cual consiste en bootear una pequeña sección de código con los códigos y direcciones necesarias para que luego se puedan levantar la siguiente parte de los archivos necesarios, en realidad es un proceso continuo pero se divide en etapas por el echo de que el primero tiene limite de 512 bytes, si bien la solución era clara al momento de poder modificar el .ld se busco un código utilizable o un ejemplo explicado en internet, códigos que variaban de muy simple a muy complicados, junto a esto sumado la nueva sintaxis y poco conocimiento en la implementación termino por frenar el proyecto a pesar de saber que registros linkear que archivos llamar y como debería conformarse el .ld pero el poco conocimiento de la sintaxis freno el desarrollo.

3. WASD: El reconocimiento de las flechas de dirección del teclado no fue posible reconocerlas, junto a esto se procedió a buscar sus códigos en ascii o reconocerlas de otra forma posible, la ausencia de avance en la solución a esto llevo a que se usara como teclas de juego las teclas W, A, S, D las que reemplazarían a las flechas de dirección del teclado.

El frenado del desarrollo por los dos puntos mencionados anteriormente provocaron una dificultad en cadena como es el no saber como implementar un .ld que nos permita bootear por diferentes stages nuestro juego semejando se a un minikernel, derivando al no poder bootearlo en no poder probarlo y con ello no tener la posibilidad de tener mas memoria para codificar y asi probar que funcione de manera correcta.

V. Conclusión

La implementación de este tipo de software que corren a nivel de maquina requieren de diferentes etapas para software con tamaño mayor a 512 bytes, lo cual implica un conocimiento principal de como bootear, como generar un archivo booteable y como preparar por medio de un linker script los diferentes stages para el booteo adecuado.

La implementación requiere de entendimientos de los registros que permiten las interrupciones como los son 0x16 para teclado o 0x10 para pantalla, ejemplos que permiten la interacción luego de las diferentes funciones o usuarios con el software, estos deben estar implementados en asm por lo tanto también se debe tener un conocimiento en este lenguaje tan necesario para la implementación de los drivers o mini-kernel.

La implementación de variables de condición es esencial cuando se ve que los procesos corren por medio de continuos

ciclos y que requieren de interrupciones para el uso de funciones como por ejemplo interrumpir o verificar que el usuario ha presionado W para llamar a la función arriba y que luego se reconozca e interrumpa nuevamente para empezar a imprimir por pantalla los píxeles de la serpiente hacia arriba.

REFERENCIAS

[1] *Juego Snake*, 1997. [Online]. Available: http://es.wikipedia.org/wiki/Snake_(videojuego)

[2] *PC emulator-x86-64*. [Online]. Available: http://es.wikipedia.org/wiki/Snake_(videojuego)

[3] *Assembly language*. [Online]. Available: http://es.wikipedia.org/wiki/Lenguaje_ensamblador

[4] *gcc gnu compiler collection*. [Online]. Available: https://gcc.gnu.org/

[5] *Comando de transformacion de archvios a bajo nivel*. [Online]. Available: http://es.wikipedia.org/wiki/Dd_%28Unix%29

[6] *Assembler and disassembler for multiples architectures*. [Online]. Available: http://www.nasm.us/

[7] *Generacion de documentacion para codigo*. [Online]. Available: http://www.stack.nl/~dimitri/doxygen/

[8] *Administracion y versionado de proyectos*. [Online]. Available: https://about.gitlab.com/

[9] AshakiranBhatter, *Writing a boot loader in Assembly and C - Part 1*. http://www.codeproject.com/Articles/664165/Writing-a-boot-loader-in-Assembly-and-C-Part, 2015.

[10] F. o. Daniel mendez, *Informe sobre aplicación booteable*, 2015.

[11] *Linker Scripts*. [Online]. Available: http://wiki.osdev.org/Linker_Scripts

[12] D. Méndez, "Tecnologías de booteo," 2015.