INFORME

TPE

Arquitectura de computadoras

(72.08)

Autores:

- Valentinuzzi Antonio

Grupo 22

Fecha de entrega: 5/11/2025

1) Resumen

El objetivo de este trabajo es crear un sistema operativo basado en PURE64 aplicando los conceptos vistos a lo largo de la cursada.

Para esto, diseñamos código dividido en Kernel y UsuarSpace y diseñamos 2 programas, una shell para poder interactuar con el sistema y un juego de TRON.

Estos 2 programas se encuentran en el UserSpace y emplean una librería llamada GlobalLib, la cual se encarga de la mayoría de las funciones de los programas y hace llamadas de syscall al kernel diseñado.

Finalmente, se diseñó el manejo de 2 excepciones, la división por 0 y la llamada de función inválida.

2) Funcionamiento

Lo primero que se desarrolló fue el kernel. En él se crearon los drivers y las Syscalls necesarias para las aplicaciones del UserSpace. Además, se desarrolló una librería paralela, NativeLib Exclusiva de Kernel para lograr esto.

Driver de video.

Con respecto al driver de video, se decidió la implementación de ‘bitmap fonts‘ para la impresión de caracteres por pantalla, debido a que se considero lo más sencillo. Esto a su vez causó que, para la impresión de caracteres más grandes (16x16) o chicos (8x8) se tuvieran que buscar distintos bitmaps de caracteres. La razón por la que se decidió de tener 3 tamaños fue para que a la hora de iniciar en una letra de tamaño mediano (12x8) se tenga la posibilidad tanto de agrandar como de achicar el tamaño de letra. Sin embargo, a pesar de

que es más sencillo, ocupa varios archivos solamente llenos con los mapas, los cuales si no fueran por los comentarios, serían muy poco claros.

Por otro lado, para el dibujo de figuras geométricas, se implementó el dibujo de rectángulos, puesto a que estos incluyen también cuadrados, y círculos. Sin embargo, esta segunda figura no es óptima y para una futura implementación es mejor realizar el dibujo de elipses para tener más potencial de figuras diferentes.

Driver de teclado

Los drivers de teclado fueron trabajados por parte del kernel. Primero realizando una rutina en ASM que nos permitiera traernos la KEY de la posición del teclado a la que deseamos acceder. Luego en C se realizó un mapeado de que tecla representa cada KEY para un teclado QWERTY. Luego, agregamos una Interrupción en la cual se es llamada cada vez que se recibe un código por parte del teclado y actualiza el valor de la última KEY recibida. Finalmente agregamos una interrupción de software que al llamarla retorna el ascii de la última tecla presionada. Es importante recalcar que el teclado no detecta la presión de 2 teclas al mismo tiempo, por lo que comandos como “Shift + letra” y otros shortcuts para indicar mayúsculas no son posibles con esta implementación.

Driver de sonido

Se implementaron funciones de encendido y apagado, las cuales acceden al puerto 0x61 del PIT para controlar su uso. Al momento de generar el sonido, se indica la frecuencia y el tiempo deseado y se lanza una función wait que espera una cantidad de tiempo antes de apagarlo.

Esta función tiene la limitante de no poder cambiar la frecuencia del sonido de forma gradual y que no pueden sonar 2 sonidos de distinta frecuencia al mismo tiempo.

Interrupciones de software

Para la conexión del kernel y el userSpace, se realizó una de Interrupción de software; la cuales se encargaron de trabajar con los drivers. Estas llamadas permiten trabajar con los drivers de Video, Teclado, Sonido. Esta Interrupción, dependiendo de los parámetros que se le entreguen trabaja con un driver distinto y retorna o modifica los valores requeridos en cada ocasión. Por parte del usuario, se realizó un solo código en ASM que llama a la interrupción con hasta 5 parámetros mediante la función Syscall().

Excepciones

En cuanto a las excepciones, se implementó un arreglo de punteros a función para que sea más sencillo identificar de cuales se tratan (por ejemplo, la número 0, cero div, es la de la posición 0). Por otro lado, para el manejo de excepciones, se implementó una “Blue Screen Of Death”, la cual indica el valor de los registros y cuál fue el error. La forma en la que 4 muestra el valor de los registros es la misma que la que sirve para mostrarlos incluso cuando se la llama sin necesidad de una excepción. Luego de mostrados estos valores, se indica que se presione una tecla para continuar y volver al sistema. Durante la entrega previa esto no fue posible pero se arregló, reactivando las interrupciones gracias a la instrucción sti. Una vez terminado esto, se regenera el stack por medio de la función getStackBase y se “reinicia” el sistema.

Registros

Los registros se pueden obtener en cualquier momento si se presiona la tecla ‘Tab’. Esto muestra una pantalla con todos los valores actuales de los registros. Esto se logra principalmente por una macro en ASM que se encarga de guardar los valores de todos los registros en un array. Luego, desde el kernel Space, se puede obtener este arreglo por medio de un getter que se encuentra en la nativeLib (o sea, la librería exclusiva de kernelSpace), que luego se guarda en una estructura (llamada registerState). Luego, estos se pueden imprimir cuando se ejecuta una excepción o cuando se presione la tecla tab. Para este último, en el administrador de system calls, se modificó la que se encarga de leer para que esta, si es que se obtuvo el carácter con el código ascii 9 (o sea, el tabulador), se despliegue el menú especial que muestra los registros, y otra vez, si se pulsa otro carácter, se sale del menú.

Este proceso puede ser optimizado, el kernel está diseñado para interrumpir al user space para mostrar los registros. Pero también muestra en pantalla, esta implementación puede ser mejorada haciendo una función que guarde los registros al presionar tab y que estos sean accesibles al user space mediante una SysCall.

Time

Al momento de obtener la fecha y la hora actual se hizo uso del RTC. Para acceder a ella primero se implementó una función en libasm.asm para cada uno de los valores más importantes (hora, minuto, segundo, día, mes y año). Dentro de cada función primero se activa el formato binario para no recibir el dato en BCD, y después se retorna el valor solicitado. Cada valor se guarda en un struct “date” para facilitar su uso.

Un detalle que descubrimos en el momento de testear esta funcionalidad. es que funciona perfectamente en QEMU, pero al momento de testear en una VM y en HW encontramos que su funcionamiento no es el esperado.

TRON

Para el desarrollo del juego de TRON se terminó implementando de manera tal que cada elemento de la cola sepa cuál es su siguiente. El tablero se guardó en 3 una matriz de 20x28, conteniendo información de si el casillero estaba había sido pasado por el jugador y cuál era el siguiente en forma de un struct. Esto se pensó para que intuitivamente a la hora de calcular una posible colisión que cause que el juego termine. A su vez, se crearon variables globales para facilitar el acceso a varios datos que utilizan muchas funciones y no tener que pasarlas por parámetro. Por último, se decidieron hacer 2 funciones separadas de “motor de juego”, debido a que se consideró que, a pesar de haber código muy similar, había que implementar una cantidad tan importante de cambios a esta que resultó más práctico hacer una distinta.

3) Benchmarking

Para este apartado, se realizaron mediciones en 3 medios distintos: QEMU, una pc de escritorio (Apéndice 5.1) y Virtual box. En todos los medios se observó su tiempo de acceso a hw, los fps que presentaban en el juego de TRON y su velocidad para realizar operaciones en punto flotante.

Performance

Se tomaron 5 mediciones en cada medio con la función PERFORMANCE de la shell.

Virtual Box	QEMU	PC
34 ms	358 ms	2953 ms
42 ms	361 ms	2954 ms
33 ms	360 ms	2954 ms
44 ms	358 ms	2953 ms
33 ms	366 ms	2953 ms

Acceso a hardware

Se tomaron 5 mediciones en cada medio con la función HARDACCESS de la shell.

Virtual Box	QEMU	PC
4	15	841
4	18	841
4	16	841
5	14	841
4	15	841

FPS

Las mediciones en esta caso fueron constantes con un valor de 11, debido a la implementación de la aplicación TRON, sus fps son iguales en los 3 medios.

4) Conclusiones

El sistema operativo es funcional, la shell logra lo que se propone en los 3 medios de prueba. Sin embargo, su rendimiento varía mucho en cada medio, haciendo que funcione muy bien en QEMU y Virtual Box pero teniendo un funcionamiento deplorable en el hardware físico.

5) Apéndice

(5.1) Especificaciones de HW

Procesador: Intel(R) Core(TM) i5-8500 CPU @ 3.00GHz, 3000 Mhz, 6 procesadores principales, 6 procesadores lógicos

Producto placa base: B360M GAMING HD

Memoria física instalada (RAM): 24,0 GB

GPU: NVIDIA GeForce GTX 1050 TI