Arquitectura de Computadoras

Informe TPE

Integrantes:

Axel Castro Benza - 62358 - acastrobenza@itba.edu.ar

Rocio D´Intino - 62341 - rdintino@itba.edu.ar

Ian Franco Tognetti - 61215 - itognetti@itba.edu.ar

Introducción

El siguiente trabajo práctico especial consiste en implementar un kernel booteable por Pure64, el cual ha sido adaptado y modificado para cumplir con los requisitos del proyecto. El objetivo principal es administrar los recursos de hardware y proporcionar una interfaz de programación de aplicaciones (API) para que los usuarios puedan utilizar dichos recursos.

Separación Kernel - Userspace

El kernel es responsable de administrar y controlar los recursos del sistema, como la memoria, los dispositivos de entrada y salida, entre otros. Además, el mismo proporciona una capa de abstracción y un conjunto de funciones o servicios que pueden ser utilizados por las aplicaciones y procesos que se ejecutan en el espacio de usuario (userspace).

En otras palabras, el kernel actúa como un intermediario entre el hardware y el software. Es el único componente capaz de interactuar de manera directa con el hardware a través de los controladores o drivers específicos para cada dispositivo, como el keyboardDriver y el videoDriver. De este modo, los ejemplos anteriormente mencionados le permiten al kernel gestionar el acceso y la comunicación con los dispositivos de hardware: el teclado y la pantalla, respectivamente.

Por otra parte, a diferencia del kernel, el userspace no tiene acceso directo al hardware. En su lugar, debe utilizar las interfaces proporcionadas por el kernel para acceder a los recursos y servicios del sistema. Esto se logra haciendo uso de la API, a través de la cual el espacio de usuario puede hacer uso de aquellas funciones del kernel que hayan sido diseñadas para realizar tareas específicas. Dicho acceso ha sido implementado a través de la interrupción de software 80h, dado que los dos módulos se encuentran en distintos espacios de memoria. Se cuenta con un handler y un dispatcher, el cual se ocupa de determinar qué syscall específica debe ser ejecutada, y en consecuencia, llama a la función que se encarga de resolver dicha tarea.

Antes de generar la interrupción, es necesario disponer de todos los parámetros requeridos para la syscall que se va a invocar. El primer parámetro es el identificador (ID) de la syscall, el cual se encarga de identificar el llamado a sistema específico que se desea ejecutar.

Manejo de las syscalls implementadas

A continuación, se detallan las acciones realizadas por cada uno de los llamados a sistema que se han implementado:

sys_write: se encarga de escribir una cadena de caracteres en la pantalla. Con el fin de facilitar y agilizar el manejo de impresión por pantalla, se ha optado por diferenciar un caso particular de los parámetros "row" y "col", que de ser iguales a -1, se muestra la cadena a partir de la posición actual del cursor. Caso contrario, se imprime en una coordenada específica en la pantalla.

sys_read: se utiliza para leer datos del teclado. Las teclas leídas se almacenan en un buffer proporcionado como parámetro, y la cantidad máxima de caracteres a leer se especifica en el parámetro "length". La función "*readKeyboard(...)*" se encarga de llevar a cabo la lectura del buffer del teclado, el cual se carga y actualiza cada vez que se presiona una tecla, es decir, al producirse una interrupción de teclado.

sys_clear: se ocupa de limpiar la pantalla. Si bien no era requisito incluir un comando que limpie la terminal, consideramos que dicha función es útil a la hora de visualizar la misma, al igual que al realizar las gráficas del juego pong.

sys_time: se utiliza para obtener la fecha y hora actual del sistema. La información se almacena en la estructura de datos "currentDate", que se pasa como parámetro y cuya información se completa por medio de la función "*getDate()*".

sys_holder: implementa una espera activa durante un tiempo determinado, haciendo uso del temporizador interno para calcular los ticks transcurridos y determinar cuándo finalizar la espera

sys_beep: se ocupa de generar un sonido o beep en el sistema durante un período de tiempo determinado.

sys_info_reg: es la encargada de obtener información sobre el estado de los registros. El valor de cada uno de los registros se almacena en un arreglo, el cual se actualiza al presionar las teclas "ctrl+R". De este modo, es posible guardar y acceder a los mismos en todo momento, incluso mientras se juega al pong.

sys_draw: se utiliza para dibujar un píxel en la pantalla en la posición determinada por los parámetros "row" y "col". De este modo, es posible dibujar diversos objetos, lo cual ha sido particularmente útil para los paddles y la pelota del juego implementado.

Las funciones previamente mencionadas son llamadas en el “syscallDispatcher()”, dependiendo del llamado a sistema que se intente realizar. La principal ventaja de este diseño es que proporciona un mecanismo centralizado para gestionar y enrutar las syscalls en un sistema operativo.

Al utilizar un dispatcher, se puede separar la lógica de implementación de cada syscall en funciones individuales y asignarles identificadores únicos. Cuando una aplicación de usuario realiza una llamada al sistema, sólo necesita proporcionar el identificador de la syscall y los parámetros necesarios. El dispatcher se encarga de identificar la syscall correspondiente según el ID proporcionado y dirigir la ejecución a la función de implementación adecuada.

En adición, al implementar cada syscall en una función individual, en términos de diseño es posible decir que la modularidad facilita la comprensión y el mantenimiento del código. En este sentido, el dispatcher permite agregar, modificar o eliminar syscalls de manera más sencilla: en caso de querer agregar una nueva funcionalidad, simplemente se implementa la syscall correspondiente y se actualiza el dispatcher para que pueda reconocerla y manejarla correctamente.

Manejo de interrupciones

Se utiliza la IDT (Interrupt Descriptor Table) para gestionar las interrupciones. Se trata de una tabla de descriptores de interrupción que contiene información sobre cómo manejar diferentes tipos de interrupciones y excepciones, cuya carga se realiza en el archivo "*idtLoader.c*". Las dos principales interrupciones manejadas en este trabajo pertenecen al teclado y al timer tick.

El manejo de la primera interrupción mencionada implica capturar el código de la tecla presionada, procesarlo y realizar las acciones correspondientes, como manejar combinaciones de teclas especiales o realizar acciones específicas según la tecla, a medida que se carga el buffer.

La interrupción del timer tick es relevante a la hora de mantener un control del tiempo, por lo que el manejo de esta interrupción implica la actualización de un contador de ticks.

Por otro lado, las rutinas de interrupción propiamente dichas se definen en el archivo "*interrupts.asm*", las cuales se ocupan de manejar y atender las interrupciones específicas.

Dentro del último archivo mencionado, se cuenta con un handler de excepciones que llama a la función *"exceptionDispatcher(...)"* ubicada en *"exceptionDispatcher.c"*, la cual logra ,determinar la excepción ocurrida gracias a un identificador: el ID correspondiente a la *"Zero Division Exception"* es 0 y para *"Invalid Code Exception"* es 6. Las ventajas de trabajar con dicho diseño son las mismas que se han mencionado en la sección anterior. Asimismo, tras determinar el tipo de excepción que se debe resolver, se imprime un mensaje de error en pantalla junto con el estado de los registros al momento de haberse generado la excepción. Luego, se vuelve a inicializar la shell junto con el intérprete de comandos.

Modo Gráfico VESA

Se utilizó el modo gráfico puesto que tiene un control simple de los píxeles de pantalla, aunque esto implicaba dibujar pixel por pixel. Por esta razón, fue necesario implementar un sistema de dibujo de texto y formas, tanto para imprimir cadenas por pantalla, así como dibujar objetos en el pong. Para ello, se creó un archivo *"font.c"* donde se maneja el formato de los caracteres, el tamaño y la tipografía, al igual que *"videoDriver.c"*, ambos en kernel.

Dentro del último archivo mencionado, se pueden encontrar funciones que van desde la impresión de un pixel, llamada *"putPixel(...)"*, pasando por otra que imprime un carácter, denominada *"putChar(...)"*, hasta una que imprime un string en una posición determinada, nombrada *"printIn(...)"* , entre otras similares. Estas funciones pueden ser accedidas en el Userspace mediante su debida syscall.

A su vez, se llevaron a cabo funciones con el fin de lograr un correcto funcionamiento del intérprete de comandos. Estas son: *"updateCoordinates()",* que se encarga de actualizar las líneas de comando, *"clearScreen()"*, *"enter()"* y *"backspace()"*.

Con el fin de lograr un buen dibujo de los caracteres, se implementó una fuente (font), la cual fue extraída de internet. Es posible apreciar que con la función *"putCharIn()"* se termina de crear el formato del carácter, imprimiendo el mismo por pantalla. Lo anteriormente mencionado se lo logra iterando sobre una matriz de bits y dibujando píxeles en función de los valores de los bits. Si un bit es cero, el píxel correspondiente se dibuja en color negro, caso contrario, el píxel se dibuja en otro color especificado por la variable color.