Instituto Tecnológico de Buenos Aires

Trabajo Práctico Especial

72.08 - Arquitectura de Computadoras

Integrantes

Madero Torres, Eduardo Federico - 59494

Otegui, Maria - 61204

Segundo Cuatrimestre del 2023

Índice

1. Introducción

En el presente trabajo se presenta la resolución de la propuesta planteada en el Trabajo Práctico Especial. El objetivo principial ha sido realizar una implementación de un Kernel basado en una arquitectura de 64 bits (64x Barebones) que se encargue de administrar los diferentes recursos de la computadora, más el armado de una API en donde los usuarios puedan utilizar el hardware sin que haya una interacción directa con el mismo.

Para cumplir con el objetivo, se han diferenciado dos espacios fundamentales: el Kernel space y el User space, Kernel y Userland respectivamente. En el Kernel se implementan todos los drivers (en donde se encuentran las funciones que el usuario utilizará desde el segundo espacio), las interrupciones y las syscalls. Mientras que en el Userland, se desarrolla la consola de comandos que el usuario podrá visualizar, y elegir dentro de una lista de opciones de módulos descriptos en el enunciado del trabajo práctico (tales cómo el juego Snake, visualización de registros, manejo de excepciones, entre otras).

2. Kernel

El Kernel de un sistema operativo sirve para administrar los recursos de hardware solicitados por los diferentes elementos de software y hacer de intermediario decidiendo a que y cuando se concede este acceso evitando así sobrecarga del sistema, recursos innecesarios y llegar a poder controlar así todo el sistema. Los recursos pueden ser accedidos mediante interrupciones que pueden ser de hardware(como por ejemplo las que suceden presionando las teclas del teclado), o las de software (como las que suceden por medio de la directiva int 80h).

El kernel cuenta con una función principal denominada `main`, en la cuál se ejecutan las funciones que inicializarán el funcionamiento del sistema. La primera función en ejecutarse será `load_idt`, en donde se carga la Interrupt Descriptor Table (IDT), en dónde cargamos las rutinas de atención de interrupciones al CPU. Una vez que ya están definidas y establecidas las interrupciones, excepciones y las systems calls con la función de precarga, se ejecuta un llamado a la función principal en el Userland, correspondiente a la consola de comandos (shell).

Se desarrollan a continuación las diferentes estructuras con mayor importancia para la implementación del kernel.

2.1. Interrupciones e IRQ Dispatcher

Las interrupciones son señales externas, provenientes de un dispositivo, como un teclado o un disco duro, hacia la CPU, que lo interrumpen y detienen inmediatamente lo que está haciendo actualmente y para realizar otra tarea. Para poder manejar las interrupciones, se utiliza una función en assembler que definimos irqHandlerMaster que dependiendo el IRQ llamará a la función IRQDispatcher para que pueda ser atendida la interrupción correspondiente.

Es importante saber que el IRQ (del inglés interrupt request, en español "petición de interrupción") es una señal recibida por el procesador de una computadora, para indicarle que debe interrumpir el curso de ejecución actual y pasar a ejecutar código específico.

La función IRQ Dispatcher es la encargada de la administración de interrupciones de hardware que se desean solicitar al CPU. En nuestra implementación, únicamente se administran dos de los recursos totales del dispositivo:

- **Keyboard Handler**: El Keyboard Handler se encarga de gestionar las interrupciones generadas por el teclado y determinar qué acciones deben realizarse en respuesta a esas interrupciones.
- **Timer Handler** : El reloj del procesador es un oscilador interno que genera señales de sincronización para coordinar las operaciones del procesador. Cada ciclo de reloj, también conocido como tick, representa una unidad básica de tiempo en la que se ejecutan las operaciones del procesador. Cada instrucción o tarea que realiza el procesador generalmente requiere varios ticks para completarse. Es por eso que el contador de ticks se utiliza para contar la cantidad de ciclos de reloj que han ocurrido desde el inicio del procesador. Esto puede ser útil en situaciones en las que se necesita realizar un seguimiento del tiempo transcurrido.

El despachador se encargará de ejecutar las funciones correspondientes al tipo de interrupción que sucedan. Al producirse una interrupción antes de ejecutar el controlador (handler), se guarda toda la información relevante en la pila (stack). Esto incluye los valores de los registros RSP (puntero de pila), RFLAGS (banderas de estado), CS (selector de segmento de código) y RIP (puntero de instrucción de retorno). El propósito de guardar esta información en la pila es preservar el contexto del programa actual antes de manejar la interrupción. Una vez que se completa el manejo de la interrupción, se restaura el contexto previo del programa a partir de la pila, lo que permite que el programa continúe su ejecución desde donde se interrumpió.

Al guardar y restaurar esta información en la pila, se asegura una transición suave entre el programa en ejecución y el controlador de interrupción, permitiendo que se maneje adecuadamente la interrupción y se retome el flujo normal del programa después de su manejo.

2.2. Exceptions

Las excepciones son eventos generados por el procesador cuando detecta una o más condiciones predefinidas al ejecturar una instrucción. Es decir, son interrupciones que son generadas por el propio procesador, al ser estas interrupciones sus rutinas de atención también están cargadas en la Interrupt Decriptor Table. Existen tres tipos de excepciones:

- **Fault**: Son excepciones que pueden ser corregidas. El procesador guarda en la pila la dirección de la instrucción que produjo la falta. Por ejemplo, supongamos que se está accediendo a una página de memoria que no está presente en la memoria física. En este caso, se produce una falta de página y el procesador genera una excepción de falta.
- **Trap**: Se utilizan para realizar accesos al sistema operativo. Un ejemplo común es la interrupción de software (software interrupt) que permite al programa hacer una solicitud al sistema operativo para realizar alguna tarea específica.
- **Abort**: No siempre se puede obtener la instrucción que causó la excepción. Reporta errores severos. Un ejemplo común de una anulación sería un fallo de hardware crítico que impide que el sistema continúe funcionando adecuadamente. En tal caso, el sistema operativo o el firmware del dispositivo puede generar una anulación para indicar que se debe detener la ejecución y reportar el error.

Para las excepciones, también se define una función ExceptionDispatcher, que se encarga de administras las excepciones. En este trabajo se ha pedido considerar dos tipos de excepciones que son las siguientes:

- **Zero Exception**: El detector de divisiones por cero se encarga de identificar cuándo se produce una división matemática entre cero. Esto se puede simular fácilmente al realizar una división entre dos variables. Su utilidad radica en la implementación de aplicaciones que involucren operaciones aritméticas, ya que permite detectar y manejar adecuadamente esta situación para evitar errores y comportamientos inesperados.
- **Invalid Operation Code Exception**: Detecta cuando se genera una operación de código inválido. Suele ocurrir con el manejo erróneo de punteros.

En el despachador de las excepciones, se imprimen además el estado de los registros justo en el momento que ocurre la excepción, el instruction pointer e información asociada al tipo de error. Luego se restaura al contexto del programa actual antes de manejar la excepción.

2.3. Syscalls

Una syscall, o llamada al sistema, es un mecanismo mediador que permite acceder a los servicios proporcionados por el kernel del sistema operativo. Funciona a través de interrupciones, donde se realiza una llamada desde el espacio de usuario a funciones internas del kernel. Estas funciones brindan acceso a diversas funcionalidades del hardware, como dibujar en pantalla o leer la información del teclado. Para garantizar un funcionamiento adecuado, se organizan controladores (drivers) que diferencian y gestionan correctamente cada syscall. Cuando un usuario ejecuta una syscall, se produce una interrupción que permite al kernel recibir la información relevante y distinguir qué syscall se está ejecutando, para luego llamar a la función del controlador correspondiente.

2.3.1. SyscallDispatcher

El despachador de syscalls desde el kernel es una componente del sistema que actúa como mediador entre las interrupciones y los recursos requeridos. Cuando se realiza la interrupción int 80h, este administrador es responsable de solicitar los recursos necesarios. El despachador de syscalls desde el kernel desempeña un papel fundamental al facilitar la comunicación y la asignación de recursos entre las interrupciones y las funciones del controlador correspondientes.

En la implementación el despachador se define en la función sysDispatcher que toma el número de syscall y redirige la llamada a la función de manejo correspondiente, utilizando la matriz syscalls que almacena las direcciones de las funciones. Esto permite una interacción controlada entre el espacio de usuario y el kernel del sistema operativo.

2.3.2. Drivers

Los drivers son componentes encargados de facilitar la implementación del acceso a funcionalidades específicas del hardware. En el contexto del sistema operativo, se incluyen los controladores necesarios para el acceso al teclado, la capacidad de dibujar en pantalla y obtener la fecha y hora actual. Esto permite una mayor flexibilidad y escalabilidad en el acceso y la gestión de los recursos del sistema operativo.

2.3.2.1 Keyboard Driver

Cuando el usuario presiona una tecla, se genera una interrupción de hardware (21h) que debe ser gestionada y su resultado guardado. Esto es responsabilidad del KeyboardDriver, un controlador encargado de administrar el teclado. Al producirse la interrupción, se ejecuta una función en C llamada keyboardHandler, la cual se encarga de recibir el código de escaneo (scan code) de la tecla presionada y realizar las siguientes acciones:

- Si el código de escaneo corresponde a la tecla de control (CONTROL), se establece una variable booleana savedRegs en true para indicar que los registros deben ser guardados.
- En caso contrario, se verifica que el buffer no esté lleno (se haya alcanzado la capacidad máxima). Si el buffer está lleno, no se realiza ninguna acción adicional.
- Si el buffer aún tiene capacidad, se almacena el código ASCII correspondiente a la tecla presionada en un buffer circular (buff). Este buffer se utiliza como una estructura de datos FIFO (First-In-First-Out), donde los nuevos elementos sobreescriben los más antiguos cuando se alcanza la capacidad máxima.
- Se actualizan las variables rear y cantElems para mantener el correcto funcionamiento del buffer circular.

Además, se implemento un manejador de código de interrupción del teclado. las siguientes funciones en lenguaje assembly para administrar el hardware del teclado. Irq01Handler: Esta función es el manejador de interrupciones para la interrupción del teclado (IRQ 1). En ella se realiza lo siguiente:

- Se guarda el estado actual de los registros en la pila mediante la instrucción pushState.
- Se lee el código de escaneo del teclado utilizando la instrucción in y se almacena en el registro al.
- Se compara el código de escaneo con el valor correspondiente a la tecla de control (1Dh). Si son iguales, se realiza un volcado de registros.
- En caso contrario, se llama a la función irqDispatcher pasando el código de escaneo como parámetro para su procesamiento.
- Se realiza la señalización de fin de interrupción (EOI) al controlador de interrupciones programable (PIC).
- Se restaura el estado de los registros mediante la instrucción popState.
- Se retorna de la interrupción utilizando la instrucción iretq.

En resumen, el KeyboardDriver es responsable de administrar el teclado y procesar las interrupciones generadas por las teclas presionadas. La función keyboardHandler se encarga de recibir los códigos de escaneo, traducirlos a códigos ASCII y manejar un buffer circular para almacenar los caracteres correspondientes a las teclas presionadas. Además, se implementa el volcado de registros cuando se detecta la tecla de control. Las funciones en lenguaje assembly permiten activar el teclado, obtener los códigos de escaneo y gestionar las interrupciones del teclado.

2.3.2.2 Video Driver

El video driver es el encargado de la administración de acceso a la pantalla utilizando el modo VBE (VESA BIOS Extension) activado manualmente desde el bootloader. Este driver maneja los píxeles visibles en la terminal, los cuales están representados por tres bytes que determinan la combinación de colores azul, verde y rojo.

Inicialmente, se implementó la funcionalidad para imprimir un píxel en una coordenada específica de la pantalla, basándose en sus valores RGB y una posición proporcionada como parámetro. Posteriormente, surgió la necesidad de escribir caracteres en la pantalla, similar a una terminal o sistema operativo. Para lograr esto, se utiliza el archivo font.c que contiene un vector de chars donde en la posicion del ASCII del caracter se encuentran indicadores binarios para dibujar dicho caracter. Donde si en el binario hay un uno, indica que debe haber un pixel y en los ceros el fondo.

Se consideró necesario proporcionar una syscall para "limpiar"la pantalla, que consiste en dibujar píxeles de color negro en toda la pantalla.

Con el manejo de píxeles, figuras y caracteres en la pantalla, se presenta la posibilidad de implementar un manejo completo de la pantalla utilizando un cursor con posiciones cartesianas, lo que permite escribir texto de manera similar a un documento o una interfaz gráfica con entrada promedio por teclado. Esta funcionalidad no se considera parte del driver de video, sino más bien de un archivo de librería que se encarga de manejar las necesidades del usuario a través de syscalls.

En este sentido, se agregaron las funcionalidades necesarias para el manejo de la limpieza de renglones de caracteres, la capacidad de desplazar el texto existente hacia arriba al finalizar un renglón y la disponibilidad de un cursor de escritura, entre otras funciones. También se incluyeron funciones para imprimir errores en pantalla, convertir números enteros a caracteres y representar valores hexadecimales.

2.3.2.3 Time Driver

Se administra el acceso al Real Time Clock (RTC) del hardware mediante un controlador. Este controlador contiene tres funciones en lenguaje assembly: getHours, getMinutes y getSeconds. Estas funciones interactúan directamente con los registros del RTC a través de las instrucciones in y out, que permiten leer y escribir en los puertos de E/S específicos del RTC. Cada función utiliza una combinación de instrucciones para seleccionar el parámetro de tiempo deseado (horas, minutos, segundos) mediante la manipulación del registro de control (port 70h) y luego obtiene el valor correspondiente del registro de datos (port 71h).

En el contexto de la syscall, se definen funciones adicionales (3getHours, 4getMinutes y 5getSeconds) que se encargan de llamar a las funciones del driver y asignar el valor obtenido a través de un puntero pasado como parámetro. Estas funciones permiten al usuario obtener el tiempo actual del RTC mediante una llamada a la syscall correspondiente.

Es importante destacar que si el usuario desea acceder a múltiples parámetros del tiempo del RTC, debe realizar múltiples syscalls, ya que cada función solo devuelve un parámetro específico. Sin embargo, se aclara que el acceso completo a la información del RTC no se utiliza con frecuencia, ya que generalmente solo se requiere información específica del tiempo. Este controlador no está diseñado para ser utilizado como un temporizador, sino como una forma de obtener información del RTC del hardware.

2.3.2.4 Sound Driver

La implementación del juego Pong, requería que se manejara el driver de sonido. Es por eso que se ha implementado un driver para reproducir sonidos utilizando el altavoz incorporado en la computadora. El driver consta de varias funciones clave. La función outb se encarga de enviar datos a un puerto de E/S específico, mientras que la función inb se utiliza para leer datos de un puerto de E/S. Estas funciones son utilizadas para comunicarse con el controlador de altavoz y realizar operaciones de entrada y salida necesarias.

La función principal del driver es playsound, la cual reproduce un sonido a través del altavoz. Esta función calcula el divisor necesario para establecer la frecuencia deseada y configura los registros del controlador de temporizador programable (PIT) utilizando las instrucciones outb. Luego, activa el altavoz utilizando la instrucción outb y establece los bits adecuados para permitir la reproducción del sonido.

Por otro lado, la función nosound se encarga de detener la reproducción del sonido. Esta función lee el valor actual del puerto correspondiente al altavoz, aplica una máscara para borrar los bits de control y luego actualiza el puerto para detener la reproducción.

Finalmente, la función beep es una utilidad conveniente que permite reproducir un sonido de duración limitada. Esta función utiliza la función playsound para iniciar la reproducción del sonido con una frecuencia especificada, pausa la ejecución durante el tiempo especificado utilizando una función no proporcionada en el código llamada sleepms, y luego detiene la reproducción del sonido utilizando la función nosound.

Este driver proporciona una interfaz para generar sonidos a través del altavoz de la computadora utilizando programación directa de hardware. Permite reproducir sonidos de diferentes frecuencias y controlar la duración de la reproducción. Se ha obtenido la información necesaria para la implementación desde la página web Osdev[1]

[1]https://wiki.osdev.org/PC_Speaker

3. Userland

Es el entorno en el que el usuario interactúa con el sistema operativo, alojando sus aplicaciones y bibliotecas. En este entorno, las aplicaciones dependen de las llamadas al sistema (syscalls) existentes para solicitar permisos y acceder a los recursos del sistema operativo. Si se desea agregar funcionalidad adicional que supere las capacidades actuales, es necesario extender las syscalls existentes.

3.1. Administración de Syscalls

Se ha implementado una interfaz de llamadas de usuario (USER CALLS). La misma contiene una serie de funciones que utilizan las syscalls del kernel para llevar a cabo diferentes tareas. Estas funciones están definidas en las bibliotecas user_syscalls.h y user_lib.h

Las funciones realizan las siguientes acciones:

- sys_clear_screen( ): Llama a la syscall correspondiente para limpiar la pantalla.
- sys_write(char * c, int color): Escribe caracteres en pantalla con un color específico.
- sys_read(char * buff): Lee una entrada del usuario y la guarda en el búfer proporcionado.
- sys_get_hours(int * hours): Obtiene la hora actual y la guarda en la variable hours.
- sys_get_minutes(int * min): Obtiene los minutos actuales y los guarda en la variable min.
- sys_get_seconds(int * sec): Obtiene los segundos actuales y los guarda en la variable sec.
- sys_new_line(): Realiza una operación de .enter.º salto de línea en pantalla.
- sys_write_dec(int c, int color): Escribe un número decimal en pantalla con un color específico.
- sys_beep(int freq, int time): Genera un sonido con una frecuencia y duración especificadas.
- sys_sleep(int ms): Suspende la ejecucion del programa durante el numero de milisegundos especificados utilizando el conteo de ticks y una función de pausa específica del sistema.
- sys_put_pixel(uint32_t color, uint32_t x, uint32_t y): Coloca un píxel en la pantalla en las coordenadas (x, y) con el color especificado.
- sys_get_screen_width(int * width): Obtiene el ancho de la pantalla y lo guarda en la variable width.
- sys_get_clean_buffer(): Limpia el búfer de pantalla.
- sys_registers(uint64_t * registers, bool * isSaved): Obtiene y guarda los registros del sistema en el array registers y el estado de guardado en la variable isSaved.
- sys_change_font_size(int size): Guarda un dígito que representa el tamaño de la fuente, puede ser 1, 2 o 3.

3.2. Shell

El intérprete de comandos del usuario es el componente central del sistema operativo, diseñado para interactuar con el usuario y proporcionar acceso a todas las funcionalidades disponibles. Al iniciar el sistema, el intérprete de comandos se inicia de forma predeterminada, actuando como el punto de entrada para el usuario. Esto es similar a la experiencia de encender una computadora sin sistema operativo, donde se presenta una terminal que permite al usuario interactuar a través de líneas de comando. Los módulos correspondientes son:

- **HELP**: Despliega en la pantalla las opciones de comandos disponibles y su descripción.
- **TIME**: Muestra la hora actual en la pantalla.
- **CLEAR**: Limpia la terminal, eliminando el contenido previo.
- **PONG**: Inicia el juego Pong, donde el usuario puede jugar al juego en la terminal.
- **REGISTERS TEST**: Realiza una prueba para verificar que los registros del sistema tengan los valores correctos. Esta implememementación la consideramos necesaria como paso intermedio y para poder verificar que los registros se guardaran correctamente.
- **DIVIDE BY ZERO**: Provoca una excepción controlada de "división por cero", generando un mensaje de error en la pantalla.
- **INVALID OPERATION**: Provoca una excepción controlada de operación inválida, generando un mensaje de error en la pantalla.
- **SNAKE**: Inicia el juego Snake, donde el usuario puede jugar al juego en la terminal, se implementó una single player versión, pero se dejó un esquema para una futura implementación de dos jugadores.
- **LETTER SIZE**: Permite el cambio en el tamaño de la fuente de la consola.

3.3. Snake

La implementación del juego snake fue hecha en un entorno de consola. El juego se basa en un bucle principal que gestiona la lógica del juego y la interacción con el jugador. Se presenta a continuación una descripción más detallada de los componentes y el esquema del juego.

Estructuras y constantes El juego utiliza las siguientes estructuras y constantes:

- **snk**:Representa el cuerpo de la serpiente y su posición en el campo de juego.
- **frt**: Representa la fruta que la serpiente debe comer para crecer.
- **V y H**: Constantes que definen las dimensiones del campo de juego.
- **N**:Constante que define el tamaño máximo del cuerpo de la serpiente.
- **ESC**:

Funciones principales El juego utiliza varias funciones principales para gestionar distintos aspectos del juego:

- **Init**: Inicializa las posiciones iniciales de la serpiente y la fruta, así como otras variables relevantes.
- **Intro_Field**: Inicializa el campo de juego con límites y espacios en blanco.
- **Intro_Data**: Inicializa la posición y la representación visual de la serpiente y la fruta en el campo.
- **loop**:Contiene el bucle principal del juego, controla la lógica del juego y actualiza la pantalla.
- **input**:Gestiona la entrada del jugador y verifica las colisiones con los límites y el cuerpo de la serpiente.
- **update**:Actualiza la posición de la serpiente y la fruta en el campo de juego.
- **input**:Representa visualmente el campo de juego en la consola.
- **checkOverlap**: Verifica si hay superposición en la posición de la serpiente en el campo.

Funciones adicionales y variables El juego también incluye algunas funciones adicionales y variables para operaciones específicas, como la generación de números aleatorios, la gestión de la pantalla y la interacción con el usuario. Además, se han incluido comentarios para futuras implementaciones del modo de dos jugadores, aunque el código actual solo soporta el modo de un solo jugador.

El juego utiliza la biblioteca syscalls para realizar operaciones en la consola, como limpiar la pantalla, obtener la entrada del teclado y escribir en la pantalla.

En resumen, el juego de Snake implementado en este código presenta una estructura y lógica sólidas para permitir que un jugador controle una serpiente en un campo de juego, con el objetivo de comer frutas y evitar colisiones con los límites y el cuerpo de la serpiente. La implementación utiliza funciones y estructuras que facilitan la manipulación y representación de los elementos del juego en la consola.

4. Desafíos

4.1. Recuperación de Excepciones

En principio se había definido una función reset()"que buscaba reestablecer el contexto original justo antes que sucediera la excepción. La función reset.[es] la siguiente:

```
call getStackBase
mov rsp, rax
call main
```

La dificultad con las excepciones en el código proporcionado radica en la incorrecta gestión de la recuperación después de una interrupción. En lugar de restablecer el contexto original de la pila, el código reinicializa el "MAIN"del kernel, lo cual no nos lleva al momento exacto justo antes de que ocurriera la interrupción. Para recuperarnos correctamente, es necesario comprender que cada vez que se ejecuta una interrupción, se empujan en la pila los registros RSP, RFLAGS, CS y RIP.

Para solucionar este problema, debemos asegurarnos de que el RSP sea la base de la pila original, lo cual se logra utilizando el "stackbase". Luego, el RIP debe ser la dirección de inicio del programa principal del "sampleCodeModule", que es donde comienza nuestro programa de usuario. Finalmente, es necesario guardar todos estos valores desde el manejador de excepciones de la siguiente manera:

```
call getStackBase
mov [rsp+3*8], rax

mov rax, 0x400000
mov [rsp], rax
iretq
```

En esta nueva versión, utilizas getStackBase para obtener la dirección base de la pila y la guardas en rax. Luego, mueves el valor de rax a [rsp+3*8], que corresponde al tercer elemento en la pila, donde originalmente se encontraba almacenado el RIP.

Después de realizar esta modificación, estableces manualmente el valor de 0x400000 en [rsp], que corresponde al segmento de código de inicio del programa principal del "sample-CodeModule".

Finalmente, utilizas la instrucción iretq para restaurar el contexto original de la interrupción y saltar al RIP original.

4.2. Recuperación de Excepciones

Se ha detectado un error que no se ha podido corregir. Si se prueba el módulo de excepciones (el de división por cero, o el de operación de código inválido), lo que sucede es que al principio el base pointer se posiciona en una dirección específica. Si a continuación vuelve a correr otra excepción, el base pointer no se restaura y se corre pocas posiciones hacia arriba. Si una vez más se produce otra excepción, el base pointer queda fijo en el valor anterior, por lo que no crece y se empieza a restaurar desde dicho valor. Es un error que no pudimos corregir, y que utilizando métodos de debuggeo no pudimos evidenciar.

4.3. Snake

La principal dificultad que enfrenta el juego Snake radica en lograr una actualización suave y continua de la pantalla a medida que el jugador avanza. Actualmente, la impresión en la pantalla durante el juego no es lo suficientemente fluida, lo que puede afectar la experiencia del usuario y la jugabilidad. Para abordar este problema y mejorar la experiencia general del juego, se requiere una optimización en la lógica de actualización de la pantalla y la representación de los movimientos de la serpiente y la fruta.

5. Conclusión

En conclusión, en este trabajo práctico se ha logrado implementar un Kernel basado en una arquitectura de 64 bits (64x Barebones) que administra los recursos de la computadora y proporciona una API para que los usuarios puedan interactuar con el hardware sin una interacción directa.

Se han diferenciado dos espacios fundamentales: el Kernel space y el User space. En el Kernel se implementan los drivers, las interrupciones y las syscalls, mientras que en el Userland se desarrolla la consola de comandos que permite al usuario elegir entre diferentes módulos y funcionalidades.

Se ha implementado el manejo de interrupciones mediante el IRQ Dispatcher, que administra las interrupciones generadas por el teclado y el temporizador del procesador. También se han implementado las excepciones, como la excepción de división por cero y la excepción de código de operación inválido, que son detectadas y manejadas adecuadamente.

Además, se ha implementado el mecanismo de syscalls, que permite a los usuarios acceder a los servicios proporcionados por el kernel a través de interrupciones. Se ha definido un Syscall Dispatcher que redirige las llamadas de syscalls a las funciones de controlador correspondientes.

En cuanto a los drivers, se ha implementado el Keyboard Driver, que administra las interrupciones generadas por el teclado y almacena las teclas presionadas en un buffer circular. También se ha desarrollado el Timer Driver, que utiliza el reloj del procesador para contar los ciclos de reloj y llevar un seguimiento del tiempo transcurrido.

En resumen, con este trabajo práctico se ha demostrado que es posible implementar un sistema operativo básico utilizando barebones y ampliar sus funcionalidades mediante la administración de recursos, el manejo de interrupciones, excepciones y syscalls. Esta implementación proporciona una base sólida para el desarrollo de sistemas operativos más completos y permite a los usuarios interactuar con el hardware de manera más precisa y controlada.