Arquitectura de Computadoras

Trabajo Práctico Especial: Informe

Grupo: 03

Integrantes:

- Pablo Gorostiaga (65148)
- Carlos Amador Vallejo Tapia (65454)
- Tiago Heras (65627)

1. Índice

2. Resumen

El siguiente informe expone el diseño, decisiones y aspectos técnicos del trabajo práctico especial de la materia Arquitectura de Computadoras del grupo 03, integrado por Pablo Gorostiaga, Carlos Amador Vallejo Tapia, y Tiago Heras. En el mismo se discutirán y explicarán los beneficios y desventajas de las decisiones tomadas, con sus respectivas justificaciones. Además de toda la información relevante sobre el desarrollo del trabajo.

3. Introducción

El objetivo fundamental de este trabajo práctico consiste en desarrollar un núcleo del sistema operativo que permita comprender cómo se administran los recursos de hardware de un equipo informático. Este núcleo es iniciado mediante Pure64, una herramienta proporcionada por la cátedra.

La arquitectura del sistema se divide en dos ámbitos claramente definidos y aislados: el espacio del Kernel y el espacio de Userland. El espacio de Kernel engloba todos los aspectos relacionados con el funcionamiento interno del sistema operativo, siendo responsable de administrar los recursos del sistema y establecer comunicación con el hardware a través de controladores específicos. Simultáneamente, suministra las funciones esenciales al espacio del usuario, cuyo propósito es facilitar la interacción entre el sistema y el usuario final. Dentro de este último espacio se ejecuta el programa principal de la terminal de comandos, que pone a disposición del usuario todas las funcionalidades del sistema.

4. Kernel

A. Drivers

Los drivers consisten en los módulos encargados de comunicarse directamente con el hardware. Que mediante interrupciones, syscalls y en ocasiones especiales excepciones puedan desplegar información al usuario.

a. Video Driver

Este driver de video implementa un sistema de gestión gráfica para modo VBE (VESA BIOS Extensions) que permite renderizar contenido en pantalla. A continuación se explican sus funciones principales organizadas por categoría:

Funciones de Renderizado de Píxeles:

vd_put_pixel()

Esta función constituye el núcleo del sistema de renderizado. Recibe un color en formato hexadecimal y coordenadas (x, y) para dibujar un píxel individual en el framebuffer. Calcula el offset correspondiente en la memoria de video considerando los bits por píxel y el pitch de la pantalla, luego descompone el color RGB en sus componentes individuales para escribirlos en memoria.

vd_put_multpixel()

Extiende la funcionalidad anterior permitiendo dibujar bloques cuadrados de píxeles. Recibe los mismos parámetros que vd_put_pixel() más un factor multiplicador que determina el tamaño del bloque (mult x mult píxeles). Esta función es fundamental para implementar el sistema de zoom de caracteres.

Funciones de Renderizado de Texto:

draw_char()

Renderiza un carácter individual en la posición actual del cursor utilizando el bitmap de fuentes almacenado en font_bitmap[]. Utiliza una máscara de bits para determinar qué píxeles del carácter deben ser dibujados con el color del texto y cuáles con el color de fondo. Aplica el factor de zoom mediante vd_put_multpixel().

vd_put_char()

Función de alto nivel que maneja la impresión de caracteres con funcionalidades adicionales como salto de línea (\n), retroceso (\b), y gestión automática del cursor. Implementa wrap automático cuando el texto excede el ancho de pantalla y desplazamiento vertical cuando se alcanza el límite inferior.

vd_put_string()

Permite imprimir cadenas completas iterando sobre cada carácter y llamando a vd_put_char(). Mantiene la consistencia en el manejo de colores según el descriptor de archivo.

Funciones de Gestión de Pantalla:

vd_clear_screen()

Limpia completamente la pantalla estableciendo todos los píxeles al color de fondo. Si el fondo es negro, utiliza _memset() para mayor eficiencia; de lo contrario, emplea vd_draw_rectangle() para llenar la pantalla con el color especificado. Reinicia la posición del cursor al origen.

scroll_screen()

Implementa el desplazamiento vertical de la pantalla cuando el contenido excede la altura disponible. Mueve todo el contenido hacia arriba una cantidad de líneas equivalente a la altura de un carácter escalado, y limpia la parte inferior liberada.

vd_set_cursor()

Permite posicionar manualmente el cursor en coordenadas específicas de píxeles, útil para crear interfaces de usuario más complejas.

Funciones de Dibujo Geométrico:

vd_draw_rectangle()

Dibuja rectángulos sólidos de color especificado. Incluye verificaciones de límites para evitar dibujar fuera de los bordes de la pantalla y recorta automáticamente las dimensiones si es necesario. Utiliza un enfoque fila por fila para optimizar el rendimiento de caché.

vd_draw_circle()

Renderiza círculos sólidos utilizando la fórmula de distancia euclidiana para determinar qué píxeles están dentro del radio especificado. Incluye verificaciones de límites tanto para el círculo completo como para píxeles individuales.

Funciones de Configuración y Utilidad

vd_set_zoom()

Ajusta el factor de escala para el texto entre los valores mínimo y máximo permitidos (1-10). Afecta tanto el tamaño de los caracteres como el espaciado entre líneas.

vd_init()

Inicializa el driver estableciendo el cursor en el origen y limpiando la pantalla. Debe llamarse antes de utilizar otras funciones del driver.

vd_get_mode_info()

Proporciona información sobre la configuración actual del modo de video, incluyendo resolución y profundidad de color.

Funciones Especializadas:

vd_show_registers()

Función de depuración que muestra el contenido de todos los registros del procesador en formato hexadecimal, útil para el diagnóstico del sistema.

b. Sound Driver

Este driver implementa un controlador de sonido básico para generar tonos a través del altavoz interno de la computadora.

Funciones de sonido:

play_sound()

Reproduce un tono continuo a una frecuencia específica. Una vez que lo activas, el sonido sigue reproduciéndose hasta que lo detengas manualmente.

beep()

Genera un "beep" o pitido que dura un tiempo determinado y luego se detiene automáticamente una vez cumplido el tiempo especificado.

c. Keyboard Driver

Este driver de teclado implementa un sistema completo de captura y procesamiento de entrada del usuario a través del teclado. Maneja la conversión de scancodes del hardware a caracteres ASCII y gestiona el estado de las teclas.

Funciones de Captura y Procesamiento:

kbd_get_key()

Esta función constituye el núcleo del sistema de captura de teclado. Obtiene el scancode del hardware mediante get_key_asm() y lo procesa para determinar si es una tecla extendida (con prefijo 0xE0), si es un make code (tecla presionada) o break code (tecla soltada). Maneja teclas

especiales como Shift, Caps Lock y flechas direccionales, convierte los scancodes a caracteres ASCII usando las tablas de mapeo, y finalmente almacena los caracteres válidos en el buffer circular.

Funciones de Gestión de Buffer:

kbd_has_char()

Verifica si existen caracteres disponibles para leer en el buffer circular comparando los punteros head y tail. Retorna un valor booleano que indica la disponibilidad de datos.

kbd_get_char()

Extrae el siguiente carácter disponible del buffer circular. Actualiza el puntero tail para avanzar al siguiente elemento y retorna el carácter obtenido. Si no hay caracteres disponibles, retorna 0.

kbd_reset_buff()

Reinicia el buffer circular estableciendo head igual a tail, efectivamente descartando todos los caracteres pendientes de procesar.

Funciones de Estado de Teclas:

kbd_is_key_down()

Consulta el array key_state[] para determinar si una tecla específica está siendo presionada en el momento actual. Recibe un carácter como parámetro y retorna su estado de presión.

B. Syscalls

Mediante las syscalls, el user space es capaz de comunicarse y mandar comandos al kernel space. Su funcionamiento consiste en llamar la funcion sys_call(), donde se pasan los parámetros necesarios para la llamada, su máxima cantidad de argumentos es de seis (6), rax, rdi, rsi, rdx, r10, r8, en ese orden. Entre estos está el principal (rax) que indica a qué syscall se llamará. Posteriormente en Assembler, se reordenan los registros para que coincidan con el estándar de linux y se realiza la interrupción 80h. Para luego llegar a sysCallDispatcher, que según los argumentos enviados decide qué syscall ejecutar.

a. Read

Permite leer datos desde un descriptor de archivo. Cuando se especifica STDIN, implementa lectura desde teclado carácter por carácter hasta completar el buffer solicitado o encontrar un

salto de línea. Utiliza interrupciones habilitadas para esperar la entrada del usuario de manera eficiente.

b. Write

Escribe datos a un descriptor de archivo. Funciona únicamente con STDOUT y STDERR, redirigiendo la salida al driver de video para mostrar el texto en pantalla. Proporciona la funcionalidad básica de impresión de texto.

c. Registers

Muestra el contenido actual de los registros del procesador en pantalla, utilizada principalmente para depuración y diagnóstico del sistema.

d. Get time

Obtiene la hora actual del sistema y la muestra en pantalla, proporcionando acceso a información temporal.

e. Set Cursor

Posiciona el cursor de texto en coordenadas específicas (x, y) de la pantalla, permitiendo controlar dónde aparecerá el próximo texto.

f. Clear Screen

Limpia completamente la pantalla, borrando todo el contenido visible y preparando el display para nueva información.

g. Set Zoom

Modifica el nivel de zoom del texto y gráficos en pantalla, permitiendo ajustar el tamaño de visualización según las necesidades del usuario.

h. Exit

Termina la ejecución del sistema enviando una señal de apagado ACPI a QEMU, efectuando un shutdown controlado del entorno de ejecución.

i. Get Mode Info

Obtiene información detallada sobre el modo de video actual, incluyendo resolución, bits por píxel y otros parámetros técnicos de la pantalla.

j. Is Key Down

Consulta el estado actual de una tecla específica, determinando si está siendo presionada en tiempo real. Útil para aplicaciones interactivas y juegos.

k. Key Ready

Verifica si hay caracteres disponibles en el buffer del teclado sin consumirlos, permitiendo implementar entrada no bloqueante.

l. Put Pixel

Dibuja un píxel individual en coordenadas específicas con un color determinado, proporcionando el control gráfico más básico para aplicaciones.

m. Draw Rect

Dibuja un rectángulo definido por posición, dimensiones y color, complementando las herramientas de dibujo disponibles.

n. Draw Circle

Renderiza un círculo en pantalla especificando centro, radio y color, ofreciendo primitivas gráficas para aplicaciones más complejas.

o. *Reset KBD Buffer*

Limpia el buffer de entrada del teclado, descartando todos los caracteres pendientes de procesamiento.

p. *Beep*

Genera un tono audible con frecuencia y duración específicas utilizando el PC speaker, proporcionando capacidades básicas de sonido para alertas y efectos.

C. Interrupciones

El funcionamiento general de estas interrupciones es, como dice el nombre, interrumpir al procesador para que ejecute una rutina específica que está guardada en la IDT. Esta tabla, según la documentación, se ubica en la posición 0x0000000000000000 de memoria con los primeros 32 lugares reservados para excepciones y después vienen las interrupciones.

Cuando arranca el kernel, se configuran con idt_loader() donde se ponen los distintos punteros a función de las diferentes interrupciones que necesita el sistema. Entre estas están el timer tick, la interrupción de teclado, y la int 80 que maneja las syscalls. Las dos primeras son de hardware y la última de software.

Después en assembler, usando una macro y el número de interrupción, se pushea todo el estado al stack y se llama a irqDispatcher() que decide qué hacer. Luego se restaura el estado del sistema. Esto es solo para interrupciones de hardware. Si es una syscall se llama a syscallDispatcher() que maneja las syscalls.

El timer tick llama al time handler que se encarga de contar los ticks del sistema. En la interrupción del teclado se guarda la tecla presionada en un buffer.

D. Excepciones

Las interrupciones de software se activan ante eventos imprevistos en el sistema. Como se mencionó, sus rutinas de atención están ubicadas al inicio de la IDT. Este kernel cuenta con soporte para dos tipos de excepciones: división por cero y operación inválida.

Ambas excepciones operan de manera similar y se integran junto con las interrupciones. Avisan la excepción que sucedio por pantalla, e imprimen todos los registros, luego volviendo a la shell.

5. Userland

A. Librería Estándar

Se dispone de una librería con todas las funciones que podrían resultar útiles al usuario. La mayor parte de estas se apoyan en las syscalls para cumplir su propósito.

El módulo libc proporciona una serie de rutinas de uso común agrupadas en varias categorías. En primer lugar, las funciones de E/S de carácter incluyen putChar, que envía un único carácter al dispositivo de salida estándar; getChar, que bloquea la ejecución hasta leer un carácter desde la entrada; e isCharReady, que permite comprobar sin bloqueo si hay datos disponibles en el búfer de entrada. Para impresión de cadenas se dispone de putString, que recorre e imprime cada carácter de una cadena terminada en nulo.

En el ámbito de operaciones con texto, strlen calcula la longitud de una cadena sin contar el terminador nulo, strcmp compara dos cadenas completas y strncmp hace lo mismo limitándose a los primeros n caracteres. Por otro lado, las conversiones numéricas se cubren con atoi, que interpreta una cadena decimal como un entero en base 10, e itoa, que genera la representación textual de un valor entero en la base especificada.

La sincronización temporal se maneja mediante getTime, que obtiene el valor del reloj del sistema, y wait_next_tick, que detiene la CPU hasta el siguiente pulso del temporizador—ideal para sincronizar actualizaciones a una tasa cercana a 60 Hz. Para el control de la pantalla, clear_screen borra todo el contenido y sitúa el cursor en la esquina superior izquierda, set_cursor posiciona el cursor en la fila y columna indicadas, y set_zoom ajusta el nivel de ampliación del modo de texto o gráfico.

Con fines de diagnóstico, showRegisters solicita la impresión del estado completo de los registros de la CPU. La generación de sonido incluye exception_sound, que emite una breve sucesión de tonos cuando ocurre una excepción, y system_start_sound, que reproduce una pequeña melodía al arrancar el sistema.

Finalmente, hay un par de funciones muy específicas para entornos de bajo nivel: hltUntil_c suspende la ejecución usando la instrucción HLT hasta que el usuario presiona la tecla ‘c’, y get_mode_info recupera los parámetros del modo de vídeo activo—como resolución y profundidad de color—almacenándolos en la estructura proporcionada.

B. Shell

La shell de este sistema se articula en torno a dos rutinas principales, shell_init y shell, declaradas en el archivo de cabecera. Al invocar shell_init, se preparan los recursos de entorno—configuración del modo de vídeo, limpieza inicial de pantalla y presentación de un mensaje de bienvenida—para a continuación entrar en la función shell, que implementa el bucle interactivo de lectura y ejecución de órdenes.

Dentro de ese bucle, la shell muestra un indicador (prompt) y captura la línea de texto que escribe el usuario utilizando las rutinas de E/S básicas de libc. A partir de la línea completa, el analizador interno separa el nombre del comando de sus argumentos, realiza una búsqueda en la tabla de funciones disponibles y, si encuentra una coincidencia, invoca la función asociada; en caso contrario, informa de que la instrucción no existe. Una vez ejecutado el comando la shell retorna al prompt para esperar una nueva entrada.

El conjunto de órdenes integradas está definido en el archivo que agrupa los comandos de la shell. Entre ellas se incluyen:

- cmd_help: que despliega la lista de comandos;
- cmd_time: que muestra la fecha y hora del sistema;
- cmd_registers: que vuelca el estado de los registros de CPU;
- pongis_init: que arranca el juego de Pongis-Golf;
- cmd_div0 e cmd_invalid_opcode: diseñados para provocar excepciones de división por cero u operación inválida con fines de prueba;
- cmd_clear: que limpia la terminal;
- cmd_zoom: que ajusta el nivel de ampliación del texto;
- cmd_exit: que finaliza la ejecución de la shell.

Gracias a esta arquitectura, la shell actúa como capa de abstracción entre el usuario y las llamadas al sistema de bajo nivel, ofreciendo un entorno interactivo versátil y extensible sin necesidad de entrar en los detalles internos de cada llamada o de la implementación de bajo nivel.

6. Pongis-Golf

El juego Pongis-Golf consiste en controlar a un personaje circular para intentar introducir una pelota en un hoyo.

A. Diseño elegido y justificación

Para el desarrollo del juego se optó por un diseño minimalista, tanto en la estética visual como en la jugabilidad. La idea principal era demostrar un juego funcional sobre un entorno desde cero

sin distraerse en complicaciones innecesarias, priorizando la simplicidad y legibilidad del código en un entorno de bajo nivel.

a. Jugabilidad

Se tomó una jugabilidad sencilla en la que se eliminó por completo la dificultad de tener un número de toques máximo, ya que se consideró tediosa y no divertida. El enfoque está en la navegación de los distintos niveles y el contador de toques sirve sólo como indicador.
En cuanto al multijugador, se eligió repetir el modo de juego de un jugador con la diferencia de que el último jugador que toca la pelota antes de que entre es el que gana.

b. Gráficos

Se evitaron los detalles y se redujo el código mediante el uso de círculos tanto para los jugadores, pelota, hoyo y obstáculos. La pantalla fue renderizada mediante a las syscall *sys_draw_rect* y *sys_draw_circle*.

c. Controles

Los controles fueron cambiados del clásico control de los juegos Pongis, en los que se controla la dirección en la que apunta el jugador con las flechas laterales y se le da una velocidad hacia la dirección donde está apuntando el personaje con la flecha de arriba. Nosotros implementamos controles básicos en los que las teclas de movimiento mueven al personaje en las 4 direcciones principales (arriba, abajo, derecha, izquierda) y sus direcciones compuestas (diagonales). Esto brinda una movilidad precisa y directa, facilitando la navegación por el campo de juego. Se tomó este esquema tradicional por su simplicidad, familiaridad para el usuario y facilidad de implementación.

B. Pros y contras

La decisión del enfoque simplificado trae diversos pros y contras a la mesa:

<u>Pros:</u>

- **Simplicidad y claridad visual:** El uso de círculos de colores permite representar los elementos del juego de forma clara y efectiva, sin necesidad de gráficos complejos.

- **Controles intuitivos:** El esquema clásico de movimiento en 8 direcciones es familiar para el usuario y fácil de implementar.

- **Autonomía completa:** El juego corre directamente sobre el sistema operativo desarrollado, sin librerías externas ni dependencias, demostrando control total del

entorno.

- **Código accesible:** La lógica minimalista favorece la legibilidad y el análisis técnico, útil para fines de depuración y revisión.

Contras:

- **Limitaciones gráficas:** La falta de primitivas avanzadas (como líneas) limita la expresividad visual y la variedad estética.

- **Sin persistencia:** No se implementaron mecanismos para guardar progreso y esto complica el completar el juego en su totalidad ya que debe terminarse en una sola sesión.

- **Sin posibilidad de perder:** No hay un máximo de toques antes de tener que intentar de nuevo, por lo que se pierde una parte de la experiencia y la dificultad del juego.

- **Falta de feedback visual o sonoro:** Al no contar con efectos visuales avanzados ni sonido, la experiencia puede sentirse básica o incompleta para el usuario final.

7. Conclusión:

Para concluir, estamos contentos con el desarrollo del proyecto y como está resuelto. Sabemos que hay funcionalidades que pueden ser mejoradas, o implementadas de otra manera. Sin embargo, considerando las prioridades del proyecto y las limitaciones en respecto al tiempo, decidimos conformarnos.

Aprendimos muchos aspectos acerca de cómo implementar un kernel, con sus respectivos drivers, syscalls, interrupciones y excepciones. Así como la implementación de un intérprete de comandos el cual se comunica con el kernel, y además un juego. Todo esto mientras se mantuvo correctamente la separación entre el kernel y user space.