Arquitectura de Computadoras

Trabajo Práctico Especial

Informe

Grupo 6

Integrantes:

- Tadeo Gorganchian (65507)
- Maria del Pilar Resek (65528)
- Eduardo Tormakh (65155)

Fecha de entrega: 5 de Noviembre de 2025

Índice:

Objetivo:

El propósito central de este trabajo fue desarrollar un Kernel booteable (utilizando el software-loader "Pure64") con la capacidad de gestionar los recursos de hardware de una computadora. Asimismo, se buscó proveer a los usuarios una API que les permitiera interactuar con dichos recursos desde el User Space.

Para alcanzar esto, fue necesario estructurar la memoria en dos segmentos bien diferenciados: uno reservado para el kernel (Kernel Space) y otro para las aplicaciones de usuario (User Space). El Kernel Space se encarga de la comunicación directa con el hardware a través de controladores o drivers para periféricos como el teclado y la pantalla. Por su parte, el User Space accede a estos recursos mediante llamadas al sistema (system calls) proporcionadas por el kernel. Se desarrolló además una biblioteca que emula funcionalidades de la biblioteca estándar de C en Linux, incorporando funciones análogas.

El proyecto también incluye un intérprete de comandos o Shell, que ofrece diversas utilidades, tales como:

- Una función de ayuda.
- Comandos para probar excepciones (división por cero y código de operación inválido).
- Visualización de la hora y fecha del sistema.
- Consulta de los valores de los registros en cualquier momento.
- Comando para la limpieza de la pantalla.
- Ejecución del juego "Tron".
- Modificación del tamaño de fuente en pantalla.
- Comando con reproducción de música.
- Comandos de benchmarking.

Separación User Space - Kernel Space

La división entre el User Space y el Kernel Space constituye un principio fundamental en el diseño del sistema. El kernel actúa como un administrador privilegiado que centraliza y controla exclusivamente el acceso a los recursos del hardware, implementando así una capa de protección y abstracción.

Para interactuar con estos recursos, el kernel proporciona una API compuesta por Llamadas al Sistema (System Calls). Estas funciones, que residen y se ejecutan en el espacio protegido del kernel (específicamente en la Tabla de Descriptores de Interrupciones, IDT),

son invocadas por las aplicaciones de usuario mediante la instrucción de software INT 80h. Cada llamada al sistema se identifica mediante un "File Descriptor" que especifica la operación solicitada.

Sobre esta base, se desarrollaron bibliotecas en el espacio de usuario que simplifican y agilizan el acceso a estas funcionalidades. Una de ellas emula el comportamiento de la biblioteca estándar de C de Linux, ofreciendo a los programadores funciones familiares como printf, scanf, putChar, entre otras, las cuales internamente realizan las llamadas al sistema necesarias.

System Calls

Se desarrollaron las System Calls que se presentan en la siguiente tabla:

id	nombre	RBX	RCX	RDX
0	sys_regs	char* buf		
1	sys_time	uint8_t* buf		
2	sys_date	uint8_t* buf		
3	sys_read	FileDescriptor fd	char *buf	uint64_t count
4	sys_write	FileDescriptor fd	const char *buf	uint64_t count
5	sys_increase_fontsize			
6	sys_decrease_fontsize			
7	sys_beep	uint32_t freq	uint64_t duration	
8	sys_screen_size	uint32_t *width	uint32_t *height	
9	sys_circle	uint64_t fill	uint64_t *info	uint32_t color
10	sys_rectangle	uint64_t fill	uint64_t *info	uint32_t color
11	sys_line	uint64_t *info	uint32_t color	
12	sys_draw_string	const char *s	uint64_t *info	uint32_t color
13	sys_clear			
14	sys_speaker_start	uint32_t freq_hz		
15	sys_speaker_stop			
16	sys_enable_textmode			
17	sys_disable_textmode			

18	sys_put_pixel	uint32_t hexColor	uint64_t x	uint64_t y
19	sys_key_status	char key		
20	sys_sleep	uint64_t milliseconds		
21	sys_clear_input_buffer			
22	sys_ticks			

Descripción breve de cada una de las System Calls:

- sys_regs: Obtiene los registros actuales del sistema.
- sys_time: Obtiene la hora actual .
- sys_date Obtiene la fecha actual.
- sys_read: Lee datos desde un FileDescriptor (STDERR o STDOUT).
- sys_write: Escribe datos en un FileDescriptor.
- sys_increase_fontsize: Aumenta el tamaño de la fuente en pantalla.
- sys_decrease_fontsize: Disminuye el tamaño de la fuente en pantalla.
- sys_beep: Genera un sonido con la frecuencia y duración especificadas.
- sys_screensize: Obtiene el alto y ancho de la pantalla.
- sys_circle: Dibuja un círculo en pantalla.
- sys_rectangle: Dibuja un rectángulo en pantalla.
- sys_line: Dibuja una línea en pantalla.
- sys_draw_string: Dibuja una cadena de texto en pantalla.
- sys_clear: Limpia la pantalla.
- sys_speaker_start: Inicia el altavoz con una frecuencia específica.
- sys_speaker_stop: Detiene el altavoz.
- sys_enable_textmode: Activa el modo texto.
- sys_disable_textmode: Activa el modo gráfico.
- sys_put_pixel: Dibuja un pixel en pantalla con color específico.
- sys_key_status: Obtiene el estado de una tecla (presionada o no).
- sys_sleep: Detiene la ejecución por una cantidad de milisegundos.
- sys_clear_input_buffer: Limpia el buffer de entrada del teclado.
- sys_ticks: Obtiene la cantidad de ticks desde el inicio del sistema.

Manejo de Interrupciones:

Para el manejo de interrupciones, se configuraron el Timer Tick y la Keyboard Interrupt, asignándolas a las entradas 0x20 y 0x21 de la IDT, respectivamente. Cada una tiene su rutina de atención de interrupción asociada, definida en el código. Además, se estableció la máscara del PIC como 0xFC, lo que permite únicamente estas dos interrupciones mientras las demás permanecen deshabilitadas.

En el caso de la interrupción del teclado, se implementó el Keyboard Driver, que se encarga de capturar las teclas presionadas. Las funcionalidades proporcionadas por este driver son utilizadas activamente por los diferentes módulos del sistema operativo.

Drivers

Driver de Video

El driver de video renderiza texto en modo grafico usando un mapa de bits por caracter. Las funciones públicas (vdPrint, vdPutChar) gestionan la ubicación del cursor y caracteres especiales (nueva línea y borrado). Cada carácter se pinta leyendo su bitmap y, por cada bit activo, se dibuja un bloque de pixeles en el framebuffer mediante putPixel. Se partió del driver de video proporcionado por la cátedra y se utilizó la fuente https://github.com/hubenchang0515/font8x16/blob/master/font8x16.h para acceder a la definición de la fuente bitmap utilizada al renderizar texto en pantalla. En la fuente se declara un arreglo bidimensional, de la forma const uint8_t font[128][FONT_HEIGHT], donde cada fila representa un carácter ASCII y cada elemento de la fila es un byte que codifica una línea horizontal del carácter.

El driver de video verifica que los píxeles a dibujar estén dentro de los límites de la pantalla, evitando errores fuera del área visible. Cuando el texto llega al final, realiza scroll desplazando el contenido hacia arriba con memcpy.

Además, permite escalar el tamaño de la fuente: cada bit del carácter se dibuja como un bloque de varios píxeles según el parámetro de tamaño, facilitando así la visualización en distintas resoluciones.

Driver de Teclado

El driver de teclado gestiona la entrada del usuario mediante interrupciones de hardware. Responde a IRQ1, mapeada en la entrada 0x21 de la IDT. Configuramos la máscara del PIC Master como 0xFC para habilitar únicamente el timer tick (IRQ0) y el teclado (IRQ1).

Cuando el usuario presiona o suelta una tecla, el hardware genera IRQ1. El handler _irq01Handler (en assembly) lee el scancode desde el puerto 0x60 y lo guarda en la variable global pressed_key.

Al presionar Left Ctrl (scancode 0x1D), capturamos un snapshot de todos los registros desde el stack (guardados por pushState). Esta información queda disponible mediante sys_regs.

Luego, irqDispatcher ejecuta handlePressedKey() en C, que procesa el scancode:

- Teclas modificadoras: detecta Left/Right Shift (0x2A/0x36) para activar/desactivar mayúsculas, y Caps Lock (0x3A) que alterna su estado.
- Scancode a ASCII: dos tablas estáticas (lowerKeys y upperKeys) mapean cada scancode a su carácter. La elección entre tablas se hace con XOR entre Shift y Caps Lock.
- Buffer circular: las teclas válidas se almacenan como ASCII en un buffer circular de 1024 bytes, controlado por buffer_start, buffer_end y buffer_current_size. Si el buffer se llena, sobrescribe los caracteres más antiguos.

Las funciones públicas incluyen: getCharFromBuffer() que extrae un carácter (retorna -1 si está vacío), read_keyboard_buffer() que copia hasta count caracteres, y clear_buffer() que vacía el buffer. Esto se expone al User Space mediante sys_read.

Además, isPressedKey() consulta pressedKeys para verificar si una letra está presionada en tiempo real, usado principalmente por aplicaciones como Tron.

Driver de Sonido

El altavoz de una computadora tiene dos estados posibles: in y out, que cambian con una cierta frecuencia para generar un sonido audible. Para implementar el driver de sonido, se manipuló el estado del altavoz utilizando el Programmable Interval Timer (PIT), que permite controlar la frecuencia de cambio de estos estados.

El canal 2 del PIT se conecta al puerto del altavoz (0x61) para modificar su estado en cada tick. Esto se logra configurando los bits correspondientes del puerto 0x61 y ajustando la frecuencia del PIT mediante los puertos 0x42 y 0x43. El divisor del PIT, calculado como *PIT_BASE_HZ / frecuencia_deseada*, determina la velocidad de los ticks y, por ende, la frecuencia del sonido emitido.

Además, se tuvo en cuenta que el altavoz necesita al menos 60 microsegundos para cambiar de estado. Si los ticks ocurren más rápido que este intervalo, el altavoz no tendrá tiempo suficiente para emitir el sonido correctamente.

Dado que el proyecto se ejecuta en QEMU en lugar de hardware real, se utilizó el comando: -audiodev pa,id=snd0 -machine pcspk-audiodev=snd0

Dependiendo del sistema operativo y el tipo de arquitectura de sonido se puede cambiar el comando *pa* (PulseAudio) por el correspondiente.

Esto simula la tarjeta de sonido y permite que el altavoz funcione correctamente. La implementación se basó en la documentación de osDev (https://wiki.osdev.org/PC_Speaker#How_It_Works).

Excepciones

El sistema implementa un mecanismo para gestionar excepciones críticas como división por cero y código de operación inválido. Estas excepciones están registradas en la IDT (Interrupt Descriptor Table) con sus rutinas de atención correspondientes.

Al ocurrir una excepción, el sistema ejecuta un dispatcher que identifica el tipo de error y activa el manejador específico. Cada manejador realiza tres acciones principales:

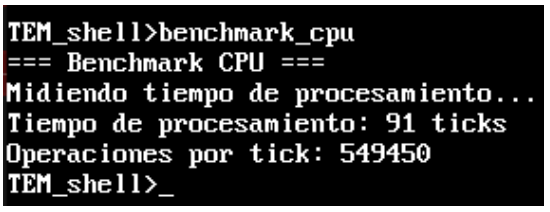
- Imprime el estado actual de todos los registros de la CPU en el momento del error
- Muestra un mensaje de error descriptivo en pantalla
- Solicita al usuario que presione Enter para continuar

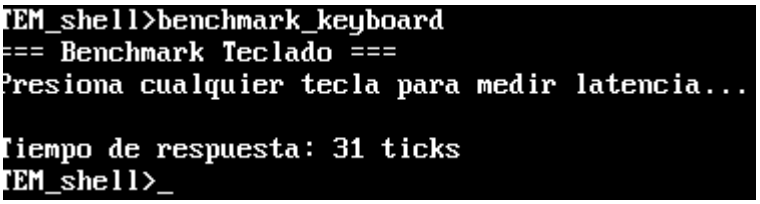
Esta implementación permite una respuesta controlada ante fallos, facilitando la depuración al proporcionar información completa del estado del sistema en el momento de la excepción, mientras mantiene la estabilidad general del entorno.

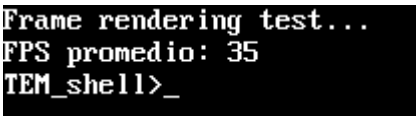
Benchmarks

En cuanto a benchmarks, implementamos 4 comandos. *benchmark_fps que* simula el renderizado de un frame para poder calcular los FPS promedio. Luego, *benchmark_cpu* que realiza operaciones enteras y de punto flotante (en torno a 50 millones de operaciones) y calcula el tiempo de procesamiento y operaciones por tick. A su vez, *benchmark_memory* mide la velocidad de acceso a memoria realizando operaciones de lectura y escritura y calcula el tiempo de acceso y operaciones por tick nuevamente. Finalmente, *benchmark_keyboard* calcula la latencia del teclado en ticks.

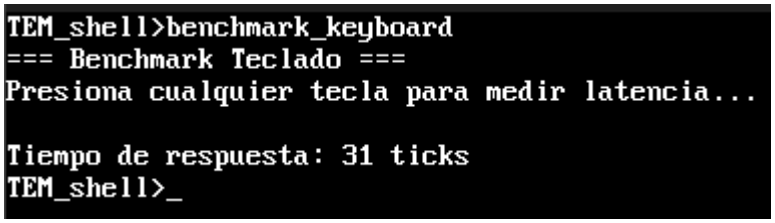
Estás funcionalidades fueron testeadas tanto en QEMU como en VirtualBox (no logramos testearlas en Hardware físico).
En QEMU se obtuvieron los siguientes resultados:







Mientras que en VirtualBox se obtuvo:



Ante estos resultados, queda claro que hay una gran diferencia en lo que respecta a FPS, rendimiento y tiempos de procesamiento, siendo QEMU ampliamente superior a VirtualBox.

Shell

La shell del TP está diseñada como un programa interactivo que permite al usuario ejecutar comandos predefinidos. Su funcionamiento se basa en un loop infinito que espera la entrada del usuario, procesa el comando ingresado y ejecuta la acción correspondiente.

La entrada del usuario se captura mediante una función que lee caracteres hasta detectar un ”\n”. Luego, la shell lo compara con la lista de comandos predefinidos y en caso de coincidir con uno de los disponibles se ejecuta la función asociada. Si no coincide, se le informa al usuario la existencia del comando “help” para visualizar todos los comandos disponibles.

Juego - Tron

Para el juego Tron se implementó una matriz de estado del tamaño del tablero (80x50 celdas) que guarda en cada posición si está ocupada y por qué jugador. Se definió un tamaño de celda fijo (CELL_SIZE = 10 píxeles), dibujando cada celda como un cuadrado mediante las syscalls gráficas disponibles.

Se implementaron dos modos de juego. En singleplayer hay 3 niveles. Al llegar a 100 puntos la velocidad se duplica, y al alcanzar 200 se triplica. Los niveles 2 y 3 incluyen obstáculos naranjas que eliminan al jugador al contacto. El nivel 3 tiene una zona verde que aumenta la velocidad mientras el jugador permanece dentro. En multiplayer se usa un sistema de puntos por rondas ganadas hasta que uno de los jugadores llegue a los 3 puntos.

Los controles se manejan con WASD para jugador 1, IJKL para jugador 2. El dibujado se realiza mediante redibujado incremental usando sys_rectangle.
El tamaño del tablero y las celdas se puede ajustar mediante los defines en config.h (BOARD_WIDTH, BOARD_HEIGHT y CELL_SIZE).

Funcionalidades extra

Por fuera de lo demandado por la cátedra, decidimos agregar un comando *“play_song”,* el cual reproduce la canción del Club Atlético Boca Juniors “Y Dale Dale Dale Dale Boca” (basada originalmente en “*Moliendo Café”* de Hugo Blanco). También, decidimos que los comandos *“print_time”* y *“print_date*” desplieguen la hora y fecha en GMT-3 (zona horaria argentina) y no en GMT-0 como devuelven las syscalls originalmente.