Informe Trabajo Práctico Especial

Asignatura: Arquitectura de Computadoras

Autores:

- Arias, Mateo Agustín 65613
- Branda, Franco Nicolás 65506
- García Puente, Manuel 65505

Grupo: 15

Año: 2025

Docentes a cargo:

- Valles, Santiago Raul
- Guagliano, Celeste Gabriela
- Ramos, Federico Gabriel

Fecha de entrega: 11/06

1. Introducción

En este informe se detalla el diseño e implementación de un sistema operativo de bajo nivel, desarrollado en el marco del Trabajo Práctico Especial (TPE) de la materia Arquitectura de Computadoras del ITBA. El proyecto tuvo como objetivo principal la implementación de un kernel booteable mediante Pure64, capaz de administrar los recursos de hardware de una computadora operando en modo protegido de 64 bits, con una separación estricta entre el espacio de usuario y el espacio de kernel. Se provee una API que permite a las aplicaciones del espacio de usuario acceder a los recursos del sistema sin realizar acceso directo al hardware. Como interfaz principal, se desarrolló una shell interactiva que permite la ejecución de diversos comandos para acceder a funcionalidades del sistema, entre las cuales se destaca Pongis Golf, demostrando la integración de gráficos, entrada por teclado y sonido.

2. Entorno de trabajo

Para el desarrollo, se utilizó Docker con la imagen provista por la cátedra (agodio/itba-so:1.0) y se empleó QEMU como emulador para probar el sistema operativo implementado. La compilación fue gestionada mediante un Makefile y se llevó a cabo en un entorno Linux (WSL 2 en sistemas Windows).

3. Estructura General del Sistema (Kernel - UserSpace)

El Kernel Space se encarga de interactuar directamente con el hardware mediante controladores específicos para video, teclado y sonido. Además, provee un conjunto de llamadas al sistema (syscalls) accesibles desde el User Space a través de la interrupción 0x80. En el User Space se incluyen programas como una shell interactiva y un juego, los cuales utilizan la API del kernel para acceder a los servicios del sistema.

3.1 Keyboard Driver

La interrupción 0x01 es interceptada al detectar la pulsación de una tecla, invocando la rutina keyboard_handler. Esta se encarga de leer el scancode desde el puerto correspondiente, traducirlo a su código ASCII en caso de tratarse de una tecla imprimible, Enter o Shift, y almacenar el resultado en un buffer circular de 64 posiciones. El acceso a la entrada de teclado se realiza a través de la función nextKey(), que retorna el siguiente carácter disponible, -2 si el buffer está vacío o 27 si se trata de la tecla ESC.

Aclaración: El controlador implementado en el kernel devuelve valores constantes para representar eventos particulares, como KEY_ESC (27) y NOT_KEY (-2), correspondientes respectivamente a la detección de la tecla ESC y a la ausencia de entrada. Al tratarse de valores fijos e independientes del estado del sistema, se optó por no exponerlos mediante una syscall. En su lugar, su interpretación se documenta de forma explícita en la documentación técnica, permitiendo que los programas en User Space comprendan su

significado sin requerir una interacción adicional con el kernel.los desarrolladores en User Space interpretar correctamente los valores devueltos por el driver sin necesidad de interacción adicional con el kernel.

3.2 Modo Gráfico y Video Driver

El sistema opera en modo gráfico para la visualización de la interfaz. El controlador de video permite renderizar texto y gráficos mediante funciones como putPixel. Se incorporaron funcionalidades adicionales, tales como ajuste de zoom en el texto de la shell (modificando el tamaño de los caracteres) y configuración de colores para texto y fondo, permitiendo personalizar la interfaz.

La interfaz gráfica simula una consola visual y es utilizada tanto por la shell como por el juego. Para optimizar el rendimiento visual y evitar parpadeos, se implementó doble buffering: el contenido se escribe primero en un buffer secundario en memoria, y posteriormente se copia al framebuffer visible mediante la syscall swapBuffers. Esta técnica permite una actualización más fluida de la pantalla, especialmente útil en la ejecución de Pongis Golf.

3.3 Sound Driver:

El controlador de sonido proporciona una interfaz sencilla para la reproducción de sonidos. Para emitir un tono, se realiza una única llamada especificando la frecuencia deseada y la duración en milisegundos. Dado que solo se trabaja con ondas cuadradas, no se incluye control de volumen.

4. Syscalls Implementadas

El kernel expone una serie de llamadas al sistema (system calls) inspiradas en la API de Linux, accesibles mediante la instrucción int 80h. Estas llamadas permiten que los programas en User Space interactúen de forma segura con el hardware a través del kernel. A continuación, se describen las más relevantes:

- **read(fd, buf, count)**: Permite leer caracteres desde la entrada estándar (STDIN). Usa un buffer y una cantidad máxima de caracteres.
- **write(fd, buf, count)**: Escribe texto en pantalla. Se debe proveer de un buffer y su longitud.
- **getDate()**: Obtiene la fecha y hora actual del sistema.
- **showRegisters()**: Función que muestra el estado actual de los registros del CPU.
- **setCursor(x, y)**: Posiciona el cursor para futuras escrituras gráficas.

- **setFontColor(color)**: Cambia el color de la fuente del texto.
- **setBackgroundFontColor(color)**: Cambia el color de fondo del texto.
- **setZoom(zoom)**: Define el nivel de zoom con el que se dibuja la fuente.
- **cleanScreen()**: Limpia completamente la pantalla.
- **putPixel(color, x, y)**: Dibuja un píxel en una posición específica.
- **sleep(microseconds)**: Suspende la ejecución durante una cantidad de microsegundos, utilizando internamente la instrucción hlt() para optimizar el uso del procesador.
- **playTone(frequency, milliseconds)**: Emite un tono a la frecuencia indicada durante una cantidad determinada de milisegundos.
- **swapBuffers()**: Intercambia el búfer actual con uno secundario previamente preparado. Esta syscall es fundamental para implementar correctamente el **double buffering**, técnica utilizada para evitar parpadeos y mejorar la fluidez del renderizado gráfico en pantalla.
- **isKeyPressed(keyCode)**: Recibe un código ASCII entre 0 y 127 y devuelve 1 si la tecla correspondiente está actualmente presionada, o 0 si no lo está.
- **getVbeInfo(struct neededInfo *)**: Provee información sobre el modo gráfico actual, útil para adaptarse a diferentes resoluciones y profundidades de color.

5. Manejo de Interrupciones y Excepciones

El sistema operativo implementado contempla el manejo de interrupciones y excepciones, elementos fundamentales para asegurar una interacción adecuada con el hardware y una respuesta controlada ante eventos imprevistos. La configuración de la Interrupt Descriptor Table (IDT) se lleva a cabo durante la fase de inicialización del kernel, cargando en ella los punteros a las rutinas correspondientes tanto para interrupciones de hardware como para excepciones del procesador.

Entre las interrupciones gestionadas se encuentran aquellas generadas por el teclado y el temporizador (timer). Para el teclado, se configuró una rutina específica que lee el scancode desde el puerto de entrada, lo traduce a su correspondiente carácter ASCII y lo almacena en un buffer circular. En cuanto a la interrupción del temporizador, aunque no se utiliza de forma intensiva en esta versión del sistema, fue configurada para permitir su uso futuro en tareas como control de tiempos o animaciones.

Durante la implementación del juego y la activación del modo multijugador, se detectó una baja sensibilidad en la detección de teclas, lo que afectaba negativamente la jugabilidad. Para mejorar este aspecto, se incorporó un buffer adicional de 128 posiciones que mantiene el estado (presionado o liberado) de cada tecla. A través de una syscall asociada, se logró una mejora significativa en la experiencia de juego, tanto en el modo de un jugador como en el de dos jugadores.

En lo referente al manejo de excepciones, se implementó el tratamiento explícito de dos tipos: división por cero e instrucción inválida. Estas excepciones son capturadas mediante entradas específicas en la IDT y redirigen la ejecución hacia rutinas que muestran el estado completo de los registros en el momento de la excepción. El sistema está diseñado para recuperarse automáticamente de dichos eventos, retornando al prompt de la shell y permitiendo al usuario continuar con el uso normal del sistema sin necesidad de reiniciarlo.

6. Pongis Golf

Con el objetivo de demostrar las capacidades del sistema, se implementó un juego denominado Pongis Golf. Este permite la participación de uno o dos jugadores simultáneamente, cada uno con un conjunto específico de teclas asignadas para movimiento y rotación. El objetivo consiste en introducir una bola en un hoyo, mientras se mantiene un marcador visible en pantalla. Se incorporaron efectos sonoros al lograr aciertos, y los gráficos, si bien simples, resultan funcionales mediante el uso de sprites y detección de colisiones. El funcionamiento del juego se encuentra documentado en detalle en el manual técnico.

7. Funcionalidades agregadas

Algunas de las funcionalidades agregadas son las ya mencionadas previamente como el **double buffer** que se implementó para evitar el **flickering** y mejorar la experiencia de usuario, o la implementación de la **syscall** del teclado para evitar que se llene el buffer y mejorar la jugabilidad, entre otras.

También se agregaron algunas funcionalidades a la consola, como la función “**color**” para cambiar el color del texto que se muestra en pantalla, pudiendo elegir entre 26 colores diferentes, la función “**echo**” que replica el mensaje enviado y la función “**clear**” que limpia la consola, indispensable para el correcto funcionamiento de la shell y del juego.

8. Posibles mejoras

Algunas de las mejoras que se podrían incorporar en un futuro para mejorar la experiencia del usuario podrían ser:

- Nuevo modo de juego Versus: incorpora un modo en el que un jugador debe intentar meter la pelota en el hoyo, mientras que el otro debe impedirlo durante un tiempo determinado.

- Optimización de dibujo mediante syscalls: utilizar llamadas al sistema que permite dibujar figuras completas en lugar de hacerlo píxel por píxel. Esto reduciría la cantidad de syscalls hechas por dibujo a una única, mejorando así el rendimiento.

- Ajuste del tamaño del búfer secundario: igualar su tamaño al del búfer principal en lugar de utilizar un valor predefinido excesivo, optimizando así el uso de memoria.

9. Conclusiones

El desarrollo del presente trabajo práctico permitió abordar en profundidad conceptos fundamentales de la materia, tales como la configuración del modo protegido de 64 bits, el manejo de interrupciones, la interacción con el hardware mediante controladores, y el diseño de interfaces de usuario de bajo nivel.

Uno de los principales desafíos consistió en establecer una separación estricta entre el espacio de usuario y el espacio de kernel, asegurando que toda interacción con el hardware se realizará exclusivamente mediante *syscalls* y de forma controlada.

Como resultado, el sistema operativo desarrollado satisface todos los requisitos funcionales definidos por la cátedra, incluyendo el manejo de excepciones, la implementación de *syscalls*, una shell interactiva y la ejecución de un programa gráfico en tiempo real. Se identifican posibles líneas de trabajo futuro orientadas a la extensión y mejora del sistema.

10. Fuentes:

Tablas y funciones que simulan la función seno y coseno:
FPGA Sine Lookup Table - Project F

Create Lookup Tables for a Sine Function - MATLAB & Simulink

Can someone please explain the Q1.15 numbering format : r/FPGA

Double buffering:

Double Buffering - OSDev Wiki

Multiple buffering - Wikipedia

Double Buffer · Sequencing Patterns · Game Programming Patterns

Sound driver:

PC Speaker and Timing Functions | Cosmodoc

Beyond Beep-Boop: Mastering the PC Speaker | Bumbershoot Software