Reporte técnico: Proyecto final de Sistemas Operativos y Laboratorio

1. Información del Proyecto

- **Título del Proyecto:** Desarrollo de un Módulo de Kernel para Control de Hardware (GPIO) en Raspberry Pi
- **Curso/Materia:** Sistemas Operativos
- **Integrantes:** John Edison Zapata Ramirez (john.zapata1@udea.edu.co)
- **Fecha de Entrega:** julio de 2025

2. Introducción

2.1. Objetivo del Proyecto

El objetivo general del proyecto es desarrollar un sistema modular en espacio de kernel que simule el funcionamiento de un sensor de temperatura (tipo DHT22) y controle una matriz de LEDs 4x4, utilizando una capa de emulación de GPIOs. Todo esto se realiza sin hardware físico, permitiendo la ejecución y prueba del sistema en una máquina virtual o sistema Linux estándar.

Se busca que el sistema reaccione de forma autónoma y eficiente a los valores de temperatura generados, mostrando visualmente el resultado a través de una matriz de LEDs. El enfoque se centra en la interacción directa entre módulos del kernel y en la simulación realista de controladores embebidos usando conceptos clave de los sistemas operativos.

2.2. Motivación y Justificación

Explique Actualmente, los sistemas embebidos y dispositivos de bajo consumo energético son esenciales para aplicaciones modernas como la domótica, el monitoreo ambiental, la automatización industrial y el Internet de las Cosas (IoT). Estos sistemas dependen fuertemente de la comunicación precisa y confiable con sensores y actuadores externos, normalmente a través de pines GPIO.

El sensor DHT22 es ampliamente utilizado por su facilidad de uso y bajo costo, pero en entornos reales, su manejo desde el espacio de usuario (por ejemplo, usando bibliotecas como RPi.GPIO) puede implicar latencias impredecibles, jitter elevado, y limitaciones en tiempo real, factores críticos en aplicaciones donde la precisión de los datos es esencial.

Este proyecto propone una solución más robusta y confiable mediante la emulación de un controlador en espacio de kernel, que interactúe con periféricos simulados a través de GPIOs

virtuales. Aunque no se accede a hardware físico real, la arquitectura diseñada imita fielmente la lógica y estructura de un sistema embebido basado en Raspberry Pi, permitiendo validar conceptos importantes como:

- Determinismo temporal y reactividad.
- Acceso directo a pines GPIO.
- Comunicación entre módulos kernel.
- Simulación de periféricos con comportamientos realistas.

Esta estrategia no solo evita la dependencia de hardware específico, sino que además profundiza en el entendimiento de las capas internas del sistema operativo, especialmente en lo relacionado con la manipulación de recursos del kernel, el modularidad, el manejo de hilos y la integración entre subsistemas.

Desde una perspectiva educativa, el proyecto se alinea perfectamente con los objetivos del curso de Sistemas Operativos, ya que permite:

- Trabajar directamente en espacio de kernel, utilizando lenguaje C.
- Comprender el flujo de lectura/escritura de periféricos simulados.
- Implementar y administrar tareas concurrentes usando kthread y workqueue.
- Simular la comunicación hardware/software por medio de intercambio de funciones y estructuras entre módulos del kernel.

2.3. Alcance del Proyecto

Defina Este proyecto tiene como alcance principal el desarrollo y validación de un sistema modular completamente funcional dentro del espacio de kernel de Linux, compuesto por cuatro módulos que interactúan entre sí para simular el comportamiento de un sistema embebido. La funcionalidad cubre desde la generación de datos de un sensor hasta el control visual de una matriz de LEDs, todo utilizando GPIOs simulados.

Funcionalidades incluidas:

- **Emulación de GPIOs en kernel space:**
  módulo llamado gpio_sim.ko que simula hasta 32 pines digitales, proporcionando funciones equivalentes a las de <linux/gpio.h> como gpio_request(), gpio_set_value() y gpio_get_value(). Esto permite que otros módulos se comporten como si interactuaran con hardware real, sin necesidad de acceso físico a pines.

- **Simulación de un sensor de temperatura tipo DHT22:**
  El módulo sensor_sim.ko genera lecturas de temperatura aleatorias dentro de un rango configurable (20 °C a 40 °C), actualizándose cada 5 segundos mediante un workqueue.

- **Control de una matriz de LEDs 4x4:**
  El módulo led_matrix.ko apara encender o apagar LEDs individuales usando pines GPIO que representan filas y columnas. El estado interno de la matriz se refleja tanto en la consola (printk) como en el estado de los GPIOs simulados.

- **Módulo de control reactivo:**
  controlador.ko para ejecutar un hilo del kernel (kthread) que lee periódicamente las temperaturas simuladas y, dependiendo del rango detectado (frío, templado o caliente), actualiza el patrón de la matriz de LEDs. Este módulo representa la lógica de negocio del sistema y centraliza la toma de decisiones basada en los datos del sensor.

- **Monitoreo del sistema en vivo:**
  Toda la interacción entre módulos se puede visualizar en tiempo real a través de dmesg -w, permitiendo observar cómo se simulan los procesos de lectura, control y visualización, como si se tratara de un entorno embebido real.

Fuera del alcance:

- No se utiliza hardware físico como Raspberry Pi ni sensores reales.

- No se incluye interfaz gráfica de usuario ni acceso desde espacio de usuario.

- No se maneja interrupciones reales ni configuraciones avanzadas del GPIO real.

Justificación del alcance:

Este alcance permite centrar el desarrollo en los aspectos más críticos del sistema operativo como comunicación entre módulos kernel, simulación de periféricos de bajo nivel y el manejo de tareas concurrentes, así como el uso de estructuras internas del kernel.

Además, garantiza que el sistema pueda ser probado y evaluado en cualquier entorno Linux sin requerimientos adicionales de hardware, facilitando su replicación, análisis y extensión futura.

3. Marco Teórico / Conceptos Fundamentales

Explique Este proyecto se basa en conceptos esenciales del desarrollo en espacio de kernel y el control de periféricos en sistemas embebidos.

- **Espacio de kernel:** Zona protegida del sistema donde se ejecutan los módulos del núcleo. Permite acceso directo a recursos como los GPIOs y control eficiente del hardware.

- **GPIO (General Purpose Input/Output):** Pines digitales que permiten interactuar con sensores y actuadores. En este proyecto se implementan como GPIOs virtuales mediante el módulo gpio_sim.ko.

- **Simulación de hardware:** Tanto el sensor DHT22 como la matriz de LEDs 4x4 fueron emulados completamente en software, permitiendo pruebas sin hardware físico.

- **Módulos del kernel:** Se desarrollaron cuatro módulos independientes que se comunican mediante funciones exportadas (EXPORT_SYMBOL()), lo que facilita el mantenimiento y la modularidad.

- **Kthreads y Workqueues:** Se usan hilos del kernel para monitoreo continuo (controlador.ko) y colas de trabajo para generar temperaturas simuladas en segundo plano (sensor_sim.ko).

- **Programación reactiva:** El sistema solo actualiza la matriz de LEDs cuando la temperatura cambia de rango (frío, templado, caliente), lo cual optimiza la respuesta.

- **Depuración con printk:** Se utiliza printk() para registrar información del sistema, como temperatura y estado de GPIOs, visible mediante dmesg.

Este enfoque permitió demostrar la interacción eficiente entre módulos del kernel para simular y controlar periféricos de forma precisa y segura además de replicable por cualquier persono sin las limitaciones que se tendrían si se hiciesen con dispositivos físicos.

4. Diseño e Implementación

4.1. Diseño de la Solución

La solución se diseñó en torno al modularidad del kernel de Linux, dividiendo la funcionalidad en cuatro módulos principales, cada uno con responsabilidades claras, acoplados mediante el uso de funciones exportadas. Esta arquitectura facilita el desarrollo, depuración y reutilización.

1. **gpio_sim.ko: Simulación de GPIOs**
   Este módulo simula una interfaz de 32 pines GPIO desde el espacio del kernel, ofreciendo funciones básicas para manejar pines virtuales:
   - ***gpio_sim_request()***: Solicita un pin.
   - ***gpio_sim_free()***: Libera un pin.
   - ***gpio_sim_direction_input()*** y ***gpio_sim_direction_output()***: Define la dirección del pin.
   - ***gpio_sim_set_value()*** y ***gpio_sim_get_value()***: Establece o lee el valor lógico (0 o 1) de un pin.

   El módulo almacena el estado de cada pin en arreglos internos. Otros módulos lo utilizan para "leer" y "escribir" en los pines, como si se tratara de hardware real. Este enfoque permite abstraer completamente la capa de hardware físico.

2. **sensor_sim.ko: Sensor de Temperatura Simulado**
   Este módulo simula el comportamiento de un sensor de temperatura tipo DHT22:
   - Cada 5 segundos genera un valor aleatorio de temperatura dentro del rango permitido (20°C a 40°C).
   - Exporta la función ***read_simulated_temperature()*** que devuelve el valor actual.

- Además, enciende o apaga un pin GPIO (GPIO [10]) dependiendo de si la temperatura supera un umbral (25°C), como lo haría un sensor con señal digital de alerta.

El módulo utiliza una *workqueue* para generar las temperaturas periódicamente sin bloquear el hilo principal del sistema.

3. **led_matrix.ko: Matriz de LEDs 4x4**

   Este módulo implementa una matriz de 4x4 LEDs controlada mediante 8 pines GPIO:
   - GPIO [0–3]: filas
   - GPIO [4–7]: columnas

   Internamente, la matriz es un arreglo de estado binario que indica qué LEDs están encendidos. Las funciones clave son:
   - ***led_on(fila, columna)***: Enciende un LED específico activando la fila y columna correspondiente con ***gpio_sim_set_value()***.
   - ***clear_matrix()***: Apaga todos los LEDs.
   - ***print_matrix_state()***: Imprime el estado de la matriz (1 = encendido, por otro lado 0 = apagado).
   - ***print_gpio_matrix_state()***: Muestra los valores actuales de los GPIOs asignados.

   Este módulo permite representar visualmente los niveles de temperatura mediante iluminación.

4. **controlador.ko: Controlador Reactivo**

   Este es el módulo principal de lógica. Ejecuta un hilo de kernel (***kthread***) que lee periódicamente la temperatura del sensor (cada 500ms):
   1. Llama a read_simulated_temperature() para obtener la última temperatura generada.
   2. Clasifica la temperatura en uno de tres rangos:
      - Frío (<25°C): enciende 4 LEDs
      - Templado (25–29°C): enciende 8 LEDs
      - Caliente (≥30°C): enciende 12 LEDs
   3. Si el rango ha cambiado desde la última lectura:
      - Llama a clear_matrix()
      - Llama a led_on() repetidamente para encender los LEDs necesarios.
      - Imprime el estado actual mediante print_matrix_state() y print_gpio_matrix_state().

   Esto simula un sistema reactivo real donde los LEDs representan un cambio significativo en el entorno (temperatura).

4.2. Tecnologías y Herramientas

- ***Lenguaje***: C (compatible con el espacio de kernel Linux)
- ***Sistema operativo***: Ubuntu 24.04 (con kernel 6.11.0-29-generic)
- ***Herramientas utilizadas***:
  - ***make***: para compilación de módulos
  - ***insmod/rmmod***: para carga y descarga de módulos
  - ***dmesg -w***: para monitorear eventos en el kernel
  - ***gcc-13***: compilador alineado con versión del kernel
- ***Mecanismos del kernel***:
  - ***EXPORT_SYMBOL()***: para compartir funciones entre módulos
  - ***printk():*** para emitir logs del kernel
  - ***kthread***: para crear hilos persistentes
  - ***workqueue***: para ejecutar tareas periódicas como lecturas del sensor

5. Pruebas y Evaluación

5.1. Metodología de Pruebas

Se cargan los módulos en orden. Se observa el comportamiento en consola (***dmesg -w***) y se verifican las actualizaciones de la matriz cada vez que cambia el rango de temperatura, al igual que se revisan los valores para los pines asignados tanto para la matriz de leds como para el sensor.

5.2. Casos de Prueba y Resultados

ID	Descripción	Resultado esperado	Resultado obtenido	Éxito/Fallo
CP-001	Temperatura < 25ºC	4 LEDs encendidos	Correcto	Éxito
CP-002	Temperatura entre 25 y 29ºC	8 LEDs encendidos	Correcto	Éxito
CP-003	Temperatura >= 30ºC	12 LEDs encendidos	Correcto	Éxito
CP-004	Temperatura cambia de rango	Matriz actualizada una vez	Correcto	Éxito
CP-005	Temperatura cambia dentro del mismo rango	Matriz no actualizada	Correcto	Éxito

5.3. Evaluación del Rendimiento (si aplica)

No aplica directamente, ya que el sistema no tiene alta carga computacional. Se verificó que el sistema es reactivo y estable en tiempo real. Por otro lado, se podría evaluar el rendimiento respecto a un controlador de espacio de usuario, pero dadas las condiciones del proyecto no sería viable o significativo los resultados al no contar con hardware real.

5.4. Problemas Encontrados y Soluciones

- **Conflictos al usar gpio.h:** Muchos sistemas carecen de soporte completo para el subsistema GPIO del kernel real, especialmente en entornos virtualizados como QEMU o VirtualBox. Esto generaba errores de compilación o ejecución. Para solucionarlo, se optó por desarrollar un módulo personalizado gpio_sim.ko, que simula 32 pines GPIO por software usando un arreglo interno. Esto permitió emular el comportamiento de entrada/salida digital sin necesidad de hardware físico.

- **Compilación cruzada vs kernel real:** Algunos errores surgieron al compilar con una versión de GCC distinta a la que compiló el kernel. Esto generaba errores como Invalid module format. Se solucionó instalando y usando explícitamente gcc-13, la misma versión usada para construir el kernel (make CC=gcc-13).

- **Visualización del estado del sistema:** Inicialmente no era claro cuántos LEDs estaban encendidos o cuál era el estado real de los pines. Se incorporaron funciones adicionales (print_matrix_state() y print_gpio_matrix_state()) que imprimen de manera clara la disposición actual de los LEDs y el valor de cada GPIO, facilitando la validación del sistema y su explicación.

- **Actualización continua de la matriz:** Se detectó que la matriz se estaba actualizando continuamente, incluso si el rango de temperatura no cambiaba. Esto generaba spam en los logs. Se mejoró el controlador para que solo actualice la matriz cuando se detecta un cambio real de estado térmico.

6. Conclusiones

Durante El proyecto alcanzó satisfactoriamente su objetivo principal: diseñar e implementar un sistema completamente funcional en espacio de kernel que simula la interacción entre un sensor de temperatura y una matriz de LEDs, utilizando pines GPIO virtuales. A través de una arquitectura modular, se desarrollaron componentes independientes que cooperan armónicamente, demostrando principios fundamentales de los sistemas operativos, como la separación entre hardware y software, el control de tareas en tiempo real mediante msleep y kthread, y la comunicación entre módulos a través de funciones exportadas.

La decisión de utilizar GPIOs simulados fue un acierto clave que permitió abstraer la capa de hardware físico, facilitando el desarrollo, pruebas y presentación del sistema sin requerir una Raspberry Pi ni otros dispositivos reales. Esta abstracción no solo aceleró el proceso de validación, sino que también posibilitó que cualquier usuario pueda replicar y probar el sistema

en su propio entorno, demostrando que la lógica de control es totalmente viable y validable desde software.

Desde el punto de vista académico, el proyecto brindó la oportunidad de comprender en profundidad el funcionamiento de un sistema embebido desde el nivel del sistema operativo, así como la interacción entre software y hardware. Aunque no se logró llevar a cabo la integración en un entorno con hardware real (como era la intención original), el trabajo permitió manipular directamente estructuras internas del kernel y simular operaciones con GPIOs, afianzando así habilidades esenciales para el desarrollo a bajo nivel. Además, se enfrentaron y resolvieron problemas típicos del entorno kernel, como incompatibilidades, errores de compilación o condiciones de carrera, lo que fortaleció la experiencia del proceso de desarrollo.

En conclusión, el proyecto no solo demostró ser funcional y didáctico, sino que también constituye una base sólida para futuras mejoras. La integración con hardware real, el manejo de interrupciones físicas y la expansión hacia nuevos periféricos como pantallas o actuadores representan líneas claras de trabajo a futuro.

8. Referencias

1. Kerrisk, M. (2010). *The Linux Programming Interface: A Linux and UNIX System Programming Handbook*. No Starch Press.
2. Rubini, A., & Corbet, J. (2001). *Linux Device Drivers* (2nd ed.). O'Reilly Media.
3. Linux Kernel Organization. (s.f.). *The Linux Kernel documentation*. Recuperado el 8 de julio de 2025, de https://www.kernel.org/doc/html/latest/

9. Anexos (Opcional)

9.1 El rol de la IA durante el desarrollo

Durante el desarrollo del proyecto, la inteligencia artificial desempeñó un papel clave como asistente técnico y guía de referencia. La IA facilitó la redacción y documentación del código, brindó apoyo para depurar errores del kernel, propuso alternativas viables cuando surgieron limitaciones técnicas y ayudó a estructurar el proyecto de manera modular y eficiente. También fue fundamental para la generación de material explicativo y técnico, como comentarios en el código, documentación en Markdown y el informe final. Gracias a su asistencia, fue posible acelerar significativamente el proceso de desarrollo y superar barreras técnicas propias del espacio de kernel, consolidando una solución funcional y didáctica.

9.2 Código

Para acceder al código y su documentación puedes acceder al siguiente enlace de GitHub (https://github.com/udea-so/proyectos_2025-1/tree/main/virtual/GR-05/codigo). Aquí podrás encontrar documentación del proyecto, el Código y una manual para ejecutar el Código.