Pregunta 4 — Propuesta de Taller/Laboratorio (Cambios al Kernel)

> Proponer un taller/laboratorio en el ámbito de **cambios al Kernel** (Unix/Linux) que podamos probar en **máquinas virtuales ARM** o en **hardware ARM como Raspberry Pi**. :contentReferenceoaicite:2

1) Laboratorio Principal: Sistema de Logs Mejorado en el Kernel

Objetivo

Implementar mejoras de **logging** en el kernel que permitan: - **Filtrar** mensajes por **nivel de severidad** (p. ej., `KERN_INFO`, `KERN_WARNING`, etc.). - **Filtrar** por **PID** (proceso específico). - **Exponer estadísticas** de logs por proceso (contadores por nivel).

Diseño (mínimo viable y defendible)

- **Punto de enganche**: utilizar la ruta de `printk()` (o capa `pr_*`) para aplicar **filtros** *antes* de encolar al buffer de `dmesg`.
- **Interfaz de control**: **sysfs** o **procfs** (evita agregar `syscall` y tocar tablas por arquitectura).
  - Ejemplo:
    - `/sys/kernel/logfilter/enabled` (0/1)
    - `/sys/kernel/logfilter/level_min` (0–7)
    - `/sys/kernel/logfilter/pid_allowlist` (lista CSV o un único PID)
    - `/proc/logstats/<pid>` → contadores por nivel
- **Estructuras simples**:
  - `level_min` (entero global atómico),
  - `allow_pid` (entero; -1 = sin filtro),
  - `per-pid stats` con `idr/hash` + `atomic64_t` por nivel (o primer MVP: un solo PID, luego extender).
- **Reglas**:
  - Si `enabled=0` → comportamiento estándar.
  - Si `enabled=1` → descartar `printk`s con `level < level_min` o `pid != allow_pid` (cuando `allow_pid >= 0`).

Por qué es interesante

- **Educativo**: delinea flujo de `printk`, subsistemas de `proc/sysfs`, atomicidad y concurrencia.
- **Práctico**: debugging más eficiente (foco por PID/nivel).
- **Verificable**: se valida con `dmesg`, `cat` en sysfs/proc y pruebas de carga.

---

2) Alternativas (con factibilidad)

1) **Driver de Dispositivo Virtual (char device)**
   - Crear `/dev/mi_dispositivo` con `read/write/ioctl`.
   - Muy bueno para aprender **VFS**, `file_operations`, memoria usuario-kernel.
   - **Nivel**: Fácil–Medio. **Riesgo** bajo. **Test** directo con `echo/cat` y `strace`.
2) **Security Module (LSM) simple**
   - Activar LSM hook limitado (p. ej., loguear `open` en `/etc/shadow` o paths "sensibles").
   - **Nivel**: Medio. Requiere Kconfig y conocer la **orden de hooks**.
   - Útil para explorar control de acceso y auditoría básica.
3) **Política de Scheduling EDF (Earliest Deadline First)**
   - Modificar `kernel/sched/` e integrar una cola EDF de juguete (no RT duro).
   - **Nivel**: Medio–Alto. Riesgo de inestabilidad si se toca el planificador principal.
   - Recomendado solo si ya se domina el subsistema `sched/*`.

   Recomendación: para cumplir tiempos y tener un **demo sólido**, priorizar el **Sistema de Logs Mejorado**, y dejar el **char device** como "plan B" si hay contratiempos.

---

3) Entorno de Desarrollo y Ejecución (ARM)

Opción A: QEMU (recomendada para iterar rápido)

Requisitos (host Linux/WSL): sudo apt-get update sudo apt-get install -y gcc-aarch64-linux-gnu qemu-system-arm qemu-system-aarch64 make bc flex bison libssl-dev libelf-dev

Compilar kernel ARM64 (ejemplo): export ARCH=arm64 export CROSS_COMPILE=aarch64-linux-gnu- make defconfig # Verificar que estén activos: CONFIG_PRINTK, CONFIG_PROC_FS, CONFIG_SYSFS make -j"$(nproc)"

Arrancar con QEMU (consola serie): qemu-system-aarch64
-M virt -cpu cortex-a57 -m 1024
-kernel arch/arm64/boot/Image
-append “console=ttyAMA0”
-nographic

Notas: - Ctrl+A, luego X para salir de QEMU con -nographic. - Si necesitas utilidades de user space, añade un initramfs mínimo.

Opción B: Raspberry Pi (hardware real)

Pasos generales: 1) Instalar Raspberry Pi OS en la SD. 2) Compilar el kernel para RPi (usar el repo/branch oficial de RPi). 3) Reemplazar kernel8.img en la partición /boot. 4) Instalar módulos si aplica: make modules_install INSTALL_MOD_PATH=

Flujo recomendado: - Validar primero en QEMU (arranque OK). - Luego probar en la Raspberry Pi.

4) Plan de Implementación por Fases

Fase 0 — “Hello kernel” (sanidad del entorno)
- Compilar kernel vanilla y **booteo OK** en QEMU (ARM/ARM64).

Fase 1 — Interfaz de control (sysfs/proc)
- Crear nodos: `enabled`, `level_min`, `allow_pid`.
- Validar lectura/escritura desde user space (`echo` / `cat`).

Fase 2 — Hook de filtrado en `printk`
- Interceptar antes de encolar/emitir al buffer de `dmesg`.
- Aplicar filtros (`enabled`, `level_min`, `allow_pid`).

Fase 3 — Estadísticas por PID
- Estructura ligera (`pid -> counters[0..7]`).
- Exponer `/proc/logstats/<pid>`.

Fase 4 — Pruebas y hardening
- Concurrencia: `spinlock_t`/atomics donde corresponda.
- Carga: spamear `pr_info/pr_warn` con distintos PIDs y niveles.
- Medición cualitativa con/sin filtros.

---

5) Plan de Pruebas (mínimo defendible)

1) **Funcionalidad básica**
   - `enabled=0` → `dmesg` estándar.
   - `enabled=1`, `level_min=4 (WARN)` → `INFO/DEBUG` no aparecen.

- `allow_pid=<PID>` → solo logs del PID indicado.

2) **Estadísticas**
   - Ejecutar 2–3 productores de logs con niveles distintos.
   - Verificar `/proc/logstats/<pid>` (totales por nivel).
3) **Estabilidad**
   - Stress con `fork` y `printf/pr_*` en bucle.
   - Sin `kernel panic`/WARNings inesperados.
4) **Rerun**
   - Reboot del kernel → filtros reseteados por defecto (configurable).

---

6) Criterios de Evaluación

- **Funcionalidad y estabilidad** del cambio (sin `panic`).
- **Documentación** clara (qué, por qué, cómo probar).
- **Originalidad / valor**: filtros por nivel + PID + stats.
- **Alineación con entregables**: informe con lecciones aprendidas, repo con código, gestión de roles y documentación de LLMs.

---

7) Roles y Gestión

- **Líder**: coordina tareas, revisa PRs, agenda pruebas.
- **Kernel dev**: implementa filtros y sysfs/proc.
- **QA/Infra**: QEMU/ARM, scripts de prueba, documentación.

---

8) Riesgos y Mitigaciones

- **Rendimiento**: filtro O(1), minimizar locks, usar atomics.
- **Condiciones de carrera**: proteger lectura/escritura de filtros con `spinlock`/barreras.
- **Portabilidad**: cambios acotados, evitar `syscall` en el MVP.
- **Tiempo**: si se complica, pivot a **driver char** (alternativa 1).

---

9) Alternativa de Respaldo

Driver char `/dev/mi_dispositivo` con `read/write/ioctl` y contador interno.
- Fácil de demostrar, bajo riesgo, aprendizaje claro de VFS.

---

10) Uso de LLMs (documentación mínima)

- Se consultó un LLM para:
  - Elegir un **MVP** factible (filtros en `printk` + sysfs/proc).
  - Recomendaciones de **entorno ARM con QEMU** y minimizar invasión (evitar `syscall`).
  - Diseño de **plan de pruebas** y **mitigaciones**.
- **Verificación local**: compilar/boot en QEMU y probar con `dmesg`/`cat`/`echo`.
- **Criterios finales de diseño**: nombres de nodos sysfs/proc y formato de estadísticas ajustados por el equipo.