Programación Hardware

Controladores de dispositivo (Linux)

Alumno: Diego Gomez Sanchez
Grado/Grupo: Ingeniería Informática - G1
Curso: 2025/2026
Profesor: David Díaz Jiménez

Fecha: 8 de febrero de 2026

Universidad de Jaén

Índice

1. Introducción y contexto

1.1. Objetivo de la práctica

El objetivo es transformar un controlador de dispositivo .[es]queleto.[en] uno funcional de caracteres. Se implementa la gestión de múltiples nodos (/dev/ECCDriver0, 1 y 2) con memoria independiente y transferencia segura de datos entre el espacio de usuario y el kernel.

1.2. Entorno de trabajo

- Sistema operativo: Ubuntu Linux (Kernel 6.x).
- Herramientas: GCC, Make, Kbuild.
- Comandos principales: `insmod`, `rmmod`, `dmesg`, `chmod`.

2. Cuestión 1: Controladores de caracteres (mem.c)

2.1. Ruta y descripción

El archivo `/drivers/char/mem.c` es la referencia para dispositivos como `/dev/null` o `/dev/zero`. Implementa la lógica de cómo el kernel debe tratar los flujos de datos básicos y sirve de base para el diseño de nuestro driver.

3. Cuestión 2: Relación entre `inode`, `file` y `cdev`

3.1. Estructuras VFS

- **inode**: Representa el archivo físico. Contiene el número *minor* necesario para identificar el dispositivo.
- **file**: Representa el archivo abierto por el proceso. Incluye el *offset* y el puntero `private_data`.
- **cdev**: Es la representación interna en el kernel del dispositivo de caracteres, vinculando el número mayor/menor con las operaciones `file_operations`.

4. Cuestión 3: Operación `open()` identificando el dispositivo

Se utiliza la función `iminor(inode)` para detectar qué dispositivo específico se está abriendo, cumpliendo con el requisito de identificación.

```
1 static int ECCopen(struct inode *inode, struct file *file) {
2     int minor = iminor(inode);
3     pr_info("ECCDriver: Abriendo dispositivo con numero menor: %d
          \n", minor);
4     return 0;
```

```
}
```

5. Cuestión 4: Funcionalidad para `read()` y `write()`

Se implementa una memoria persistente mediante una estructura de datos global `ECC_devices`.

Listing 1: Implementación de ECCwrite

```
static ssize_t ECCwrite(struct file *file, const char __user *
    buffer, size_t count, loff_t *f_pos) {
    int minor = iminor(file_inode(file));
    memset(ECC_devices[minor].buffer, 0, BUFFER_SIZE);
    if (count > BUFFER_SIZE) count = BUFFER_SIZE;
    if (copy_from_user(ECC_devices[minor].buffer, buffer, count))
        {
        return -EFAULT;
    }
    ECC_devices[minor].length = count;
    pr_info("ECCDriver: Guardados %zu bytes en el dispositivo %d\
        n", count, minor);
    return count;
}
```

Listing 2: Implementación de ECCread

```
static ssize_t ECCread(struct file *file, char __user *buffer,
    size_t count, loff_t *f_pos) {
    int minor = iminor(file_inode(file));
    if (*f_pos >= ECC_devices[minor].length) return 0;
    if (count > ECC_devices[minor].length - *f_pos)
        count = ECC_devices[minor].length - *f_pos;
    if (copy_to_user(buffer, ECC_devices[minor].buffer + *f_pos,
        count))
        return -EFAULT;
    *f_pos += count;
    return count;
}
```

6. Cuestión 5: Propuesta personal de funcionalidad

6.1. Diseño de "Búfer de Mensaje Único"mediante `memset`

Como propuesta personal, se ha implementado la limpieza total del búfer mediante `memset` previo a cada escritura. Esto garantiza que cada mensaje sea independiente y no contenga restos de ejecuciones anteriores, mejorando la integridad de la información.

7. Pruebas realizadas (Protocolo de verificación)

7.1. Compilación y carga del módulo

En primer lugar, se compila el código fuente mediante la utilidad `make`. Una vez generado el archivo `.ko`, se carga en el kernel:

```
make
sudo insmod ECCDriver.ko
```

7.2. Asignación de permisos

Para permitir que un usuario sin privilegios de administrador pueda interactuar con los dispositivos, se modifican los permisos de los nodos creados en `/dev`:

```
sudo chmod 666 /dev/ECCDriver*
```

7.3. Verificación de persistencia independiente

Se realizan pruebas cruzadas escribiendo mensajes diferentes en distintos dispositivos para comprobar que los búferes son independientes:

```
# Escritura en dispositivos distintos
echo "Mensaje para dispositivo 0" > /dev/ECCDriver0
echo "Datos del dispositivo 1" > /dev/ECCDriver1

# Lectura y comprobacion
cat /dev/ECCDriver0   # Resultado esperado: Mensaje para
   dispositivo 0
cat /dev/ECCDriver1   # Resultado esperado: Datos del dispositivo
    1
```

7.4. Verificación de identificación y logs del sistema

Para asegurar que el driver identifica correctamente el número *minor* (Cuestión 3) y la escritura (Cuestión 4), se examina el buffer de mensajes del kernel:

```
sudo dmesg | tail -n 10
```

En la salida se observa la trazabilidad de las operaciones, indicando exactamente qué dispositivo ha sido abierto y cuántos bytes se han almacenado en su memoria interna.

7.5. Descarga del módulo

Finalmente, se comprueba que el driver se descarga correctamente liberando los recursos:

```
sudo rmmod ECCDriver
```