Sistemas Operativos 2

Unidad 2: Administración de dispositivos de E/S

Arquitectura de la entrada salida

René Ornelis
Primer semestre de 2025

Contenido

Índice de figuras

Arquitectura de la entrada/salida

1 Módulos del administrador de entrada/salida

El sistema operativo o como cualquier pieza de software es una pieza de software compleja la cual se tiene que estructurar en módulos que hagan eficiente su administración. Adicionalmente recordemos que el sistema operativo debe de buscar el principio de independencia de los dispositivos por lo que esta arquitectura interna también debe orientarse a este objetivo. Como se ve en la Figura 1, en general los sistemas operativos estructura sus módulos de siguiente manera (de abajo hacia arriba):

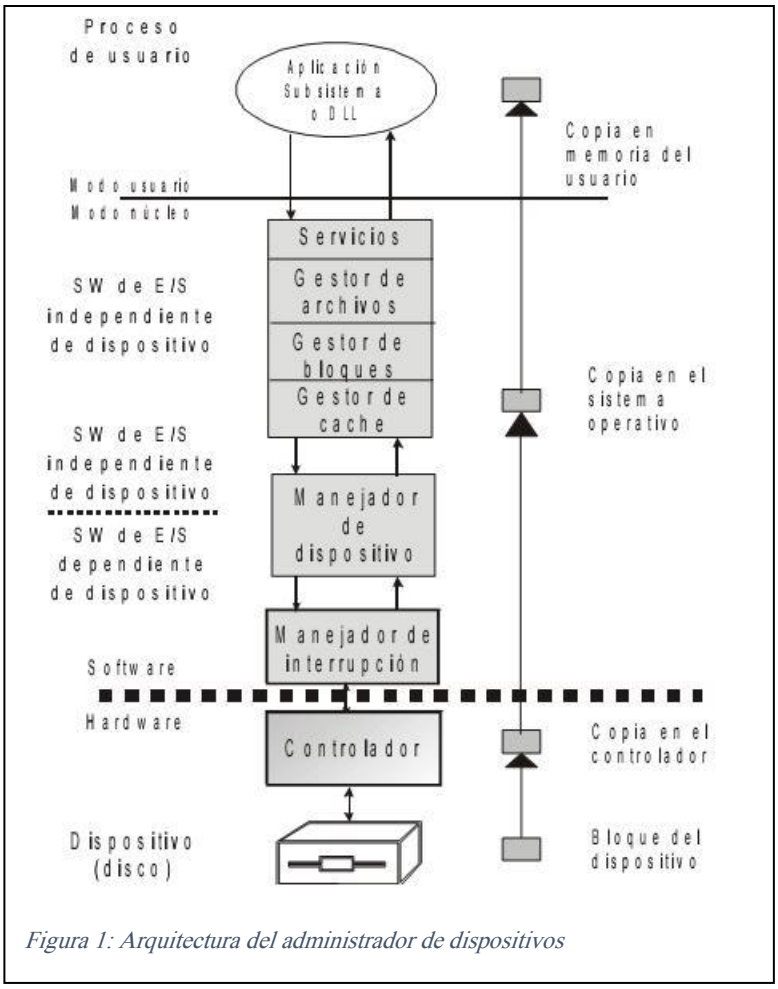

Figura 1: Arquitectura del administrador de dispositivos

- Manejador de interrupciones
- Manejadores de dispositivos
- Subsistema de gestión
- Interfaz del SO
- Aplicaciones del usuario

1.1 Manejadores de interrupciones

Son el nivel más bajo de la arquitectura. El módulo de administración de interrupciones es el que controla la asignación de interrupciones por dispositivo de forma que cuando recibe una interrupción, invoca la rutina de servicio definida por el *driver* registrado para ese dispositivo. Este módulo está muy vinculado al hardware ya que debe tener en cuenta las características de cada procesador y dispositivo, por lo que decimos que es la parte de la arquitectura *dependiente del dispositivo*.

Este módulo está obligado a correr en modo *kernel* (modo supervisor o modo privilegiado), ya que necesita tener acceso a todo el hardware, la memoria (con el MMU deshabilitado) y los puertos de entrada/salida.

1.2 Manejadores de dispositivos (*driver*)

Tienen el código específico para cada dispositivo. Su función es recibir información del dispositivo y trasladarla a las capas superiores. Asimismo, acepta solicitudes generales del sistema operativoy asegurarse que se cumpla la requisición. También convierte los datos del dispositivo al formato general del SO.

El *driver* marca la línea divisoria entre el software dependiente del dispositivo y el software independiente del dispositivo. Parte del *driver* y el manejador de interrupciones conforman la parte dependiente del dispositivo, ya que ambos módulos deben trabajar con las características de los dispositivos y el procesador y se debe manejar los comandos específicos para cada dispositivo. Otra parte del *driver* debe trabajar con comandos genéricos del sistema operativo y forma parte del software independiente del dispositivo.

1.3 Subsistema de gestión

Este subsistema está conformado por todos los módulos necesarios para la administración genérica de los dispositivos. Por ejemplo, la administración de bloques, el sistema de archivos la administración de la caché, las colas de prioridades, el *buffering*, etc.. que trabajan y administran sin interactuar directamente con los dispositivos ya que esto se hace a través del *driver*. Este subsistema es parte del software independiente del dispositivo

1.4 Interfaz del SO

Es la parte del sistema operativo por medio de la cual presenta a los procesos los servicios para que estos puedan hacer uso de los recursos del sistema (dispositivos). El API de todo sistema operativo debe buscar ser independiente del dispositivo, de forma que las complejidades de cada dispositivo quedan ocultas dentro de la arquitectura del sistema operativo.

Por ejemplo, en el caso de acceder un archivo:

1. El proceso, a través del API del sistema le indica que quiere leer un bloque de 10k del archivo c:\texto.txt. Estos parámetros son enviados como un código de servicio (leer de archivo), nombre del archivo y buffer donde el proceso requiere que quede la información leída.
2. A través del sistema de archivos convierte el nombre de archivo (c:\texto.txt) en una dirección de bloque dentro del disco en forma de cilindro-cabeza-sector.
3. Se revisa si la información correspondiente ya está en la memoria cache. De ser así, ya no se lee del dispositivo, sino que se toma de la memoria cache y se transfiere al proceso en la dirección de memoria solicitada por éste.
4. Si la información no está en caché, el sistema prepara un buffer para recibir la información y agrega el requerimiento a la cola de atención del *driver*.
5. El *driver* toma el requerimiento y lo convierte en instrucciones específicas para el disco y el DMA.
6. El disco trabaja en el requerimiento, en conjunto con el DMA, y cuando finaliza la petición, el DMA genera la interrupción de notificación de aviso.
7. El *driver* informa a las capas superiores de la finalización del requerimiento

8. El módulo de gestión de buffers traslada la información en el buffer del *kernel* (con la información leída) y lo traslada al buffer del proceso
9. Se le notifica al proceso que la petición está completa.

1.5 Aplicaciones del usuario

Las aplicaciones del usuario, tiene acceso al API del SO para trabajar con los dispositivos de E/S. Sin embargo, existen otras aplicaciones como los **demonios**, o **bibliotecas** que hacen procesos de **manejo por cola**, como colas de impresión, servidores de correo, etc.

2 Control de acceso a dispositivos compartidos

Como se ve en el estudio de los interbloqueos, existen recursos (dispositivos) son exclusivos e inapropiativos, es decir que el dispositivo sólo puede ser utilizado por un proceso a la vez (exclusivo) y el sistema no desasignar el recurso dado a un proceso (inapropiativo). Estos dispositivos representan para el programador y para el sistema operativo complejidad adicional, tanto para el control del interbloqueo como para la coordinación del acceso. Por eso, gran parte de la arquitectura del sistema de entrada salida está basada en abstraer y ocultar de los procesos el uso compartido de dispositivos. En general, cualquier dispositivo será visto por los procesos como un dispositivo compartido, ya que no tiene que coordinar con otros procesos es acceso al mismo, sino solo enviar un comando y esperar por la respuesta.

Si los procesos accedieran directamente a los dispositivos tendrían que coordinar con los demás procesos el acceso. Por ejemplo: acceder a una impresora implica que una vez concedido a un proceso y este inicie la impresión, los demás procesos deberán esperar hasta que el proceso favorecido termine de imprimir. Esto era evidente en sistemas operativos simples como el DOS donde cada proceso accedía directamente a la impresora y el usuario no podía realizar nada más ya que este sistema carecía de la habilidad de procesos concurrentes.

Este concepto es el SPOOL (del inglés *Simultaneous Peripheral Operation Online*) que consiste en que el API del sistema operativo presenta a los procesos un servicio compartido y este servicio es el único que accede al dispositivo exclusivo. Por ejemplo, en caso de una impresora (ver Figura 2) los procesos no acceden a la impresora, sino que invocan al servicio de SPOOL y este es el que se encarga de ordenar los requerimientos, priorizar y organizar la impresión. Al mismo tiempo reporta a los procesos que el requerimiento fue recibido, por lo que estos pueden continuar con otros trabajos.

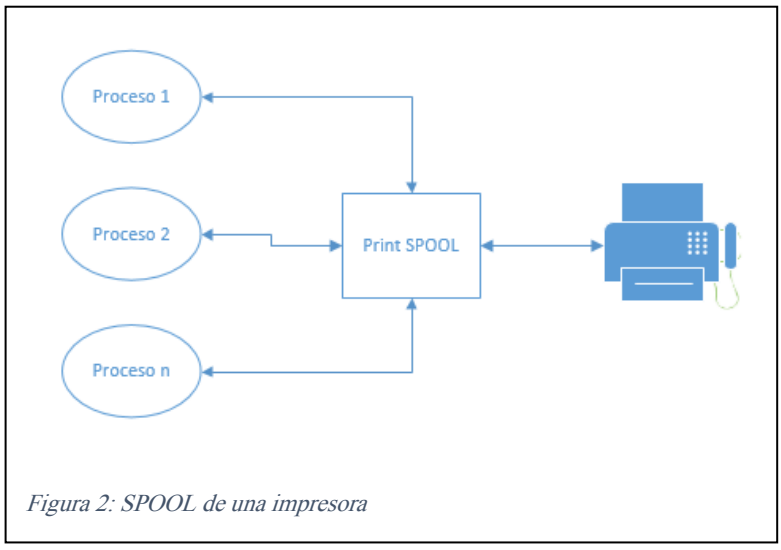

Figura 2: SPOOL de una impresora

Este concepto se puede generalizar a cualquier dispositivo y tiene la ventaja de que se pueden implementar más fácilmente las políticas de acceso a los dispositivos.

3 Plug and Play (PnP)

El concepto de *Plug and Play (PnP)* se refiere a una tecnología y un conjunto de estándares diseñados para facilitar la conexión y configuración automática de dispositivos en un sistema informático sin necesidad de intervención manual del usuario. El objetivo principal de Plug and Play es **simplificar la adición de nuevos hardware**, reduciendo la complejidad del proceso de instalación y configuración.

Plug and Play permite que un sistema operativo detecte automáticamente nuevos dispositivos de hardware cuando se conectan al ordenador, configurarlos adecuadamente, e integrarlos en el sistema sin requerir que el usuario realice ajustes manuales. Esto incluye la asignación de recursos, como direcciones de entrada/salida (I/O), interrupciones (IRQ), y canales de DMA (Direct Memory Access).

3.1 Funcionamiento de Plug and Play

3.1.1 Detección de dispositivos

- **Cuando se conecta un nuevo dispositivo**: El sistema operativo detecta automáticamente la conexión del nuevo hardware. Esto puede suceder a través de diversos métodos, como conexiones físicas (USB, PCI) o mediante la identificación del dispositivo en un bus de expansión.
- **Protocolos de comunicación:** Los dispositivos compatibles con Plug and Play utilizan protocolos estándar para comunicar su presencia y capacidades al sistema operativo.

3.1.2 Identificación del dispositivo

- **ID del dispositivo**: Cada dispositivo Plug and Play tiene identificadores únicos (como el ID de hardware) que permiten al sistema operativo identificar el tipo y la marca del dispositivo (ver Figura 3). El código de fabricante es concedido por la organización *Unified Extensible Firmware Interface Forum* (uefi.org).
- **Información del dispositivo**: El dispositivo puede proporcionar información adicional sobre sus capacidades y requerimientos a través de datos de configuración.

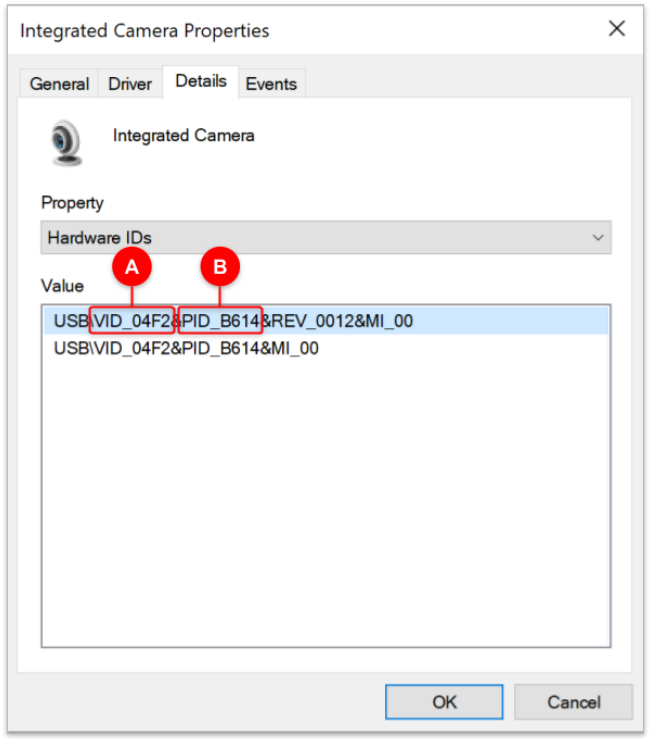

Figura 3: Código PnP de un dispositivo

3.1.3 Instalación de controladores

- **Búsqueda y carga de controladores:** El sistema operativo busca controladores adecuados para el dispositivo. Estos controladores son programas que permiten al sistema operativo y a las aplicaciones interactuar correctamente con el hardware.
- **Instalación automática**: Si el controlador adecuado está disponible, el sistema operativo lo instala automáticamente. En algunos casos, puede solicitar al usuario que proporcione un controlador si no se encuentra uno adecuado en el sistema.

3.1.4 Asignación de recursos

- **Recursos del sistema**: El sistema operativo asigna recursos necesarios para el dispositivo, como direcciones de memoria, puertos de I/O, y líneas de interrupción. Esto asegura que el nuevo dispositivo no entre en conflicto con otros dispositivos ya presentes en el sistema.
- **Configuración dinámica**: La configuración de recursos se realiza de manera dinámica para evitar conflictos y para maximizar el uso eficiente de los recursos del sistema.

3.1.5 Configuración del dispositivo

- **Configuración y prueba**: Una vez instalados los controladores y asignados los recursos, el dispositivo es configurado y probado para asegurar que funcione correctamente. El sistema operativo puede realizar ajustes adicionales según sea necesario para optimizar el rendimiento del dispositivo.

3.1.6 Integración y uso

- **Integración en el sistema**: El dispositivo se integra en el sistema operativo y está disponible para su uso por las aplicaciones y el usuario.

- **Gestión continua:** El sistema operativo continúa gestionando el dispositivo y puede actualizar su configuración si se realizan cambios en el hardware o en los requisitos del sistema.

3.2 Beneficios del Plug and Play

- **Facilidad de uso**: Reduce la necesidad de intervención manual por parte del usuario, facilitando la instalación de nuevos dispositivos.
- **Reducción de errores:** Minimiza la posibilidad de conflictos de recursos y errores de configuración, ya que el sistema operativo maneja automáticamente estos aspectos.
- **Compatibilidad:** Mejora la compatibilidad de hardware con diferentes sistemas operativos y configuraciones.

3.3 Ejemplos y aplicaciones

- **Conexiones USB:** La tecnología Plug and Play es ampliamente utilizada en dispositivos USB, como teclados, ratones, impresoras, y unidades de almacenamiento externas.
- **Tarjetas de expansión PCI/PCIe**: Los sistemas modernos utilizan Plug and Play para tarjetas de expansión, como tarjetas de video y tarjetas de sonido, para facilitar la integración con la placa base.

4 Buffering

Tal como se indicó anteriormente, el subsistema de gestión de la entrada/salida, incluye el manejo de los buffers del sistema. Recordemos que cuando un proceso solicita una operación de entrada salida uno de los parámetros es la dirección de memoria donde está la información a escribir o donde se solicita que queden los datos leídos. Este buffer del proceso está en la **memoria del proceso** y es una dirección lógica.

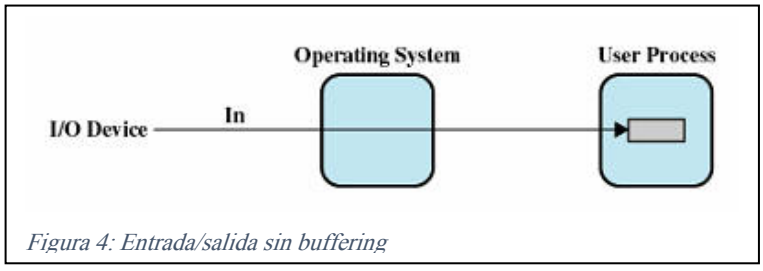

Figura 4: Entrada/salida sin buffering

Tal como se ve en la Figura 4, si la operación de entrada salida se realiza sin la utilización de buffering, implica que la página del proceso debe ser excluida del swapping de memoria virtual. Recordemos que las páginas de los procesos están dentro de los marcos de página, que es el área designada para uso de memoria virtual. Aunque una página puede ser marcada como "siempre en memoria", supone o un desperdicio de espacio, si el buffer no ocupa toda la página, o un aumento de la hiperpaginación al descartar de la política de reemplazo la página (o páginas) donde reside el buffer.

Por esto es necesario que en toda operación de entrada salida, el DMA/dispositivo transfieran la información a un buffer del sistema operativo, y al finalizar se traslada a la memoria del proceso, tal como se muestra en la Figura 5. Esto tiene la ventaja que, dado que los procesos deben esperar hasta que las operaciones de E/S terminen, el proceso puede ser

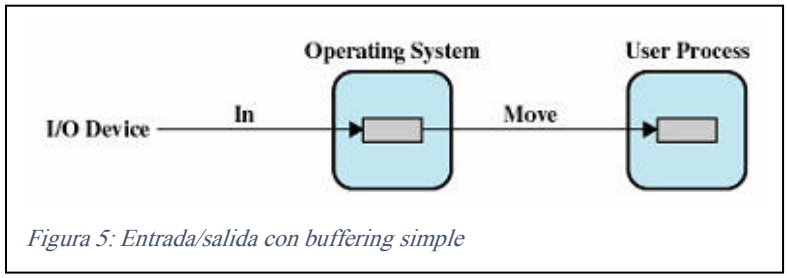

Figura 5: Entrada/salida con buffering simple

suspendido y sus páginas descargadas a disco sin problema, hasta que finalice la operación de entrada/salida.

4.1 Orientaciones del buffering

El manejo del buffering depende en gran manera del tipo de dispositivo para el cual se utilice, ya que la complejidad de manejo varía entre los dispositivos de bloque y los dispositivos carácter.

El manejo de un buffer para un dispositivo de bloque tiene la ventaja que los bloques son de tamaño fijo y se maneja un bloque a la vez. Por su parte el buffer para un dispositivo de carácter debe manejar un flujo de datos constante, lo cual no se puede manejar como un solo bloque, por lo que lo usual es utilizar buffer como una cola circular de información. En este caso, la rutina de servicio de interrupción solo ingresa información al buffer, mientras que otra parte del *driver* se encargar leer de la cola y dar el debido proceso. Esto representa un problema de productor-consumidor, con la complejidad de que la rutina de servicio de interrupción no puede utilizar ningún tipo de coordinación interproceso (IPC) como semáforos, monitores y otros, ya que de ninguna forma puede suspenderse y la información en la cola debe mantenerse íntegra.

4.2 Tipos de buffering

Ya vimos la necesidad de mantener un buffering para el manejo de la entrada/salida y con ello, el primer tipo de buffering; **buffering simple**, el cual funciona y permite que los procesos requirientes puedan ser suspendidos mientras finaliza la operación de entrada/salida. Sin embargo, mientras el sistema transfiere la información del buffer del sistema a la página del proceso (implica posiblemente subir la página del proceso desde disco), no se puede realizar otra operación con el dispositivo. Entonces surge la idea de utilizar un **buffering doble**, el cual, tal como se ilustra en la Figura 6, es conformado por un buffer de entrada y un buffer de salida. Así, un dispositivo puede iniciar otra operación de entrada/salida mientras la información de la operación anterior es transferida a la memoria del proceso solicitante, lo cual mejora el rendimiento de la entrada/salida.

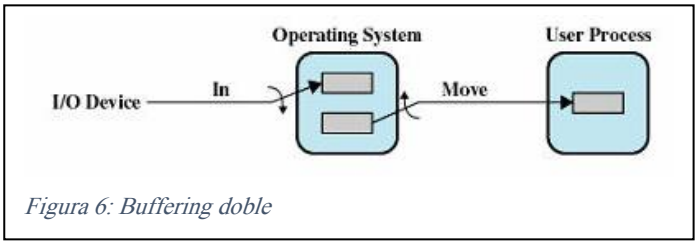

Figura 6: Buffering doble

Este mismo concepto, se puede expandir a **buffering múltiple**, el cual consiste en múltiples espacios de memoria que se utilizan como buffer para la operación con un dispositivo (ver Figura 7) los cuales se utilizan de forma cíclica.

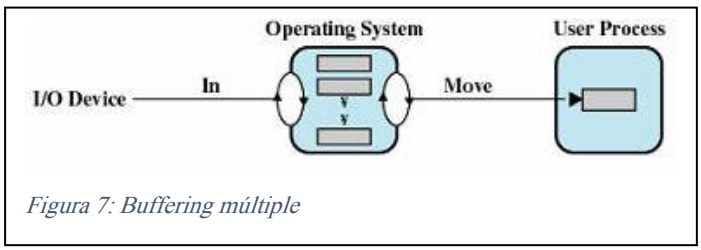

Figura 7: Buffering múltiple

Este esquema es muy útil en caso de dispositivos con alta demanda, como un disco duro, el cual que

atiende a muchos procesos, por lo que puede estar transfiriendo información a muchos procesos mientras realiza una operación de entrada/salida.