AVANCE 16/05/2025

Núñez Anrubio Jonathan Mahonri & Peña Razo Christian Yael

# 5.1 Principios del hardware de E/S

El control de los dispositivos conectados a una computadora es una preocupación clave para los diseñadores de sistemas operativos. Debido a la gran variedad de funciones y velocidades de los dispositivos de E/S (como un ratón, un disco duro o un robot de cinta), se necesitan métodos diferentes para controlarlos. Estos métodos conforman el **subsistema de E/S** del núcleo (kernel), que aísla al resto del sistema operativo de la complejidad del manejo de dispositivos.

Existen dos tendencias opuestas en la tecnología de E/S:

1. **Estandarización creciente** de interfaces de hardware y software, lo que facilita integrar nuevas generaciones de dispositivos.
2. **Aumento en la diversidad** de dispositivos, algunos tan distintos que su integración es compleja.

Para enfrentar este reto, se combinan técnicas de hardware (como puertos, buses y controladores) y software. El sistema operativo utiliza **controladores de dispositivos (device drivers)**, que encapsulan los detalles específicos de cada dispositivo y ofrecen una **interfaz uniforme** de acceso, al igual que las llamadas al sistema lo hacen entre aplicaciones y el sistema operativo.

# 5.1.1 Dispositivos de E/S

Las computadoras interactúan con diversos dispositivos, clasificados principalmente como dispositivos de almacenamiento, transmisión e interfaz humana. Estos dispositivos se conectan al sistema mediante puertos, y si comparten cables, se utiliza un **bus**, como el **bus PCI**, que emplea un protocolo y señales eléctricas definidas. Una **conexión en cadena** enlaza varios dispositivos en serie, funcionando como un bus.

Un **controlador** es el conjunto electrónico que opera puertos, buses o dispositivos. Algunos son simples, como el de puerto serie; otros, complejos, como el de **canal de fibra (FC)**, que puede requerir su propia tarjeta (HBA). Muchos dispositivos, como los discos duros, incluyen su **controlador integrado**, que maneja funciones avanzadas como la gestión de errores, almacenamiento en búfer y caché.

# 5.1.2 Controladores de Dispositivos

Un controlador de dispositivo es un software cuya función es la de indicar al [**sistema operativo**](https://www.neoguias.com/sistema-operativo/) y otros Softwares el modo en el cual comunicarse con una pieza de [**hardware**](https://www.neoguias.com/hardware/).

Por ejemplo, los controladores de impresoras comunican al sistema operativo, así como también a cualquiera sea el programa en el cual se encuentre abierto el documento a imprimir, cómo imprimir la información de manera exacta.

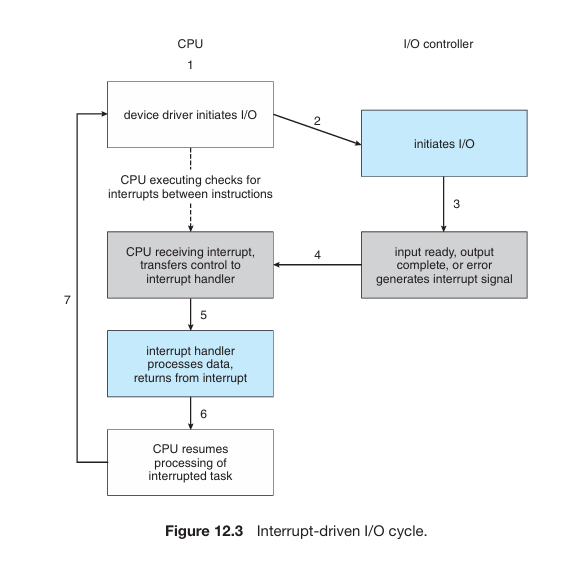
La misma idea general se aplica a [**tarjetas gráficas**](https://www.neoguias.com/que-es-tarjeta-grafica/), [**teclados**](https://www.neoguias.com/que-es-teclado/), [**monitores**](https://www.neoguias.com/que-es-monitor/), etc.

# 5.1.3 Interrupciones

El **mecanismo de interrupciones** permite que el CPU responda a eventos asíncronos, como cuando un dispositivo necesita atención. El CPU detecta una señal en la línea de petición de interrupción, guarda su estado, salta a una rutina especial llamada **manejador de interrupciones**, atiende el evento y luego retoma la ejecución previa.

**Características clave del sistema de interrupciones:**

* **Tipos de interrupciones**:
  + **No enmascarables**: para errores críticos como fallos de memoria.
  + **Enmascarables**: pueden desactivarse temporalmente para evitar interrupciones durante operaciones críticas.
* **Interrupciones vectoreadas**: se usa una tabla (vector de interrupciones) con direcciones de manejadores específicos, evitando tener que revisar todos los dispositivos.
  + Si hay más dispositivos que entradas en el vector, se usa **encadenamiento de interrupciones**, donde cada entrada enlaza a varios manejadores.
* **Priorización**: el sistema puede **clasificar interrupciones** por niveles de prioridad para que las más importantes se atiendan primero.
* **Trampas (software interrupts)**: se usan para eventos como fallos de página o errores como división por cero. También son utilizadas por llamadas al sistema hechas por programas.



**Ejecución y manejo:**

* **Manejadores de interrupciones en dos niveles**:
  + **FLIH (First-Level Interrupt Handler)**: realiza el cambio de contexto y registra la interrupción.
  + **SLIH (Second-Level Interrupt Handler)**: se encarga del trabajo completo, como copiar datos o finalizar una operación.
* **Aplicaciones del mecanismo de interrupciones**:
  + Manejo de errores, memoria virtual (fallos de página), llamadas al sistema y control de flujos en el kernel.
  + En operaciones como la lectura de disco, se usan varios niveles de interrupciones para balancear urgencia y eficiencia.
* **Ejemplo: Solaris**: usa **hilos del núcleo** para manejar interrupciones con diferentes prioridades, permitiendo ejecución concurrente en sistemas multiprocesador y priorizando tareas críticas.

# 5.1.4 Acceso de memoria directo (DMA)

El **DMA (Direct Memory Access)** permite transferencias de datos entre dispositivos (como discos) y memoria sin involucrar directamente al CPU, lo que mejora el rendimiento del sistema frente al método tradicional **PIO (Programmed I/O)**, donde el CPU gestiona manualmente cada byte transferido.

**Funcionamiento del DMA:**

1. **Inicio del DMA**:
   * El CPU escribe un bloque de comandos DMA en memoria con:
     + Dirección de origen y destino.
     + Cantidad de bytes a transferir.
   * Este bloque puede incluir direcciones no contiguas (**scatter–gather**).
   * Luego, el CPU informa al controlador DMA y continúa con otras tareas.
2. **Transferencia**:
   * El controlador DMA toma el control del bus de memoria y realiza las transferencias.
   * Usa cables especiales (**DMA-request y DMA-acknowledge**) para coordinarse con el controlador del dispositivo.
   * Una vez finalizada la transferencia, el DMA **interrumpe al CPU** para notificar la finalización.
3. **Impacto en el CPU**:
   * Durante la transferencia, el CPU puede ser momentáneamente bloqueado del acceso a la RAM (**cycle stealing**), pero puede seguir usando su caché.
4. **Dirección de la memoria**:
   * Generalmente, la transferencia se hace en espacio de **memoria del kernel**.
   * Si se requiere acceder desde **espacio de usuario**, se necesita una **copia adicional**, lo que es ineficiente.
   * Para evitarlo, los sistemas modernos usan **mapeo de memoria** para permitir transferencias directas hacia/desde el espacio de usuario.
5. **Acceso y seguridad**:
   * En sistemas con protección de memoria, solo el sistema operativo puede emitir comandos a los dispositivos.
   * En sistemas sin protección, los procesos pueden acceder directamente a los controladores, lo que mejora el rendimiento, pero compromete la **seguridad y estabilidad**.

# 5.2 Principios del software de E/S

Dejando de lado los principios del hardware de E/S, ahora tenemos que tomar en cuenta los **principios del software de E/S**, el cual abarca un conjunto de rutinas del sistema operativo que ayuda a gestionar la comunicación entre sistema y dispositivos (memoria y CPU). **El principal propósito es abstraer la complejidad del hardware, facilitar el acceso a dispositivos y optimizar el rendimiento del sistema.** Esto parte en la organización del software estableciendo capas, existiendo dos capas básicas pero importantes:

1. Las **capas inferiores** manejan o interaccionan directamente con el hardware, se encargan de los controladores de dispositivos y gestionar interrupciones, ocultando peculiaridades del hardware para que las capas superiores no las detecten.
2. Las **capas superiores** son las encargadas de ofrecer una interfaz sencilla y uniforme a los programas de usuario, ocultando los detalles del hardware. Estas capas se comunican con las capas inferiores para realizar las operaciones físicas, pero desde el punto de vista del usuario y del programador, solo se necesita trabajar con operaciones abstractas como leer, escribir o abrir un archivo.

Analizaremos el tema a profundidad, iniciando por los principales objetivos del software de E/S y posteriormente las distintas maneras en las que se puede realizar la E/S desde el punto de vista del sistema operativo.

5.2.1 Objetivos del software de E/S

El **objetivo principal del software de E/S** es lograr la **independencia de dispositivos**, es decir, permitir que los programas puedan acceder a distintos dispositivos (discos duros, CD-ROM, USB, etc.) sin necesidad de que sean modificados para cada tipo de hardware, por ejemplo, un programa que lee un archivo debe funcionar igual si el archivo está en un disco duro o en una memoria USB.

También se busca una **denominación uniforme**, en donde todos los archivos y dispositivos deben identificarse mediante nombres simples (cadenas o enteros), sin necesidad de saber qué tipo de dispositivo hay detrás. En sistemas como UNIX, todos los dispositivos se integran como en una jerarquía de archivos, de modo que, una memoria USB puede montarse en un directorio como */usr/ast/respaldo*, y los archivos escritos allí se almacenan en ese dispositivo, sin que el usuario tenga que preocuparse por ello.

Otro aspecto esencial es el **manejo de errores**, los errores deben tratarse en los niveles más cercanos al hardware, ya que, si un controlador detecta un error de lectura, debe intentar corregirlo automáticamente. Muchos de los errores son temporales y se solucionan repitiendo la operación. Solo si los niveles más bajos no pueden resolver el problema, este debe ser notificado a los niveles superiores.

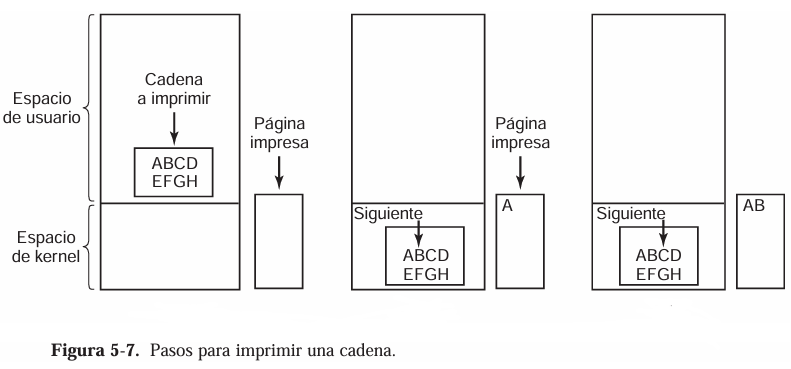
Las **transferencias síncronas y asíncronas** son otro tema importante, aunque la mayoría de las operaciones de E/S son asíncronas (la CPU inicia la operación y continúa con otra tarea hasta recibir una interrupción), los programas de usuario suelen trabajar con operaciones de **bloqueo** (síncronas), donde el programa se suspende hasta que los datos están disponibles. El sistema operativo se encarga de simular este comportamiento para simplificar el desarrollo de programas.

El **uso de búferes** también es crucial, ya que muchas veces los datos no pueden ir directamente del dispositivo a su destino final, así que se almacenan temporalmente en búferes, por ejemplo, tenemos los paquetes de red primero se almacenan y examinan antes de saber a dónde enviarlos. En dispositivos con restricciones de tiempo, como los de audio digital, los búferes ayudan a mantener un flujo constante de datos y evitan problemas como sub-desbordamientos, sin embargo, el uso de búferes implica copiado de datos y puede afectar el rendimiento.

Finalmente, se distingue entre **dispositivos compartidos** (como discos, que pueden ser usados por varios usuarios a la vez) y **dispositivos dedicados** (como unidades de cinta, que deben ser usados por un solo usuario a la vez). El sistema operativo debe gestionar ambos tipos correctamente, evitando conflictos como los interbloqueos cuando múltiples usuarios compiten por un recurso exclusivo.

5.2.2 E/S programadas

La **E/S programada** es la forma más simple de realizar operaciones de entrada/salida, donde **la CPU realiza todo el trabajo directamente**, sin encargar en mecanismos como interrupciones o DMA (acceso directo a memoria). En este modelo, la CPU interactúa de forma activa con el dispositivo, supervisando constantemente su estado y realizando la transferencia de datos. Para entenderlo mejor, se utiliza el ejemplo de un proceso que desea imprimir la cadena “ABCDEFGH”:



1. **Preparación en espacio de usuario:**
   * El proceso coloca la cadena en un **búfer** dentro de su espacio de usuario.
   * Luego solicita acceso a la impresora a través de una llamada al sistema. Si la impresora está ocupada, el proceso puede recibir un error o quedar bloqueado hasta que esté disponible.
2. **Intervención del sistema operativo:**
   * Una vez concedido el acceso, el sistema operativo **copia el búfer** del espacio de usuario al **espacio de kernel**, ya que el kernel necesita acceso directo a los datos.
   * Se verifica si la impresora está lista. Si no lo está, el sistema espera.
3. **Transferencia de caracteres:**
   * Cuando la impresora está disponible, se copia el **primer carácter** al **registro de datos** del dispositivo. Esto activa la impresora.
   * La impresora puede no imprimir de inmediato, ya que algunas almacenan los datos en un búfer interno (por línea o página).
   * La CPU **sondea** continuamente el **registro de estado** de la impresora para saber si está lista para recibir otro carácter.
   * Cuando el estado indica que está lista, el siguiente carácter se transfiere, y así sucesivamente hasta que se imprime toda la cadena.
4. **Ciclo de impresión:**
   * Este proceso de sondeo continuo se repite hasta completar la operación.
   * Finalmente, el control se devuelve al proceso de usuario.

**Características clave**

* El **aspecto esencial** de la E/S programada es el uso del **sondeo (polling)** o **ocupado en espera**, donde la CPU verifica constantemente el estado del dispositivo sin hacer otra cosa mientras tanto.
* Esto implica que la CPU **no puede realizar otras tareas** hasta que toda la operación de E/S haya terminado.

**Ventajas y desventajas**

* **Ventajas:**
  + Simplicidad de implementación.
  + Útil en sistemas embebidos o dedicados donde la CPU no tiene otra carga de trabajo.
  + Puede ser eficiente si el tiempo de E/S es muy corto (por ejemplo, solo copiar a un búfer).
* **Desventajas:**
  + Muy **ineficiente en sistemas multitarea**, ya que **bloquea la CPU** durante toda la operación de E/S.
  + No permite aprovechar la CPU para otras tareas mientras se espera a que el dispositivo esté listo.

La E/S programada es adecuada para sistemas simples o de propósito único, pero no es práctica en entornos multitarea o de alto rendimiento. En esos casos, se requieren mecanismos más eficientes y prácticos como **interrupciones** o **DMA**, que se analizaran en secciones posteriores.

5.2.3 E/S manejadas por interrupciones

Otra manera en la que se puede realizar la E/S desde el punto de vista del sistema operativo es la **E/S manejada por interrupciones**, la cual mejora la eficiencia del sistema al permitir que la CPU no espere activamente a que un dispositivo esté listo, como en la E/S programada. En lugar de eso, el dispositivo interrumpe a la CPU solo cuando necesita atención, por ejemplo, al imprimir una cadena, el sistema operativo copia los datos del proceso al espacio de kernel y envía el primer carácter a la impresora. Luego, la CPU puede ejecutar otro proceso mientras la impresora trabaja. Cuando la impresora termina de imprimir un carácter, **genera una interrupción**.

La interrupción detiene temporalmente el proceso en ejecución, se ejecuta una rutina especial que envía el siguiente carácter (si queda alguno), y luego se reanuda el proceso interrumpido. Si no hay más caracteres, se desbloquea el proceso que pidió la impresión.

Este método **libera a la CPU de estar ocupada en espera**, lo que mejora el rendimiento del sistema, especialmente en entornos multitarea.

5.2.4 E/S usando DMA

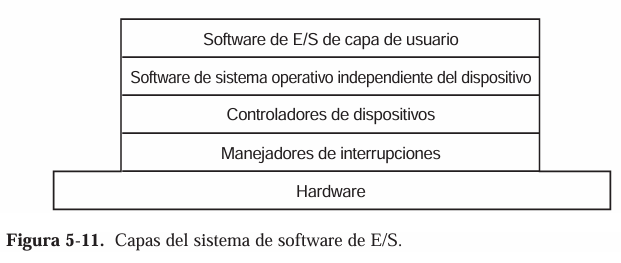
Una limitación de la E/S controlada por interrupciones es que genera **una interrupción por cada carácter**, lo cual consume tiempo de CPU. Para ello, se puede optimizar este proceso se utiliza **DMA (Acceso Directo a Memoria)**, donde un **controlador de DMA** se encarga de transferir los datos directamente entre la memoria y el dispositivo de E/S, **sin involucrar a la CPU en cada paso**. La CPU solo interviene al inicio para configurar la transferencia y al final para recibir una interrupción que indica que todo el búfer se ha procesado. Esto **reduce drásticamente el número de interrupciones**, pasando de una por carácter a una por cada bloque o búfer completo. Esto representa una mejora considerable, especialmente cuando se manejan grandes cantidades de datos o cuando las interrupciones son costosas.

Sin embargo, **el DMA no es perfecto**: el controlador de DMA suele ser más lento que la CPU. Si no puede manejar el dispositivo a su máxima velocidad, o si la CPU no tiene otra tarea útil durante la espera, puede ser más eficiente usar E/S por interrupciones o incluso E/S programada.

A pesar de esto, **en la mayoría de los casos, DMA es preferible**, ya que **libera a la CPU** para realizar otras tareas mientras se realiza la E/S. El concepto básico es similar a la E/S programada, pero **el trabajo lo hace el controlador de DMA en lugar de la CPU**, lo cual requiere hardware adicional.

5.3 Capas de Software

Por lo general, el software de E/S se organiza en cuatro capas. Cada capa tiene una función bien definida que realizar, y una interfaz bien definida para los niveles adyacentes. La funcionalidad y las interfaces difieren de un sistema a otro, por lo que el análisis que veremos a continuación, que examina todas las capas empezando desde el inferior, no es específico de una sola máquina.



5.3.1 Manejador de Interrupciones

Aunque la E/S programada es útil en algunos casos, en la mayoría de las operaciones se utilizan interrupciones, que son inevitables y deben manejarse de forma eficiente y discreta dentro del sistema operativo.

Manejo básico de interrupciones:

El controlador que inicia la E/S se bloquea hasta que la operación se completa mediante:

o down en un semáforo

o wait en una variable de condición

o receive de un mensaje

• Al completarse la operación, el procedimiento de interrupción:

o Desbloquea al controlador (por up, signal o envío de mensaje).

o Permite que el controlador reanude su ejecución.

o Idealmente, los controladores deben funcionar como procesos del kernel con sus propios estados.

Pasos generales del procesamiento de una interrupción:

1. Guardar los registros no salvados por el hardware.

2. Establecer el contexto del procedimiento de servicio (TLB, MMU, tablas).

3. Crear una pila para el procedimiento.

4. Reconocer la interrupción o rehabilitarla si no hay controlador centralizado.

5. Copiar registros a la tabla de procesos.

6. Ejecutar el procedimiento de servicio (leer registros del dispositivo).

7. Elegir el siguiente proceso a ejecutar (posible cambio por prioridad).

8. Configurar la MMU y el TLB para el nuevo proceso.

9. Cargar los registros y el PSW del nuevo proceso.

10. Comenzar la ejecución del nuevo proceso.

5.3.2 Controladores de los dispositivos (drivers)

Los controladores de dispositivos (drivers) son programas que permiten al sistema operativo comunicarse con el hardware. Cada dispositivo de E/S tiene su propio conjunto de **registros** que se utilizan para enviar comandos y leer su estado. La estructura y funcionalidad de estos registros varía mucho entre dispositivos.

Por ejemplo:

* Un **ratón** informa sobre desplazamientos y botones presionados.
* Un **disco** requiere control sobre sectores, pistas, cabezas, motores, etc.

Dado que los dispositivos son tan variados, cada uno necesita **su propio código especializado**, es decir, su **driver**, que normalmente es proporcionado por el fabricante del hardware para distintos sistemas operativos.

**Tipos y estructura de drivers**

Un driver puede controlar:

* Un **tipo específico de dispositivo** (por ejemplo, ratón).
* Una **clase de dispositivos similares** (por ejemplo, varios discos SCSI).

Aunque en teoría un mismo driver podría controlar dispositivos distintos, esto **no es recomendable** por razones de diseño y mantenimiento.

**Ubicación y ejecución**

* Tradicionalmente, los drivers se ejecutan en el **núcleo (kernel)** del sistema operativo para acceder directamente al hardware.
* Algunos sistemas avanzados como **MINIX 3** ejecutan los drivers en **espacio de usuario**, aumentando la fiabilidad al aislar fallos de hardware del núcleo.

**Interfaz y arquitectura del sistema operativo**

El sistema operativo necesita una **arquitectura clara y modular** para permitir la instalación de drivers de terceros. Los drivers suelen ubicarse por debajo del resto del sistema operativo, sirviendo como puente hacia el hardware.

Los sistemas operativos clasifican los drivers en:

* **Dispositivos de bloque**: como discos, con datos direccionables en bloques.
* **Dispositivos de carácter**: como teclados o impresoras, que manejan flujos de caracteres.

Generalmente, existe una **interfaz estándar** que todos los drivers deben seguir, definiendo procedimientos para operaciones comunes como leer y escribir.

**Modelos de carga de controladores**

* **Modelos antiguos (UNIX)**: los drivers estaban integrados directamente en el kernel y requerían recompilación para cambios.
* **Modelos modernos (como en Windows o Linux)**: los drivers se **cargan dinámicamente** durante la ejecución, facilitando el soporte de una amplia variedad de hardware.

**Funciones del controlador**

Además de leer y escribir, un driver debe:

* Inicializar el dispositivo.
* Administrar recursos.
* Manejar eventos y errores.
* Traducir parámetros abstractos (por ejemplo, bloques lógicos) a parámetros físicos del hardware (por ejemplo, cabeza, pista y sector de un disco).

El controlador también debe gestionar el **estado del dispositivo**, verificar si está ocupado y poner en cola las solicitudes si es necesario. Si el dispositivo está disponible, se le envían comandos apropiados, escritos en sus registros.

**Flujo típico de operación**

1. Validación de parámetros de entrada.
2. Traducción de direcciones lógicas a físicas (si aplica).
3. Verificación del estado del dispositivo.
4. Envío de comandos.
5. Espera (bloqueo) o ejecución inmediata.
6. Verificación de errores tras la operación.
7. Paso de datos al software superior.
8. Procesamiento de nuevas solicitudes o bloqueo en espera.

**Complejidades reales**

En la práctica, el diseño de drivers se complica por varios factores:

* **Reentrancia**: un driver puede ser interrumpido mientras está en ejecución por el mismo dispositivo, por lo que debe poder manejar múltiples ejecuciones concurrentes.
* **Conexión en caliente**: en sistemas que permiten agregar o quitar dispositivos mientras están encendidos, los drivers deben gestionar la aparición/desaparición dinámica de hardware.
* **Gestión de recursos**: al agregar/quitar dispositivos, puede ser necesario reasignar líneas de interrupción u otros recursos compartidos.

**Interacción con el kernel**

Aunque los drivers no pueden hacer llamadas al sistema directamente, pueden utilizar ciertas **funciones del kernel** como:

* Asignación de memoria.
* Control de la MMU.
* Gestión de temporizadores.
* Interacción con el controlador de DMA y de interrupciones.

5.3.3 Software en modo usuario para E/S

Aunque la mayor parte del software de entrada/salida (E/S) reside dentro del sistema operativo, también existe una porción significativa que opera en el **modo de usuario**, fuera del kernel. Esta porción incluye tanto **procedimientos de biblioteca** como programas completos ejecutándose como **procesos de usuario**.

**Procedimientos de biblioteca**

Las **llamadas al sistema**, incluyendo las de E/S, suelen hacerse a través de funciones de biblioteca. Por ejemplo, en un programa en C, la llamada:

cuenta = write(da, buffer, nbytes);

es una función de biblioteca (en este caso, write) que se vincula con el programa en tiempo de compilación y forma parte del binario en memoria durante la ejecución. Esta colección de funciones forma parte esencial del subsistema de E/S.

Algunas de estas funciones hacen algo más que invocar directamente al sistema operativo. Por ejemplo, la función printf no sólo pasa datos al sistema, sino que **da formato** a la salida. Una llamada como:

printf("El cuadrado de %3d es %6d\n", i, i\*i);

transforma variables en una cadena con formato específico antes de escribirla en pantalla usando write.

Por otro lado, scanf es una función de entrada que lee datos y los almacena en variables según un patrón de formato, utilizando también funciones de E/S. Todas estas funciones forman parte de la **biblioteca estándar de E/S** y operan en **modo usuario**.

**Sistemas de colas (spooling)**

No todo el software de E/S en modo usuario son funciones de biblioteca. Otra categoría importante incluye **programas dedicados** como los **demonios** de E/S que gestionan dispositivos dedicados en sistemas multiprogramados mediante un mecanismo llamado **spooling** (sistema de colas).

Un ejemplo clásico es el uso de **impresoras**. Permitir que cualquier proceso abra el dispositivo de la impresora directamente es problemático. Un proceso podría bloquear la impresora si no la utiliza inmediatamente. Para evitar esto, se emplea un sistema en el que:

* El usuario crea un archivo que desea imprimir.
* Este archivo se coloca en un **directorio especial de cola de impresión**.
* Un **demonio de impresión**, el único con permiso para acceder directamente a la impresora, se encarga de imprimir secuencialmente los archivos en la cola.

Este enfoque protege el dispositivo contra el uso irresponsable y asegura una utilización ordenada.

**Spooling en redes y otros dispositivos**

El modelo de colas también se usa para **E/S en redes**. Por ejemplo, para enviar archivos a través de una red, el archivo se deposita en una **cola de red**. Luego, un **demonio de red** toma el archivo y lo transfiere.

Una aplicación notable de este concepto fue **USENET**, una red de distribución de noticias con miles de grupos temáticos. Los mensajes publicados se colocaban en colas antes de transmitirse automáticamente a otras máquinas. Todo esto se manejaba por software en modo usuario, no por el núcleo del sistema operativo.

**Resumen del sistema de E/S**

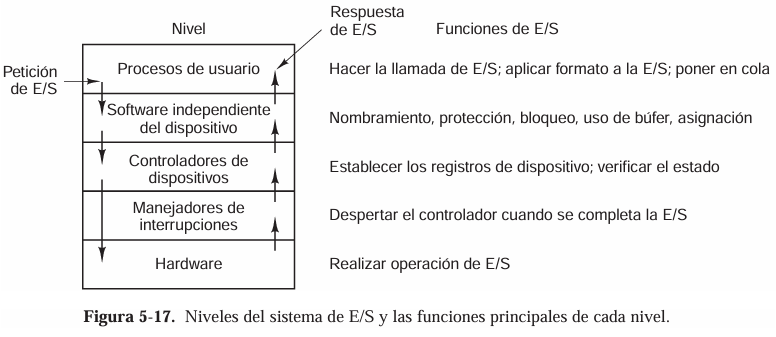
La figura 5-17 (descrita en el texto) ilustra el modelo completo de E/S como una jerarquía de niveles:

1. **Hardware**
2. **Manejadores de interrupciones**
3. **Controladores de dispositivos**
4. **Software independiente del dispositivo**
5. **Procesos de usuario**

Las **flechas** entre niveles representan el flujo de control. Por ejemplo, cuando un usuario solicita leer un archivo, el sistema operativo maneja la llamada y el software independiente del dispositivo revisa si el bloque está en la caché de búfer. Si no lo está, se hace una solicitud al controlador del dispositivo para leerlo del disco, y el proceso se bloquea hasta que la operación termine.

Cuando el **dispositivo** (como el disco) termina su tarea, **genera una interrupción**. El **manejador de interrupciones** identifica el dispositivo, extrae su estado y despierta al proceso bloqueado para que finalice su operación y continúe.

Este resumen conserva la riqueza del contenido original y presenta de forma clara los diferentes componentes del software de E/S en modo usuario y su interacción con el sistema operativo.



# Bibliografía:

1. Silberschatz, A., Galvin, P. B., & Gagne, G. (1983). *Operating system concepts*. <http://ci.nii.ac.jp/ncid/BA35822586>
2. *¿Qué es un controlador de dispositivo? | Neoguias*. (2019, 27 mayo). Neoguias | Sistemas, Aplicaciones, Dispositivos y Programación A Todos los Niveles. <https://www.neoguias.com/que-es-controlador-dispositivo/#Mas_informacion_acerca_de_los_controladores_de_dispositivo>
3. *Unidad 5 Administración de Dispositivos de E/S 5.1 Principios de*. (2015, noviembre 13). studylib.es. <https://studylib.es/doc/708953/unidad-5-administraci%C3%B3n-de-dispositivos-de-e-s-5.1-princi...?p=7>
4. *Tanenbaum, A. S., & Van Steen, M. (2009). Sistemas operativos modernos (3.ª ed., A. V. R. Elizondo, Trad.; J. R. R. Sánchez & A. J. Govea, Revisión técnica). Pearson Educación.* [*https://www.pearson.com*](https://www.pearson.com)