Actividad 2: Gestión de Entrada/Salida y de Memoria (Material complementario)

El sistema operativo es el orquestador de tu computadora, y dos de sus responsabilidades más críticas son:

1. **Gestionar la Entrada/Salida (E/S):** Permitir que el software (tus programas) interactúe con el hardware (teclado, disco, impresora, etc.).
2. **Gestionar la Memoria:** Asignar y proteger el espacio en la RAM para que los programas puedan ejecutarse sin interferencias.

Ambas funciones son vitales para la estabilidad, el rendimiento y la seguridad del sistema.

1. Gestión de Entrada/Salida (E/S): El Puente entre Software y Hardware

La E/S es cómo tu computadora "habla" con el mundo exterior y con sus propios componentes internos. Desde que pulsas una tecla hasta que guardas un archivo, todo es una operación de E/S. El sistema operativo gestiona este flujo de información a través de una **jerarquía de E/S**, una serie de capas que abstraen la complejidad del hardware para los programas.

1.1. Jerarquía de Entrada/Salida: Las Capas de Abstracción

Piensa en la jerarquía de E/S como una serie de traductores, cada uno entendiendo un nivel de lenguaje diferente, desde el más abstracto (lo que tú ves) hasta el más concreto (el hardware).

- **Aplicaciones de Usuario (Capa Superior):**
    - Son tus programas (navegador web, procesador de texto, juego).
    - Ellos no "saben" cómo funciona un disco duro. Solo hacen solicitudes de alto nivel, como "guardar este documento" o "leer estos datos de internet".
    - Utilizan las **Bibliotecas de Entrada/Salida** para comunicarse.
- **Bibliotecas de Entrada/Salida (API de E/S):**
    - Son conjuntos de funciones estándar (como `fopen()`, `fclose()`, `printf()` en C, o métodos equivalentes en Python/Java).
    - Proporcionan una interfaz consistente para los programadores. Tú llamas a `fopen("mi_archivo.txt", "w")` y la biblioteca se encarga de llamar al sistema operativo.

    - Están por encima del sistema operativo en el sentido de que son parte del entorno de programación, pero dependen de las **llamadas al sistema** del SO.
- **Sistema de Archivos:**
    - Una capa crucial que organiza los datos en los dispositivos de almacenamiento.
    - Traduce las solicitudes de "abrir archivo" o "leer bloque X" en operaciones lógicas sobre el disco.
    - Gestiona directorios, nombres de archivos, permisos y la estructura de los datos en el disco (FAT32, NTFS, EXT4, etc.).
- **Sistema de E/S Genérico (Device-Independent I/O Software):**
    - Esta es una capa dentro del **kernel** del sistema operativo.
    - Se encarga de tareas comunes a todos los dispositivos, independientemente de su tipo:
        - **Cola de Solicitudes de E/S:** Gestiona el orden en que las solicitudes llegan a los dispositivos.
        - **Buffering:** Utiliza áreas de memoria temporal (buffers) para compensar las diferencias de velocidad entre la CPU/RAM y los dispositivos de E/S (ej. el teclado es lento, el disco es más rápido pero aun así más lento que la RAM). Cuando lees un archivo, los datos se leen en bloques en un buffer antes de ser entregados al programa.
        - **Caching:** Almacena copias de datos usados frecuentemente en una memoria más rápida (la caché) para acelerar futuros accesos. Una caché de disco guarda bloques de datos que ya se leyeron por si se necesitan de nuevo.
        - **Asignación de Dispositivos:** Decide qué proceso tiene acceso a un dispositivo en un momento dado.
        - **Manejo de Errores:** Gestiona los errores que pueden ocurrir durante las operaciones de E/S.
- **Controladores de Dispositivos (Device Drivers - Software):**
    - **¡Atención a la distinción!** En el contexto de un sistema operativo, un **controlador de dispositivo (driver)** es un **programa de software** que permite al sistema operativo comunicarse con un hardware específico.

    - Traduce las solicitudes genéricas del Sistema de E/S Genérico en **instrucciones específicas** que el hardware del dispositivo puede entender. Por ejemplo, el driver de una impresora sabe cómo enviar los comandos exactos para que la impresora imprima una página.
    - Cada dispositivo requiere su propio driver, escrito por el fabricante del hardware.
- **Manejadores de Interrupciones (Interrupt Handlers):**
    - Parte del kernel, estrechamente ligada a los drivers.
    - Cuando un dispositivo de hardware completa una tarea o necesita atención (ej. una tecla presionada, el disco terminó de leer), genera una **interrupción de hardware**.
    - Los manejadores de interrupciones son rutinas especiales en el kernel que se activan para procesar estas interrupciones rápidamente y devolver el control al proceso que estaba esperando.
- **Hardware (Capa Inferior):**
    - Los dispositivos físicos (teclado, ratón, disco duro, tarjeta de red, impresora, pantalla, etc.).
    - También incluye los **controladores de dispositivo (físicos o controladoras)** que son chips o circuitos en la placa base o en el propio dispositivo (ej. la controladora SATA en la placa madre) que interpretan las señales eléctricas y gestionan el flujo de datos físicos. **No confundir con los "drivers" de software.**

1.2. Anillos de Protección: Seguridad en la Interacción con el Hardware

Los **anillos de protección** son un mecanismo de seguridad implementado por el hardware (CPU) y utilizado por el sistema operativo para controlar el nivel de acceso que tienen los diferentes programas a los recursos del sistema.

- **Anillo 0 (Modo Kernel):** El nivel de privilegio más alto. Las instrucciones ejecutadas en este anillo tienen acceso completo a todo el hardware y a todas las direcciones de memoria. El **kernel del sistema operativo** reside y se ejecuta en este anillo.
- **Anillo 3 (Modo Usuario):** El nivel de privilegio más bajo. Las aplicaciones de usuario se ejecutan aquí. No tienen acceso directo al hardware ni a las áreas protegidas de la

memoria. Si una aplicación necesita algo privilegiado, debe hacer una **llamada al sistema**, que es un cambio controlado al modo kernel.

- **¿Dónde deben ejecutarse los Controladores de Dispositivos (Drivers de Software)?**
    - Los drivers de software necesitan interactuar muy de cerca con el hardware. Por lo tanto, necesitan un nivel de privilegio que les permita hacerlo.
    - Tradicionalmente, en la mayoría de los sistemas operativos (como Linux o Windows), los **drivers de dispositivos se ejecutan en el Anillo 0 (modo kernel)**. Esto les da el acceso directo que necesitan.
    - **El Riesgo:** Sin embargo, esto también los convierte en un punto crítico de fallo. Un driver mal diseñado o malicioso puede comprometer todo el sistema, ya que opera con los mismos privilegios que el propio kernel. Esto es una de las razones de las "pantallas azules" o "kernel panics".
    - **Soluciones y Arquitecturas Modernas (como los Microkernels):** Algunos diseños de sistemas operativos (como los **Microkernels**) intentan mover los drivers a anillos de protección intermedios o incluso al **modo usuario**. Esto mejora la modularidad y la seguridad (si un driver falla, solo se cae ese driver, no todo el sistema), pero puede introducir una ligera sobrecarga de rendimiento debido a las comunicaciones adicionales entre los diferentes anillos.
    - El objetivo es encontrar un equilibrio entre la necesidad de acceso del driver al hardware y la seguridad y estabilidad del sistema.

2. Gestión de Memoria: El Espacio Vital para los Procesos

La **gestión de memoria** es el subsistema del sistema operativo encargado de controlar y coordinar la memoria principal (RAM). Su objetivo es optimizar el uso de la RAM, proteger los procesos entre sí y permitir que se ejecuten programas incluso si son más grandes que la memoria física disponible.

2.1. Problemas Fundamentales: Protección y Reubicación

1. **Protección:**
    - **Problema:** ¿Cómo evitamos que un proceso (o un error en un proceso) acceda o corrompa la memoria de otro proceso o la del propio sistema operativo?

- **Solución:** El SO, con la ayuda del hardware (la **Unidad de Gestión de Memoria - MMU** en la CPU), impone límites. Cada proceso tiene su propio espacio de memoria aislado.

2. **Reubicación:**
    - **Problema:** Cuando se carga un programa en memoria, ¿dónde se coloca? Queremos que el SO pueda ubicarlo en cualquier lugar disponible en la RAM, sin que el programa necesite saber dónde estará. Además, si un programa tiene que ser movido (reubicado) en la memoria mientras se ejecuta (para hacer espacio a otro), ¿cómo lo hacemos de forma transparente para el programa?
    - **Solución:** La **traducción de direcciones virtuales a reales**. Los programas trabajan con **direcciones virtuales (lógicas)**, y el hardware se encarga de mapearlas a **direcciones reales (físicas)** en la RAM.

2.2. Traducción de Direcciones: El Motor de la Gestión de Memoria

El procesador utiliza la **Unidad de Gestión de Memoria (MMU)** para traducir las direcciones virtuales generadas por el programa a direcciones físicas en la RAM. Esta traducción es transparente para el programa.

2.3. Intercambio (Swapping): Mover Procesos Completos

El intercambio es uno de los métodos más básicos para la gestión de memoria y la multitarea.

- **Idea Principal:** Mueve procesos completos entre la **RAM** (memoria principal) y el **almacenamiento secundario** (disco duro o SSD, que actúa como "espacio de intercambio" o "swap space").
- **Funcionamiento con Registros Base y Límite:**
    - **Registro Base:** Contiene la dirección física inicial en la RAM donde un proceso está cargado.
    - **Registro Límite:** Define el tamaño máximo del espacio de memoria asignado a ese proceso.
    - **Traducción:** Cuando el programa genera una **dirección virtual (offset)**, la MMU la compara primero con el Registro Límite para la **protección** (si es mayor, hay un error). Si es válida, la suma al Registro Base para obtener la **dirección física real**.

- Direccio´n_Fı´sica=Registro_Base+Direccio´n_Virtual

- **Limitaciones:**
  - **Lento:** Mover procesos completos de RAM a disco y viceversa es una operación muy lenta.
  - **Fragmentación Externa:** Con el tiempo, la memoria RAM puede quedar con pequeños "agujeros" libres entre procesos, lo que impide cargar procesos grandes, incluso si la suma de los agujeros es suficiente.
  - **Ineficiente:** Un proceso grande puede ocupar mucha RAM, incluso si solo usa una pequeña parte de ella en un momento dado.

2.4. Paginación: Dividir para Conquistar (El Estándar Moderno)

La paginación es el método predominante en los sistemas operativos modernos porque es mucho más flexible y eficiente.

- **Idea Principal:** Divide tanto el **espacio de direcciones virtuales** de un proceso como la **memoria física (RAM)** en bloques de tamaño fijo:
  - **Páginas:** Bloques de tamaño fijo en el espacio de direcciones virtuales de un proceso.
  - **Marcos (Frames):** Bloques de tamaño fijo en la memoria física (RAM), del mismo tamaño que las páginas.
- **Traducción de Dirección Virtual a Real con Tablas de Páginas:**
  - **Dirección Virtual:** Una dirección virtual generada por el programa se divide en dos partes:
    - **Número de Página (Page Number):** Identifica a qué página virtual pertenece la dirección.
    - **Desplazamiento dentro de la Página (Offset):** Indica la posición exacta dentro de esa página.
  - **Tabla de Páginas:** Cada proceso tiene su propia **tabla de páginas**, una estructura de datos mantenida por el sistema operativo (y a menudo almacenada en la RAM). Esta tabla mapea cada **número de página virtual** a un **número de marco físico** en la RAM.
  - **Proceso de Traducción:**

- La MMU toma el **número de página virtual**.
- Consulta la **tabla de páginas** para encontrar el **número de marco físico** correspondiente.
- Combina el **número de marco físico** con el **desplazamiento dentro de la página** para construir la **dirección física real**.
- Direccio´n_Fı´sica=(Nu´mero_Marco_Fı´sico×Taman~o_Pa´gina) +Desplazamiento

- **Protección en Paginación:**
  - Cada entrada en la tabla de páginas no solo contiene el número de marco, sino también **bits de permisos** (lectura, escritura, ejecución).
  - Si un proceso intenta realizar una operación (ej. escribir) en una página para la que no tiene permiso, la MMU detecta la violación y genera una **excepción (Page Fault)**, que el sistema operativo maneja (generalmente terminando el proceso).
  - También hay un bit de **presencia/ausencia** que indica si la página está actualmente en la RAM o ha sido "paginada" al disco. Si se intenta acceder a una página no presente, se genera una interrupción que le dice al SO que debe cargarla.
- **Ventajas de la Paginación sobre el Intercambio:**
  - **No Contiguo:** Las páginas de un proceso **no necesitan estar en bloques contiguos** en la memoria física. Pueden estar dispersas por toda la RAM, lo que elimina la **fragmentación externa**.
  - **Memoria Virtual Eficiente:** La paginación es la base de la **memoria virtual**. Permite que un programa use un espacio de direcciones virtuales mucho más grande que la RAM física disponible. Solo las páginas que se están usando activamente se cargan en RAM; el resto pueden estar en el disco de intercambio (swap).
  - **Mejor Uso de la RAM:** Permite que más procesos se ejecuten simultáneamente, ya que solo se cargan las partes necesarias de cada proceso en la RAM.
  - **Eficiencia:** Aunque la traducción de direcciones es más compleja, la MMU del hardware la realiza a velocidades muy altas. Además, se utilizan **TLBs**

(Translation Lookaside Buffers), cachés especiales dentro de la MMU, para acelerar aún más las traducciones de páginas usadas frecuentemente.

La gestión de E/S y de memoria son funciones vitales del sistema operativo. La jerarquía de E/S permite una interacción estructurada y segura con el hardware, mientras que la paginación (y en menor medida el intercambio) resuelve los desafíos de protección y reubicación, haciendo posible la multitarea eficiente y segura en cualquier computadora moderna.